UNIVERSIDAD NACIONAL DEL ALTIPLANO

ESCUELA DE POSGRADO

DOCTORADO EN CIENCIAS DE LA COMPUTACIÓN

TESIS

DEEP LEARNING PARA LA VISIÓN ARTIFICIAL E IDENTIFICACIÓN DEL PERSONAL ADMINISTRATIVO Y DOCENTE DE LA UNIVERSIDAD NACIONAL MICAELA BASTIDAS DE APURÍMAC 2018

PRESENTADA POR:

ERECH, ORDOÑEZ RAMOS

PARA OPTAR EL GRADO ACADÉMICO DE:

DOCTORIS SCIENTIAE EN CIENCIAS DE LA COMPUTACIÓN

PUNO, PERÚ

2020

UNIVERSIDAD NACIONAL DEL ALTIPLANO

ESCUELA DE POSGRADO

DOCTORADO EN CIENCIAS DE LA COMPUTACIÓN

TESIS

DEEP LEARNING PARA LA VISIÓN ARTIFICIAL E IDENTIFICACIÓN DEL PERSONAL ADMINISTRATIVO Y DOCENTE DE LA UNIVERSIDAD NACIONAL MICAELA BASTIDAS DE APURÍMAC 2018

PRESENTADA POR:

ERECH, ORDOÑEZ RAMOS

PARA OPTAR EL GRADO ACADÉMICO DE:

DOCTORIS SCIENTIAE EN CIENCIAS DE LA COMPUTACIÓN

APROBADA POR EL JURADO SIGUIENTE:

PRESIDENTE

DR. JUAN REYNALDO PAREDES QUISPE

PRIMER MIEMBRO

DR. HENRY IVAN CONDORI ALEJO

SEGUNDO MIEMBRO

DR. ELVIS ALIAGA PAYEHUANCA

ASESOR DE TESIS

DR. YALMAR PONCE ATENCIO

Puno, 07 de febrero de 2020

ÁREA: Ciencias de la Computación
TEMA: Deep Learning
LÍNEA: Computación Gráfica y Visión Computacional

DEDICATORIA

A mis seres más queridos, mis hijos: Erick, Stephanie, Arnold, Albert y Adrian.

A mis maestros del doctorado, quienes nunca desistieron de enseñarnos, sin importarles las distancias de viajes que consumían sus energías para desarrollar sus clases, que al final contribuyeron en nuestra formación.

A todo el personal docente y administrativo de la UNAMBA que apoyaron directa e indirectamente para concluir esta tesis.

Para todos ellos es esta dedicatoria de tesis.

AGRADECIMIENTOS

Agradezco de manera especial a Jesús Utrera Burgal por sus artículos y publicaciones que fueron fuente de inspiración y admiración.

Agradezco también a todos los amigo y colegas cada quien con su experiencia en su especialidad, aportaron en la ejecución de esta tesis.

Finalmente, agradezco las autoridades de la UNAMBA por haber permitido usar los ambientes de la Universidad y lograr que se concretice el trabajo de investigación.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

Pág.

ÍNDICE DE ANEXOS

Pág.

RESUMEN

El objetivo principal de esta investigación fue lograr la proporción más alta de precisión en la identificación del personal administrativo y docente de la Universidad Nacional Micaela Bastidas de Apurímac (UNAMBA), usando Deep Learning; para lo cual, la investigación fue dividida en dos etapas. La primera etapa consistió en entrenar en una arquitectura clásica como es VGG16 y este modelo evolucionó hasta el propuesto VGG16UNAMBA con la que se entrenó las imágenes obtenida a través de una cámara de video. En la segunda etapa se usó una arquitectura más moderna como lo es DenseNet121, la cual evoluciono hasta obtener DenseNet121UNAMBA, con la cual se entrenó la misma cantidad de imágenes, para finalmente, escoger la proporción más alta de precisión generada entre ambas arquitecturas.

La investigación se desarrolló en los ambientes de la UNAMBA en el año 2019 en la sede académica de Tamburco – Abancay a una cantidad de 242 personas (administrativos y docentes); este proceso requería decenas de imágenes por persona para el entrenamiento de la Red Neuronal Convolucional; por lo que se usó técnicas de *Video Scraping* y *data augmentation* para lograr 27,996 imágenes, las cuales se dividieron en 19,700 imágenes para el entrenamiento y 8,296 para la validación de los modelos.

En cuanto a los resultados en la primera etapa, se entrenó el modelo VGG16UNAMBA con el cual se logró obtener una proporción de 0.9805 de precisión; mientras que en la segunda etapa se usó a DenseNet121UNAMBA, con dicho modelo se logró una proporción de 0.9932 de precisión.

Palabras clave

Aprendizaje de máquinas, aprendizaje profundo, entrenamiento, inteligencia artificial, propagación hacia atrás, red neuronal convolucional.

ABSTRACT

The main objective of this research was to achieve the highest proportion of accuracy in the identification of the administrative and teaching staff of the National University Micaela Bastidas de Apurímac (UNAMBA), using Deep Learning; for which, the investigation was divided into two stages. The first stage consisted of training in a classical architecture such as VGG16 and this model evolved to the proposed VGG16UNAMBA with which the images obtained through a video camera were trained. In the second stage, a more modern architecture was used, such as DenseNet121, which evolved to obtain DenseNet121UNAMBA, with which the same amount of images was trained, to finally choose the highest proportion of accuracy generated between both architectures.

The research was carried out in the environments of UNAMBA in the year 2019 at the academic headquarters of Tamburco - Abancay to an amount of 242 people (administrative and teaching); this process required dozens of images per person for the training of the Convolucional Neural Network; therefore Video Scraping and data augmentation techniques were used to achieve 27,996 images, which were divided into 19,700 images for training and 8,296 for the validation of models.

As for the results in the first stage, the VGG16UNAMBA model was trained with which a proportion of 0.9805 accuracy was achieved; while DenseNet121UNAMBA was used in the second stage, with this model a proportion of 0.9932 accuracy was achieved.

Keywords

Artificial Intelligence, backpropagation, convolutional neural network, deep learning, machine learning, training.

INTRODUCCIÓN

Hoy en día la Inteligencia Artificial está abordando casi todo lo que hace un ser humano y en este camino, la identificación de personas es una tarea casi natural del ser humano, pero para una maquina es un proceso muy difícil a lo que los algoritmos planteados por el ser humano para este propósito han quedado relegados, con la evolución de las Redes Neuronales en afán de emular el funcionamiento del cerebro y lograr superar a los algoritmos tradicionales donde se definía la manera de funcionar para una tarea determinada, a lograr que el mismo algoritmo a través de miles de iteraciones en millones de neuronas artificiales, logre un aprendizaje automático para una tarea dada.

El objetivo de esta investigación fue lograr obtener una proporción alta de precisión en la identificación del personal docente y administrativo de la Universidad Nacional Micaela Bastidas de Apurímac (UNAMBA), entrenando millones de neuronas reunidas en un modelo de arquitectura especifica en consideración que existen modelos que fueron probados en bases de datos generales para la investigación pero no en imágenes reales como en este caso particular.

Este trabajo de investigación fue desarrollado y estructurado de la siguiente manera: El Capítulo I describe el marco teórico conceptual relacionado a la problemática de la Inteligencia Artificial, Machine Learning y Deep Learning, juntamente con los modelos más exitosos en las Redes Convolucionales y finalmente la forma de evaluar la performance de un modelo de Machine Learning. El capítulo II menciona el planteamiento del problema, la pregunta de investigación planteada, la justificación y los objetivos de esta investigación. El capítulo III describe la metodología utilizada en esta investigación, la forma de cómo se obtuvo los miles y miles de imágenes del personal administrativo y docente de la UNAMBA, luego se realizaron los entrenamientos en dos de los modelos más destacados en el ámbito del reconocimiento de imágenes. El capítulo IV se muestran los resultados obtenidos, analizando, describiendo, interpretando y explicando la información obtenida en esta investigación.

CAPÍTULO I

REVISIÓN DE LA LITERATURA

1.1 Marco teórico

1.1.1 Deep Learning, Machine Learning e Inteligencia Artificial

El Deep Learning, el Machine Learning y la Inteligencia Artificial, son conceptos que están íntimamente ligado, como sub conjuntos como se muestra en la figura 1 (Chollet, 2018).

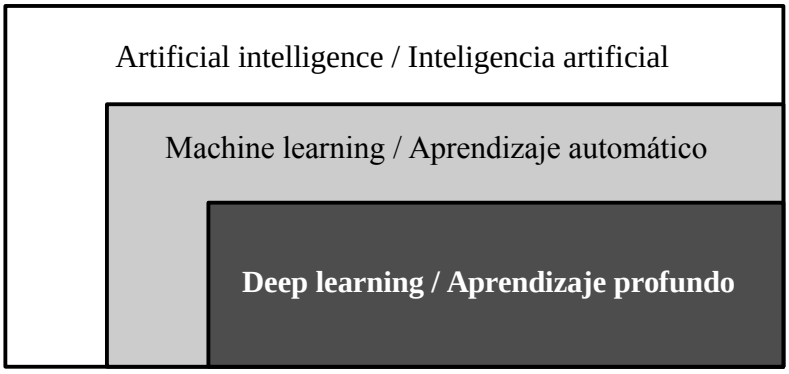


Figura 1. Inteligencia Artificial, Machine Learning y Deep Learning

Fuente: (Chollet, 2018)

Está claro que en la presente investigación nos centramos en el Deep Learning; sin embargo, es preciso conocer conceptos básicos del entorno conceptual de donde procede.

1.1.2 Inteligencia Artificial (IA)

Podemos definir a la Inteligencia Artificial como el proceso de simulación de la inteligencia humana mediante máquinas y sistemas informáticos especiales, que incluyen el aprendizaje, razonamiento y auto corrección (TutorialsPoint, 2018).

Es un campo de investigación muy amplio, donde las máquinas muestran cognitivas capacidades tales como conductas de aprendizaje, interacción proactiva con el medio ambiente, inferencia y deducción, visión por computadora, reconocimiento de voz, resolución de problemas, representación del conocimiento, percepción y muchos otros (Russell y Norvig, 2008). Más coloquialmente, la IA ve cualquier actividad donde las máquinas imitan comportamientos inteligentes típicamente mostrado por humanos. La inteligencia artificial se inspira en elementos de informática, matemática y estadística.

1.1.2.1 Los temas fundamentales de la Inteligencia Artificial

Según Ponce (2010) el campo de la IA se compone de varias áreas de estudio, las más comunes e importantes son:

a) Búsqueda de soluciones

b) Sistemas expertos

c) Procesamiento del lenguaje natural

d) Reconocimiento de modelos

e) Robótica

f) **Aprendizaje de las Máquinas (Machine Learning)**

g) Lógica

h) Incertidumbre y "lógica difusa"

Como se destaca la investigación está dentro del área del Aprendizaje de las Máquinas o Machine Learning.

1.1.2.2 Ramas de la Inteligencia Artificial

Ponce (2010), manifiesta que, existen varios elementos que componen la ciencia de la IA, dentro de los cuales se pueden encontrar tres grandes ramas:

a) Lógica difusa

b) Redes Neurales Artificiales

c) Algoritmos genéticos

Dentro de estas ramas es notorio que la investigación está centrada en las Redes Neuronales Artificiales y más concretamente en **Redes Neuronales Convolucionales**, ya

que éstas permiten el procesamiento de imágenes como se verá en temas siguientes.

1.1.2.3 Redes Neuronales Artificiales (RNA)

Ponce (2010) afirma que las RNA se definen como sistemas de mapeos no lineales cuya estructura se basa en principios observados en los sistemas nerviosos de humanos y animales.

La idea detrás de una Red Neuronal Artificial es simular el comportamiento de una red neuronal biológica. Para ello se emula con fórmulas matemáticas una neurona a la que le van a llegar señales de entrada con distintos pesos, que se sumarán, y se emitirá una señal de salida que dependerá de una determinada función de activación como se muestra en la Figura 2 (Ponce, 2010).

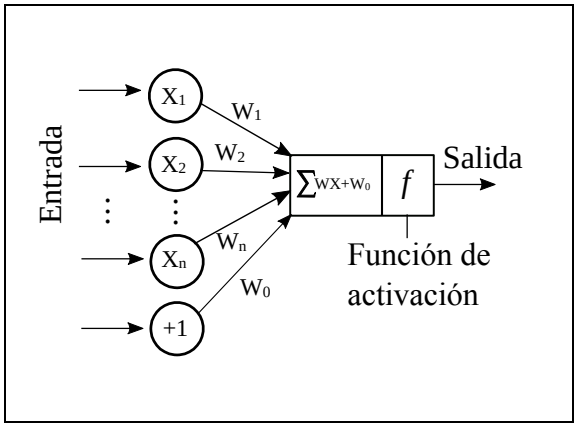


Figura 2. Neurona artificial

Fuente: (Ponce, 2010)

Existen diferentes tipos de funciones de activación las más relevantes para esta investigación son: Sigmoid (Sigmoide) y ReLU (Rectified Lineal Unit).

Sigmoid: $f(x) = \frac{2}{1+e^{-2x}} - 1$

ReLU: $f(x) = \max(0, x) = \begin{cases} 0 \; for \; x < 0 \\ x \; for \; x \geq 0 \end{cases}$

Una red neuronal artificial estará formada por un conjunto de neuronas, y que se somete a un proceso de aprendizaje, para enseñarle a reconocer formas (clasificación) o hacer predicciones (Regresión).

La función ReLU se usa en las capas de entrada y ocultas; mientras que, en la capa de salida se usa la función Sigmoid para la identificación de personas.

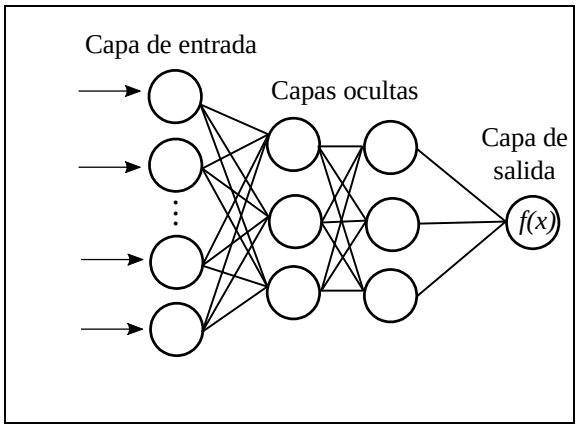


Figura 3. Red Neuronal Artificial

Fuente: (Ponce, 2010)

Cada unidad recibe entradas de otros nodos y genera una salida simple escalar que depende de la información local disponible, guardada internamente o que llega a través de las conexiones con pesos. Pueden realizarse muchas funciones complejas dependiendo de las conexiones.

1.1.3 Machine Learning

El Machine Learning o Aprendizaje Automático es un tipo de inteligencia artificial (IA) que proporciona a las computadoras la capacidad de aprender, sin ser programadas explícitamente. El aprendizaje automático se centra en el desarrollo de programas informáticos que pueden cambiar cuando se exponen a nuevos datos (Gori, 2017).

Dentro del Machine Learning existen tres enfoques para aprender, el aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por reforzamiento (Hurwitz y Kirsch, 2018).

a) Aprendizaje supervisado

El aprendizaje supervisado generalmente comienza con un conjunto establecido de datos y una cierta comprensión de cómo se clasifican esos datos; por ejemplo, si le damos abundante información de imágenes de animales (perros y gatos), y etiquetamos cada imagen como perro o gato, entonces el sistema aprenderá a identificar un gato o un perro en otra imagen cualquiera distinta con la que fue entrenado (Hurwitz y Kirsch, 2018).

b) Aprendizaje no supervisado

El aprendizaje no supervisado es más adecuado cuando el problema requiere una gran cantidad de datos sin etiqueta; por ejemplo, le damos abundante información de imágenes de (gatos y perros), pero no le decimos al sistema que son gatos o perros, por lo que la comprensión del significado detrás de estas imágenes, requiere algoritmos que pueden comenzar a comprender el significado de las imágenes para poder clasificar los "gatos o perros" en cualquier otra imagen (Hurwitz y Kirsch, 2018).

c) Aprendizaje por reforzamiento

El aprendizaje por reforzamiento es un modelo de aprendizaje conductual, donde el algoritmo recibe retroalimentación del análisis de los datos para que el usuario sea guiado hacia el mejor resultado; es decir, se aprende con estímulos de ponderación alta si se acerca al objetivo o ponderaciones menores si comete errores (Hurwitz y Kirsch, 2018).

En esta investigación se utiliza el aprendizaje supervisado.

1.1.3.1 Aprendizaje supervisado

Como vimos anteriormente, el aprendizaje supervisado está destinado a encontrar patrones en los datos que correspondan a una etiqueta que define el significado de los datos; por ejemplo, podría haber millones de imágenes de animales e incluir una explicación de qué es cada animal y luego puede crear una aplicación de aprendizaje automático que distinga a un animal de otro.

Existes dos tipos principales de problemas del Aprendizaje Automático supervisado, denominados **clasificación y regresión** (Andreas y Sarah, 2017).

Hurwitz y Kirsch (2018) afirman que cuando la etiqueta es constante, es una **regresión**; y cuando los datos provienen de un conjunto finito de valores, se conoce como **clasificación**. En esencia, utilizar la regresión para el aprendizaje supervisado lo ayuda a comprender la correlación entre las variables. Un ejemplo de aprendizaje supervisado, mediante el uso del análisis de regresión, es el pronóstico del tiempo; el pronóstico del tiempo tiene en cuenta los patrones climáticos históricos conocidos y las condiciones actuales para proporcionar una predicción sobre el clima y otro ejemplo de aprendizaje supervisado, mediante el uso de análisis de clasificación sería la clasificación de animales.

Los algoritmos se entrenan utilizando ejemplos pre-procesados, y en este punto, el rendimiento de los algoritmos se evalúa con datos de validación. Ocasionalmente, los patrones que se identifican en un subconjunto de datos no se pueden detectar en la población más grande de datos. Si el modelo es apto para representar solo los patrones que existen en el subconjunto de entrenamiento, entonces se crea un problema llamado *overfitting* (sobre-ajuste).

El sobre-ajuste significa que su modelo está ajustado con demasiada precisión para los datos de entrenamiento, pero no puede ser aplicable para otros conjuntos de datos diferentes al entrenado. Para protegerse contra el sobre-ajuste, las validaciones deben realizarse con datos desconocidos. El uso de datos desconocidos en el conjunto de validación puede ayudarlo a evaluar la precisión del modelo para predecir resultados. Los modelos de aprendizaje supervisados tienen una amplia aplicabilidad a una variedad de problemas, incluida la detección de identidad, soluciones a recomendaciones, reconocimiento de voz o análisis de riesgos.

En esta investigación se investiga e implementa modelos CNN de clasificación (identificación) de personas, concretamente del personal docente y administrativo de la UNAMBA.

1.1.4 Deep Learning o Aprendizaje Profundo

El Deep Learning es un sub-campo del Machine Learning en el que los algoritmos en cuestión se inspiran en la estructura y función del cerebro llamadas redes neuronales artificiales (TutorialsPoint, 2018).

Según Goodfellow et al. (2016) se refieren al Deep Learning "como un enfoque de la Inteligencia Artificial, un tipo de aprendizaje automático que alcanza gran potencia y flexibilidad mediante el aprendizaje de la representación del mundo, a través de conceptos jerárquicamente anidados. Se trata de formar conceptos complejos mediante la extracción y concatenación de conceptos muy simples". Para estos autores, el aprendizaje automático es el único enfoque viable, que permite construir sistemas de Inteligencia Artificial que pueden operar en los complicados ambientes del mundo en que vivimos.

La constante investigación que se ha tenido en el aprendizaje automático a través de los años, ha dado lugar a una gran cantidad de algoritmos y modelos, muchos relacionados entre sí y generando infinidad de aplicaciones en conjunto (Goodfellow et al., 2016).

Según García (2015) el Deep Learning es un conjunto de técnicas y procedimientos algorítmicos basados en Machine Learning para lograr que una máquina aprenda de la misma forma que lo haría un ser humano. Siendo más precisos, hablamos de una familia de algoritmos cuyo propósito es simular el comportamiento que lleva a cabo nuestro cerebro para reconocer imágenes, palabras o sonidos. Son algoritmos que funcionan en base a "un proceso por capas". El aprendizaje profundo simula el funcionamiento básico del cerebro, que se realiza a través de las neuronas.

1.1.4.1 Principales algoritmos del Deep Learning

Dependiendo del tipo de aplicación que se trate, hay que emplear el tipo de algoritmo más adecuado para ello. Según Deng (2014), el tipo de aprendizaje, de acuerdo a su arquitectura y a su finalidad que persiguen los clasifican en tres grupos:

1) **Redes profundas para aprendizaje no supervisado o generativo**

   En estas redes, el propósito principal es el análisis de patrones, la síntesis de los datos observados, o bien una agrupación, sin que se tenga una etiqueta para cada clase de patrón u objetivo.

2) **Redes profundas para aprendizaje supervisado**

   En estas redes, se cuenta con patrones conocidos y bien categorizados, con el fin de clasificar de forma directa.

3) **Redes profundas híbridas**

   Es una mezcla de las anteriores, con la meta de poder tener la capacidad de discriminar, auxiliado del aprendizaje no supervisado. Y esto podría llevar a un mejor performance que las redes supervisadas.

1.1.5 Redes Neuronales Profundas (DNN)

Una red neuronal profunda (Deep Neural Network) es una red neuronal artificial con varias capas ocultas entre las capas de entrada y salida figura 4 con la finalidad de modelar relaciones no lineales complejas (Ponce, 2010).

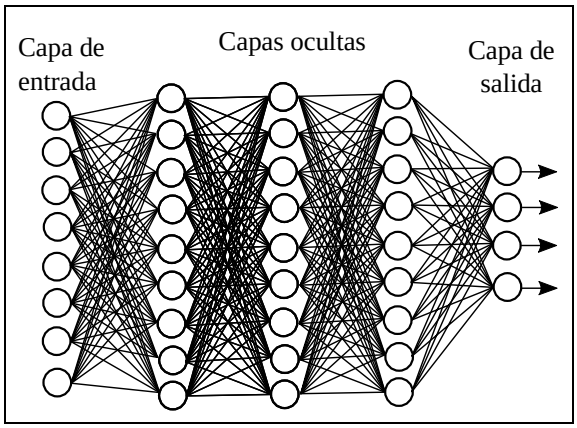


Figura 4. Arquitectura de una Red Neuronal Artificial Profunda

Fuente: (Ponce, 2010)

En TutorialsPoint (2018) se destacan dos tipos importantes de redes neuronales profundas:

- Convolutional Neural Networks (CNN) / Redes Neuronales Convolucionales
- Recurrent Neural Networks / Redes Neuronales Recurrentes

De las cuales, para los propósitos de la investigación es de suma relevancia las Redes Convolucionales.

1.1.6 Convolutional Neural Networks (CNN)

Las redes neuronales convolucionales son una variante del perceptrón multicapa, con la diferencia que las CNN's realizan operaciones de "convolución" entre los parámetros de la red y los datos de entrada, en lugar de productos punto.

Las CNNs son de los algoritmos más populares en el paradigma del Deep Learning, cuando se trata de reconocimiento de objetos en imágenes. El desempeño que han tenido se ubica entre los mejores en la actualidad. Tienen sus orígenes en las investigaciones de Lecun et al. (1998) y con el empleo de GPU's para su entrenamiento originaron un despunte importante en el estado del arte a partir del 2011, en cuanto a los resultados que se obtenían en las tareas de clasificación de objetos y reconocimiento de dígitos escritos a mano, ver Springer (2017). A partir de aquí se incrementó la investigación en este tipo de redes, hasta obtener las variantes en su arquitectura y resultados obtenidos de la actualidad.

Hearty (2016) explica que el diseño de Redes Neuronales Convolucionales se inspira en el área del cerebro (corteza visual) que procesa la información visual, que la corteza visual

tiene varias especializaciones que le permiten procesar eficazmente los datos visuales y que contiene muchas células receptoras que detectan la luz en regiones superpuestas del campo visual; asimismo, estas células receptoras están sujetas a la misma operación de convolución; es decir, todas procesan su entrada de la misma manera.

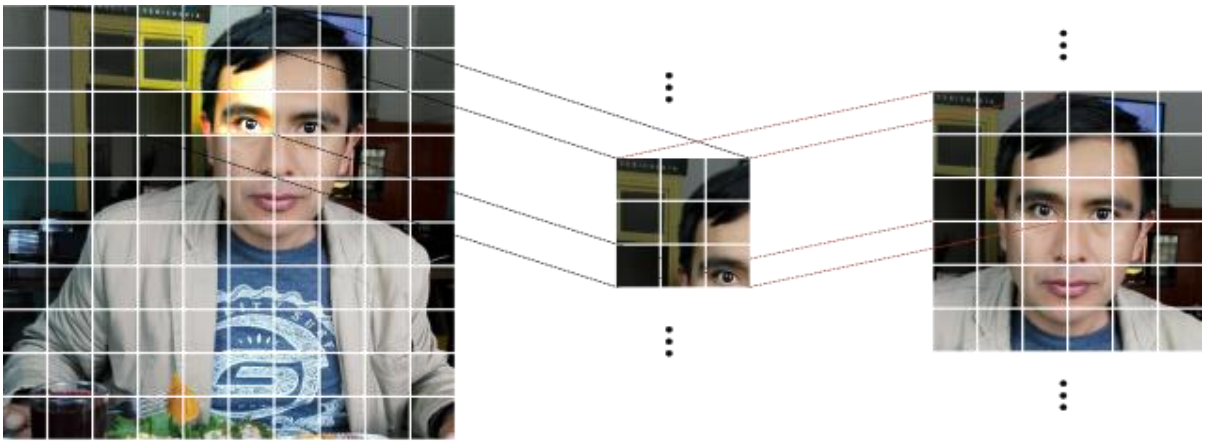
1.1.6.1 Topología de las (CNN)

La arquitectura de la Red Neuronal Convolucional es un gráfico acíclico compuesto por capas de cada vez menos nodos, donde cada capa se alimenta de la siguiente. Hearty (2016) hace dos diferencias en las CNN, en la primera diferencia hace notar que las CNN y la mayoría de las otras redes es que todas las neuronas en una conexión son idénticas; es decir, todas las neuronas poseen los mismos parámetros y valores de peso y la segunda diferencia entre las CNN y otras arquitecturas es que la conectividad entre los nodos son limitadas; en otras palabras, las entradas a un nodo dado se limitarán solo a aquellos nodos cuyos campos receptores son contiguos.

Para ilustrar esto más claramente, tomemos una imagen de entrada de ejemplo y analicemos cómo una CNN podría procesar partes de esa imagen en nodos específicos. A los nodos en la primera capa de una CNN se les asignarán subconjuntos de la imagen de entrada, en este caso, digamos que toman un subconjunto de 3 x 3 píxeles de la imagen cada uno, así nuestra cobertura cubre toda la imagen sin superposición entre las áreas tomadas como entrada por los nodos y sin espacios, generando una versión transformada de esa entrada.

Esta salida generalmente es recogida por una segunda capa de nodos. En este caso, supongamos que nuestra segunda capa está tomando un subconjunto de todas las salidas de los nodos en la primera capa. Por ejemplo, podría estar tomando un subconjunto contiguo de 6 x 6 píxeles de la imagen original; es decir, tiene un campo receptivo que cubre las salidas de exactamente cuatro nodos de la capa anterior. Esto se vuelve un poco más intuitivo cuando se explica visualmente (Hearty, 2016):

Figura 5. Convolución que alimenta a la siguiente capa como entrada



Fuente: (Hearty, 2016)

De acuerdo a Duval (2016) hay tres componentes básicos como requisito previo para definir una red convolucional básica:

a) The convolutional layer

b) The Pooling layer (optional)

c) The output layer

Las cuales lo explicamos a mayor detalle a continuación:

a) Capas de convolución (convolution layers)

Para evitar que cada nodo aprenda un conjunto impredecible de parámetros libres, los pesos en una capa se comparten en toda la capa; es decir, los filtros aplicados en una capa convolucional son un conjunto único de filtros, que se deslizan (convolucionan) a través del conjunto de datos de entrada. Esto produce un mapa de activación bidimensional de la entrada, que se conoce como mapa de características (Hearty, 2016).

El filtro en sí está sujeto a cuatro hiperparámetros:

- El tamaño (size)
- La profundidad (depth)
- La zancada (stride) y
- El relleno de ceros (zero-padding).

1) **El tamaño (size)**

Es el área del filtro que recorrerá la imagen, un filtro no necesariamente es cuadrado. Los filtros más grandes tenderán a superponerse más, y como veremos, esto puede mejorar la precisión de la clasificación; sin embargo, crucialmente, aumentar el tamaño del filtro creará salidas cada vez más grandes; por lo que, administrar el tamaño de los resultados de las capas convolucionales es un factor importante para controlar la eficiencia de una red (Hearty, 2016).

2) **La profundidad (depth)**

Define el número de nodos en la capa. La profundidad a veces se conoce como una dimensión por derecho propio; casi se relaciona con la complejidad de una imagen, no en términos de su contenido, sino en términos de la cantidad de canales necesarios para describirla con precisión, establecer el parámetro de profundidad en un valor demasiado pequeño tiende a conducir a malos resultados porque la red no tiene la profundidad expresiva (en términos de conteo de canales) requerida para caracterizar con precisión los datos de entrada; por lo que es necesario probar el valor de profundidad apropiado durante la configuración de la red mediante la optimización de hiperparámetros (Hearty, 2016).

3) **La zancada (stride)**

Es una medida de espacio entre las neuronas. Un valor de zancada de uno hará que cada elemento de la entrada (para una imagen, potencialmente cada píxel) sea el centro de una instancia de filtro. Naturalmente, esto conduce a un alto grado de superposición y resultados muy grandes. El aumento de la zancada causa una menor superposición en los campos receptivos y se reduce el tamaño de la salida (Hearty, 2016).

El siguiente diagrama muestra gráficamente tanto la profundidad como la zancada:

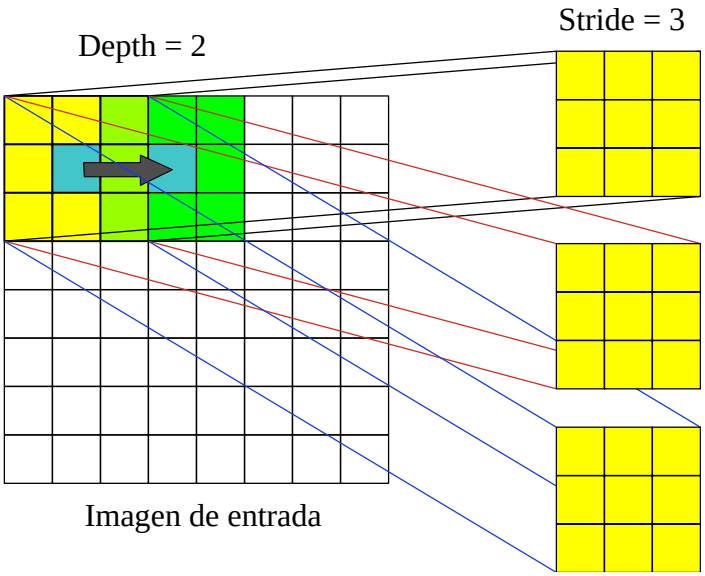


Figura 6. Depth and stride

Fuente: (Hearty, 2016)

4) El relleno cero (zero-padding),

Es el proceso de establecer los valores externos (los bordes) de cada campo en cero, lo que tiene el efecto de reducir el tamaño de salida para esa capa. Es posible establecer uno o varios píxeles alrededor del borde del campo a cero, lo que reduce el tamaño de salida en consecuencia. En términos más generales, aumentar el grado de relleno cero puede causar una disminución en la efectividad, que está vinculada a la mayor dificultad de las características de aprendizaje a través de la codificación gruesa (Hearty, 2016).

Sin embargo, el zero-padding es muy útil porque nos permite ajustar los tamaños de entrada y salida y garantizar que el tamaño de la capa de entrada y la capa de salida sean iguales; así poder gestionar fácilmente los valores de zancada y de profundidad; además, el relleno cero mejora el rendimiento, ya que, sin él, una CNN tenderá a degradar gradualmente el contenido en los bordes del filtro (Hearty, 2016).

Hearty (2016) nos propone que para calibrar el número de nodos, la zancada apropiada y el relleno para capas sucesivas cuando definimos nuestra conexión, necesitamos conocer el tamaño de la salida de la capa anterior; para lo cual, se calcula el tamaño espacial de la salida de una capa (O) en función del tamaño de la imagen de entrada (W), el tamaño del filtro (F), la zancada (S) y la cantidad de

relleno cero aplicado (P), de la siguiente manera:

$$O = \frac{W - F + 2P}{S} + 1$$

Si O no es un número entero, los filtros no se colocan en mosaico a través de la entrada de forma ordenada y en su lugar se extienden sobre el borde de la entrada. Esto puede causar algunos problemas cuando se entrena (normalmente implica lanzar excepciones) Al ajustar el valor de paso, uno puede encontrar una solución de número entero para O y entrenar de manera efectiva. Es normal que la zancada se limite a lo que sea conveniente dados los otros valores del hiperparámetro y el tamaño de entrada.

Hemos discutido los hiperparámetros involucrados en la configuración correcta de la capa convolucional, pero aún no hemos discutido el proceso de convolución en sí. La convolución es un operador matemático, como la suma o la derivación, que se usa mucho en aplicaciones de procesamiento de señales y en muchos otros contextos donde su aplicación ayuda a simplificar ecuaciones complejas (Hearty, 2016).

Hablando en términos generales, la convolución es una operación sobre dos funciones, como producir una tercera función que es una versión modificada de una de las dos funciones originales. En el caso de convolución dentro de una CNN, el primer componente es la entrada de la red. En el caso de convolución aplicada a imágenes, la convolución se aplica en dos dimensiones (el ancho y la altura de la imagen). La imagen de entrada suele tener tres matrices de píxeles, una para cada uno de los canales de color rojo, azul y verde, con valores que oscilan entre 0 y 255 en cada canal (Hearty, 2016).

La segunda entrada a la operación de convolución es el núcleo de convolución, una matriz única de números de coma flotante que actúa como filtro en las matrices de entrada. El resultado de esta operación de convolución es el mapa de características. La operación de convolución funciona deslizando el filtro a través de la entrada, calculando el producto escalar de los dos argumentos en cada instancia, que se escribe en el mapa de características. En los casos en que el paso de la capa convolucional es uno, esta operación se realizará en cada píxel de la imagen de entrada (Hearty, 2016).

La principal ventaja de la convolución es que reduce la necesidad de ingeniería de características. Crear y administrar núcleos complejos y realizar los procesos de ingeniería de características altamente especializados necesarios es una tarea exigente, que se vuelve más difícil por el hecho de que los procesos de ingeniería de características que funcionan bien en un contexto pueden funcionar mal en la mayoría de los demás (Hearty, 2016).

b) Capas de agrupación (pooling layers)

El apilamiento de capas convolucionales nos permite crear una topología que efectivamente crea características como mapas de características para datos de entrada complejos y ruidosos. Sin embargo, las capas convolucionales no son el único componente de una red profunda. Es común conectar capas convolucionales con capas de agrupación o reducción en ese orden. Su utilidad principal radica en la reducción de las dimensiones espaciales ancho y alto del volumen de entrada para la siguiente capa convolucional, principalmente utilizando una operación máxima (max-pooling), media (mean-pooling) o sumatoria (sum-pooling) y no afecta a la dimensión de profundidad del volumen (Duval, 2016).

La agrupación es una operación sobre mapas de entidades, donde se agregan múltiples valores de entidades en un solo valor,

Si no agrupamos, tendemos a encontrarnos con una gran cantidad de características; por lo cual, su uso disminuye el tamaño con una menor sobrecarga de cálculo para las próximas capas de la red; así como también, reduce el sobre ajuste (overfitting).

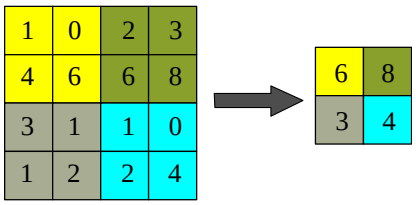


Figura 7. Max-Pooling

Fuente: (Duval, 2016)

En la figura 7 se muestra la acción de Max-pooling en una matriz 4 x 4 con una zancada (stride) de 2 pixeles, en donde obtiene el valor máximo de las submatrices.

c) Capas de salida (The output layer)

Luego de haber procesado múltiples capas de convolución y pooling, necesitamos la salida en forma de una clase; por lo que las capas de convolución y pooling simplemente serían capaces de extraer características y disminuir el número de parámetros de las imágenes reales, generalmente la capa de salida tiene una función de pérdida como la categorical cross-entropy, para calcular el error en la predicción.

Cabe mencionar también que al final de las capas convolucionales y de pooling, los modelos de redes utilizan generalmente capas completamente conectados. Esta última capa clasificadora tendrá tantas neuronas como el número de clases que se debe clasificar. Normalmente esta capa clasificadora utiliza la función "softmax" o "sigmoide", que dentro de estas usan una función de pérdida como la categorical cross-entropy (Duval, 2016).

1.1.7 Modelos de clasificación de imágenes

Como se vio anteriormente el modelo de una red neuronal convolucional.es sumamente sencillo y entendible, pero la combinación de capas a nivel longitudinal y de profundidad en modelos más complejos es la parte difícil del uso de redes neuronales convolucionales en la práctica es cómo diseñar arquitecturas de modelos que utilicen mejor los elementos básicos; por lo que es importante ver modelos que en un entorno determinado dieron buenos resultados.

1.1.7.1 Arquitecturas de red clásicas

a) LeNet-5

Esta arquitectura es una de las primeras aplicaciones exitosas y ampliamente conocida de las redes neuronales convolucionales. Esta arquitectura fue descrita por Lecun et al. (1998).

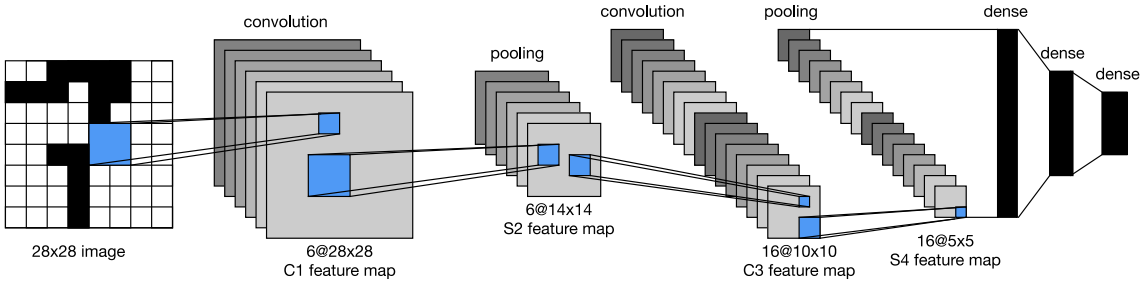


Figura 8. Arquitectura del modelo LeNet-5

Fuente: (Lecun et al., 1998)

b) AlexNet

Fue diseñado por Alex Krizhevsky, publicada con Ilya Sutskever y se convirtió en uno de los modelos que introdujo el uso de GPU para mejorar su performance. En la figura 9 se puede apreciar la comunicación entre las GPU en la capa C3, F1, F2 y la capa de salida (Krizhevsky et al., 2017).

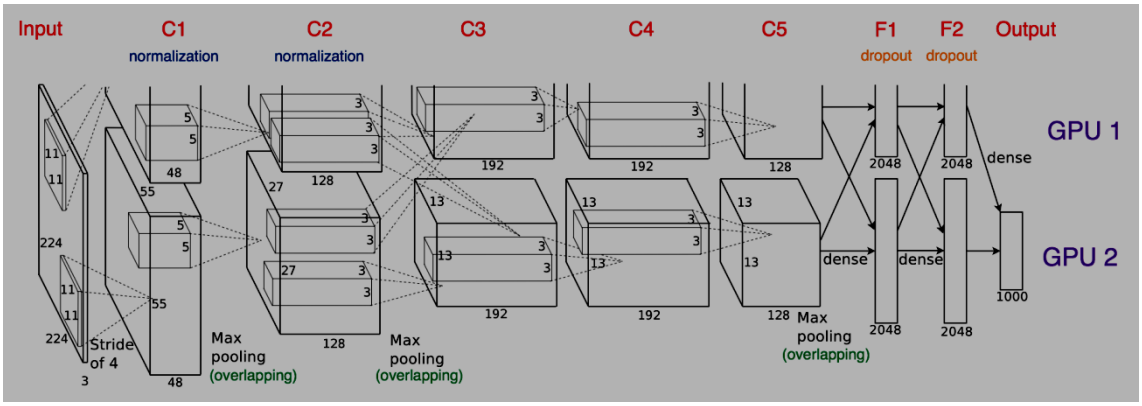


Figura 9. Arquitectura del modelo AlexNet

Fuente: (Krizhevsky et al., 2017)

c) VGG16

Es considera una de las mejores arquitecturas de modelos para la visión artificial. Lo diferente de VGG16 es el uso de capas de convolución de filtros, capas completamente conectadas y finalmente una capa de salida con softmax como función de activación (Simonyan y Zisserman, 2015).

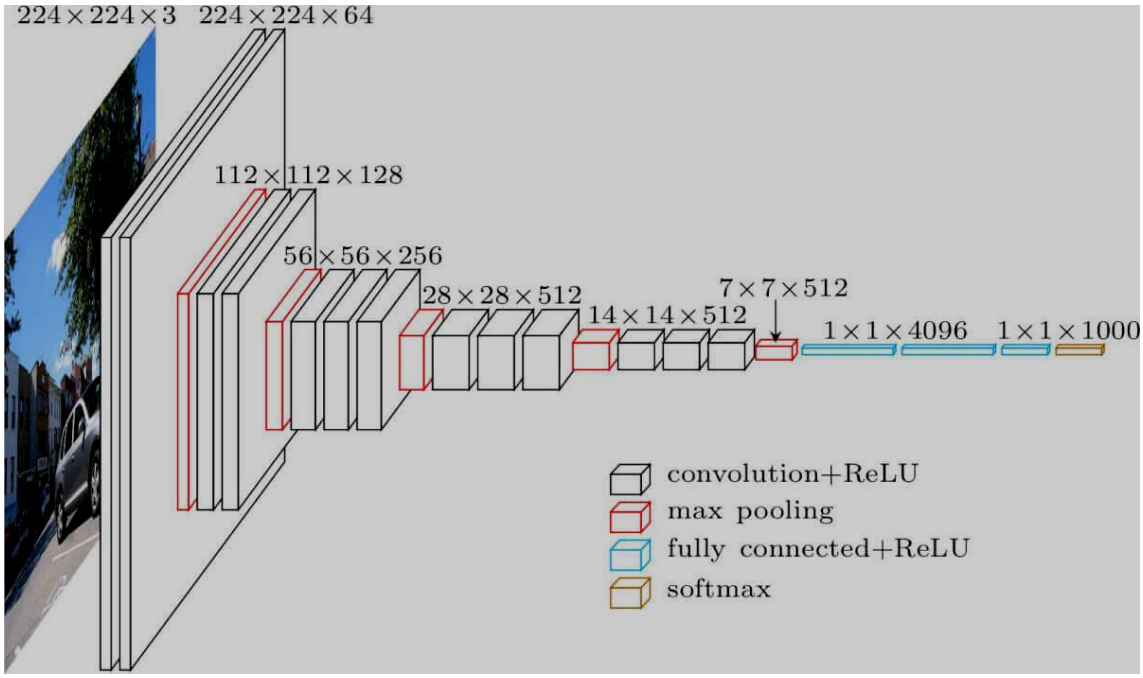


Figura 10. Arquitectura del modelo VGG16

Fuente: (Simonyan y Zisserman, 2015)

1.1.7.2 Arquitecturas de red modernas

a) Inception

Fue introducida por los investigadores de Google en el año 2014 y ocupó el primer lugar en la competencia ImageNet del 2014 por desafíos de clasificación y detección.

Debido a lo complejo del modelo es que vemos de manera comprimida para tener una idea de su arquitectura en la figura 11, el modelo se compone de una unidad básica denominada "celda de inicio" en la que se realizan una serie de convoluciones a diferentes escalas y luego se agrega los resultados. Para guardar el cálculo, se utilizan convoluciones 1x1 para reducir la profundidad del canal de entrada. Para cada celda, se usa un conjunto de filtros 1x1, 3x3 y 5x5 que pueden aprender a extraer características a diferentes escalas de la entrada. También se usa la agrupación máxima, aunque con el "relleno" mismo para preservar las dimensiones de modo que la salida se pueda concatenar adecuadamente (Szegedy et al., 2014).

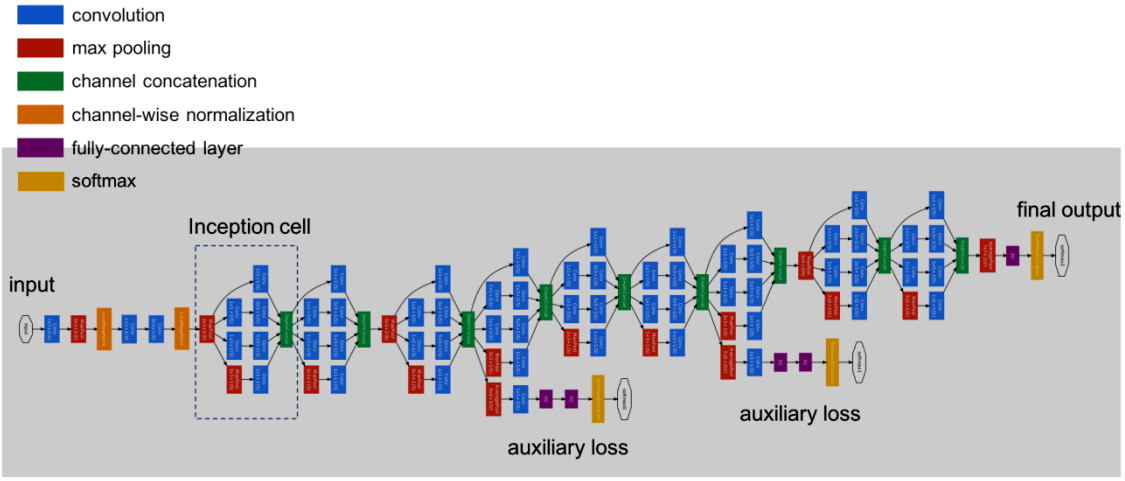


Figura 11. Arquitectura general de Inseption

Fuente: (Szegedy et al., 2014)

b) ResNet

Ante el problema de tener una red muy profunda con muchas capas ocultas para supuestamente aumentar la precisión, este proceso lleva al camino hacia el problema de gradientes de desaparición / explosión que en realidad hace que la red neuronal sea inutilizable. Entonces, ¿cómo se sabe el número correcto de capas ocultas para usar?

El entrenamiento de redes neuronales profundas con optimizadores basados en gradientes y métodos de aprendizaje puede causar la desaparición y la explosión de gradientes durante la propagación hacia atrás. Como su nombre indica, los gradientes que desaparecen es el fenómeno en el que los pesos se actualizan tan lentamente que dejan de actualizarse después de un tiempo y, como puede suponer, los gradientes explosivos son el fenómeno en el que se realizan grandes actualizaciones.

Con la ayuda de bloques residuales, Resnet aumenta el número de capas ocultas tanto como se desee sin preocuparnos por el problema de gradientes de fuga-explosión. Los bloques residuales permiten que la red conserve lo que había aprendido previamente (si no hay nada que aprender) al tener una función de ponderación de mapeo de identidad donde la salida es igual a la entrada, preservando lo que la red neuronal ha aprendido al no aplicar transformaciones decrecientes.

Por tanto los bloques residuales pueden usarse junto con las capas convolucionales para maximizar la precisión.

La función residual crea un duplicado de la entrada dada nombrada como acceso directo para preservar la salida anterior de las posibles transformaciones desastrosas. Puede ver que después de almacenar el valor original de Y, se somete a una serie de operaciones de convolución mientras intenta maximizar el aprendizaje. Como los pesos originales de Y se conservan (acceso directo), finalmente se agregan con la Y transformada para eliminar los posibles efectos negativos de las transformaciones. Aquí, mediante la eliminación de los efectos negativos, queremos decir que los pesos actúan como una función de identidad (He et al., 2015).

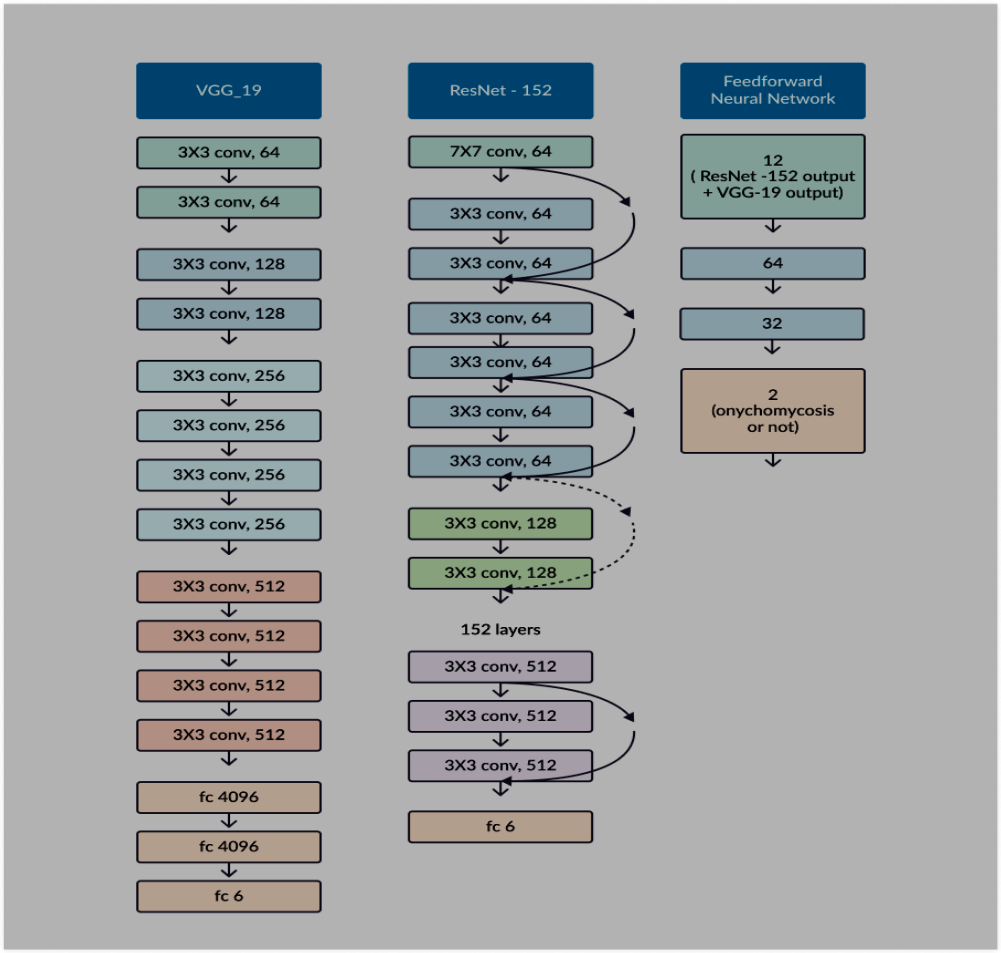


Figura 12. Arquitectura de ResNet

Fuente: (He et al., 2015)

c) DenseNet

La idea detrás de las redes convolucionales densas es simple: puede ser útil hacer referencia a mapas de características desde antes en la red. Por lo tanto, el mapa de características de cada capa se concatena a la entrada de cada capa sucesiva dentro de un bloque denso. Esto permite que las capas posteriores dentro de la red aprovechen directamente las características de las capas anteriores, fomentando la reutilización de características dentro de la red. Los autores afirman que "la concatenación de mapas de características aprendidos por diferentes capas aumenta la variación en la entrada de las capas posteriores y mejora la eficiencia" (Huang et al., 2018)

Cuando me encontré con este modelo por primera vez, pensé que tendría una cantidad absurda de parámetros para soportar las densas conexiones entre capas. Sin embargo, debido a que la red es capaz de usar directamente cualquier mapa de características anterior, los autores descubrieron que podían trabajar con profundidades de canal de salida muy pequeñas; es decir, 12 filtros por capa, reduciendo enormemente el número total de parámetros necesarios. Los autores se refieren al número de filtros utilizados en cada capa convolucional como una "tasa de crecimiento" k, ya que cada capa sucesiva tendrá k más canales que el anterior (como resultado de acumular y concatenar todas las capas anteriores a la entrada).

De acuerdo a los autores afirman que DenseNet logra un mejor rendimiento con menos complejidad que ResNet.

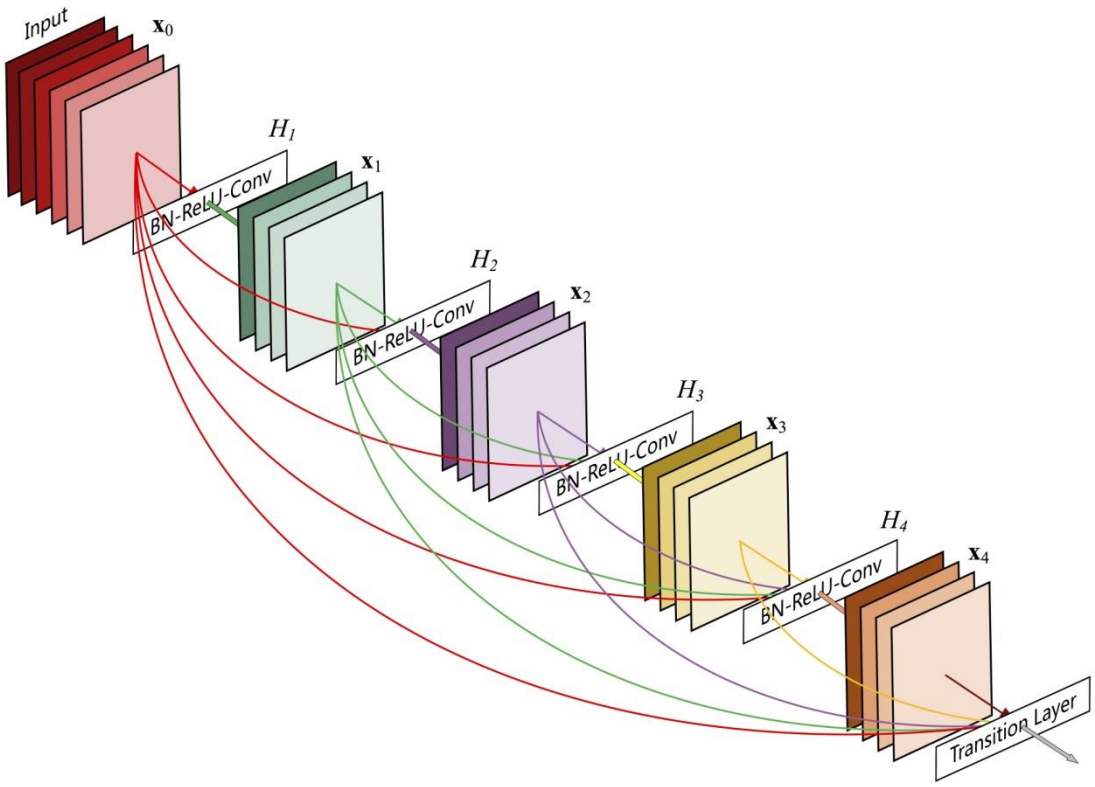


Figura 13. Arquitectura de DenseNet

Fuente: (Huang et al., 2018)

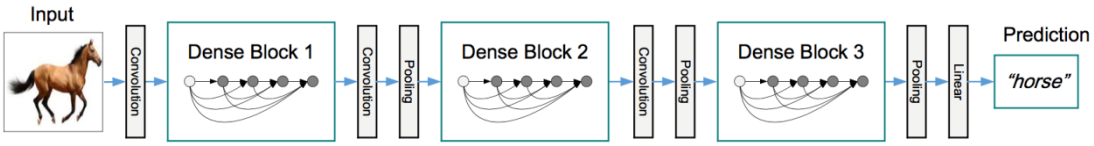


Figura 14. Arquitectura resumida de DenseNet

Fuente: (Huang et al., 2018)

1.1.8 Evaluación de la efectividad de un modelo de Deep Learning

Para evaluar de manera significativa el desempeño de nuestro modelo, debemos definir los criterios de prueba y una puntuación de rendimiento adecuada.

Para hacer esto, necesitaremos tres cosas:

a) Una comprensión de lo que nuestro modelo va a generar

b) Una medida que podemos utilizar para cuantificar el rendimiento del modelo.

c) Algunos datos objetivo que podemos usar para calificar el desempeño del modelo de acuerdo con nuestra medida

Una forma de decidir qué medida de rendimiento es adecuada para la tarea en cuestión es considerar la matriz de confusión. Una matriz de confusión es una tabla de contingencias; en el contexto del modelado estadístico, típicamente describen la predicción de la etiqueta versus las etiquetas reales. Es común generar una matriz de confusión (particularmente para problemas multiclase con más clases) para un modelo entrenado, ya que puede proporcionar información valiosa sobre fallas de clasificación por tipo y clase de falla (Hearty, 2016).

1.1.8.1 Matriz de confusión

En el ámbito de la Inteligencia Artificial una matriz de confusión es una herramienta que permite ver el desempeño de un algoritmo al mostrar la proporción de números de instancias entre las clases reales y las predichas verificando que instancias fueron clasificadas erróneamente o no. La matriz de confusión, normalmente se usa en el aprendizaje supervisado. El nombre de Matriz de confusión se da en consideración a que muestra si el sistema está confundiendo dos clases (Visa et al., 2011).

Tabla 1

Matriz de confusión

		Clases predichas	
		Clase=si	Clase=no
Clases reales	Clase=si	TP = True Positive	FN = False Negative
	Clase=no	FP = False Positive	TN= True Negative

Fuente: (Visa et al., 2011)

Dónde:

TP: Número correcto de predicciones que la instancia es positiva.

FN: Número incorrecto de predicciones que la instancia es negativa.

FP: Número incorrecto de predicciones que la instancia es positiva.

TN: Número correcto de predicciones que la instancia es negativa.

a) Precisión (Accurancy)

Cuando nos referimos a la precisión o en inglés ***"accurancy"***, hablamos de la dispersión del conjunto de valores que se obtienen a partir de mediciones repetidas de una magnitud; entonces, cuanto menor es la dispersión mayor es la precisión.

La fórmula para ver la precisión es la siguiente:

$$\text{Precisión (Accurancy)} = \frac{\text{Número correcto de predicciones}}{\text{Número total de imágenes}}$$

b) Exactitud

Por otro lado, la Exactitud o, en inglés ***"Precision"*** se refiere a lo cerca que está el resultado de una medición del valor verdadero. En términos estadísticos, la exactitud está relacionada con el sesgo de una estimación. Se representa por la proporción entre los positivos reales predichos por el algoritmo y todos los casos positivos

$$\text{Exactitud} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

c) Sensibilidad/True Positive Rate/Recall (TPR)

Es conocida también como Tasa de Verdaderos Positivos *(True Positive Rate)* o en ingles simplemente ***"Recall"*** y representa la proporción de casos positivos, que fueron correctamente identificadas por el modelo. Se calcula de acuerdo a la siguiente ecuación:

$$\text{TPR} = \frac{\text{TP}}{\text{FN} + \text{TP}}$$

d) False Positive Rate (FPR)

La tasa de Falsos Positivos, en ingles ***"False Positive Rate"*** es la proporción de casos negativos que fueron erróneamente clasificados como positivos por el modelo. Se calcula según la siguiente ecuación:

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

e) Especificidad/True Negative Rate (TNR)

La Especificidad, es la Tasa de Verdaderos Negativos, en ingles ***"True Negative Rate"*** y se trata de los casos negativos que el modelo ha clasificado correctamente. Se calcula de acuerdo a la siguiente ecuación:

$$\text{TNR} = \frac{\text{TN}}{\text{FP} + \text{TN}}$$

f) False Negative Rate (FNR)

La tasa de Falsos Negativos en ingles ***"False Negative Rate"*** es la proporción de casos positivos incorrectamente clasificados por el modelo y su ecuación es:

$$\text{FNR} = \frac{\text{FN}}{\text{TP} + \text{FN}}$$

g) F1-Score/Puntuación F1

Finalmente tenemos la puntuación F1, que tiene en cuenta tanto la ***"Precisión"*** como el ***"Recall"*** para medir como última instancia la precisión del modelo

$$\text{F1} = \frac{2 \times \text{Precisión} \times \text{Recall}}{\text{Precisión} + \text{Recall}}$$

h) Misclassification Rate/Tasa de error

La tasa de error muestra que proporción de la data clasifica incorrectamente. Su ecuación es la siguiente:

$$\text{Misclassification Rate} = \frac{\text{FP} + \text{FN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

1.2 Antecedentes

1.2.1 A nivel internacional

El Deep Learning es hacer que la computadora aprenda de manera automática usando Redes Neuronales para un propósito en particular y las investigaciones respecto a un aprendizaje específico son varias, por lo cual mencionaremos algunas investigaciones actuales:

En el artículo científico de Barrera et al. (2015) titulado: *"Prediction system of erythemas for phototypes I and II, using deep-learning"*, se presentan los resultados del desarrollo

de un sistema de predicción del tiempo de exposición de una persona, a rayos UV provenientes del sol, que pueden producir eritemas sobre la piel, utilizando la estandarización del índice ultravioleta y los límites de dosis de radiación permitidos para los fototipos cutáneos I y II, con el objetivo de prever la generación de este tipo de lesiones. Esto lo realizaron a través de la implementación de algoritmos de inteligencia artificial como Deep Belief Networks y Backpropagation, basados en la técnica de Deep Learning. En esta ocasión utilizan como parámetros de entrenamiento para la Red Neuronal, los datos meteorológicos como el índice de cielo despejado, la radiación sobre la superficie horizontal de la tierra, y la temperatura promedio del día, provistos por National Aeronautics and Space Administration (NASA). Con estos datos se entrenó una red neuronal con el objeto de pronosticar el índice UV del año posterior a los datos de entrada, adicionalmente se aplicaron algunas regresiones matemáticas permitiendo de esta manera, obtener una aproximación al comportamiento del índice UV a lo largo del día. De igual manera, esta información se empleó para estimar los tiempos de exposición solar máximos, para el periodo de tiempo comprendido entre las 6:00 de la mañana y las 6:00 de la tarde.

Otros investigadores como Rojas y Trujillo (2018) afirman que estudiar las Redes Neuronales Artificiales (RNA) es uno de los más activos en la comunidad científica con múltiples aplicaciones recientes y por lo cual ellos se centran en el algoritmo *Firefly*, el cual ha sido empleado con éxito en el pre-entrenamiento de RNAs con el objetivo de evitar la convergencia en mínimos locales de métodos de entrenamiento convencionales como el algoritmo *Stochastic Gradient Descent* (SGD); sin embargo, en redes con un considerable número de parámetros, el pre-entrenamiento pasa a ser un problema de optimización en espacios de elevada dimensionalidad, y la aplicación del algoritmo *Firefly*, así como cualquier meta-heurística, presenta limitaciones computacionales a considerar, por lo que investigan una variante del algoritmo *Firefly* que permite entrenar una RNA con un subconjunto del conjunto de patrones de entrenamiento original sin disminuir la precisión.

Aproximándonos un poco más al objeto de estudio, encontramos a Vizcaya (2018) en su tesis “Deep Learning para la detección de peatones y vehículos sobre FPGA” donde se platearon entrenar y probar una Red Neuronal Convolucional, con la intensión de valorar los tiempos de entrenamiento y el error en la clasificación de imágenes de señales de tránsito vehicular empleadas en México y otros países, el observa que para el experimento

en el que se cuenta con una mayor cantidad de información (banco de 12,000 imágenes), el entrenamiento converge en menor cantidad de épocas que en el caso que se tiene menor cantidad de información (8,000 imágenes), esto para todos los casos de su experimentación; También afirma la viabilidad de implementar Redes Neuronales Convolucionales con arquitecturas mucho más grandes en FPGAs. Un factor importante a considerar es el tamaño de la Red Neuronal Convolucional a implementar, en comparación con los recursos disponibles en el dispositivo FPGA seleccionado, afirmando que con la constante investigación que se realiza en esta área, es de esperarse que en un corto plazo se cuente con frameworks enfocados en ese punto, y con ello facilitar la implementación de Redes Neuronales Convolucionales existentes directamente en FPGAs.

Acercándonos un poco más a los fines de la investigación vemos que la detección de rostros ha progresado significativamente después del trabajo extraordinario en su momento de Viola y Jones (2004), el clasificador que propusieron, fue efectivo en la detección de rostros frontales; sin embargo, aun no se había solucionado el problema de la detección en diferentes ángulos, hasta que Mathias et al. (2014), Chen et al. (2014) y otros investigadores mejoran la detección de Viola y Jones (2004) en consideración a que pueden detectar rostros en diferentes ángulos; de la misma manera, con la intención de lograr mejoras, encontramos la investigación de Liu et al. (2019) en su artículo *"Center and Scale Prediction: A Box-free Approach for Object Detection"*, en cuya investigación detecta el centro de objetos y a partir de allí se calcula el cuadro característico que encierra al objeto en una escala determinada, cuando se hace una busque en una imagen.

Avanzando un poco más a la detección de humanos por parte de un robot en un contexto social, encontramos el trabajo de Mateus et al. (2019), en su artículo *"Efficient and robust Pedestrian Detection using Deep Learning for Human-Aware Navigation"* en la cual utiliza el aprendizaje profundo para la identificación de humanos, previamente identifica objetos con el método ACF (Aggregate Channel Features) y luego los analiza en una red convolucional profunda para finalmente detectar personas que transitan a su alrededor.

En el artículo de los investigadores Xiao et al. (2017) *"Joint Detection and Identification Feature Learning for Person Search"*, investigan un nuevo marco de aprendizaje profundo para la búsqueda de personas; por lo que manejan conjuntamente la detección e identificación en una sola Red Convolucional y propusieron una función de pérdida de coincidencias de instancias en línea, para capacitar a la red de manera efectiva. Su naturaleza no paramétrica les permitió una convergencia más rápida y mejor.

Dentro del campo de la agricultura, vemos la investigación de Tsampikos et al. (2019) en su artículo *"Deep learning-based visual recognition of rumex for robotic precisión farming"* aborda el problema de reconocer el muelle de hoja ancha (Rumex obtusifolius L.) en los pastizales a partir de imágenes 2D de alta resolución, presentan los factores determinantes para desarrollar e implementar algoritmos de reconocimiento visual de malezas mediante el aprendizaje profundo. La implementación explota las técnicas de transferencia de aprendizaje para la extracción de características basadas en el aprendizaje profundo, en combinación con un clasificador para el reconocimiento de malezas, utilizaron una plataforma robótica prototipo para poner a disposición un conjunto de datos de imágenes de una granja lechera que contiene muelles de hoja ancha. La evaluación del algoritmo propuesto en este conjunto de datos muestra que supera a los métodos competidores de reconocimiento de malezas / plantas en precisión de reconocimiento, al tiempo que produce bajas tasas de falsos positivos en condiciones de operación del mundo real.

Se han encontrado también investigaciones relacionadas a la morfología de las formaciones de lutitas (rocas) en donde los investigadores Kamrava et al. (2019) en su artículo *"Enhancing images of shale formations by a hybrid stochastic and deep Learning algorithm"*, proponen un nuevo algoritmo híbrido mediante el cual se utiliza un método de reconstrucción estocástica para generar una gran cantidad de imágenes de una formación de lutita, utilizando muy pocas imágenes de entrada a muy bajo costo, y luego entrenar una red convolucional de aprendizaje profundo. Los autores se refirieren al método como algoritmo híbrido estocástico de aprendizaje profundo (HSDL).

Dentro del campo del reconocimiento de patrones y letras encontramos el trabajo de investigación de Kastrati et al. (2019) en su artículo *"Integrating word embeddings and document topics with deep learning in a video classification framework"*, donde usan técnicas de aprendizaje profundo para administrar y clasificar contenido educativo para diversas aplicaciones de búsqueda y recuperación con el fin de proporcionar una experiencia de aprendizaje más personalizada; por lo cual, proponen un marco que aprovecha las representaciones de características y el aprendizaje profundo para clasificar las conferencias de video en un entorno MOOC para ayudar a la búsqueda y recuperación efectivas. El marco consta de tres módulos principales. En el primer módulo se preocupan del pre procesamiento y la conversión de video a texto. El segundo módulo es la representación de la transcripción, que representa el texto en las transcripciones de

conferencias en el espacio vectorial mediante la explotación de diferentes técnicas de representación que incluyen la bolsa de palabras, las incrustaciones, el aprendizaje de transferencia y el modelado de temas. El módulo final cubre clasificadores cuyo objetivo es etiquetar las conferencias de video en las categorías apropiadas. Como parte del módulo clasificador, se examinan dos modelos de aprendizaje profundo, a saber, la red neuronal profunda (DNN) y la red neuronal convolucional (CNN). Se realizan simulaciones múltiples en un conjunto de datos real a gran escala utilizando diversas representaciones de características y técnicas de clasificación para probar y validar el marco propuesto.

Ye et al. (2019) en su artículo *"Projecting Australia's forest cover dynamics and exploring influential factors using deep learning"*, presentan una aplicación de técnicas de aprendizaje profundo en la captura de dinámicas de cobertura forestal a largo plazo y continuas en el tiempo a escala continental. Desarrollaron un modelo de conjunto espacialmente explícito para proyectar el cambio de la cubierta forestal de Australia utilizando redes neuronales de aprendizaje profundo de memoria a corto plazo (LSTM) aplicadas a un conjunto de datos espaciotemporales multidimensionales de alta resolución y ejecutados en un clúster informático de alto rendimiento. Según los autores el aprendizaje profundo superó ampliamente a un modelo econométrico espacial de última generación a escala continental, estatal y de celdas de cuadrícula. Por ejemplo, a escala continental, en comparación con el modelo econométrico espacial, el modelo de aprendizaje profundo mejoró el rendimiento de la proyección en un 44% (error cuadrático medio) y un 12% (pseudo R cuadrado). Los resultados ilustran la robustez y efectividad del modelo LSTM. Este trabajo proporciona una herramienta confiable para proyectar la cubierta forestal y la producción agrícola en escenarios futuros dados, apoyando la toma de decisiones en el desarrollo, manejo y conservación sostenible de la tierra.

Dentro del campo de la seguridad de la información en Internet Turkey (2019) presenta un nuevo enfoque de clasificación de datos de firewall. Este enfoque utiliza 10 casos para obtener resultados numéricos. El enfoque propuesto consiste en la adquisición de datos de Firewall, selección de características y pasos de clasificación. En primer lugar, obtuvo los datos de un firewall, Luego eliminó las características redundantes y estas características lo normalizaron mediante la normalización min – max. Los conjuntos de características finales obtenidos se envían a los clasificadores. En los casos definidos, la memoria de corto plazo (LSTM), la memoria de corto plazo bidireccional (Bi-LSTM) y la máquina de vectores de soporte (SVM) se utilizan como clasificadores. Por lo que el

enfoque de aprendizaje profundo fue más exitoso que el clasificador SVM y la precisión de clasificación más alta se calculó como 97.38% utilizando la red híbrida Bi-LSTM-LSTM. Para lograr un sistema de monitoreo inteligente para la seguridad de la red.

Zhu et al. (2019) en su artículo *"Deep learning for identifying radiogenomic associations in breast cancer"*, cuyo objetivo principal fue determinar si los modelos de aprendizaje profundo pueden distinguir entre los subtipos moleculares del cáncer de mama basados en imágenes de resonancia magnética con contraste dinámico (DCE-MRI). Utilizaron imágenes DCE-MR de 270 pacientes y las lesiones de interés fueron identificadas por los radiólogos. La tarea consistía en determinar automáticamente si el tumor es del subtipo Luminal A o de otro subtipo en función de los parches de imagen de RM que representan el tumor. Se utilizaron tres enfoques diferentes de aprendizaje profundo para clasificar el tumor de acuerdo con sus subtipos moleculares: aprender desde cero donde solo se usaron parches tumorales para el entrenamiento, aprendizaje de transferencia donde las redes pre-entrenadas en imágenes naturales se afinaron usando parches tumorales, y fuera de características profundas en el estante donde las características extraídas por redes neuronales entrenadas en imágenes naturales se utilizaron para la clasificación con una máquina de vectores de soporte. Las arquitecturas de red utilizadas en los experimentos fueron GoogleNet, VGG y CIFAR. Finalmente concluyen que el aprendizaje profundo puede desempeñar un papel importante en el descubrimiento de asociaciones radiogenómicas en el cáncer de mama.

Luo et al. (2019) en su artículo *"Temporal and spatial deep learning network for infrared thermal defect detection"*, presentan estudios de materiales compuestos que son materiales estructurales importantes en la industria aeroespacial, palas de turbinas eólicas, etc. Los defectos producidos durante la fabricación y en servicio de los materiales compuestos laminados conducen a un peligro potencial para la seguridad; por lo tanto, las pruebas y evaluaciones no destructivas (NDT y E) son la tecnología clave para garantizar la calidad de los principales equipos de construcción y la seguridad de la operación. En este artículo, se propone un híbrido de arquitectura de aprendizaje profundo espacial y temporal para la detección automática de defectos de termografía. Los autores afirman que la integración de la estrategia de aprendizaje de redes cruzadas tiene la capacidad de minimizar significativamente la iluminación desigual y mejorar la tasa de detección. Los resultados muestran que la estructura de aprendizaje cruzado de grupo de geometría visual-Unet (VGG-Unet) puede mejorar significativamente el contraste entre

las regiones defectuosas y no defectuosas. Además, se realiza una investigación de los diferentes métodos de extracción de características en los que se integra el aprendizaje profundo para optimizar la estructura de aprendizaje. Para investigar la eficacia y robustez del método propuesto, se han llevado a cabo estudios experimentales para defectos de desacoplamiento interno en muestras de polímero reforzado con fibra de carbono (CFRP) de forma regular e irregular.

Dentro de la clasificación de algunos cultivos hortícolas, como los niveles de banano o uvas, se agrupan en grupos en lugar de la clasificación individual, por lo que en las investigaciones de Le et al. (2019), se estudian este tipo de cultivos al tener una compleja estructura física, desarrollaron una clasificación no invasiva de aprendizaje profundo de plátano en racimo dada una sola característica de imagen. Usaron redes neuronales de convolución basadas en regiones de máscara, también conocidas como Máscara R-CNN. Con la máscara R-CNN, la detección de la fruta de plátano compleja dentro de una imagen predice la clase de plátano y al mismo tiempo genera una máscara que separa la fruta de su fondo. Se utiliza un conjunto de datos real basado en niveles de banana y el modelo desarrollado discrimina niveles normales de niveles anormales. El modelo propuesto por los investigadores, obtuvo una precisión promedio mejor del 92.5%. Con el aumento de datos, el modelo mejoró ligeramente con una precisión del 93.8% al clasificar la clase de rechazo y un 96.5% para la precisión general.

1.2.2 A nivel nacional

En las investigaciones nacionales, encontramos a Arenas y Marino (2016) que se plantearon como objetivo “Modelar, analizar y diseñar un sistema para la clasificación de señales de tránsito vehicular”, logrando Modelar la arquitectura de una Red Neuronal Convolucional para el mapeo de señales de tránsito vehicular; así como también lograron analizar y calcular los parámetros relacionados a la arquitectura de la Red Neuronal Convolucional.

De la misma manera Fernández (2017) en su investigación “Identificación Automática de Acciones Humanas en Secuencias de Video para soporte de video vigilancia”, se plantea como objetivo principal, identificar acciones humanas en secuencias de video vigilancia usando técnicas de aprendizaje profundo con la finalidad de apoyar a los sistemas de monitoreo por video. En dicha investigación realiza varios experimentos con bases de datos conocidas como: UCF101 y VIRAT 2.0 Ground, de esa investigación llega a la

conclusión de que la naturaleza espacio-temporal de las acciones en video requieren métodos que sean capaces de capturar características de los frames de video y de la transición temporal entre frames, ya que los experimentos no fueron del todos exitosos.

En esta misma línea del entrenamiento de redes neuronales, Ascarza (2018) de la Universidad Pontificia Católica del Perú, en su tesis: "Segmentación automática de textos, mediante redes neuronales convolucionales en imágenes de documentos históricos" se traza como objetivo principal, desarrollar un modelo basado en una red neuronal convolucional profunda para segmentación de imágenes de documentos históricos. Por lo que hizo experimentos en el dataset público Parzival; pero, no obtuvo resultados satisfactorios, ya que en su mayoría hubieron confusiones por parches de tipo "pagina" y que existe un desbalance entre el número de parches de tipo "comentario" y "decoración" con respecto a la cantidad de parches de tipo "texto" y "pagina".

Finalmente se revisó el trabajo de Reátegui y Velasco (2018) de la Universidad Nacional de la Amazonia del Perú, quien en su trabajo de tesis titulado: "Aplicación informática para reconocimiento de la especie Camu Camu (Myrciaria Dubia) a través de Redes Neuronales Convolucionales", se plantearon como objetivo principal, Implementar un software, creado a partir del uso de técnicas de redes Neuronales Convolucionales, que permita el reconocimiento de plantas de Camu Camu a partir de las hojas, y en el transcurso de la investigación, logran reconocer a las plantas a partir de sus hojas, implementando un software que tiene la capacidad de reconocer en un 100% a las imágenes que efectivamente corresponden a la hoja de la especie Camu Camu, y a la vez indicar en un 97% que la hoja no es una imagen de Camu Camu.

CAPÍTULO II

PLANTEAMIENTO DEL PROBLEMA

2.1 Identificación del problema

A nivel general una de las formas de controlar al personal de una entidad, es identificándolos en las entradas y salidas de sus centros laborales, y esta labor es encomendada generalmente a los de portería o alguna oficina de seguridad o control en dicha entidad. La forma de hacerlo es identificando de manera visual o con algún medio de identificación implementado en la entidad; pero resulta que la operación de control por estos mecanismos depende mucho de la efectividad de las personas asignadas a esa labor. Para registrar el control, el rostro es uno de los principales focos de atención, centramos la atención visual en las características faciales y estructura física de la persona, somos capaces de identificar cientos de personas, incluyendo el de aquellas personas que no hemos visto durante algún tiempo, llevando a cabo esta tarea en una fracción de segundo y eso ha llevado a múltiples estudios de reconocimiento facial dentro de la Inteligencia Artificial y por ende diferentes estudios del reconocimiento automático de la identidad de la persona, esta problemática del reconocimiento facial y de la identidad de la persona de manera automática es una labor que constantemente los investigadores desarrollan diferentes técnicas y algoritmos para asemejarse al del ojo humano; sin embargo son sólo aproximaciones con ventajas y desventajas en cada uno de ellos.

La Universidad Nacional Micaela Bastidas de Apurímac (UNAMBA) es una entidad pública que como cualquier otra tiene personal administrativo y docente laborando dentro de esta, con un horario y control determinado; sin embargo existe la necesidad de saber la fluctuación de entradas y salidas del personal docente y administrativo para fines de control.

Aunque la detección de personas implica la detección de algunas características principales y propias de las personas como el sexo, la edad y el rostro; implementar y enseñar a un computador a realizar dicha detección es una tarea muy difícil en comparación a los humanos que lo hacemos casi de manera natural, sin demasiado esfuerzo.

La simple detección del rostro sin la identidad es una tarea difícil, a causa de la alta variabilidad de la apariencia del rostro. Podemos catalogar a los rostros como objetos no rígidos y dinámicos con una diversidad grande en la forma, el color y la textura; así como también a factores como la pose de la cabeza, iluminación (contraste, las sombras), expresiones faciales, las oclusiones (lentes) y otras características faciales como el maquillaje, la barba y prendas de cabeza.

En el tema de reconocimiento facial o de patrones, se ha trabajado por años, con diferentes enfoques y algoritmos. En la última década, el paradigma del Deep Learning ha revolucionado el estado del arte en tareas como reconocimiento de voz, el procesamiento del lenguaje natural y la visión artificial; en consideración a estos últimos avances en la Inteligencia Artificial y propiamente dicho en el Deep Learning se pretende implementar y entrenar Redes Neuronales Convolucionales para apoyar al control con la identificación del personal administrativo y docente de la UNAMBA.

2.2 Enunciado del problema

La identificación del problema nos permitió formular la siguiente pregunta:

¿Cuál será la proporción de precisión en la identificación del personal administrativo y docente de la UNAMBA, usando Deep Learning?

2.3 Justificación

El presente trabajo de investigación, se realizó con fines de contribuir en el control del personal administrativo y docente de la UNAMBA, usando el Deep Learning para implementar la visión Artificial como un paradigma de la Inteligencia Artificial; además de ello sirve para mostrar la forma de implementar el Deep Learning usando librerías avanzadas para estos propósitos como Keras, Tensorflow u otros y finalmente culminarlo en un análisis de proporción alto en la identificación del personal de la UNAMBA; de esta manera contribuimos al control del personal de la UNAMBA a través de la implementación de técnicas de Visión Artificial avanzadas, que sean capaces de auto-

programarse, que aprendan de su propia experiencia, y además que contribuirá a presentes y futuras investigaciones en el marco del Aprendizaje Automático ***(Machine Learning)*** y particularmente en el ***Deep Learning***.

2.4 Objetivos

2.4.1 Objetivo general

Lograr la proporción más alta de precisión en la identificación del personal administrativo y docente de la UNAMBA usando *Deep Learning* en el campo de la Visión Artificial.

2.4.2 Objetivos específicos

- Determinar en una primera etapa de aprendizaje la proporción de precisión de una arquitectura clásica de CNN en la identificación del personal administrativo y docente de la UNAMBA.
- Determinar en una segunda etapa de aprendizaje la proporción de precisión de una arquitectura moderna de CNN en la identificación del personal administrativo y docente de la UNAMBA.
- Comparar la proporción de la primera etapa con la segunda etapa.

2.5 Hipótesis

2.5.1 Hipótesis General

Si se usa *Deep Learning*, entonces la proporción de la precisión en la identificación del personal administrativo y docente de la UNAMBA a través de la visión artificial es alta.

CAPÍTULO III

MATERIALES Y MÉTODOS

3.1 Lugar de estudio

La investigación se realizó en el año 2019, en la Universidad Nacional Micaela Bastidas de Apurímac, Ubicada en el departamento de Apurímac, provincia de Abancay, distrito de Tamburco. La toma de imágenes correspondientes a docentes y administrativos se realizó en la puerta de acceso a la Universidad.

La localización del lugar de estudios lo podemos apreciar en la Figura 15

Figura 15. Localización geográfica de la UNAMBA, Tamburco

Fuente: Google Maps

3.2 Población

La población está conformada por el conjunto de personal administrativo y docente que labora dentro de la sede principal de la Universidad Nacional Micaela Bastidas de Apurímac (Tamburco), los cuales ascienden a $N = 300$.

3.3 Muestra

Hallamos la muestra de la siguiente manera:

$$n' = \frac{N}{p^2(N-1)+1}$$

Dónde:

n': Tamaño de la muestra no ajustada

N: Tamaño de la población

p: Probabilidad de error (5% = 0.05)

n: Tamaño de la muestra

$$n' = \frac{300}{0.05^2 * 300 - 1 + 1} = 171{,}67$$

Luego reajustamos la muestra como sigue:

$$n = \frac{n'}{\frac{(n'-1)}{N} + 1}$$

Por lo cual $n = 109.18 \approx 110$

El valor de n indica que el número minimo de personas a ser estudiadas, ascienden a 110 entre administrativos y docentes a una taza de error del 5% para ser representativa a la población; sin embargo, la personas captadas en la puerta de ingreso a la universidad, para fines de la investigación fue $n = 242$ personas entre administrativos y docentes de la UNAMBA; por lo que, los experimentos de entrenamiento fueron realizados con esta cantidad.

Por lo tanto $n = 242$

3.4 Métodos

3.4.1 Estrategia de captura y colección de imágenes

En el campo del Deep Learning y específicamente en el campo del reconocimiento de imágenes, dada la problemática que se desee resolver, se requiere de una base de datos gigante entre miles y millones de imágenes relacionados a la problemática a resolver; por lo cual los investigadores para descubrir modelos más eficientes usan algunas bases

de datos generales disponibles algunas de manera libre, otras pagadas en universidades o instituciones dedicadas al Machine Learning.

En esta investigación al no disponer de ninguna base de datos de imágenes relacionadas al personal docente y administrativo de la Universidad Nacional Micaela Bastidas de Apurímac, se utilizó algoritmos de Visión Artificial ya definidos en bibliotecas libres como es OpenCV y se usó la técnica de *Video Scrapping* con la cual se obtuvo datos que utilizaremos para alimentar a tu modelo de *Deep Learning*. Importa mucho la calidad y cantidad de información que se consiga, ya que impactará directamente en lo bien o mal que luego funcione nuestro modelo.

3.4.1.1 Técnica de Video Scraping

Esta técnica consiste en implementar un software que permita capturar información de videos a los cuales podremos coleccionarlos y manipularlos a nuestras necesidades.

Para esta investigación se desarrolló un pequeño software usando librerías de OpenCV y algoritmos de *Haar-Cascade* pre entrenado para propósitos generales.

Por la necesidad de obtener imágenes de medio cuerpo del personal docente y administrativo de la UNAMBA, es que se utilizó:

- haarcascade_mcs_upperbody.xml y
- haarcascade_fullbody.xml

Para lograr una visión Artificial parcial, logrando obtenerse Verdaderos Positivos y una gran cantidad de Falsos Negativos, considerando que el algoritmo de Haar no es preciso en sus objetivos.

A continuación mostramos la forma del método scraping en la cámara de video vigilancia, instalado en la puerta de entrada de la Universidad.

Figura 16. Captura de imágenes de personas

Fuente: Propia

3.4.2 Preparar las imágenes

Una vez capturada la imágenes a través de la técnica de video scraping, se tuvo que limpiar todas aquellas imágenes correspondientes a falsos negativos (Imágenes capturadas que no son personas), imágenes de personas que no corresponden a los propósitos de la investigación (Personas que no son docentes o administrativos de la sede central de la UNAMBA); así como también imágenes que no lograron tener buena resolución, ya que la información para el entrenamiento del modelo convolucional de Deep Learning tiene que ser buena, de esta manera se logró recopilar 27,996 imágenes de medio cuerpo correspondientes a las 242 clases.

Una vez limpiada las imágenes, se clasificó cada clase (persona) en un directorio con su nombre correspondiente Figura 18.

Seguidamente se partió la cantidad de imágenes que tenemos en cada directorio (clases), para que sea representativo, ya que si no, el aprendizaje podría ser tendencioso hacia un tipo de respuesta y cuando nuestro modelo intente generalizar la identificación fallaría.

Finalmente en esta etapa se separó las imágenes en dos grupos: uno para entrenamiento y otro para la evaluación del modelo como se aprecia en la Figura 17, en donde en cada carpeta existe 242 sub carpetas con imágenes diferentes de cada persona; es decir la data original se dividió en un 70.4% para entrenamiento, 19,700 imágenes en total en la carpeta entrenamiento y 29.6% para la validación, 8,296 para la evaluación de los modelos.

La data original se puede dividir en un 80% para el entrenamiento y 20% para la validación si la información es abundante o en 70% para el entrenamiento y 30% para la validación. En esta investigación se aproximó al último caso.

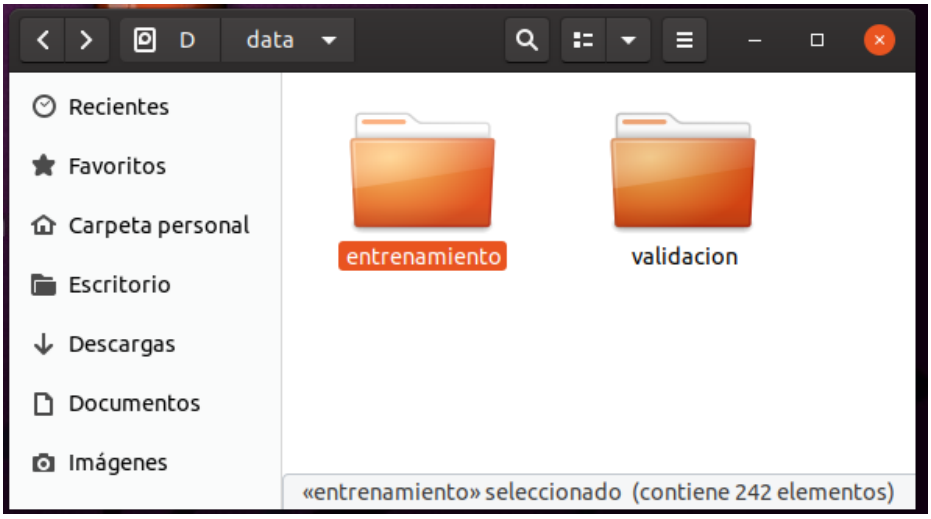


Figura 17. Grupos de imágenes, entrenamiento y validación

Fuente: Propia

En la figura 18 se muestra como queda organizada la información de cada persona (clase) en carpetas con sus respectivos nombres. Cabe resaltar que las carpetas de entrenamiento y validación tienen en su interior la misma cantidad de sub carpetas.

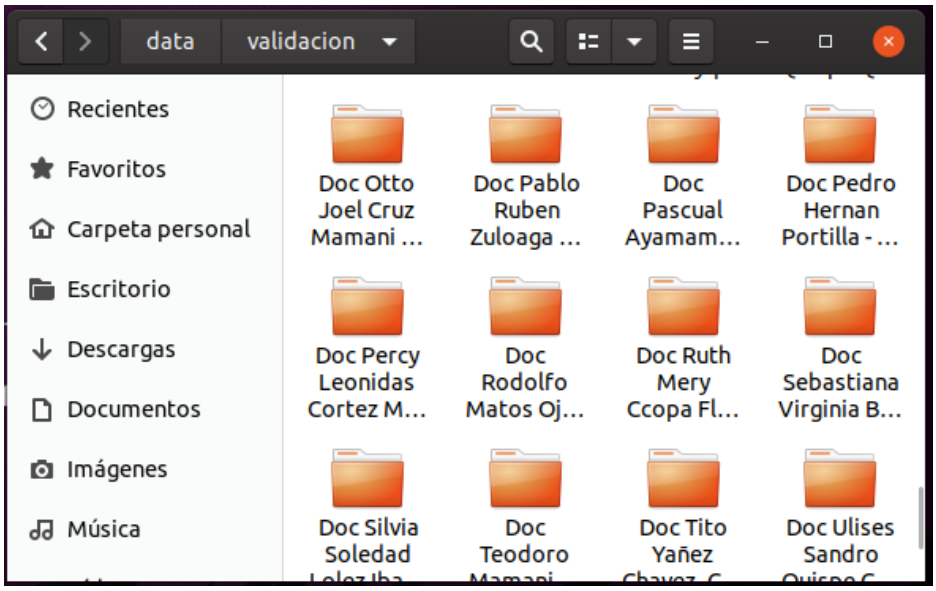


Figura 18. Muestra de clases

Fuente: Propia

3.4.3 Elegir el modelo

En el mundo del Machine Learning, existen diversos modelos que podemos elegir de acuerdo al objetivo que tengamos, por ejemplo: algoritmos de clasificación, predicción, regresión lineal, clustering, Deep Learning, bayesiano, etc. y podrá haber variantes si lo que vamos a procesar son imágenes, sonido, texto o valores numéricos.

Esta investigación tiene el propósito de usar modelos de Deep Learning, usando Redes Neuronales Convolucionales (CNN) y dentro de este grupo de CNNs, existen varios modelos probados exitosamente, como también podríamos crear el nuestro; por lo que se eligió dos arquitecturas de redes Neuronales como referencia VGG16 y DenseNet121, los cuales fueron puestos en práctica para nuestros propósitos, sin embargo se tuvo que hacer algunas variaciones para lograr el objetivo de esta investigación, los cuales serán detallados en la sección de resultados

3.4.4 Entrenar el modelo de CNN

Una vez elegido el modelo se requiere configurarlo y compilarlo en una máquina de alto rendimiento en consideración a los millos de neuronas que se requieren entrenar. Normalmente se y utiliza procesamiento en GPU ya que el CPU es inadecuado y podría tardar días en completar un entrenamiento.

Para los propósitos de entrenamiento de los modelos elegidos en el paso anterior, se usó como herramienta a Google Colaboratory, este es un entorno gratuito de Jupyter Notebook que no requiere configuración y que se ejecuta completamente en la nube, dando soporte de CPU, GPU e incluso TPU para la ejecución de los algoritmos de Machine Learning u otros que se quieran probar y que requieran alto poder de performance, permite escribir y ejecutar código, guardar y compartir tus análisis, todo de forma gratuita desde el navegador.

Para este propósito se usa el conjunto de datos de entrenamiento (Figura 17) que en nuestro caso asciende a 19,673 imágenes de entrenamiento y 8,285 imágenes de validación de los modelos para entrenar a 242 clases entre docentes y administrativos de la UNAMBA.

3.4.5 Evaluación del modelo

Debemos comprobar el modelo entrenado con nuestro grupo de datos de evaluación (Fig. 17) que en nuestro caso asciende a 8,285 imágenes a ser validadas, estas imágenes son desconocidas para el modelo, por lo que permite verificar la precisión de nuestro modelo ya entrenado. Si alcanzamos un 90% o más podremos tener confianza en los resultados que nos otorga el modelo de no ser así pasaremos al siguiente paso.

3.4.6 Parameter Tuning (configuración de parámetros)

Una vez entrenado el modelo se revisan los resultados obtenidos y si no obtuvimos buenas predicciones, probablemente tuvimos problemas de overfitting o underfitting las cuales se solucionan incrementando la cantidad de veces que iteramos nuestros datos de entrenamiento (épocas) y/o ajustando los parámetros de compilación como: el redimensionamiento del tamaño de las imágenes de entrada, el Learning Rate (tasa de aprendizaje), etc.; de la misma forma se hacen cambios en el modelo CNN como la cantidad de capas a usar, el tipo de función de activación (Relu), optimizadores de la función de coste (*Stochastic Gradient Descent*, *Adam* u otros) o las funciones de predicción (Sigmoid, Softmax u otros), etc. y volver a compilar n veces hasta lograr los objetivos.

Esta manipulación del modelo sigue siendo más un arte que una ciencia y se irá mejorando a medida que experimentamos más y más con gran esfuerzo y paciencia para dar con buenos resultados.

3.4.7 Predicción o inferencia

Si logramos el porcentaje alto esperado de precisión, habremos entrenado Redes Neuronales Convolucionales exitosamente para pasar a otro etapa (desarrollo de software) y ponerlo en práctica con nueva información y comenzar a predecir o inferir resultados en tiempo real para los fines de control o el que se le haya dado.

3.4.8 Herramientas y materiales

Para el desarrollo y ejecución de la investigación se utilizaron las siguientes herramientas y materiales:

- Laptop con las siguientes características:
    - Sistema Operativo: Ubuntu 19.10 de 64 bits
    - Memoria: 8 Gb.
    - Procesador: Intel® Core™ i3-2350M CPU @ 2.30GHz × 4
- Cámara IP Hikvision: Configurada a 5 fotogramas por segundo, en consideración a que se tenía que analizar 5 macro imágenes por segundo (Con calidad de 2 MP) para ser pre analizadas y recortar imágenes de personas para su almacenamiento en una etapa de recolección o identificación en una etapa de entrenamientos, por medio de modelos y arquitecturas de redes neuronales convolucionales. La cámara puede ser configurada entre 25-30 fotogramas por segundo (por defecto) dependiendo de la potencia y cantidad procesadores que lleva la computadora de análisis, siendo mucho mejor la utilización de una tarjeta gráfica que incluya cientos o miles de GPUs.

  En esta investigación la cámara de video vigilancia se configuró de esa manera por las limitadas características de la computadora de análisis.
- Entorno de desarrollo de programación:
    - Jupyter Notebook (a nivel local)
    - Spyder (a nivel local)

- Google Colab (Entorno gratuito de Jupyter Notebook que se ejecuta en la nube con GPUs.)

- Google Drive como plataforma de almacenamiento y nexo con Google Colab.

- Lenguaje de programación: Python versión 3

- Anaconda (una distribución open-source para el desarrollo de Machine Learning) y librerías de Inteligencia Artificial

  - Numpy
  - Matplotlib
  - Sklearn
  - Pandas
  - Opencv
  - Tensor Flow
  - Keras

- Análisis de datos: Se usaron las librerías: Confusion_matrix y Classification_report de Sklearn, Pandas, Numpy, Seaborn y Pyplot de Matplotlib.

CAPÍTULO IV

RESULTADOS Y DISCUSIÓN

En este capítulo se presentan los resultados finales del proceso de la investigación hasta llegar a las discusiones.

4.1 Análisis de resultados

4.1.1 Primera etapa

Investigación y entrenamiento Deep Learning con VGG16 personalizado

El modelo propuesto para los propósitos de la investigación en esta etapa, fue usar como modelo la arquitectura clásica de VGG16; sin embargo, este modelo no se adecua a los propósitos de la investigación, por ejemplo la entrada de imágenes por defecto es de 224 de alto x 224 de largo, la función de la última capa es softmax, lo cual no es adecuado a los propósitos de la investigación, por lo cual se realizó diferentes experimentos hasta llegar al presente modelo (VGG16UNAMBA) funcional, donde se modificó lo siguiente:

- Tamaño de entrada de las imágenes a 160 x 160 (longitud x altura)
- Penúltima capa del modelo de Flatten() a GlobalAveragePooling2D()
- Y finalmente la última capa de clasificación que tenía por defecto a “softmax” se cambió por “sigmoid”.

La función VGG16UNAMBA propuesta en código python se resume a lo siguiente:

```
def VGG16UNAMBA(nclases=242,longitud=160,altura=160):
    base_model = vgg16.VGG16(include_top=False,
        weights='imagenet',
        input_shape=(longitud, altura, 3),
        classes=nclases)
    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dense(4096, activation='relu')(x)
    predictions = Dense(nclases, activation='sigmoid') (x)
    model = Model(inputs=base_model.input,outputs=predictions)
    return model
```

Si queremos ver todas las capas del modelo CNN VGG16UNAMBA ejecutamos el siguiente código en python

```
modelo=  VGG16UNAMBA()
modelo.summary()
```

Y veremos el siguiente resultado:

Model: "model_1"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 160, 160, 3)	0
block1_conv1 (Conv2D)	(None, 160, 160, 64)	1792
block1_conv2 (Conv2D)	(None, 160, 160, 64)	36928
block1_pool (MaxPooling2D)	(None, 80, 80, 64)	0
block2_conv1 (Conv2D)	(None, 80, 80, 128)	73856
block2_conv2 (Conv2D)	(None, 80, 80, 128)	147584
block2_pool (MaxPooling2D)	(None, 40, 40, 128)	0
block3_conv1 (Conv2D)	(None, 40, 40, 256)	295168

block3_conv2 (Conv2D)	(None, 40, 40, 256)	590080
block3_conv3 (Conv2D)	(None, 40, 40, 256)	590080
block3_pool (MaxPooling2D)	(None, 20, 20, 256)	0
block4_conv1 (Conv2D)	(None, 20, 20, 512)	1180160
block4_conv2 (Conv2D)	(None, 20, 20, 512)	2359808
block4_conv3 (Conv2D)	(None, 20, 20, 512)	2359808
block4_pool (MaxPooling2D)	(None, 10, 10, 512)	0
block5_conv1 (Conv2D)	(None, 10, 10, 512)	2359808
block5_conv2 (Conv2D)	(None, 10, 10, 512)	2359808
block5_conv3 (Conv2D)	(None, 10, 10, 512)	2359808
block5_pool (MaxPooling2D)	(None, 5, 5, 512)	0
global_average_pooling2d_1 (	(None, 512)	0
dense_1 (Dense)	(None, 4096)	2101248
dense_2 (Dense)	(None, 242)	991474

==========

Total params: 17,807,410
Trainable params: 17,807,410
Non-trainable params: 0

Un modelo CNN con 17,807,410 parámetros de entrenamiento, en donde cada neurona se encargara de entrenar un parámetro; por lo cual es válido afirmar que se dispone de 17,807,410 neuronas.

En el apartado de Anexos podemos ver el entrenamiento final N° 05 (etapa I) que corresponde al planteado en esta etapa de investigación con mayor detalle del entrenamiento y procesamiento de la información.

Evaluación del modelo VGG16UNAMBA

El modelo fue evaluado con el conjunto de datos de la carpeta "validación" con 8,296 imágenes y clasificó correctamente 8,135 imágenes correspondientes a las 224 clases; por lo cual la primera proporción hallada fue de:

$$p_1 = \frac{8135}{8296} = 0.9805$$

Podemos afirmar que el modelo genera una proporción de 0.9805 de precisión en la identificación de 242 personas, entre administrativos y docentes de la UNAMBA.

Intervalo de confianza para p_1 con un 99% de confiabilidad:

$$Z_{\alpha/2} = Z_{0.01/2} = Z_{0.005} \rightarrow p = 0.995$$

$$Z_t = 2.576$$

$$p_1 \pm Z_{\alpha/2}\sqrt{\frac{p_1(1-p_1)}{n}}$$

$$0.9805 \pm 2.576\sqrt{\frac{0.9805(0.0195)}{8296}}$$

$$0.9805 \pm 0.003910683$$

$$[0.9766\ ,\ 0.9844]$$

Podemos afirmar que el modelo en la etapa I, identifica al personal administrativo y docente dentro del intervalo de confianza con una variación muy pequeña para p_1.

4.1.2 Segunda etapa

Investigación y entrenamiento Deep Learning con DenseNet personalizado

Con el objetivo de mejorar la proporción de la etapa I, se buscó un modelo con una arquitectura moderna como DenseNet121.

Este nuevo modelo tenía una arquitectura moderna; sin embargo no era adecuado para los fines de la investigación; por lo cual se tuvo que generar un nuevo modelo personalizado a los propósitos de la investigación.

Se hicieron los siguientes cambios:

- Tamaño de entrada de las imágenes a 150 x 150 (longitud x altura)
- Se cambió la penúltima capa de Flatten() por GlobalAveragePooling2D()
- Y finalmente la capa de clasificación que tenía por defecto a “softmax” se cambió por “sigmoid”.

La función DenseNet121UNAMBA propuesta en código python se resume a lo siguiente:

```
def DenseNet121UNAMBA(nclases=242, longitud=150, altura=150):
    base_model = DenseNet121(include_top=False,
        weights='imagenet',
        input_shape=(longitud, altura, 3),
        classes=nclases)
    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dense(1024, activation='relu')(x)
    predictions = Dense(nclases, activation='sigmoid') (x)
    model = Model(inputs=base_model.input, outputs=predictions)
    return model
```

Si queremos ver todas las capas del modelo CNN DenseNet121UNAMBA exitoso, nos referenciamos a anexo 7 (entrenamiento 07 etapa II), donde se muestra a detalle los parámetros, procedimientos y reportes generados por el modelo; sin embargo, mostramos un resumen de dicho modelo en cuanto a la cantidad de parámetros que usó.

Total params: 12,227,378

Trainable params: 12,143,730

Non-trainable params: 83,648

En anexo 7 se podrá apreciar la complejidad de las capas de convolución usadas por el modelo a diferencia de la etapa uno; sin embargo este modelo muestra una disminución notable de parámetros a entrenar 12,227,378 frente a 17,807,410 en la etapa I, reduciéndose sustancialmente los parámetros a entrenar.

Evaluación del modelo DenseNet121UNAMBA

Este modelo propuesto, también fue evaluado con el conjunto de datos de la carpeta "validación" con 8,296 imágenes y clasificó correctamente 8,240 imágenes correspondientes a las 224 clases; por lo cual la segunda proporción hallada fue de:

$$p_2 = \frac{8240}{8296} = \frac{1030}{1037} = 0.9932$$

Podemos afirmar que el modelo genera una proporción de 0.9932 de precisión en la identificación de 242 personas, entre administrativos y docentes de la UNAMBA, mejorando la proporción de la primera etapa.

Intervalo de confianza para p_2 con un 99% de confiabilidad:

$$Z_{\alpha/2} = Z_{0.01/2} = Z_{0.005} \rightarrow p = 0.995$$

$$Z_t = 2.576$$

$$p_2 \pm Z_{\alpha/2}\sqrt{\frac{p_2(1-p_2)}{n}}$$

$$0.9932 \pm 2.576\sqrt{\frac{0.9932(0.0068)}{8296}}$$

$$0.9932 \pm 0.002324258$$

$$[0.990875742\ ,\ 0.995524258]$$

Podemos afirmar que el modelo en la etapa II identifica al personal administrativo y docente dentro de un intervalo de confianza muy ceñido, por lo que la proporción p_2, se mantiene en un 0.99, lo cual es muy satisfactorio.

4.1.3 Prueba de hipótesis

De los resultados obtenidos en la etapas I y II, nos hacen pensar que las proporciones obtenidas son similares (iguales estadísticamente), por lo que es necesario una comprobación a través de una prueba de hipótesis.

Planteamos la prueba de hipótesis de la siguiente manera:

i. Planteamiento de la prueba de hipótesis

$H_0: p_1 = p_2$

$H_1: p_1 \neq p_2$

La hipótesis nula (H_0) nos indica que la proporción p_1 es igual a la proporción p_2 y la hipótesis alterna (H_1) nos indica que la proporción p_1 es diferente a la proporción p_2.

ii. Nivel de significancia

Para comprobar la prueba de hipótesis, trabajamos con un nivel de significancia del 99%; por lo que:

$\alpha = 0.01$

$Z_t = \pm 2.576$ (Zeta tabulado)

Figura 19. Campana de Gauss

Fuente: Propia

iii. Estadístico de prueba

Por tratarse de proporciones y n grande ($n = 8{,}296$), se usa:

$$Z_c = \frac{p_1 - p_2}{\sqrt{\frac{p(1-p)}{n_1} + \frac{p(1-p)}{n_2}}}$$

La estimación combinada de p se puede hallar de la siguiente manera:

$$p = \frac{x_1 + x_2}{n_1 + n_2}$$

Dónde:

p = proporción muestral

x_1 = Numero de aciertos en la muestra uno

x_2 = Numero de aciertos en la muestra dos

n_1 = Numero de observaciones (imágenes) en la muestra uno

n_2 = Numero de observaciones (imágenes) en la muestra dos

Calculamos el estadístico de prueba:

$$p = \frac{x_1 + x_2}{n_1 + n_2} = \frac{8135 + 8240}{8296 + 8296} = 0.9869$$

$$Z_c = \frac{p_1 - p_2}{\sqrt{\frac{p(1-p)}{n_1} + \frac{p(1-p)}{n_2}}} = \frac{0.9805 - 0.9932}{\sqrt{\frac{0.9869(0.0131)}{8296} + \frac{0.9869(0.0131)}{8296}}}$$

$$Z_c = \frac{-0.0127}{0.00176544} = -7.19$$

iv. Conclusión estadística

Como $(|Z_c| = 7.19) > (|Z_t| = 2.576)$, se rechaza H_0, como se puede corroborar en la figura 19, el valor $|Z_c| = 7.19$, esta fuera del area de haceptación de H_0.

Significa que estadísticamente a un nivel de significancia de 0.01, la proporción p_1 es diferente a la proporción p_2.

4.2 Discusión de los resultados

a) Como se puede apreciar en la parte de evaluación de los modelos en las etapas I y II, la proporción "más alta" hallada, corresponde a la segunda etapa con un 0.9932 frente a 0.9805 de precisión en la identificación del personal administrativo y docente de la UNAMBA; al igual que a Barrera et al. (2015), la implementación de técnicas de inteligencia artificial como el Deep Learning, les permitió desarrollar aplicaciones, orientadas a pronosticar los efectos negativos de la radiación solar en la salud humana, establecer relaciones entre las variables meteorológicas y las condiciones naturales necesarias para la aparición de este tipo de adversidades de manera exitosa. La diferencia de esta investigación y la investigación de Barrera et al. (2015) radica en que los investigadores señalados usan la regresión como forma de aprendizaje y nosotros la clasificación; pero en ambos casos se usa el Deep Learning de manera exitosa.

b) Los modelos propuestos (VGG16UNAMBA y DenseNet121UNAMBA) dieron grandes resultados y destacaremos que una de las modificaciones más resaltantes en dichos modelos fue reducir el nivel de falsos positivos, pues la función Softmax (que venía por defecto en VGG16 y DenseNet121) devolvía predicciones que al ser sumadas en su totalidad siempre sumaban uno, por lo que mantener esta función traía como consecuencia identificar una persona siempre de manera positiva, aunque esa persona estuviera fuera de las clases entrenadas, con lo cual se incrementaba los falsos positivos. Al usar la función Sigmoid que devuelve una predicción de cada clase de forma independiente eliminamos los falsos positivos, pues al ingresar una persona que no está en las clases entrenadas, simplemente devolverá cero en su predicción; de esta manera podemos corroborar la utilización

exitosa de la función de activación Sigmoid en la investigación de Vizcaya (2018)en la clasificación de imágenes de señales de tránsito vehicular empleadas en México y otros países, obteniendo resultados exitosos hasta en 96.75% de exactitud en 12,000 imágenes de entrenamiento.

c) También podemos afirmar que la cantidad de parámetros a entrenar (neuronas) en la etapa uno de 17,807,410 queda compensado con la abrumadora complejidad del modelo en la segunda etapa, la cual a su vez redujo los parámetros entrenables a 12,143,730. Ambos modelos llegaron a un alto nivel de proporción alto en la identificación del personal administrativo y docente de la UNAMBA. El modelo I logró sus objetivos gracias a la cantidad inmensa de neuronas utilizadas; pero poca complejidad en las capas de convolución; sin embargo el modelo II logró sus objetivos gracias a la complejidad de capas dentro de su arquitectura y en contraparte utilizo menos cantidad de neuronas que el modelo I; en ambos modelos fue de mucha importancia los parámetros utilizados: longitud=150, altura=150 de la imagen, Learning Rate=0.01 y función de optimización SGD (Stochastic Gradient Descent); en contraparte encontramos el trabajo de Fernández (2017) “Identificación automática de acciones humanas en secuencias de video para soporte de video vigilancia” en donde utilizo la arquitectura de Inception, diferente a la planteada en esta investigación, y los parámetros que utilizo fueron: función de optimización: Adam, con un learning rate de 0,002478752 y una función de pérdida de tipo categorical_crossentropy. Con los parámetros mencionados anteriormente se obtienen un accuracy (Presisión) de 0.7510, el cual es relativamente bajo en consideración a la diferencia de parámetros y el modelo empleado; por lo cual elegir el modelo y ajustar los parámetros es de suma importancia en un modelo de Deep Learning.

CONCLUSIONES

- En una primera etapa se implementó un modelo con arquitectura clásica VGG16UNAMBA, con lo cual se logró obtener una proporción de 0.9805 de precisión en la identificación del personal administrativo y docente de la UNAMBA.
- En una segunda etapa se implementó un modelo con arquitectura moderna DenseNet121UNAMBA con la cual se logró una proporción alta de 0.9932 de precisión en la identificación del personal administrativo y docente de la UNAMBA.
- Comparando la primera etapa y la segunda, se ha logrado alcanzar la proporción más alta de 0.9932 de precisión en la identificación del personal administrativo y docente de la Universidad Nacional Micaela Bastidas de Apurímac, entrenando 242 clases en un conjunto de 19,700 imágenes de entrenamiento, 8,296 imágenes de validación y un total de 27,996 imágenes que se obtuvieron gracias a la técnica de *Video Scraping* y *data augmentation*, que se aplicó a las imágenes iniciales obtenidas por las técnicas antes mencionadas.
- Finalmente manifestar que se logró aplicar el Deep Learning a través de las Redes Neuronales Convolucionales de manera satisfactoria y se propuso dos modelos modificados VGG16UNAMBA y DenseNet121UNAMBA, siendo ambas muy efectivas en la identificación del personal de la UNAMBA y resaltando que de este último se logró la proporción más alta (0.9932) de precisión en la identificación del personal de la UNAMBA.

RECOMENDACIONES

- Si se quiere continuar trabajando con el aprendizaje supervisado, se recomienda entrenar Redes Neuronales Convolucionales en la fase de recolección de imágenes (fotos de cada persona) en consideración a que en esta investigación se usó clasificadores en cascada basados en funciones de Haar de OpenCV para el reconocimiento de cuerpos humanos en una imagen, pero se experimentó que no son efectivos, ya que generan demasiados falsos negativos al confundir como persona a sombras o formas parecidos al cuerpo humano.
- Para otras investigaciones, se recomienda trabajar con el aprendizaje no supervisado, para evitar, una recolección casi manual de imágenes.

BIBLIOGRAFÍA

Andreas, C., y Sarah, G. (2017). *Introduction to Machine Learning with Python* (Third Realese). O'Reilly Media.

Arenas, A., y Marino, K. J. (2016). *Diseño de un sistema de clasificación de señales de tránsito vehicular utilizando redes neuronales convolucionales* [Tesis para optar el título profesional de Ingeniería de Sistemas]. San Ignacio de Loyola.

Ascarza, F. (2018). *Segmentación automática de textos, mediante redes neuronales convolucionales en imágenes documentos históricos* [Tesis para optar el grado de Magíster en Informática con mención en Ciencias de la Computación]. Pontificia Universidad Católica del Perú.

Barrera, J. F. P., Hurtado, D. A., y Moreno, R. J. (2015). Prediction system of erythemas for phototypes I and II, using deep-learning. *Revista de la Facultad de Ciencias Farmacéuticas y Alimentarias*, *22*(3), 188-196. https://doi.org/10.17533/udea.vitae.v22n3a03

Chen, D., Ren, S., Wei, Y., Cao, X., y Sun, J. (2014). Joint Cascade Face Detection and Alignment. *Computer Vision – ECCV*, 109-122.

Chollet, F. (2018). *Deep learning with Python*. Manning.

Deng, L. (2014). Deep Learning: Methods and Applications. *Foundations and Trends in Signal Processing*, *7*(3-4), 197-387. https://doi.org/10.1561/2000000039

Duval, F. (2016). *Introduction to Neuronal Networks* (First Printing). Davies Company.

Fernández, L. (2017). *Identificación automática de acciones humanas en secuencias de video para soporte de videovigilancia* [Tesis para optar el grado académico de Magíster en Informática con mención en Ciencias de la Computación]. Pontificia Universidad Católica Del Perú.

García, B. (2015). *Implementación de Técnicas de Deep Learning* [Trabajo de Fin de Grado, Universidad de la Laguna]. http://riull.ull.es/xmlui/handle/915/1409

Goodfellow, I., Bengio, Y., y Courville, A. (2016). *Deep Learning*. MIT Press.

Gori, M. (2017). *Machine Learning: A Constraint-Based Approach*. Morgan Kaufmann.

He, K., Zhang, X., Ren, S., y Sun, J. (2015). Deep Residual Learning for Image Recognition. *arXiv:1512.03385 [cs]*. http://arxiv.org/abs/1512.03385

Hearty, J. (2016). *Advanced Machine Learning with Python* (Primera edición). Packt Publishing Ltd.

Huang, G., Liu, Z., van der Maaten, L., y Weinberger, K. Q. (2018). Densely Connected Convolutional Networks. *arXiv:1608.06993 [cs]*.

Hurwitz, J., y Kirsch, D. (2018). *Machine Learning For Dummies*. John Wiley y Sons.

Kamrava, S., Tahmasebi, P., y Sahimi, M. (2019). Enhancing images of shale formations by a hybrid stochastic and deep Learning algorithm. *NeuralNetworks*, *118*, 310-320.

Kastrati, Z., Imran, A. S., y Kurti, A. (2019). Integrating word embeddings and document topics with deep learning in a video classification framework. *Pattern Recognition Letters*, *128*, 85-92.

Krizhevsky, A., Sutskever, I., y Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, *60*(6), 84-90. https://doi.org/10.1145/3065386

Le, T.-T., Lin, C., y Piedad, E. (2019). Deep learning for noninvasive classification of clustered horticultural crops – A case for banana fruit tiers. *Postharvest Biology and Technology Magazine*, *156*.

Lecun, y., Bottou, L., Bengio, Y., y Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278-2324. https://doi.org/10.1109/5.726791

Liu, W., Liao, S., y Hasan, I. (2019, abril 23). Center and Scale Prediction: A Box-free Approach for Object Detection. *ArXiv.Org*, *arXiv:1904.02948v2[cs.CV]*.

Luo, Q., Gao, B., Woo, W., y Yang, Y. (2019). Temporal and spatial deep learning network for infrared thermal defect detection. *NDT and E International*, *108*.

Mateus, A., Ribeiro, D., Miraldo, P., y Nascimento, J. (2019). Efficient and robust Pedestrian Detection using Deep Learning for Human-Aware Navigation. *Robotics and Autonomous Systems*, *113*, 23-37.

Mathias, M., Benenson, R., Pedersoli, M., y Gool, L. V. (2014). Face Detection without Bells and Whistles. *Springer International Publishing Switzerland*, 720–735.

Ponce, P. (2010). *Inteligencia Artificial con aplicaciones a la ingeniería.* (Primera Edición). Alfaomega.

Reátegui, A., y Velasco, M. (2018). *Aplicación Informática para reconocimiento de la especie Camu Camu (Myrciaria Dubia) a través de Redes Neuronales Convolucionales* [Tesis para optar el grado académico de Maestro en Ingeniería de Sistemas]. Universidad Nacional de la Amazonía Peruana.

Rojas, J., y Trujillo, R. (2018). Algoritmo meta-heurístico Firefly aplicado al pre-entrenamiento de redes neuronales artificiales. *Revista Cubana de Ciencias Informáticas, 12*(1), 14-27.

Russell, S. J., y Norvig, P. (2008). *Inteligencia artificial: Un enfoque moderno*. Pearson Educación.

Simonyan, K., y Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv:1409.1556 [cs]*. http://arxiv.org/abs/1409.1556

Springer, S. (2017). *Encyclopedia of Machine Learning and Data Mining, Sammut y Webb, 2nd Ed, 2017: Encyclopedia of Machine Learning and Data Mining*. Bukupedia.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., y Rabinovich, A. (2014). Going Deeper with Convolutions. *arXiv:1409.4842 [cs]*.

Tsampikos, K., Triantafyllidis, G., y Nalpantidis, L. (2019). Deep learning-based visual recognition of rumex for robotic precision farming. *Computers and Electronics in Agriculture, 165*, 85-90.

Turkey, E. (2019). An efficient hybrid deep learning approach for internet security. *Physica A, 535*.

TutorialsPoint, D. (2018). *Tensor Flow, Simply Easy Learning*.

Viola, P., y Jones, M. J. (2004). Robust Real-Time Face Detection. *International Journal of Computer Vision*.

Visa, S., Ramsay, B., y Ralescu, A. (2011). *Confusion Matrix- based Feature Selection*.

Vizcaya, R. (2018). *Deep Learning para la Detección de Peatones y Vehículos* [Maestro en Ciencias de la Computación]. Universidad Autónoma del Estado de México.

Xiao, T., Li, S., Wang, B., Lin, L., y Wang, X. (2017, abril 6). Joint Detection and Identification Feature Learning for Person Search. *arXiv.org*, *arXiv:1604.01850v3 [cs.CV]*.

Ye, L., Gao, L., Martinez, R., Mallants, D., y Bryan, B. (2019). Projecting Australia's forest cover dynamics and exploring influential factors using deep learning. *Environmental Modelling y Software Magazine*, *119*, 407-417.

Zhu, Z., Albadawy, E., Saha, A., Zhang, J., y Harowicz, M. (2019). Deep learning for identifying radiogenomic associations in breast cancer. *Computers in Biology and Medicine Magazine*, *109*, 85-90.

ANEXOS

Anexos 1: Etapa I (entrenamiento 01)

Datos relevantes del entrenamiento (fallido):

Épocas=15	(Épocas del entrenamiento)
Longitud = 96	(Longitud de la imagen)
Altura =96	(Altura de la imagen)
Batch_size = 12	(Cantidad de imágenes a ser procesadas en un lote)
Pasos = 1000	(Pasos de aprendizaje en una época)
N clases = 242	(Número de personas a identificar)
Lr=1e-4	(Learning Rate)
Activation='sigmoid'	(Función sigmoide para la predicción)
Adam	(Optimizador Adam)

1. Modelo generado VGG16 adecuado a los propósitos de la investigación:

Model: "sequential_1" _____
Layer (type) Output Shape Param #
==========
conv2d_1 (Conv2D) (None, 96, 96, 64) 1792
_____
conv2d_2 (Conv2D) (None, 96, 96, 64) 36928
_____
max_pooling2d_1 (MaxPooling2 (None, 48, 48, 64) 0
_____
conv2d_3 (Conv2D) (None, 48, 48, 128) 73856
_____
conv2d_4 (Conv2D) (None, 48, 48, 128) 147584
_____
max_pooling2d_2 (MaxPooling2 (None, 24, 24, 128) 0
_____
conv2d_5 (Conv2D) (None, 24, 24, 256) 295168
_____
conv2d_6 (Conv2D) (None, 24, 24, 256) 590080
_____
conv2d_7 (Conv2D) (None, 24, 24, 256) 590080
_____
max_pooling2d_3 (MaxPooling2 (None, 12, 12, 256) 0
_____
conv2d_8 (Conv2D) (None, 12, 12, 512) 1180160
_____
conv2d_9 (Conv2D) (None, 12, 12, 512) 2359808
_____
conv2d_10 (Conv2D) (None, 12, 12, 512) 2359808
_____
max_pooling2d_4 (MaxPooling2 (None, 6, 6, 512) 0
_____
conv2d_11 (Conv2D) (None, 6, 6, 512) 2359808
_____
conv2d_12 (Conv2D) (None, 6, 6, 512) 2359808
_____
conv2d_13 (Conv2D) (None, 6, 6, 512) 2359808
_____
max_pooling2d_5 (MaxPooling2 (None, 3, 3, 512) 0
_____

flatten_1 (Flatten) (None, 4608) 0

_____

dense_1 (Dense) (None, 4096) 18878464

_____

dense_2 (Dense) (None, 4096) 16781312

_____

dense_3 (Dense) (None, 242) 991474

==========

Total params: 51,365,938
Trainable params: 51,365,938
Non-trainable params: 0

Comentario: El modelo emplea 51,365,938 neuronas, dado que tiene que entrenar esa misma cantidad de parámetros.

2. Evaluación del modelo entrenando

Loss	**Acc**	**Mean_scuared_error**

[5.378613274503937, **0.01124546553808948**, 0.19604898167024068]

Comentario: Como se aprecia se tiene una proporción de 0.01124546553808948 de precisión (Accurancy), por lo que el modelo no logra clasificar casi nada, el entrenamiento es un verdadero fracaso.

Anexos 2: Etapa I (entrenamiento 02)

Datos relevantes del entrenamiento (fallido):

Épocas=15	(Épocas del entrenamiento)
Longitud = 160	(Longitud de la imagen)
Altura = 160	(Altura de la imagen)
Batch_size = 12	(Cantidad de imágenes a ser procesadas en un lote)
Pasos = 1000	(Pasos de aprendizaje en una época)
Nclases = 242	(Número de personas a identificar)
Lr=1e-4	(Learning Rate)
Activation='sigmoid'	(Función sigmoide para la predicción)
Adam	(Optimizador Adam)

1. Modelo generado VGG16 adecuado a los propósitos de la investigación:

En este entrenamiento se hace una modificación en la longitud y altura de las imágenes de (90 x 90) a (160 x 160).
Seguidamente mostramos el modelo utilizado:

Model: "sequential_1"

_____

Layer (type) Output Shape Param #

==========

conv2d_1 (Conv2D) (None, 160, 160, 64) 1792

_____

conv2d_2 (Conv2D) (None, 160, 160, 64) 36928

_____

max_pooling2d_1 (MaxPooling2 (None, 80, 80, 64) 0

_____

conv2d_3 (Conv2D) (None, 80, 80, 128) 73856

_____

conv2d_4 (Conv2D) (None, 80, 80, 128) 147584

_____

max_pooling2d_2 (MaxPooling2 (None, 40, 40, 128) 0

_____

conv2d_5 (Conv2D) (None, 40, 40, 256) 295168

_____

conv2d_6 (Conv2D) (None, 40, 40, 256) 590080

_____

conv2d_7 (Conv2D) (None, 40, 40, 256) 590080

_____

max_pooling2d_3 (MaxPooling2 (None, 20, 20, 256) 0

_____

conv2d_8 (Conv2D) (None, 20, 20, 512) 1180160

_____

conv2d_9 (Conv2D) (None, 20, 20, 512) 2359808

_____

conv2d_10 (Conv2D) (None, 20, 20, 512) 2359808

_____

max_pooling2d_4 (MaxPooling2 (None, 10, 10, 512) 0

_____

conv2d_11 (Conv2D) (None, 10, 10, 512) 2359808

_____

conv2d_12 (Conv2D) (None, 10, 10, 512) 2359808
_____
conv2d_13 (Conv2D) (None, 10, 10, 512) 2359808
_____
max_pooling2d_5 (MaxPooling2 (None, 5, 5, 512) 0
_____
flatten_1 (Flatten) (None, 12800) 0
_____
dense_1 (Dense) (None, 4096) 52432896
_____
dense_2 (Dense) (None, 4096) 16781312
_____
dense_3 (Dense) (None, 242) 991474
==========

Total params: 84,920,370
Trainable params: 84,920,370
Non-trainable params: 0

Comentario: Como se ve el modelo hace un incremento de parámetros de 51,365,938 en el primer entrenamiento a 84,920,370 por solo aumentar el tamaño de estrada de las imágenes.

2. Evaluación del modelo

Loss	Acc	Mean_scuared_error

[5.377772446728098, **0.01124546553808948**, 0.19300543459395833]

Comentario: Como se aprecia el modelo no logra mejorar, respecto al primer entrenamiento y muestra una proporción de 0.01124546553808948 de precisión (Accurancy) similar al entrenamiento anterior, por lo que no logra clasificar casi nada, también es un fracaso el entrenamiento.

Anexos 3: Etapa I (entrenamiento 03)

Datos relevantes del entrenamiento (fallido):

Épocas=15	(Épocas del entrenamiento)
Longitud = 160	(Longitud de la imagen)
Altura =160	(Altura de la imagen)
Batch_size = 12	(Cantidad de imágenes a ser procesadas en un lote)
Pasos = 1000	(Pasos de aprendizaje en una época)
Nclases = 242	(Número de personas a identificar)
Lr=0.01	(Learning Rate)
Activation='sigmoid'	(Función sigmoide para la predicción)
Adam	(Optimizador Adam)

1. Modelo generado VGG16 adecuado a los propósitos de la investigación:

Comentario: El modelo emplea 84,920,370 neuronas, al igual que el entrenamiento N.º 02; pero, se hace una modificación en el Learning Rate a (**lr=0.01**) de lr=0.0001 buscando que el modelo converja y logre aprender.

2. Evaluación del modelo

Loss	**Acc**	**Mean_scuared_error**
[15.16714131941331,	**0.006027000964320154**,	0.04901396246221035]

Comentario: Otro intento más de fracaso, el modelo propuesto no logra clasificar las imágenes de entrada con sus respectivas clases (personas).

Anexos 4: Etapa I (entrenamiento 04)

Datos relevantes del entrenamiento (fallido):

Épocas=15	(Épocas del entrenamiento)
Longitud = 160	(Longitud de la imagen)
Altura =160	(Altura de la imagen)
Batch_size = 12	(Cantidad de imágenes a ser procesadas en un lote)
Pasos = 1000	(Pasos de aprendizaje en una época)
Nclases = 242	(Número de personas a identificar)
Lr=0.1	(Learning Rate)
Activation='sigmoid'	(Función sigmoide para la predicción)
SGD	(Optimizador: Stochastic Gradient Descent)

1. Modelo generado VGG16 adecuado a los propósitos de la investigación:

Comentario: El modelo es similar al entrenamiento Nº 03 con la variación del Learning Rate a 0.1 y el cambio de optimizador de Adam a Stochastic Gradient Descent.

2. Evaluación del modelo

Loss	**Acc**	**Mean_scuared_error**

[15.494299119444252, **0.004957678355501814**, 0.049211028368722425]

Comentario: Como se aprecia en la evaluación el modelo no logra converger (aprender) y no ha mejorado respecto a los intentos 01,02 y 03 de entrenamiento, dados los cambios realizados.

Anexos 5: Etapa final I (entrenamiento 05)

Datos relevantes del entrenamiento (Exitoso):

Épocas=15	(Épocas del entrenamiento)
Longitud = 150	(Longitud de la imagen)
Altura =150	(Altura de la imagen)
Batch_size = 12	(Cantidad de imágenes a ser procesadas en un lote)
Pasos = 1000	(Pasos de aprendizaje en una época)
N clases = 242	(Número de personas a identificar)
Lr=0.01	(Learning Rate)
Activation='sigmoid'	(Función sigmoide para la predicción)
SGD	(Optimizador: Stochastic Gradient Descent)

1. Modelo VGG16UNAMBA adecuado a los propósitos de la investigación:

El modelo es similar al entrenamiento Nº 04 con las siguientes variaciones: se cambia la longitud y la altura a 150 x 150 respectivamente, el modelo carga por defecto los pesos de entrenamiento que vienen con el modelo VGG16 (weights='imagenet'), se cambia el Learning Rate Lr= 0.01 y se cambia dentro del modelo el Flatten() por defecto por GlobalAveragePooling2D() con estos cambios el modelo personalizado se ve de la siguiente manera:

a) Código fuente:

```
from keras.applications import vgg16
from keras.layers import GlobalAveragePooling2D
def VGG16Custom():
  base_model = vgg16.VGG16(include_top=False, weights='imagenet',
                   input_shape=(longitud, altura, 3), classes=nclases)
  x = base_model.output
  x = GlobalAveragePooling2D()(x)
  x = Dense(4096, activation='relu')(x)
  predictions = Dense(nclases, activation='sigmoid') (x)

  model = Model(inputs=base_model.input, outputs=predictions)
return model
```

b) Model: "VGG16UNAMBA"

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	(None, 150, 150, 3)	0
block1_conv1 (Conv2D)	(None, 150, 150, 64)	1792
block1_conv2 (Conv2D)	(None, 150, 150, 64)	36928
block1_pool (MaxPooling2D)	(None, 75, 75, 64)	0
block2_conv1 (Conv2D)	(None, 75, 75, 128)	73856
block2_conv2 (Conv2D)	(None, 75, 75, 128)	147584

block2_pool (MaxPooling2D)	(None, 37, 37, 128)	0
block3_conv1 (Conv2D)	(None, 37, 37, 256)	295168
block3_conv2 (Conv2D)	(None, 37, 37, 256)	590080
block3_conv3 (Conv2D)	(None, 37, 37, 256)	590080
block3_pool (MaxPooling2D)	(None, 18, 18, 256)	0
block4_conv1 (Conv2D)	(None, 18, 18, 512)	1180160
block4_conv2 (Conv2D)	(None, 18, 18, 512)	2359808
block4_conv3 (Conv2D)	(None, 18, 18, 512)	2359808
block4_pool (MaxPooling2D)	(None, 9, 9, 512)	0
block5_conv1 (Conv2D)	(None, 9, 9, 512)	2359808
block5_conv2 (Conv2D)	(None, 9, 9, 512)	2359808
block5_conv3 (Conv2D)	(None, 9, 9, 512)	2359808
block5_pool (MaxPooling2D)	(None, 4, 4, 512)	0
global_average_pooling2d_2 (	(None, 512)	0
dense_3 (Dense)	(None, 4096)	2101248
dense_4 (Dense)	(None, 242)	991474

==========

Total params: 17,807,410
Trainable params: 17,807,410
Non-trainable params: 0

Comentario: El modelo propuesto ha disminuido notablemente el número de parámetros de 84,920,370 en los entrenamientos 2,3 y 4 a 17,807,410. Esta reducción se debe gracias a la introducción de la función *GlobalAveragePooling2D().*

2. Evaluación del modelo entrenado

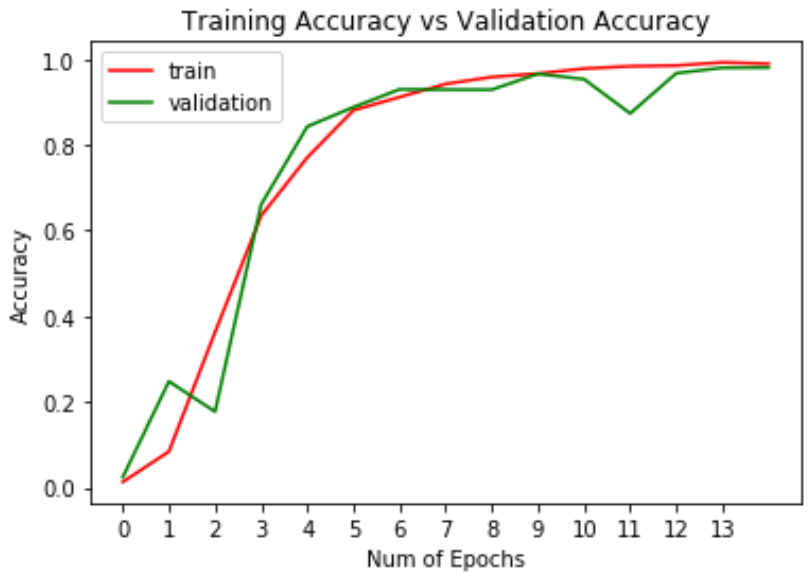
Loss	Acc	Mean_scuared_error

[0.07763004006213177, **0.9805930568948891**, 0.0010402486381952797]

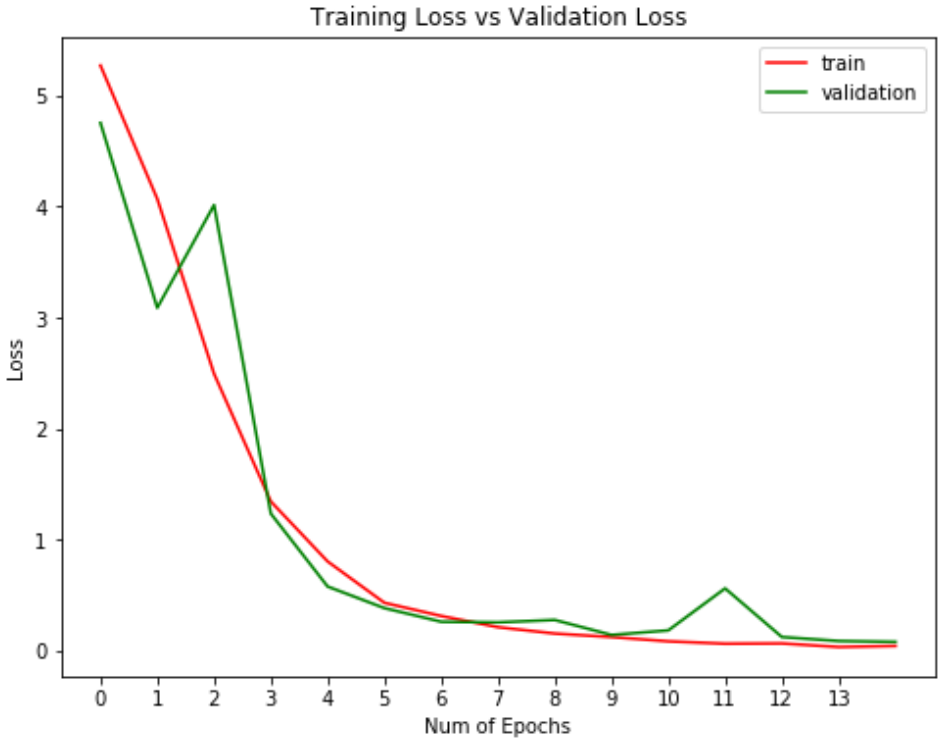
Comentario: Como se aprecia en la evaluación el modelo logra converger y alcanzar una proporción de **0.9805930568948891** de Precisión (Accurancy), usando 8,296 imágenes de validación.

3. Gráficas del proceso de entrenamiento final (etapa I)

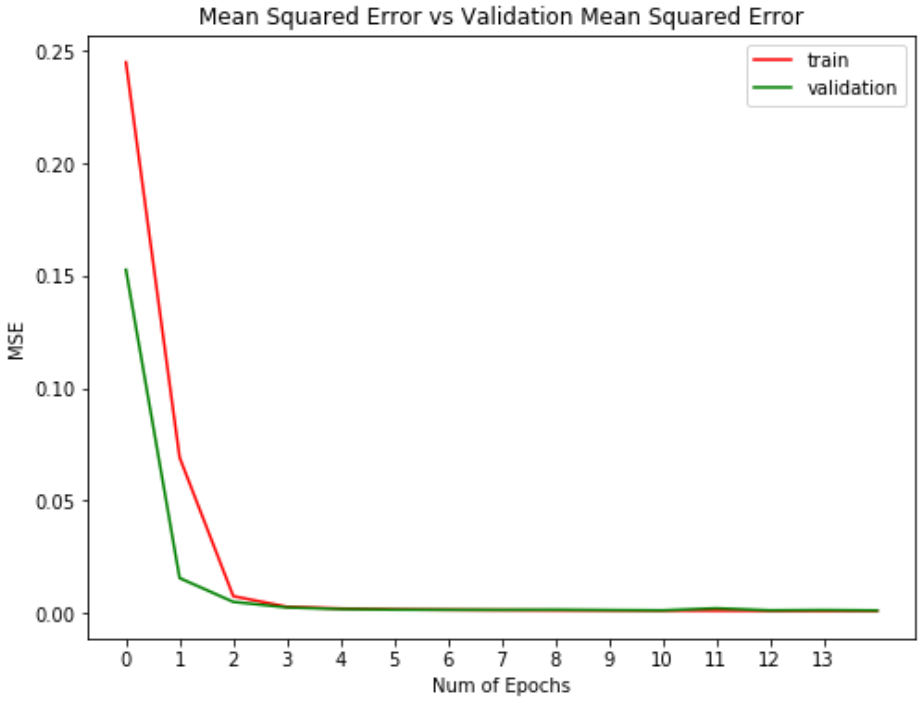
Gráficas que muestra el proceso de entrenamiento con las imágenes de entrenamiento ver sus validación, hasta lograr un nivel de precisión alto a través de las diferentes épocas.



Loss (pérdida): En el siguiente gráfico mostramos la evolución del entrenamiento visto desde una perspectiva de la perdida de información (Loss), tendiendo a cero en las épocas finales; es decir, en un inicio se tuvo altas tasas de perdida (no se lograba identificar a las personas).

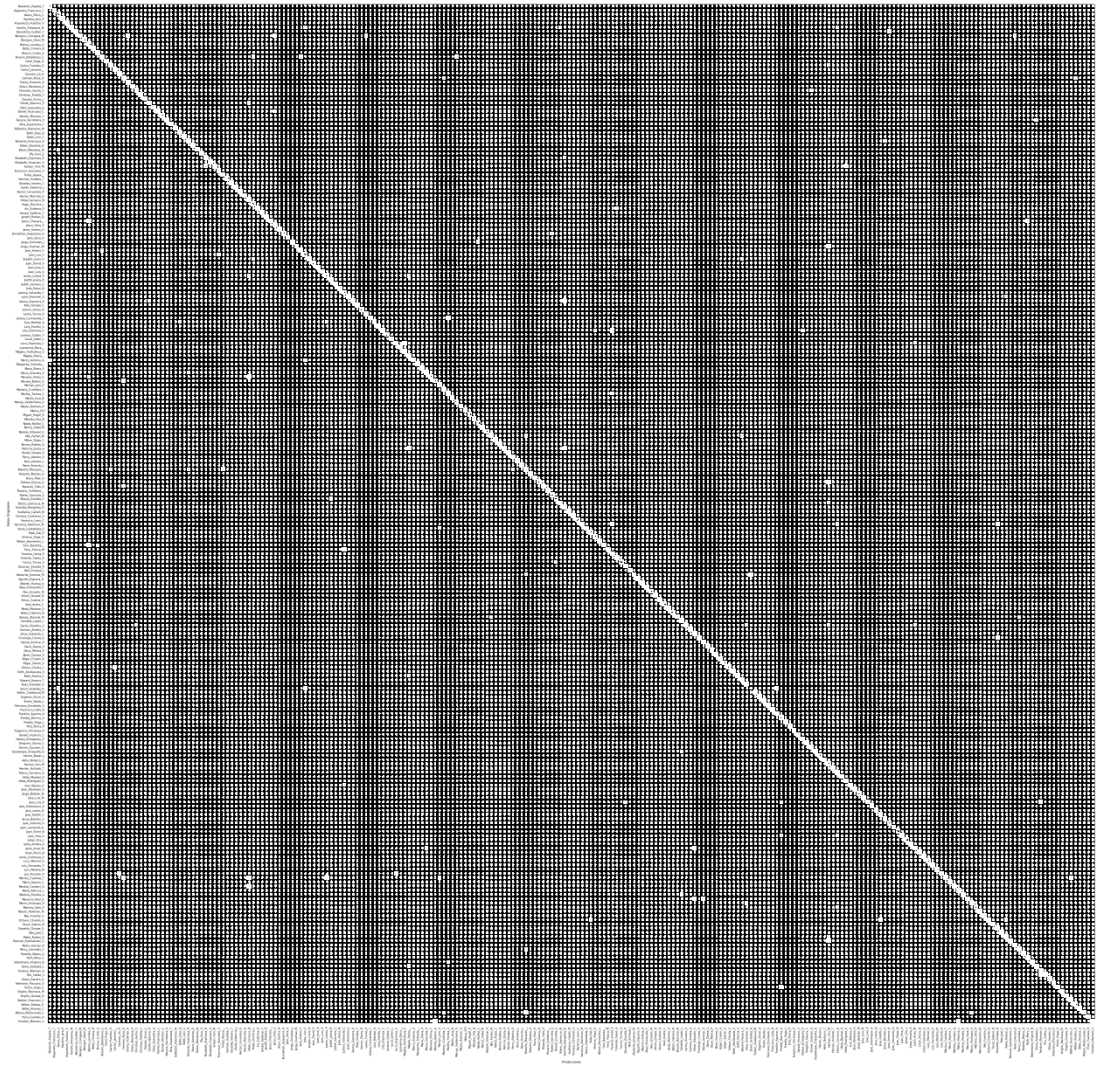


Error cuadrático medio (ECM): Es un estimador que mide el promedio de los errores al cuadrado, en otras palabras, la diferencia entre el estimador y lo que se estima. El ECM es una función de riesgo, correspondiente al valor esperado de la pérdida del error al cuadrado o pérdida cuadrática, donde se aprecia que inicialmente los errores son altos a al pasar las épocas van disminuyendo hasta solaparse en la época 3.

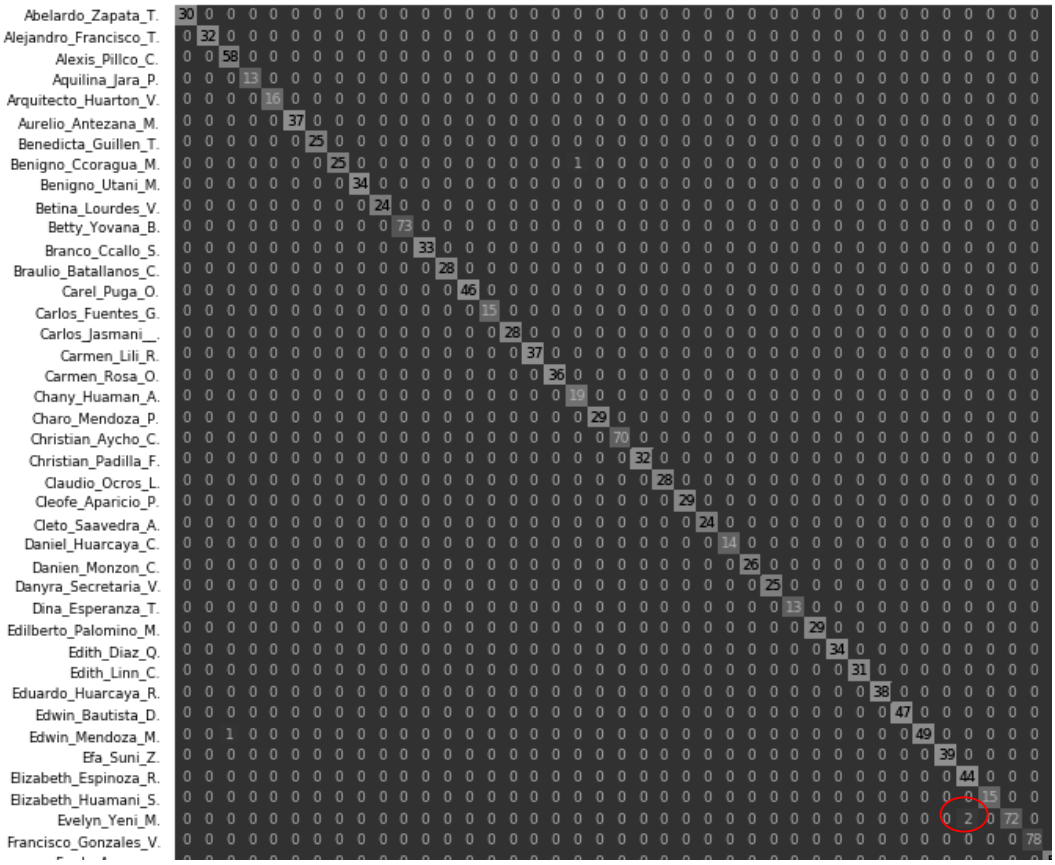


Gráfica de la matriz de confusiones

Una matriz de confusión es una herramienta que nos permite ver el desempeño del modelo en forma general, donde cada columna de la matriz representa la identificación de una persona (clase) lo que el modelo predice, mientras que cada fila representa el ingreso de una imagen real, en total 242 filas (242 personas). Uno de los beneficios de las matrices de confusión es que facilitan ver si el sistema está confundiendo dos clases o no, por lo que en nuestro modelo apreciamos casi una diagonal perfecta, salvo algunos puntos en donde el modelo se equivocó, es decir si se introdujo 50 imágenes en la fila 10, que representa a la persona 10, la salida o la predicción del modelo, también debería mostrarse en la columna 10; si proseguimos en cada fila los resultados devueltos por el modelo deberían salir en el mismo número de columna, lográndose todas las salidas en la diagonal principal (para un caso perfecto), sin embargo vemos varios puntos que no están precisamente en la diagonal principal, son los casos en que el sistema se confundió.



Para cuestiones de muestra ampliaremos un trozo de la imagen (parte superior izquierda) para ver el contenido a detalle de los aciertos y confusiones del modelo, encerramos en un círculo dos confusiones (Evelin_Yeni_M.) en la penúltima fila del gráfico siguiente:



...

⋮

4. Reporte de clasificación del modelo I, propuesto (VGG16UNAMBA)

N.º	Administrativos y docentes (clases)	Precision (Exactitud)	Recall (Sensibilidad)	F1-Score (Puntaje F1)	Support (Imágenes de entrada correctas)	Numero de predicciones del modelo
1	Abelardo_Zapata_T.	0.968	1.000	0.984	30	30
2	Alejandro_Francisco_T.	1.000	1.000	1.000	32	32
3	Alexis_Pillco_C.	0.967	1.000	0.983	58	58
4	Aquilina_Jara_P.	1.000	1.000	1.000	13	13
5	Arquitecto_Huarton_V.	1.000	1.000	1.000	16	16
6	Aurelio_Antezana_M.	1.000	0.925	0.961	40	37
7	Benedicta_Guillen_T.	0.962	0.962	0.962	26	25
8	Benigno_Ccoragua_M.	1.000	0.833	0.909	30	25
9	Benigno_Utani_M.	1.000	1.000	1.000	34	34
10	Betina_Lourdes_V.	0.828	1.000	0.906	24	24
11	Betty_Yovana_B.	1.000	1.000	1.000	73	73
12	Branco_Ccallo_S.	0.971	1.000	0.985	33	33
13	Braulio_Batallanos_C.	0.966	0.903	0.933	31	28
14	Carel_Puga_O.	1.000	1.000	1.000	46	46
15	Carlos_Fuentes_G.	0.938	0.938	0.938	16	15
16	Carlos_Jasmani__.	0.966	1.000	0.982	28	28
17	Carmen_Lili_R.	0.974	1.000	0.987	37	37
18	Carmen_Rosa_O.	0.900	0.947	0.923	38	36
19	Chany_Huaman_A.	0.950	1.000	0.974	19	19

20	Charo_Mendoza_P.	1.000	1.000	1.000	29	29
21	Christian_Aycho_C.	0.986	0.986	0.986	71	70
22	Christian_Padilla_F.	1.000	1.000	1.000	32	32
23	Claudio_Ocros_L.	1.000	1.000	1.000	28	28
24	Cleofe_Aparicio_P.	0.967	0.967	0.967	30	29
25	Cleto_Saavedra_A.	1.000	1.000	1.000	24	24
26	Daniel_Huarcaya_C.	1.000	0.933	0.966	15	14
27	Danien_Monzon_C.	1.000	1.000	1.000	26	26
28	Danyra_Secretaria_V.	1.000	0.962	0.980	26	25
29	Dina_Esperanza_T.	1.000	1.000	1.000	13	13
30	Edilberto_Palomino_M.	1.000	1.000	1.000	29	29
31	Edith_Diaz_Q.	0.971	1.000	0.986	34	34
32	Edith_Linn_C.	1.000	1.000	1.000	31	31
33	Eduardo_Huarcaya_R.	0.950	0.974	0.962	39	38
34	Edwin_Bautista_D.	1.000	1.000	1.000	47	47
35	Edwin_Mendoza_M.	1.000	0.980	0.990	50	49
36	Efa_Suni_Z.	1.000	1.000	1.000	39	39
37	Elizabeth_Espinoza_R.	0.957	0.978	0.967	45	44
38	Elizabeth_Huamani_S.	1.000	1.000	1.000	15	15
39	Evelyn_Yeni_M.	1.000	0.947	0.973	76	72
40	Francisco_Gonzales_V.	0.987	1.000	0.994	78	78
41	Fredy_Apaza__.	0.966	1.000	0.983	57	57
42	German_Huallpa_T.	1.000	1.000	1.000	26	26
43	Griselda_Venero_T.	1.000	1.000	1.000	22	22
44	Guido_Valencia_J.	1.000	1.000	1.000	15	15
45	Hector_Cervantes_P.	1.000	1.000	1.000	29	29
46	Hector_Marcilla_G.	1.000	1.000	1.000	52	52
47	Hilda_Carrasco_B.	0.680	1.000	0.810	17	17
48	Hugo_Quichca_L.	0.967	1.000	0.983	59	59
49	Iris_Eufemia_P.	1.000	0.980	0.990	51	50
50	Ismael_Saldivar_T.	1.000	1.000	1.000	79	79
51	Janeth_Robles_O.	1.000	1.000	1.000	16	16
52	Jesus_Chacara__.	1.000	0.857	0.923	28	24
53	Jesus_Ortiz_S.	0.959	1.000	0.979	70	70
54	Jesus_Santos_H.	1.000	1.000	1.000	24	24
55	Jhonathan_Sequeiros_H	1.000	0.960	0.980	25	24
56	John_Silva_V.	1.000	1.000	1.000	82	82
57	Jorge_Gonzales_J.	1.000	0.964	0.982	28	27
58	Jorge_Guzman_M.	1.000	0.941	0.970	34	32
59	Jose_Antero_F.	0.947	0.947	0.947	19	18
60	Jose_Luis_F.	0.943	0.971	0.957	68	66
61	Joseph_Louis_H.	1.000	0.944	0.971	36	34
62	Juan_David_V.	1.000	1.000	1.000	25	25
63	Juan_Jose_A.	1.000	1.000	1.000	45	45
64	Juan_Luis_A.	1.000	1.000	1.000	37	37
65	Juana_Loaisa_P.	0.878	0.956	0.915	45	43
66	Judith_Juana_R.	0.977	1.000	0.989	43	43
67	Judith_Serrano_C.	1.000	1.000	1.000	14	14
68	Julia_Perez_N.	1.000	1.000	1.000	15	15
69	Justina_Valverde_C.	0.949	1.000	0.974	56	56

70	Justo_Rommel_S.	1.000	0.976	0.988	41	40
71	Karina_Gamarra_P.	1.000	0.941	0.970	51	48
72	Katy_Quispe_C.	1.000	1.000	1.000	55	55
73	Leonor_Garay_R.	1.000	1.000	1.000	48	48
74	Leslie_Torres_P.	0.950	1.000	0.974	19	19
75	Liliana_Curinambe_T.	1.000	0.914	0.955	35	32
76	Lina_Maribel_A.	1.000	0.917	0.957	24	22
77	Lirio_Portillo_S.	1.000	1.000	1.000	36	36
78	Lita_Espinoza_C.	1.000	0.923	0.960	52	48
79	Lorenzo_Quibio_Y.	1.000	1.000	1.000	28	28
80	Lucas_Julian_L.	1.000	1.000	1.000	19	19
81	Lucio_Espinoza_C.	0.979	0.939	0.958	49	46
82	Luzmarina_Baca__.	1.000	1.000	1.000	16	16
83	Magbis_Huillcahua_S.	0.857	1.000	0.923	12	12
84	Magda_Paliza__.	0.915	1.000	0.956	54	54
85	Marco_Antonio_A.	1.000	0.946	0.972	37	35
86	Margarita_Yolanda__.	1.000	1.000	1.000	37	37
87	Maria_Elena_V.	1.000	1.000	1.000	34	34
88	Maria_Graciela_P.	0.978	0.978	0.978	46	45
89	Mariana_Vilma_C.	1.000	0.897	0.945	29	26
90	Mariela_Bañon_A.	0.966	0.966	0.966	58	56
91	Marilyn_Juro_V.	0.943	1.000	0.971	33	33
92	Marleny_Ccoñislla_C.	0.857	1.000	0.923	12	12
93	Martha_Teresa_C.	0.922	0.979	0.949	48	47
94	Martin_Inca_B.	1.000	1.000	1.000	31	31
95	Maruja_Valderrama_V.	0.984	1.000	0.992	63	63
96	Mayte_Damian_H.	1.000	1.000	1.000	37	37
97	Melisa_OCI	1.000	1.000	1.000	48	48
98	Miguel_Angel_R.	1.000	1.000	1.000	24	24
99	Miluska_Vila_B.	1.000	1.000	1.000	18	18
100	Nalda_Nuñez_C.	0.960	1.000	0.980	24	24
101	Nancy_Vidal_M.	1.000	1.000	1.000	14	14
102	Natalie_Villaroel_H.	1.000	1.000	1.000	56	56
103	Nilo_Farfan_M.	0.979	0.958	0.968	48	46
104	Nilton_Rojas_C.	1.000	1.000	1.000	34	34
105	Norma_Robles_Q.	1.000	1.000	1.000	25	25
106	Patricia_Quiza_A.	1.000	0.886	0.939	35	31
107	Paulet_Vargas_S.	1.000	1.000	1.000	30	30
108	Percy_Alberto_E.	0.941	1.000	0.970	16	16
109	Raul_Arbieto_L.	1.000	1.000	1.000	21	21
110	Rene_Romulo_L.	1.000	1.000	1.000	63	63
111	Roberto_Marquez__.	0.939	0.939	0.939	82	77
112	Roberto_Mestas_G.	1.000	1.000	1.000	33	33
113	Rocio_Pilar_H.	1.000	1.000	1.000	21	21
114	Rofwel_Quisca_Q.	1.000	0.960	0.980	50	48
115	Rolando_Tello_P.	1.000	0.972	0.986	36	35
116	Rosario_Collabino__.	1.000	1.000	1.000	36	36
117	Ruben_Sanchez_E.	0.957	1.000	0.978	22	22
118	Sheyla_Zunilda_C.	0.985	0.985	0.985	67	66
119	Silano_Llamocca_M.	0.976	0.976	0.976	41	40

120	Sumilda_Bengolea_D.	0.933	1.000	0.966	70	70
121	Svetlania_Callalli_M.	1.000	1.000	1.000	36	36
122	Tomaza_Contreras_S.	1.000	1.000	1.000	46	46
123	Veronica_Leon_C.	1.000	1.000	1.000	26	26
124	Veronica_Palomino_M.	1.000	0.921	0.959	38	35
125	Viane_Castañeda_S.	1.000	0.964	0.982	28	27
126	Vidal_Del_P.	0.963	1.000	0.981	26	26
127	Vinerva_Vega_O.	0.985	1.000	0.992	66	66
128	Wilson_Barrientos_L.	1.000	1.000	1.000	13	13
129	Yeni_Bautista__.	1.000	0.952	0.976	63	60
130	Yeny_Yesica_M.	1.000	0.956	0.977	45	43
131	Yesenia_Ustúa_P.	0.932	1.000	0.965	55	55
132	Yolanda_Yepez_A.	0.950	1.000	0.974	19	19
133	Yurica_Torres_S.	1.000	0.985	0.992	65	64
134	Zacarias_Zavalla_P.	0.971	1.000	0.985	33	33
135	Abel_Enrique_J.	1.000	1.000	1.000	18	18
136	Abelardo_Jimenez_M.	1.000	0.900	0.947	30	27
137	Agustín_Elguera_H.	1.000	1.000	1.000	19	19
138	Alberto_Huihua_A.	1.000	1.000	1.000	50	50
139	Alejo_Pumacallo_F.	1.000	1.000	1.000	12	12
140	Alex_Ernesto_M.	1.000	1.000	1.000	26	26
141	Arturo_Quispe_Q.	1.000	1.000	1.000	23	23
142	Arturo_Suárez_O.	1.000	1.000	1.000	68	68
143	Axel_Andre_C.	1.000	1.000	1.000	54	54
144	Beda_Marlene_O.	1.000	1.000	1.000	28	28
145	Belen_Cabrera_N.	1.000	1.000	1.000	67	67
146	Braulio_Barzola_M.	1.000	0.900	0.947	20	18
147	Candida_Lopez_L.	0.949	1.000	0.974	37	37
148	Carlos_Rivelino_S.	1.000	0.857	0.923	28	24
149	Carmen_Amelia_A.	1.000	1.000	1.000	32	32
150	César_Eduardo_C.	0.946	1.000	0.972	35	35
151	Crisologo_Conza_A.	1.000	0.969	0.984	64	62
152	Daniel_Amilcar_P.	0.933	1.000	0.966	14	14
153	Dario_Dante_S.	1.000	1.000	1.000	15	15
154	Deysi_Melisa_P.	1.000	1.000	1.000	32	32
155	Ebert_Gomez_A.	1.000	1.000	1.000	14	14
156	Edgar_Crispin_H.	1.000	1.000	1.000	20	20
157	Edgar_Zenón_V.	1.000	1.000	1.000	30	30
158	Edison_Chiclla_C.	1.000	0.976	0.988	42	41
159	Edith_Abollaneda_S.	1.000	1.000	1.000	18	18
160	Edith_Karina_C.	1.000	0.938	0.968	16	15
161	Edward_Ilasaca_C.	0.957	1.000	0.978	44	44
162	Elias_Aranibar_A.	0.923	1.000	0.960	24	24
163	Erech_Ordoñez_R.	0.978	0.946	0.962	93	88
164	Esther_Calatayud_M.	1.000	1.000	1.000	15	15
165	Eugenio_Oscar_A.	1.000	1.000	1.000	26	26
166	Evelin_Naida_L.	1.000	1.000	1.000	31	31
167	Feliciano_Escobedo_S.	1.000	1.000	1.000	37	37
168	Francisco_Calle_R.	1.000	1.000	1.000	34	34
169	Franklin_Aguirre_H.	0.938	1.000	0.968	30	30

170	Freddy_Barrios_S.	0.920	1.000	0.958	46	46
171	Freddy_Vega_L.	1.000	1.000	1.000	31	31
172	Fritz_Percy_P.	1.000	1.000	1.000	14	14
173	Fulgencio_Vilcanqui_P.	1.000	1.000	1.000	11	11
174	Gerald_Vicencio_C.	1.000	1.000	1.000	32	32
175	Gladys_Echegaray_P.	0.980	1.000	0.990	48	48
176	Gregorio_Gauna_C.	1.000	1.000	1.000	15	15
177	Grover_Guzman_G.	0.923	1.000	0.960	12	12
178	Guadalupe_Chaquilla_	1.000	0.969	0.984	32	31
179	Hector_Basan_J.	1.000	1.000	1.000	36	36
180	Helio_Nolasco_C.	1.000	1.000	1.000	15	15
181	Hernan_Yari_M.	0.722	1.000	0.839	26	26
182	Hernán_Hurtado_T.	1.000	1.000	1.000	42	42
183	Hilario_Carrasco_K.	0.818	1.000	0.900	18	18
184	Hilda_Maribel_H.	1.000	1.000	1.000	43	43
185	Hilda_Rodriguez_A.	0.913	1.000	0.955	21	21
186	Ivon_Nieves_A.	1.000	0.971	0.985	34	33
187	Jhon_Abraham_A.	1.000	1.000	1.000	20	20
188	Jorge_Beltrán_M.	1.000	1.000	1.000	40	40
189	Jose_Luis_M.	1.000	1.000	1.000	12	12
190	Jose_Luis_P.	1.000	0.907	0.951	43	39
191	Jose_Sotomayor_A.	1.000	1.000	1.000	17	17
192	Jose_carlos_N.	1.000	1.000	1.000	51	51
193	José_Adolfo_C.	0.982	1.000	0.991	54	54
194	Josue_Benites_V.	0.938	1.000	0.968	15	15
195	Juan_Antonio_H.	0.968	1.000	0.984	30	30
196	Juan_Leonardo_D.	1.000	1.000	1.000	15	15
197	Juan_Silver_B.	1.000	1.000	1.000	54	54
198	Juan_Viza_A.	1.000	0.953	0.976	43	41
199	Julian_Ore_L.	1.000	1.000	1.000	34	34
200	Justa_Amalia_A.	1.000	1.000	1.000	28	28
201	Justo_Arias_M.	0.941	0.941	0.941	34	32
202	Lilian_Rocio_B.	1.000	1.000	1.000	13	13
203	Lintol_Contreras_S.	1.000	1.000	1.000	33	33
204	Lucy_Marisol_G.	1.000	1.000	1.000	58	58
205	Luis_Fernando_P.	1.000	1.000	1.000	41	41
206	Luis_Medina_M.	1.000	1.000	1.000	49	49
207	Luis_Ricardo_P.	1.000	0.952	0.976	42	40
208	Mariluz_Cuentas_T.	1.000	0.645	0.784	31	20
209	Mario_Aquino_C.	1.000	1.000	1.000	22	22
210	Maritza_Condori_Q.	1.000	0.857	0.923	14	12
211	María_Patricia_L.	0.978	1.000	0.989	45	45
212	Marleny_Peralta_A.	1.000	0.958	0.979	24	23
213	Mauricio_Raul_E.	1.000	0.955	0.977	44	42
214	Mauro_Huayapa_H.	0.971	0.971	0.971	34	33
215	Máximo_Soto_P.	1.000	0.980	0.990	49	48
216	Nelson_Palemón_M.	1.000	1.000	1.000	16	16
217	Niki_Franklin_F.	1.000	1.000	1.000	13	13
218	Octavio_Chambi_A.	1.000	0.941	0.970	51	48
219	Oscar_Sabino_B.	1.000	1.000	1.000	27	27

220	Oswaldo_Quispe_Q.	0.906	1.000	0.951	29	29
221	Otto_Joel_C.	1.000	1.000	1.000	31	31
222	Pablo_Ruben_Z.	0.957	0.978	0.968	46	45
223	Pascual_Ayamamani_C	1.000	0.923	0.960	26	24
224	Pedro_Hernan_P.	0.962	1.000	0.980	25	25
225	Percy_Leonidas_C.	0.964	0.964	0.964	28	27
226	Rodolfo_Matos_O.	1.000	1.000	1.000	30	30
227	Ruth_Mery_C.	0.939	1.000	0.969	31	31
228	Sebastiana_Virginia_B.	1.000	0.923	0.960	13	12
229	Silvia_Soledad_L.	0.963	0.963	0.963	27	26
230	Teodoro_Mamani_A.	0.907	1.000	0.951	39	39
231	Tito_Yañez_C.	1.000	0.875	0.933	16	14
232	Ulises_Sandro_Q.	1.000	1.000	1.000	37	37
233	Valeriano_Paucara_O.	1.000	1.000	1.000	15	15
234	Victor_Hugo_S.	1.000	0.867	0.929	15	13
235	Virgilio_Machaca_M.	1.000	1.000	1.000	17	17
236	Virgilio_Quispe_D.	1.000	1.000	1.000	16	16
237	Walker_Huaccani_C.	0.920	1.000	0.958	23	23
238	Wilber_Robles_D.	0.960	1.000	0.980	24	24
239	Willie_Alvarez_C.	1.000	1.000	1.000	12	12
240	Wilson_Mollocondo_F.	1.000	0.918	0.957	49	45
241	Yhon_Fuentes_H.	1.000	1.000	1.000	17	17
242	Yonatan_Mamani_C.	1.000	0.824	0.903	17	14
	Macro avg (Promedio)	**0.981185**	**0.980602**	**0.979901**		
	TOTAL				**8296**	**8135**
	Accuracy (Precisión)		**0.980593**			

Promedio de muestras	**34.3**
Desviación estándar	**16.6**
Mediana	**31**
Mínima muestra	**11**
Máxima muestra	**93**

Comentario: Por lo cual podemos concluir que **0.980593** es la proporción de precisión (Accurancy) más alta alcanzada hasta el momento con el modelo de red neuronal convolucional propuesto en esta etapa final I (VGG16UNAMBA).

$$p_1 = \frac{\text{Número correcto de predicciones}}{\text{Número total de imágenes}} = \frac{8135}{8296} = 0.980593$$

Donde p_1 es la primera proporción hallada.

Anexos 6: Etapa II (entrenamiento 06)

Datos relevantes del entrenamiento (No satisfactorio):

Épocas=15	(Épocas del entrenamiento)
Longitud = 128	(Longitud de la imagen)
Altura =128	(Altura de la imagen)
Batch_size = 12	(Cantidad de imágenes a ser procesadas en un lote)
Pasos = 1000	(Pasos de aprendizaje en una época)
N clases = 242	(Número de personas a identificar)
Lr=1e-4	(Learning Rate)
Activation='sigmoid'	(Función sigmoide para la predicción)
Adam	(Optimizador Adam (lr=1e-4, decay=1e-9))

1. Modelo generado personalizado (DenseNet121UNAMBA) para los propósitos de la investigación:

Comentario: Es un modelo de CNN con arquitectura moderna pero con un nivel de profundidad alto (121 capas) la modificación que se hizo son básicamente los que se muestran en el código fuente python de la siguiente manera:

a) Código fuente

```
from keras.applications import DenseNet121
from keras.layers import GlobalAveragePooling2D
from keras.models import Model
def DenseNet121UNAMBA():
  base_model = DenseNet121(include_top=False, weights='imagenet',
                    input_shape=(longitud, altura, 3), classes=nclases)
  x = base_model.output
  x = GlobalAveragePooling2D()(x)
  x = Dense(1024, activation='relu')(x)
  predictions = Dense(nclases, activation='sigmoid') (x)
  model = Model(inputs=base_model.input, outputs=predictions)
  return model
```

Model: "DenseNet121UNAMBA"

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 128, 128, 3)	0	
zero_padding2d_1 (ZeroPadding2D	(None, 134, 134, 3)	0	input_1[0][0]
conv1/conv (Conv2D)	(None, 64, 64, 64)	9408	zero_padding2d_1[0][0]
conv1/bn (BatchNormalization)	(None, 64, 64, 64)	256	conv1/conv[0][0]
conv1/relu (Activation)	(None, 64, 64, 64)	0	conv1/bn[0][0]
zero_padding2d_2 (ZeroPadding2D	(None, 66, 66, 64)	0	conv1/relu[0][0]

pool1 (MaxPooling2D) (None, 32, 32, 64) 0 zero_padding2d_2[0][0]

conv2_block1_0_bn (BatchNormali (None, 32, 32, 64) 256 pool1[0][0]

conv2_block1_0_relu (Activation (None, 32, 32, 64) 0 conv2_block1_0_bn[0][0]

conv2_block1_1_conv (Conv2D) (None, 32, 32, 128) 8192 conv2_block1_0_relu[0][0]

conv2_block1_1_bn (BatchNormali (None, 32, 32, 128) 512 conv2_block1_1_conv[0][0]

conv2_block1_1_relu (Activation (None, 32, 32, 128) 0 conv2_block1_1_bn[0][0]

conv2_block1_2_conv (Conv2D) (None, 32, 32, 32) 36864 conv2_block1_1_relu[0][0]

conv2_block1_concat (Concatenat (None, 32, 32, 96) 0 pool1[0][0] conv2_block1_2_conv[0][0]

conv2_block2_0_bn (BatchNormali (None, 32, 32, 96) 384 conv2_block1_concat[0][0]

conv2_block2_0_relu (Activation (None, 32, 32, 96) 0 conv2_block2_0_bn[0][0]

conv2_block2_1_conv (Conv2D) (None, 32, 32, 128) 12288 conv2_block2_0_relu[0][0]

conv2_block2_1_bn (BatchNormali (None, 32, 32, 128) 512 conv2_block2_1_conv[0][0]

conv2_block2_1_relu (Activation (None, 32, 32, 128) 0 conv2_block2_1_bn[0][0]

conv2_block2_2_conv (Conv2D) (None, 32, 32, 32) 36864 conv2_block2_1_relu[0][0]

conv2_block2_concat (Concatenat (None, 32, 32, 128) 0 conv2_block1_concat[0][0] conv2_block2_2_conv[0][0]

conv2_block3_0_bn (BatchNormali (None, 32, 32, 128) 512 conv2_block2_concat[0][0]

conv2_block3_0_relu (Activation (None, 32, 32, 128) 0 conv2_block3_0_bn[0][0]

conv2_block3_1_conv (Conv2D) (None, 32, 32, 128) 16384 conv2_block3_0_relu[0][0]

conv2_block3_1_bn (BatchNormali (None, 32, 32, 128) 512 conv2_block3_1_conv[0][0]

conv2_block3_1_relu (Activation (None, 32, 32, 128) 0 conv2_block3_1_bn[0][0]

conv2_block3_2_conv (Conv2D) (None, 32, 32, 32) 36864 conv2_block3_1_relu[0][0]

conv2_block3_concat (Concatenat (None, 32, 32, 160) 0 conv2_block2_concat[0][0] conv2_block3_2_conv[0][0]

conv2_block4_0_bn (BatchNormali (None, 32, 32, 160) 640 conv2_block3_concat[0][0]

conv2_block4_0_relu (Activation (None, 32, 32, 160) 0 conv2_block4_0_bn[0][0]

conv2_block4_1_conv (Conv2D) (None, 32, 32, 128) 20480 conv2_block4_0_relu[0][0]

conv2_block4_1_bn (BatchNormali (None, 32, 32, 128) 512 conv2_block4_1_conv[0][0]

conv2_block4_1_relu (Activation (None, 32, 32, 128) 0 conv2_block4_1_bn[0][0]

conv2_block4_2_conv (Conv2D) (None, 32, 32, 32) 36864 conv2_block4_1_relu[0][0]

conv2_block4_concat (Concatenat (None, 32, 32, 192) 0 conv2_block3_concat[0][0] conv2_block4_2_conv[0][0]

conv2_block5_0_bn (BatchNormali (None, 32, 32, 192) 768 conv2_block4_concat[0][0]

conv2_block5_0_relu (Activation (None, 32, 32, 192) 0 conv2_block5_0_bn[0][0]

conv2_block5_1_conv (Conv2D) (None, 32, 32, 128) 24576 conv2_block5_0_relu[0][0]

conv2_block5_1_bn (BatchNormali (None, 32, 32, 128) 512 conv2_block5_1_conv[0][0]

conv2_block5_1_relu (Activation (None, 32, 32, 128) 0 conv2_block5_1_bn[0][0]

conv2_block5_2_conv (Conv2D) (None, 32, 32, 32) 36864 conv2_block5_1_relu[0][0]

conv2_block5_concat (Concatenat (None, 32, 32, 224) 0 conv2_block4_concat[0][0] conv2_block5_2_conv[0][0]

conv2_block6_0_bn (BatchNormali (None, 32, 32, 224) 896 conv2_block5_concat[0][0]

conv2_block6_0_relu (Activation (None, 32, 32, 224) 0 conv2_block6_0_bn[0][0]

conv2_block6_1_conv (Conv2D) (None, 32, 32, 128) 28672 conv2_block6_0_relu[0][0]

conv2_block6_1_bn (BatchNormali (None, 32, 32, 128) 512 conv2_block6_1_conv[0][0]

conv2_block6_1_relu (Activation (None, 32, 32, 128) 0 conv2_block6_1_bn[0][0]

conv2_block6_2_conv (Conv2D) (None, 32, 32, 32) 36864 conv2_block6_1_relu[0][0]

conv2_block6_concat (Concatenat (None, 32, 32, 256) 0 conv2_block5_concat[0][0] conv2_block6_2_conv[0][0]

pool2_bn (BatchNormalization) (None, 32, 32, 256) 1024 conv2_block6_concat[0][0]

pool2_relu (Activation) (None, 32, 32, 256) 0 pool2_bn[0][0]

pool2_conv (Conv2D) (None, 32, 32, 128) 32768 pool2_relu[0][0]

pool2_pool (AveragePooling2D) (None, 16, 16, 128) 0 pool2_conv[0][0]

conv3_block1_0_bn (BatchNormali (None, 16, 16, 128) 512 pool2_pool[0][0]

conv3_block1_0_relu (Activation (None, 16, 16, 128) 0 conv3_block1_0_bn[0][0]

conv3_block1_1_conv (Conv2D) (None, 16, 16, 128) 16384 conv3_block1_0_relu[0][0]

conv3_block1_1_bn (BatchNormali (None, 16, 16, 128) 512 conv3_block1_1_conv[0][0]

conv3_block1_1_relu (Activation (None, 16, 16, 128) 0 conv3_block1_1_bn[0][0]

conv3_block1_2_conv (Conv2D) (None, 16, 16, 32) 36864 conv3_block1_1_relu[0][0]

conv3_block1_concat (Concatenat (None, 16, 16, 160) 0 pool2_pool[0][0] conv3_block1_2_conv[0][0]

conv3_block2_0_bn (BatchNormali (None, 16, 16, 160) 640 conv3_block1_concat[0][0]

conv3_block2_0_relu (Activation (None, 16, 16, 160) 0 conv3_block2_0_bn[0][0]

conv3_block2_1_conv (Conv2D) (None, 16, 16, 128) 20480 conv3_block2_0_relu[0][0]

conv3_block2_1_bn (BatchNormali (None, 16, 16, 128) 512 conv3_block2_1_conv[0][0]

conv3_block2_1_relu (Activation (None, 16, 16, 128) 0 conv3_block2_1_bn[0][0]

conv3_block2_2_conv (Conv2D) (None, 16, 16, 32) 36864 conv3_block2_1_relu[0][0]

conv3_block2_concat (Concatenat (None, 16, 16, 192) 0 conv3_block1_concat[0][0] conv3_block2_2_conv[0][0]

conv3_block3_0_bn (BatchNormali (None, 16, 16, 192) 768 conv3_block2_concat[0][0]

conv3_block3_0_relu (Activation (None, 16, 16, 192) 0 conv3_block3_0_bn[0][0]

conv3_block3_1_conv (Conv2D) (None, 16, 16, 128) 24576 conv3_block3_0_relu[0][0]

conv3_block3_1_bn (BatchNormali (None, 16, 16, 128) 512 conv3_block3_1_conv[0][0]

conv3_block3_1_relu (Activation (None, 16, 16, 128) 0 conv3_block3_1_bn[0][0]

conv3_block3_2_conv (Conv2D) (None, 16, 16, 32) 36864 conv3_block3_1_relu[0][0]

conv3_block3_concat (Concatenat (None, 16, 16, 224) 0 conv3_block2_concat[0][0] conv3_block3_2_conv[0][0]

conv3_block4_0_bn (BatchNormali (None, 16, 16, 224) 896 conv3_block3_concat[0][0]

conv3_block4_0_relu (Activation (None, 16, 16, 224) 0 conv3_block4_0_bn[0][0]

conv3_block4_1_conv (Conv2D) (None, 16, 16, 128) 28672 conv3_block4_0_relu[0][0]

conv3_block4_1_bn (BatchNormali (None, 16, 16, 128) 512 conv3_block4_1_conv[0][0]

conv3_block4_1_relu (Activation (None, 16, 16, 128) 0 conv3_block4_1_bn[0][0]

conv3_block4_2_conv (Conv2D) (None, 16, 16, 32) 36864 conv3_block4_1_relu[0][0]

conv3_block4_concat (Concatenat (None, 16, 16, 256) 0 conv3_block3_concat[0][0] conv3_block4_2_conv[0][0]

conv3_block5_0_bn (BatchNormali (None, 16, 16, 256) 1024 conv3_block4_concat[0][0]

conv3_block5_0_relu (Activation (None, 16, 16, 256) 0 conv3_block5_0_bn[0][0]

conv3_block5_1_conv (Conv2D) (None, 16, 16, 128) 32768 conv3_block5_0_relu[0][0]

conv3_block5_1_bn (BatchNormali (None, 16, 16, 128) 512 conv3_block5_1_conv[0][0]

conv3_block5_1_relu (Activation (None, 16, 16, 128) 0 conv3_block5_1_bn[0][0]

conv3_block5_2_conv (Conv2D) (None, 16, 16, 32) 36864 conv3_block5_1_relu[0][0]

conv3_block5_concat (Concatenat (None, 16, 16, 288) 0 conv3_block4_concat[0][0] conv3_block5_2_conv[0][0]

conv3_block6_0_bn (BatchNormali (None, 16, 16, 288) 1152 conv3_block5_concat[0][0]

conv3_block6_0_relu (Activation (None, 16, 16, 288) 0 conv3_block6_0_bn[0][0]

conv3_block6_1_conv (Conv2D) (None, 16, 16, 128) 36864 conv3_block6_0_relu[0][0]

conv3_block6_1_bn (BatchNormali (None, 16, 16, 128) 512 conv3_block6_1_conv[0][0]

conv3_block6_1_relu (Activation (None, 16, 16, 128) 0 conv3_block6_1_bn[0][0]

conv3_block6_2_conv (Conv2D) (None, 16, 16, 32) 36864 conv3_block6_1_relu[0][0]

conv3_block6_concat (Concatenat (None, 16, 16, 320) 0 conv3_block5_concat[0][0] conv3_block6_2_conv[0][0]

conv3_block7_0_bn (BatchNormali (None, 16, 16, 320) 1280 conv3_block6_concat[0][0]

conv3_block7_0_relu (Activation (None, 16, 16, 320) 0 conv3_block7_0_bn[0][0]

conv3_block7_1_conv (Conv2D) (None, 16, 16, 128) 40960 conv3_block7_0_relu[0][0]

conv3_block7_1_bn (BatchNormali (None, 16, 16, 128) 512 conv3_block7_1_conv[0][0]

conv3_block7_1_relu (Activation (None, 16, 16, 128) 0 conv3_block7_1_bn[0][0]

conv3_block7_2_conv (Conv2D) (None, 16, 16, 32) 36864 conv3_block7_1_relu[0][0]

conv3_block7_concat (Concatenat (None, 16, 16, 352) 0 conv3_block6_concat[0][0] conv3_block7_2_conv[0][0]

conv3_block8_0_bn (BatchNormali (None, 16, 16, 352) 1408 conv3_block7_concat[0][0]

conv3_block8_0_relu (Activation (None, 16, 16, 352) 0 conv3_block8_0_bn[0][0]

conv3_block8_1_conv (Conv2D) (None, 16, 16, 128) 45056 conv3_block8_0_relu[0][0]

conv3_block8_1_bn (BatchNormali (None, 16, 16, 128) 512 conv3_block8_1_conv[0][0]

conv3_block8_1_relu (Activation (None, 16, 16, 128) 0 conv3_block8_1_bn[0][0]

conv3_block8_2_conv (Conv2D) (None, 16, 16, 32) 36864 conv3_block8_1_relu[0][0]

conv3_block8_concat (Concatenat (None, 16, 16, 384) 0 conv3_block7_concat[0][0] conv3_block8_2_conv[0][0]

conv3_block9_0_bn (BatchNormali (None, 16, 16, 384) 1536 conv3_block8_concat[0][0]

conv3_block9_0_relu (Activation (None, 16, 16, 384) 0 conv3_block9_0_bn[0][0]

conv3_block9_1_conv (Conv2D) (None, 16, 16, 128) 49152 conv3_block9_0_relu[0][0]

conv3_block9_1_bn (BatchNormali (None, 16, 16, 128) 512 conv3_block9_1_conv[0][0]

conv3_block9_1_relu (Activation (None, 16, 16, 128) 0 conv3_block9_1_bn[0][0]

conv3_block9_2_conv (Conv2D) (None, 16, 16, 32) 36864 conv3_block9_1_relu[0][0]

conv3_block9_concat (Concatenat (None, 16, 16, 416) 0 conv3_block8_concat[0][0] conv3_block9_2_conv[0][0]

conv3_block10_0_bn (BatchNormal (None, 16, 16, 416) 1664 conv3_block9_concat[0][0]

conv3_block10_0_relu (Activatio (None, 16, 16, 416) 0 conv3_block10_0_bn[0][0]

conv3_block10_1_conv (Conv2D) (None, 16, 16, 128) 53248 conv3_block10_0_relu[0][0]

conv3_block10_1_bn (BatchNormal (None, 16, 16, 128) 512 conv3_block10_1_conv[0][0]

conv3_block10_1_relu (Activatio (None, 16, 16, 128) 0 conv3_block10_1_bn[0][0]

conv3_block10_2_conv (Conv2D) (None, 16, 16, 32) 36864 conv3_block10_1_relu[0][0]

conv3_block10_concat (Concatena (None, 16, 16, 448) 0 conv3_block9_concat[0][0] conv3_block10_2_conv[0][0]

conv3_block11_0_bn (BatchNormal (None, 16, 16, 448) 1792 conv3_block10_concat[0][0]

conv3_block11_0_relu (Activatio (None, 16, 16, 448) 0 conv3_block11_0_bn[0][0]

conv3_block11_1_conv (Conv2D) (None, 16, 16, 128) 57344 conv3_block11_0_relu[0][0]

conv3_block11_1_bn (BatchNormal (None, 16, 16, 128) 512 conv3_block11_1_conv[0][0]

conv3_block11_1_relu (Activatio (None, 16, 16, 128) 0 conv3_block11_1_bn[0][0]

conv3_block11_2_conv (Conv2D) (None, 16, 16, 32) 36864 conv3_block11_1_relu[0][0]

conv3_block11_concat (Concatena (None, 16, 16, 480) 0 conv3_block10_concat[0][0] conv3_block11_2_conv[0][0]

conv3_block12_0_bn (BatchNormal (None, 16, 16, 480) 1920 conv3_block11_concat[0][0]

conv3_block12_0_relu (Activatio (None, 16, 16, 480) 0 conv3_block12_0_bn[0][0]

conv3_block12_1_conv (Conv2D) (None, 16, 16, 128) 61440 conv3_block12_0_relu[0][0]

conv3_block12_1_bn (BatchNormal (None, 16, 16, 128) 512 conv3_block12_1_conv[0][0]

conv3_block12_1_relu (Activatio (None, 16, 16, 128) 0 conv3_block12_1_bn[0][0]

conv3_block12_2_conv (Conv2D) (None, 16, 16, 32) 36864 conv3_block12_1_relu[0][0]

conv3_block12_concat (Concatena (None, 16, 16, 512) 0 conv3_block11_concat[0][0] conv3_block12_2_conv[0][0]

pool3_bn (BatchNormalization) (None, 16, 16, 512) 2048 conv3_block12_concat[0][0]

pool3_relu (Activation) (None, 16, 16, 512) 0 pool3_bn[0][0]

pool3_conv (Conv2D) (None, 16, 16, 256) 131072 pool3_relu[0][0]

pool3_pool (AveragePooling2D) (None, 8, 8, 256) 0 pool3_conv[0][0]

conv4_block1_0_bn (BatchNormali (None, 8, 8, 256) 1024 pool3_pool[0][0]

conv4_block1_0_relu (Activation (None, 8, 8, 256) 0 conv4_block1_0_bn[0][0]

conv4_block1_1_conv (Conv2D) (None, 8, 8, 128) 32768 conv4_block1_0_relu[0][0]

conv4_block1_1_bn (BatchNormali (None, 8, 8, 128) 512 conv4_block1_1_conv[0][0]

conv4_block1_1_relu (Activation (None, 8, 8, 128) 0 conv4_block1_1_bn[0][0]

conv4_block1_2_conv (Conv2D) (None, 8, 8, 32) 36864 conv4_block1_1_relu[0][0]

conv4_block1_concat (Concatenat (None, 8, 8, 288) 0 pool3_pool[0][0] conv4_block1_2_conv[0][0]

conv4_block2_0_bn (BatchNormali (None, 8, 8, 288) 1152 conv4_block1_concat[0][0]

conv4_block2_0_relu (Activation (None, 8, 8, 288) 0 conv4_block2_0_bn[0][0]

conv4_block2_1_conv (Conv2D) (None, 8, 8, 128) 36864 conv4_block2_0_relu[0][0]

conv4_block2_1_bn (BatchNormali (None, 8, 8, 128) 512 conv4_block2_1_conv[0][0]

conv4_block2_1_relu (Activation (None, 8, 8, 128) 0 conv4_block2_1_bn[0][0]

conv4_block2_2_conv (Conv2D) (None, 8, 8, 32) 36864 conv4_block2_1_relu[0][0]

conv4_block2_concat (Concatenat (None, 8, 8, 320) 0 conv4_block1_concat[0][0] conv4_block2_2_conv[0][0]

conv4_block3_0_bn (BatchNormali (None, 8, 8, 320) 1280 conv4_block2_concat[0][0]

conv4_block3_0_relu (Activation (None, 8, 8, 320) 0 conv4_block3_0_bn[0][0]

conv4_block3_1_conv (Conv2D) (None, 8, 8, 128) 40960 conv4_block3_0_relu[0][0]

conv4_block3_1_bn (BatchNormali (None, 8, 8, 128) 512 conv4_block3_1_conv[0][0]

conv4_block3_1_relu (Activation (None, 8, 8, 128) 0 conv4_block3_1_bn[0][0]

conv4_block3_2_conv (Conv2D) (None, 8, 8, 32) 36864 conv4_block3_1_relu[0][0]

conv4_block3_concat (Concatenat (None, 8, 8, 352) 0 conv4_block2_concat[0][0] conv4_block3_2_conv[0][0]

conv4_block4_0_bn (BatchNormali (None, 8, 8, 352) 1408 conv4_block3_concat[0][0]

conv4_block4_0_relu (Activation (None, 8, 8, 352) 0 conv4_block4_0_bn[0][0]

conv4_block4_1_conv (Conv2D) (None, 8, 8, 128) 45056 conv4_block4_0_relu[0][0]

conv4_block4_1_bn (BatchNormali (None, 8, 8, 128) 512 conv4_block4_1_conv[0][0]

conv4_block4_1_relu (Activation (None, 8, 8, 128) 0 conv4_block4_1_bn[0][0]

conv4_block4_2_conv (Conv2D) (None, 8, 8, 32) 36864 conv4_block4_1_relu[0][0]

conv4_block4_concat (Concatenat (None, 8, 8, 384) 0 conv4_block3_concat[0][0] conv4_block4_2_conv[0][0]

conv4_block5_0_bn (BatchNormali (None, 8, 8, 384) 1536 conv4_block4_concat[0][0]

conv4_block5_0_relu (Activation (None, 8, 8, 384) 0 conv4_block5_0_bn[0][0]

conv4_block5_1_conv (Conv2D) (None, 8, 8, 128) 49152 conv4_block5_0_relu[0][0]

conv4_block5_1_bn (BatchNormali (None, 8, 8, 128) 512 conv4_block5_1_conv[0][0]

conv4_block5_1_relu (Activation (None, 8, 8, 128) 0 conv4_block5_1_bn[0][0]

conv4_block5_2_conv (Conv2D) (None, 8, 8, 32) 36864 conv4_block5_1_relu[0][0]

conv4_block5_concat (Concatenat (None, 8, 8, 416) 0 conv4_block4_concat[0][0] conv4_block5_2_conv[0][0]

conv4_block6_0_bn (BatchNormali (None, 8, 8, 416) 1664 conv4_block5_concat[0][0]

conv4_block6_0_relu (Activation (None, 8, 8, 416) 0 conv4_block6_0_bn[0][0]

conv4_block6_1_conv (Conv2D) (None, 8, 8, 128) 53248 conv4_block6_0_relu[0][0]

conv4_block6_1_bn (BatchNormali (None, 8, 8, 128) 512 conv4_block6_1_conv[0][0]

conv4_block6_1_relu (Activation (None, 8, 8, 128) 0 conv4_block6_1_bn[0][0]

conv4_block6_2_conv (Conv2D) (None, 8, 8, 32) 36864 conv4_block6_1_relu[0][0]

conv4_block6_concat (Concatenat (None, 8, 8, 448) 0 conv4_block5_concat[0][0] conv4_block6_2_conv[0][0]

conv4_block7_0_bn (BatchNormali (None, 8, 8, 448) 1792 conv4_block6_concat[0][0]

conv4_block7_0_relu (Activation (None, 8, 8, 448) 0 conv4_block7_0_bn[0][0]

conv4_block7_1_conv (Conv2D) (None, 8, 8, 128) 57344 conv4_block7_0_relu[0][0]

conv4_block7_1_bn (BatchNormali (None, 8, 8, 128) 512 conv4_block7_1_conv[0][0]

conv4_block7_1_relu (Activation (None, 8, 8, 128) 0 conv4_block7_1_bn[0][0]

conv4_block7_2_conv (Conv2D) (None, 8, 8, 32) 36864 conv4_block7_1_relu[0][0]

conv4_block7_concat (Concatenat (None, 8, 8, 480) 0 conv4_block6_concat[0][0] conv4_block7_2_conv[0][0]

conv4_block8_0_bn (BatchNormali (None, 8, 8, 480) 1920 conv4_block7_concat[0][0]

conv4_block8_0_relu (Activation (None, 8, 8, 480) 0 conv4_block8_0_bn[0][0]

conv4_block8_1_conv (Conv2D) (None, 8, 8, 128) 61440 conv4_block8_0_relu[0][0]

conv4_block8_1_bn (BatchNormali (None, 8, 8, 128) 512 conv4_block8_1_conv[0][0]

conv4_block8_1_relu (Activation (None, 8, 8, 128) 0 conv4_block8_1_bn[0][0]

conv4_block8_2_conv (Conv2D) (None, 8, 8, 32) 36864 conv4_block8_1_relu[0][0]

conv4_block8_concat (Concatenat (None, 8, 8, 512) 0 conv4_block7_concat[0][0] conv4_block8_2_conv[0][0]

conv4_block9_0_bn (BatchNormali (None, 8, 8, 512) 2048 conv4_block8_concat[0][0]

conv4_block9_0_relu (Activation (None, 8, 8, 512) 0 conv4_block9_0_bn[0][0]

conv4_block9_1_conv (Conv2D) (None, 8, 8, 128) 65536 conv4_block9_0_relu[0][0]

conv4_block9_1_bn (BatchNormali (None, 8, 8, 128) 512 conv4_block9_1_conv[0][0]

conv4_block9_1_relu (Activation (None, 8, 8, 128) 0 conv4_block9_1_bn[0][0]

conv4_block9_2_conv (Conv2D) (None, 8, 8, 32) 36864 conv4_block9_1_relu[0][0]

conv4_block9_concat (Concatenat (None, 8, 8, 544) 0 conv4_block8_concat[0][0] conv4_block9_2_conv[0][0]

conv4_block10_0_bn (BatchNormal (None, 8, 8, 544) 2176 conv4_block9_concat[0][0]

conv4_block10_0_relu (Activatio (None, 8, 8, 544) 0 conv4_block10_0_bn[0][0]

conv4_block10_1_conv (Conv2D) (None, 8, 8, 128) 69632 conv4_block10_0_relu[0][0]

conv4_block10_1_bn (BatchNormal (None, 8, 8, 128) 512 conv4_block10_1_conv[0][0]

conv4_block10_1_relu (Activatio (None, 8, 8, 128) 0 conv4_block10_1_bn[0][0]

conv4_block10_2_conv (Conv2D) (None, 8, 8, 32) 36864 conv4_block10_1_relu[0][0]

conv4_block10_concat (Concatena (None, 8, 8, 576) 0 conv4_block9_concat[0][0] conv4_block10_2_conv[0][0]

conv4_block11_0_bn (BatchNormal (None, 8, 8, 576) 2304 conv4_block10_concat[0][0]

conv4_block11_0_relu (Activatio (None, 8, 8, 576) 0 conv4_block11_0_bn[0][0]

conv4_block11_1_conv (Conv2D) (None, 8, 8, 128) 73728 conv4_block11_0_relu[0][0]

conv4_block11_1_bn (BatchNormal (None, 8, 8, 128) 512 conv4_block11_1_conv[0][0]

conv4_block11_1_relu (Activatio (None, 8, 8, 128) 0 conv4_block11_1_bn[0][0]

conv4_block11_2_conv (Conv2D) (None, 8, 8, 32) 36864 conv4_block11_1_relu[0][0]

conv4_block11_concat (Concatena (None, 8, 8, 608) 0 conv4_block10_concat[0][0] conv4_block11_2_conv[0][0]

conv4_block12_0_bn (BatchNormal (None, 8, 8, 608) 2432 conv4_block11_concat[0][0]

conv4_block12_0_relu (Activatio (None, 8, 8, 608) 0 conv4_block12_0_bn[0][0]

conv4_block12_1_conv (Conv2D) (None, 8, 8, 128) 77824 conv4_block12_0_relu[0][0]

conv4_block12_1_bn (BatchNormal (None, 8, 8, 128) 512 conv4_block12_1_conv[0][0]

conv4_block12_1_relu (Activatio (None, 8, 8, 128) 0 conv4_block12_1_bn[0][0]

conv4_block12_2_conv (Conv2D) (None, 8, 8, 32) 36864 conv4_block12_1_relu[0][0]

conv4_block12_concat (Concatena (None, 8, 8, 640) 0 conv4_block11_concat[0][0] conv4_block12_2_conv[0][0]

conv4_block13_0_bn (BatchNormal (None, 8, 8, 640) 2560 conv4_block12_concat[0][0]

conv4_block13_0_relu (Activatio (None, 8, 8, 640) 0 conv4_block13_0_bn[0][0]

conv4_block13_1_conv (Conv2D) (None, 8, 8, 128) 81920 conv4_block13_0_relu[0][0]

conv4_block13_1_bn (BatchNormal (None, 8, 8, 128) 512 conv4_block13_1_conv[0][0]

conv4_block13_1_relu (Activatio (None, 8, 8, 128) 0 conv4_block13_1_bn[0][0]

conv4_block13_2_conv (Conv2D) (None, 8, 8, 32) 36864 conv4_block13_1_relu[0][0]

conv4_block13_concat (Concatena (None, 8, 8, 672) 0 conv4_block12_concat[0][0] conv4_block13_2_conv[0][0]

conv4_block14_0_bn (BatchNormal (None, 8, 8, 672) 2688 conv4_block13_concat[0][0]

conv4_block14_0_relu (Activatio (None, 8, 8, 672) 0 conv4_block14_0_bn[0][0]

conv4_block14_1_conv (Conv2D) (None, 8, 8, 128) 86016 conv4_block14_0_relu[0][0]

conv4_block14_1_bn (BatchNormal (None, 8, 8, 128) 512 conv4_block14_1_conv[0][0]

conv4_block14_1_relu (Activatio (None, 8, 8, 128) 0 conv4_block14_1_bn[0][0]

conv4_block14_2_conv (Conv2D) (None, 8, 8, 32) 36864 conv4_block14_1_relu[0][0]

conv4_block14_concat (Concatena (None, 8, 8, 704) 0 conv4_block13_concat[0][0] conv4_block14_2_conv[0][0]

conv4_block15_0_bn (BatchNormal (None, 8, 8, 704) 2816 conv4_block14_concat[0][0]

conv4_block15_0_relu (Activatio (None, 8, 8, 704) 0 conv4_block15_0_bn[0][0]

conv4_block15_1_conv (Conv2D) (None, 8, 8, 128) 90112 conv4_block15_0_relu[0][0]

conv4_block15_1_bn (BatchNormal (None, 8, 8, 128) 512 conv4_block15_1_conv[0][0]

conv4_block15_1_relu (Activatio (None, 8, 8, 128) 0 conv4_block15_1_bn[0][0]

conv4_block15_2_conv (Conv2D) (None, 8, 8, 32) 36864 conv4_block15_1_relu[0][0]

conv4_block15_concat (Concatena (None, 8, 8, 736) 0 conv4_block14_concat[0][0] conv4_block15_2_conv[0][0]

conv4_block16_0_bn (BatchNormal (None, 8, 8, 736) 2944 conv4_block15_concat[0][0]

conv4_block16_0_relu (Activatio (None, 8, 8, 736) 0 conv4_block16_0_bn[0][0]

conv4_block16_1_conv (Conv2D) (None, 8, 8, 128) 94208 conv4_block16_0_relu[0][0]

conv4_block16_1_bn (BatchNormal (None, 8, 8, 128) 512 conv4_block16_1_conv[0][0]

conv4_block16_1_relu (Activatio (None, 8, 8, 128) 0 conv4_block16_1_bn[0][0]

conv4_block16_2_conv (Conv2D) (None, 8, 8, 32) 36864 conv4_block16_1_relu[0][0]

conv4_block16_concat (Concatena (None, 8, 8, 768) 0 conv4_block15_concat[0][0] conv4_block16_2_conv[0][0]

conv4_block17_0_bn (BatchNormal (None, 8, 8, 768) 3072 conv4_block16_concat[0][0]

conv4_block17_0_relu (Activatio (None, 8, 8, 768) 0 conv4_block17_0_bn[0][0]

conv4_block17_1_conv (Conv2D) (None, 8, 8, 128) 98304 conv4_block17_0_relu[0][0]

conv4_block17_1_bn (BatchNormal (None, 8, 8, 128) 512 conv4_block17_1_conv[0][0]

conv4_block17_1_relu (Activatio (None, 8, 8, 128) 0 conv4_block17_1_bn[0][0]

conv4_block17_2_conv (Conv2D) (None, 8, 8, 32) 36864 conv4_block17_1_relu[0][0]

conv4_block17_concat (Concatena (None, 8, 8, 800) 0 conv4_block16_concat[0][0] conv4_block17_2_conv[0][0]

conv4_block18_0_bn (BatchNormal (None, 8, 8, 800) 3200 conv4_block17_concat[0][0]

conv4_block18_0_relu (Activatio (None, 8, 8, 800) 0 conv4_block18_0_bn[0][0]

conv4_block18_1_conv (Conv2D) (None, 8, 8, 128) 102400 conv4_block18_0_relu[0][0]

conv4_block18_1_bn (BatchNormal (None, 8, 8, 128) 512 conv4_block18_1_conv[0][0]

conv4_block18_1_relu (Activatio (None, 8, 8, 128) 0 conv4_block18_1_bn[0][0]

conv4_block18_2_conv (Conv2D) (None, 8, 8, 32) 36864 conv4_block18_1_relu[0][0]

conv4_block18_concat (Concatena (None, 8, 8, 832) 0 conv4_block17_concat[0][0] conv4_block18_2_conv[0][0]

conv4_block19_0_bn (BatchNormal (None, 8, 8, 832) 3328 conv4_block18_concat[0][0]

conv4_block19_0_relu (Activatio (None, 8, 8, 832) 0 conv4_block19_0_bn[0][0]

conv4_block19_1_conv (Conv2D) (None, 8, 8, 128) 106496 conv4_block19_0_relu[0][0]

conv4_block19_1_bn (BatchNormal (None, 8, 8, 128) 512 conv4_block19_1_conv[0][0]

conv4_block19_1_relu (Activatio (None, 8, 8, 128) 0 conv4_block19_1_bn[0][0]

conv4_block19_2_conv (Conv2D) (None, 8, 8, 32) 36864 conv4_block19_1_relu[0][0]

conv4_block19_concat (Concatena (None, 8, 8, 864) 0 conv4_block18_concat[0][0] conv4_block19_2_conv[0][0]

conv4_block20_0_bn (BatchNormal (None, 8, 8, 864) 3456 conv4_block19_concat[0][0]

conv4_block20_0_relu (Activatio (None, 8, 8, 864) 0 conv4_block20_0_bn[0][0]

conv4_block20_1_conv (Conv2D) (None, 8, 8, 128) 110592 conv4_block20_0_relu[0][0]

conv4_block20_1_bn (BatchNormal (None, 8, 8, 128) 512 conv4_block20_1_conv[0][0]

conv4_block20_1_relu (Activatio (None, 8, 8, 128) 0 conv4_block20_1_bn[0][0]

conv4_block20_2_conv (Conv2D) (None, 8, 8, 32) 36864 conv4_block20_1_relu[0][0]

conv4_block20_concat (Concatena (None, 8, 8, 896) 0 conv4_block19_concat[0][0] conv4_block20_2_conv[0][0]

conv4_block21_0_bn (BatchNormal (None, 8, 8, 896) 3584 conv4_block20_concat[0][0]

conv4_block21_0_relu (Activatio (None, 8, 8, 896) 0 conv4_block21_0_bn[0][0]

conv4_block21_1_conv (Conv2D) (None, 8, 8, 128) 114688 conv4_block21_0_relu[0][0]

conv4_block21_1_bn (BatchNormal (None, 8, 8, 128) 512 conv4_block21_1_conv[0][0]

conv4_block21_1_relu (Activatio (None, 8, 8, 128) 0 conv4_block21_1_bn[0][0]

conv4_block21_2_conv (Conv2D) (None, 8, 8, 32) 36864 conv4_block21_1_relu[0][0]

conv4_block21_concat (Concatena (None, 8, 8, 928) 0 conv4_block20_concat[0][0] conv4_block21_2_conv[0][0]

conv4_block22_0_bn (BatchNormal (None, 8, 8, 928) 3712 conv4_block21_concat[0][0]

conv4_block22_0_relu (Activatio (None, 8, 8, 928) 0 conv4_block22_0_bn[0][0]

conv4_block22_1_conv (Conv2D) (None, 8, 8, 128) 118784 conv4_block22_0_relu[0][0]

conv4_block22_1_bn (BatchNormal (None, 8, 8, 128) 512 conv4_block22_1_conv[0][0]

conv4_block22_1_relu (Activatio (None, 8, 8, 128) 0 conv4_block22_1_bn[0][0]

conv4_block22_2_conv (Conv2D) (None, 8, 8, 32) 36864 conv4_block22_1_relu[0][0]

conv4_block22_concat (Concatena (None, 8, 8, 960) 0 conv4_block21_concat[0][0] conv4_block22_2_conv[0][0]

conv4_block23_0_bn (BatchNormal (None, 8, 8, 960) 3840 conv4_block22_concat[0][0]

conv4_block23_0_relu (Activatio (None, 8, 8, 960) 0 conv4_block23_0_bn[0][0]

conv4_block23_1_conv (Conv2D) (None, 8, 8, 128) 122880 conv4_block23_0_relu[0][0]

conv4_block23_1_bn (BatchNormal (None, 8, 8, 128) 512 conv4_block23_1_conv[0][0]

conv4_block23_1_relu (Activatio (None, 8, 8, 128) 0 conv4_block23_1_bn[0][0]

conv4_block23_2_conv (Conv2D) (None, 8, 8, 32) 36864 conv4_block23_1_relu[0][0]

conv4_block23_concat (Concatena (None, 8, 8, 992) 0 conv4_block22_concat[0][0] conv4_block23_2_conv[0][0]

conv4_block24_0_bn (BatchNormal (None, 8, 8, 992) 3968 conv4_block23_concat[0][0]

conv4_block24_0_relu (Activatio (None, 8, 8, 992) 0 conv4_block24_0_bn[0][0]

conv4_block24_1_conv (Conv2D) (None, 8, 8, 128) 126976 conv4_block24_0_relu[0][0]

conv4_block24_1_bn (BatchNormal (None, 8, 8, 128) 512 conv4_block24_1_conv[0][0]

conv4_block24_1_relu (Activatio (None, 8, 8, 128) 0 conv4_block24_1_bn[0][0]

conv4_block24_2_conv (Conv2D) (None, 8, 8, 32) 36864 conv4_block24_1_relu[0][0]

conv4_block24_concat (Concatena (None, 8, 8, 1024) 0 conv4_block23_concat[0][0] conv4_block24_2_conv[0][0]

pool4_bn (BatchNormalization) (None, 8, 8, 1024) 4096 conv4_block24_concat[0][0]

pool4_relu (Activation) (None, 8, 8, 1024) 0 pool4_bn[0][0]

pool4_conv (Conv2D) (None, 8, 8, 512) 524288 pool4_relu[0][0]

pool4_pool (AveragePooling2D) (None, 4, 4, 512) 0 pool4_conv[0][0]

conv5_block1_0_bn (BatchNormali (None, 4, 4, 512) 2048 pool4_pool[0][0]

conv5_block1_0_relu (Activation (None, 4, 4, 512) 0 conv5_block1_0_bn[0][0]

conv5_block1_1_conv (Conv2D) (None, 4, 4, 128) 65536 conv5_block1_0_relu[0][0]

conv5_block1_1_bn (BatchNormali (None, 4, 4, 128) 512 conv5_block1_1_conv[0][0]

conv5_block1_1_relu (Activation (None, 4, 4, 128) 0 conv5_block1_1_bn[0][0]

conv5_block1_2_conv (Conv2D) (None, 4, 4, 32) 36864 conv5_block1_1_relu[0][0]

conv5_block1_concat (Concatenat (None, 4, 4, 544) 0 pool4_pool[0][0] conv5_block1_2_conv[0][0]

conv5_block2_0_bn (BatchNormali (None, 4, 4, 544) 2176 conv5_block1_concat[0][0]

conv5_block2_0_relu (Activation (None, 4, 4, 544) 0 conv5_block2_0_bn[0][0]

conv5_block2_1_conv (Conv2D) (None, 4, 4, 128) 69632 conv5_block2_0_relu[0][0]

conv5_block2_1_bn (BatchNormali (None, 4, 4, 128) 512 conv5_block2_1_conv[0][0]

conv5_block2_1_relu (Activation (None, 4, 4, 128) 0 conv5_block2_1_bn[0][0]

conv5_block2_2_conv (Conv2D) (None, 4, 4, 32) 36864 conv5_block2_1_relu[0][0]

conv5_block2_concat (Concatenat (None, 4, 4, 576) 0 conv5_block1_concat[0][0] conv5_block2_2_conv[0][0]

conv5_block3_0_bn (BatchNormali (None, 4, 4, 576) 2304 conv5_block2_concat[0][0]

conv5_block3_0_relu (Activation (None, 4, 4, 576) 0 conv5_block3_0_bn[0][0]

conv5_block3_1_conv (Conv2D) (None, 4, 4, 128) 73728 conv5_block3_0_relu[0][0]

conv5_block3_1_bn (BatchNormali (None, 4, 4, 128) 512 conv5_block3_1_conv[0][0]

conv5_block3_1_relu (Activation (None, 4, 4, 128) 0 conv5_block3_1_bn[0][0]

conv5_block3_2_conv (Conv2D) (None, 4, 4, 32) 36864 conv5_block3_1_relu[0][0]

conv5_block3_concat (Concatenat (None, 4, 4, 608) 0 conv5_block2_concat[0][0] conv5_block3_2_conv[0][0]

conv5_block4_0_bn (BatchNormali (None, 4, 4, 608) 2432 conv5_block3_concat[0][0]

conv5_block4_0_relu (Activation (None, 4, 4, 608) 0 conv5_block4_0_bn[0][0]

conv5_block4_1_conv (Conv2D) (None, 4, 4, 128) 77824 conv5_block4_0_relu[0][0]

conv5_block4_1_bn (BatchNormali (None, 4, 4, 128) 512 conv5_block4_1_conv[0][0]

conv5_block4_1_relu (Activation (None, 4, 4, 128) 0 conv5_block4_1_bn[0][0]

conv5_block4_2_conv (Conv2D) (None, 4, 4, 32) 36864 conv5_block4_1_relu[0][0]

conv5_block4_concat (Concatenat (None, 4, 4, 640) 0 conv5_block3_concat[0][0] conv5_block4_2_conv[0][0]

conv5_block5_0_bn (BatchNormali (None, 4, 4, 640) 2560 conv5_block4_concat[0][0]

conv5_block5_0_relu (Activation (None, 4, 4, 640) 0 conv5_block5_0_bn[0][0]

conv5_block5_1_conv (Conv2D) (None, 4, 4, 128) 81920 conv5_block5_0_relu[0][0]

conv5_block5_1_bn (BatchNormali (None, 4, 4, 128) 512 conv5_block5_1_conv[0][0]

conv5_block5_1_relu (Activation (None, 4, 4, 128) 0 conv5_block5_1_bn[0][0]

conv5_block5_2_conv (Conv2D) (None, 4, 4, 32) 36864 conv5_block5_1_relu[0][0]

conv5_block5_concat (Concatenat (None, 4, 4, 672) 0 conv5_block4_concat[0][0] conv5_block5_2_conv[0][0]

conv5_block6_0_bn (BatchNormali (None, 4, 4, 672) 2688 conv5_block5_concat[0][0]

conv5_block6_0_relu (Activation (None, 4, 4, 672) 0 conv5_block6_0_bn[0][0]

conv5_block6_1_conv (Conv2D) (None, 4, 4, 128) 86016 conv5_block6_0_relu[0][0]

conv5_block6_1_bn (BatchNormali (None, 4, 4, 128) 512 conv5_block6_1_conv[0][0]

conv5_block6_1_relu (Activation (None, 4, 4, 128) 0 conv5_block6_1_bn[0][0]

conv5_block6_2_conv (Conv2D) (None, 4, 4, 32) 36864 conv5_block6_1_relu[0][0]

conv5_block6_concat (Concatenat (None, 4, 4, 704) 0 conv5_block5_concat[0][0] conv5_block6_2_conv[0][0]

conv5_block7_0_bn (BatchNormali (None, 4, 4, 704) 2816 conv5_block6_concat[0][0]

conv5_block7_0_relu (Activation (None, 4, 4, 704) 0 conv5_block7_0_bn[0][0]

conv5_block7_1_conv (Conv2D) (None, 4, 4, 128) 90112 conv5_block7_0_relu[0][0]

conv5_block7_1_bn (BatchNormali (None, 4, 4, 128) 512 conv5_block7_1_conv[0][0]

conv5_block7_1_relu (Activation (None, 4, 4, 128) 0 conv5_block7_1_bn[0][0]

conv5_block7_2_conv (Conv2D) (None, 4, 4, 32) 36864 conv5_block7_1_relu[0][0]

conv5_block7_concat (Concatenat (None, 4, 4, 736) 0 conv5_block6_concat[0][0] conv5_block7_2_conv[0][0]

conv5_block8_0_bn (BatchNormali (None, 4, 4, 736) 2944 conv5_block7_concat[0][0]

conv5_block8_0_relu (Activation (None, 4, 4, 736) 0 conv5_block8_0_bn[0][0]

conv5_block8_1_conv (Conv2D) (None, 4, 4, 128) 94208 conv5_block8_0_relu[0][0]

conv5_block8_1_bn (BatchNormali (None, 4, 4, 128) 512 conv5_block8_1_conv[0][0]

conv5_block8_1_relu (Activation (None, 4, 4, 128) 0 conv5_block8_1_bn[0][0]

conv5_block8_2_conv (Conv2D) (None, 4, 4, 32) 36864 conv5_block8_1_relu[0][0]

conv5_block8_concat (Concatenat (None, 4, 4, 768) 0 conv5_block7_concat[0][0] conv5_block8_2_conv[0][0]

conv5_block9_0_bn (BatchNormali (None, 4, 4, 768) 3072 conv5_block8_concat[0][0]

conv5_block9_0_relu (Activation (None, 4, 4, 768) 0 conv5_block9_0_bn[0][0]

conv5_block9_1_conv (Conv2D) (None, 4, 4, 128) 98304 conv5_block9_0_relu[0][0]

conv5_block9_1_bn (BatchNormali (None, 4, 4, 128) 512 conv5_block9_1_conv[0][0]

conv5_block9_1_relu (Activation (None, 4, 4, 128) 0 conv5_block9_1_bn[0][0]

conv5_block9_2_conv (Conv2D) (None, 4, 4, 32) 36864 conv5_block9_1_relu[0][0]

conv5_block9_concat (Concatenat (None, 4, 4, 800) 0 conv5_block8_concat[0][0] conv5_block9_2_conv[0][0]

conv5_block10_0_bn (BatchNormal (None, 4, 4, 800) 3200 conv5_block9_concat[0][0]

conv5_block10_0_relu (Activatio (None, 4, 4, 800) 0 conv5_block10_0_bn[0][0]

conv5_block10_1_conv (Conv2D) (None, 4, 4, 128) 102400 conv5_block10_0_relu[0][0]

conv5_block10_1_bn (BatchNormal (None, 4, 4, 128) 512 conv5_block10_1_conv[0][0]

conv5_block10_1_relu (Activatio (None, 4, 4, 128) 0 conv5_block10_1_bn[0][0]

conv5_block10_2_conv (Conv2D) (None, 4, 4, 32) 36864 conv5_block10_1_relu[0][0]

conv5_block10_concat (Concatena (None, 4, 4, 832) 0 conv5_block9_concat[0][0] conv5_block10_2_conv[0][0]

conv5_block11_0_bn (BatchNormal (None, 4, 4, 832) 3328 conv5_block10_concat[0][0]

conv5_block11_0_relu (Activatio (None, 4, 4, 832) 0 conv5_block11_0_bn[0][0]

conv5_block11_1_conv (Conv2D) (None, 4, 4, 128) 106496 conv5_block11_0_relu[0][0]

conv5_block11_1_bn (BatchNormal (None, 4, 4, 128) 512 conv5_block11_1_conv[0][0]

conv5_block11_1_relu (Activatio (None, 4, 4, 128) 0 conv5_block11_1_bn[0][0]

conv5_block11_2_conv (Conv2D) (None, 4, 4, 32) 36864 conv5_block11_1_relu[0][0]

conv5_block11_concat (Concatena (None, 4, 4, 864) 0 conv5_block10_concat[0][0] conv5_block11_2_conv[0][0]

conv5_block12_0_bn (BatchNormal (None, 4, 4, 864) 3456 conv5_block11_concat[0][0]

conv5_block12_0_relu (Activatio (None, 4, 4, 864) 0 conv5_block12_0_bn[0][0]

conv5_block12_1_conv (Conv2D) (None, 4, 4, 128) 110592 conv5_block12_0_relu[0][0]

conv5_block12_1_bn (BatchNormal (None, 4, 4, 128) 512 conv5_block12_1_conv[0][0]

conv5_block12_1_relu (Activatio (None, 4, 4, 128) 0 conv5_block12_1_bn[0][0]

conv5_block12_2_conv (Conv2D) (None, 4, 4, 32) 36864 conv5_block12_1_relu[0][0]

conv5_block12_concat (Concatena (None, 4, 4, 896) 0 conv5_block11_concat[0][0] conv5_block12_2_conv[0][0]

conv5_block13_0_bn (BatchNormal (None, 4, 4, 896) 3584 conv5_block12_concat[0][0]

conv5_block13_0_relu (Activatio (None, 4, 4, 896) 0 conv5_block13_0_bn[0][0]

conv5_block13_1_conv (Conv2D) (None, 4, 4, 128) 114688 conv5_block13_0_relu[0][0]

conv5_block13_1_bn (BatchNormal (None, 4, 4, 128) 512 conv5_block13_1_conv[0][0]

conv5_block13_1_relu (Activatio (None, 4, 4, 128) 0 conv5_block13_1_bn[0][0]

conv5_block13_2_conv (Conv2D) (None, 4, 4, 32) 36864 conv5_block13_1_relu[0][0]

conv5_block13_concat (Concatena (None, 4, 4, 928) 0 conv5_block12_concat[0][0] conv5_block13_2_conv[0][0]

conv5_block14_0_bn (BatchNormal (None, 4, 4, 928) 3712 conv5_block13_concat[0][0]

conv5_block14_0_relu (Activatio (None, 4, 4, 928) 0 conv5_block14_0_bn[0][0]

conv5_block14_1_conv (Conv2D) (None, 4, 4, 128) 118784 conv5_block14_0_relu[0][0]

conv5_block14_1_bn (BatchNormal (None, 4, 4, 128) 512 conv5_block14_1_conv[0][0]

conv5_block14_1_relu (Activatio (None, 4, 4, 128) 0 conv5_block14_1_bn[0][0]

conv5_block14_2_conv (Conv2D) (None, 4, 4, 32) 36864 conv5_block14_1_relu[0][0]

conv5_block14_concat (Concatena (None, 4, 4, 960) 0 conv5_block13_concat[0][0] conv5_block14_2_conv[0][0]

conv5_block15_0_bn (BatchNormal (None, 4, 4, 960) 3840 conv5_block14_concat[0][0]

conv5_block15_0_relu (Activatio (None, 4, 4, 960) 0 conv5_block15_0_bn[0][0]

conv5_block15_1_conv (Conv2D) (None, 4, 4, 128) 122880 conv5_block15_0_relu[0][0]

conv5_block15_1_bn (BatchNormal (None, 4, 4, 128) 512 conv5_block15_1_conv[0][0]

conv5_block15_1_relu (Activatio (None, 4, 4, 128) 0 conv5_block15_1_bn[0][0]

conv5_block15_2_conv (Conv2D) (None, 4, 4, 32) 36864 conv5_block15_1_relu[0][0]

conv5_block15_concat (Concatena (None, 4, 4, 992) 0 conv5_block14_concat[0][0] conv5_block15_2_conv[0][0]

conv5_block16_0_bn (BatchNormal (None, 4, 4, 992) 3968 conv5_block15_concat[0][0]

conv5_block16_0_relu (Activatio (None, 4, 4, 992) 0 conv5_block16_0_bn[0][0]

conv5_block16_1_conv (Conv2D) (None, 4, 4, 128) 126976 conv5_block16_0_relu[0][0]

conv5_block16_1_bn (BatchNormal (None, 4, 4, 128) 512 conv5_block16_1_conv[0][0]

conv5_block16_1_relu (Activatio (None, 4, 4, 128) 0 conv5_block16_1_bn[0][0]

conv5_block16_2_conv (Conv2D) (None, 4, 4, 32) 36864 conv5_block16_1_relu[0][0]

conv5_block16_concat (Concatena (None, 4, 4, 1024) 0 conv5_block15_concat[0][0] conv5_block16_2_conv[0][0]

bn (BatchNormalization) (None, 4, 4, 1024) 4096 conv5_block16_concat[0][0]

relu (Activation) (None, 4, 4, 1024) 0 bn[0][0]

global_average_pooling2d_1 (Glo (None, 1024) 0 relu[0][0]

dense_1 (Dense) (None, 1024) 1049600 global_average_pooling2d_1[0][0]

dense_2 (Dense) (None, 242) 248050 dense_1[0][0]

==========

Total params: 8,335,154
Trainable params: 8,251,506
Non-trainable params: 83,648

2. Proceso de entrenamiento en Google Colab:

Epoch 1/15 1000/1000 [==========] - 338s 338ms/step - loss: 3.1772 - acc: 0.3735 - mean_squared_error: 0.0287 - val_loss: 1.0979 - val_acc: 0.7475 - val_mean_squared_error: 0.0035 Epoch 00001: val_loss improved from inf to 1.09785, saving model to ./checkpoint09.h5

Epoch 2/15 1000/1000 [==========] - 311s 311ms/step - loss: 0.6898 - acc: 0.8547 - mean_squared_error: 0.0039 - val_loss: 0.3103 - val_acc: 0.9294 - val_mean_squared_error: 0.0034 Epoch 00002: val_loss improved from 1.09785 to 0.31031, saving model to ./checkpoint09.h5

Epoch 3/15 1000/1000 [==========] - 311s 311ms/step - loss: 0.2639 - acc: 0.9443 - mean_squared_error: 0.0039 - val_loss: 0.1727 - val_acc: 0.9597 - val_mean_squared_error: 0.0033 Epoch 00003: val_loss improved from 0.31031 to 0.17274, saving model to ./checkpoint09.h5

Epoch 4/15 1000/1000 [==========] - 312s 312ms/step - loss: 0.1237 - acc: 0.9762 - mean_squared_error: 0.0038 - val_loss: 0.1429 - val_acc: 0.9634 - val_mean_squared_error: 0.0032 Epoch 00004: val_loss improved from 0.17274 to 0.14288, saving model to ./checkpoint09.h5

Epoch 5/15 1000/1000 [==========] - 311s 311ms/step - loss: 0.1237 - acc: 0.9715 - mean_squared_error: 0.0037 - val_loss: 0.1502 - val_acc: 0.9602 - val_mean_squared_error: 0.0029 Epoch 00005: val_loss did not improve from 0.14288

Epoch 6/15 1000/1000 [==========] - 312s 312ms/step - loss: 0.0728 - acc: 0.9842 - mean_squared_error: 0.0035 - val_loss: 0.1322 - val_acc: 0.9661 - val_mean_squared_error: 0.0028

Epoch 00006: val_loss improved from 0.14288 to 0.13218, saving model to ./checkpoint09.h5 Epoch 7/15 1000/1000 [==========] - 312s 312ms/step - loss: 0.0870 - acc: 0.9792 - mean_squared_error: 0.0032 - val_loss: 0.0828 - val_acc: 0.9778 - val_mean_squared_error: 0.0022

Epoch 00007: val_loss improved from 0.13218 to 0.08282, saving model to ./checkpoint09.h5

Epoch 8/15 1000/1000 [==========] - 312s 312ms/step - loss: 0.0856 - acc: 0.9792 - mean_squared_error: 0.0031 - val_loss: 0.1768 - val_acc: 0.9519 - val_mean_squared_error: 0.0021
Epoch 00008: val_loss did not improve from 0.08282

Epoch 9/15 1000/1000 [==========] - 311s 311ms/step - loss: 0.0574 - acc: 0.9856 - mean_squared_error: 0.0026 - val_loss: 0.1708 - val_acc: 0.9548 - val_mean_squared_error: 0.0024 Epoch 00009: val_loss did not improve from 0.08282

Epoch 10/15 1000/1000 [==========] - 312s 312ms/step - loss: 0.0696 - acc: 0.9813 - mean_squared_error: 0.0027 - val_loss: 0.1324 - val_acc: 0.9642 - val_mean_squared_error: 0.0020

Epoch 00010: val_loss did not improve from 0.08282
Epoch 00010: early stopping
Guardando el modelo

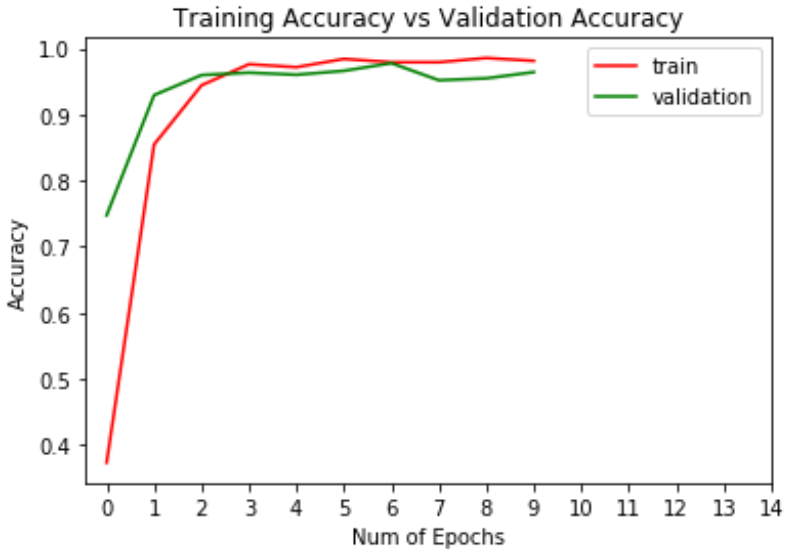
3. Evaluación del modelo:

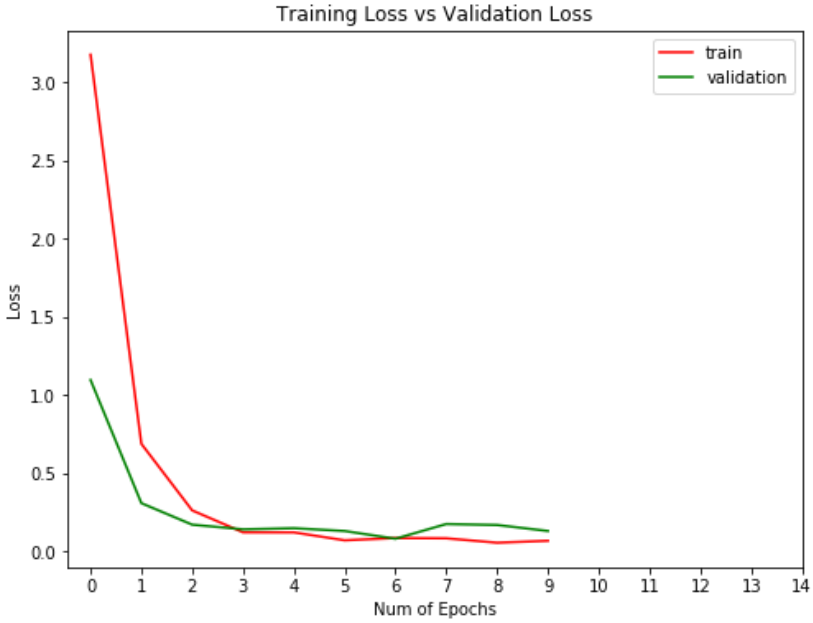
Loss, **Acc,** **Mean_scuared_error**

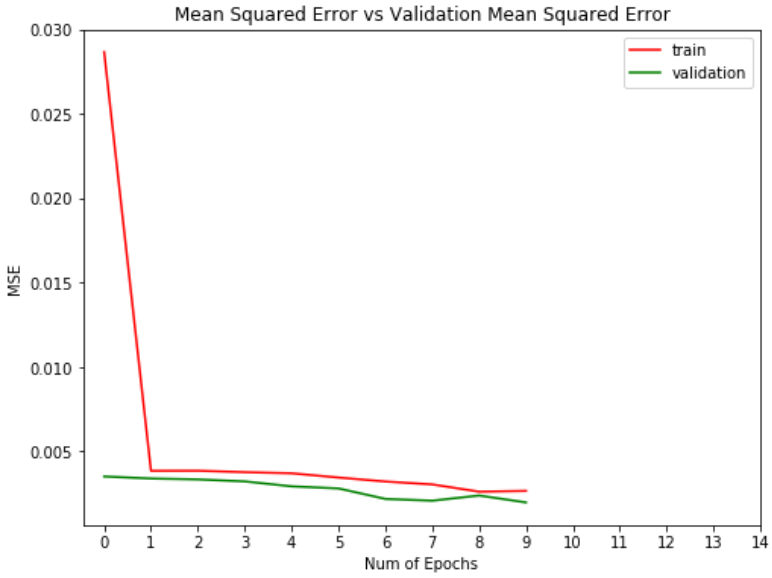
[0.1313690784498222, **0.9648023143683703**, 0.0019444343377356059]

Comentario: El modelo propuesto con una arquitectura moderna, ha logrado una proporción de 0.96 de precisión (Accurancy), lo cual es bajo respecto al obtenido en el último experimento de la etapa I.

4. Gráficas del modelo







Comentario: Las gráficas muestran la evolución del modelo en cuanto a la precisión (accurancy), la pedida (Loss) y El error cuadrático medio**.**

Anexos 7: Etapa final II (entrenamiento 07)

Datos relevantes del entrenamiento (exitoso):

Épocas=15	(Épocas del entrenamiento)
Longitud = 150	(Longitud de la imagen)
Altura =150	(Altura de la imagen)
Batch_size = 12	(Cantidad de imágenes a ser procesadas en un lote)
Pasos = 1000	(Pasos de aprendizaje en una época)
Nclases = 242	(Número de personas a identificar)
Lr=0.01	(Learning Rate)
Activation='sigmoid'	(Función sigmoide para la predicción)
SGD	(Optimizador: Stochastic Gradient Descent)

1. Modelo generado personalizado (DenseNet121UNAMBA) para los propósitos de la investigación:

Comentario: Este modelo a nivel de código fuente, es el mismo que el entrenamiento 06, variamos en la longitud y la altura de las imágenes de entrada (150 x 150) y la utilización del optimizador Stochastic Gradient Descent (SGD) en vez de Adam, por lo que la cantidad de neuronas se incrementa notablemente de 8,251,506 a 12,143,730 parámetros entrenables, básicamente por el tamaño de entrada de las imágenes.

Model: "DenseNet121UNAMBA"

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 150, 150, 3)	0	
zero_padding2d_1 (ZeroPadding2D	(None, 156, 156, 3)	0	input_1[0][0]
conv1/conv (Conv2D)	(None, 75, 75, 64)	9408	zero_padding2d_1[0][0]
conv1/bn (BatchNormalization)	(None, 75, 75, 64)	256	conv1/conv[0][0]
conv1/relu (Activation)	(None, 75, 75, 64)	0	conv1/bn[0][0]
zero_padding2d_2 (ZeroPadding2D	(None, 77, 77, 64)	0	conv1/relu[0][0]
pool1 (MaxPooling2D)	(None, 38, 38, 64)	0	zero_padding2d_2[0][0]
conv2_block1_0_bn (BatchNormali	(None, 38, 38, 64)	256	pool1[0][0]
conv2_block1_0_relu (Activation	(None, 38, 38, 64)	0	conv2_block1_0_bn[0][0]
conv2_block1_1_conv (Conv2D)	(None, 38, 38, 128)	8192	conv2_block1_0_relu[0][0]
conv2_block1_1_bn (BatchNormali	(None, 38, 38, 128)	512	conv2_block1_1_conv[0][0]
conv2_block1_1_relu (Activation	(None, 38, 38, 128)	0	conv2_block1_1_bn[0][0]
conv2_block1_2_conv (Conv2D)	(None, 38, 38, 32)	36864	conv2_block1_1_relu[0][0]
conv2_block1_concat (Concatenat	(None, 38, 38, 96)	0	pool1[0][0] conv2_block1_2_conv[0][0]
conv2_block2_0_bn (BatchNormali	(None, 38, 38, 96)	384	conv2_block1_concat[0][0]
conv2_block2_0_relu (Activation	(None, 38, 38, 96)	0	conv2_block2_0_bn[0][0]
conv2_block2_1_conv (Conv2D)	(None, 38, 38, 128)	12288	conv2_block2_0_relu[0][0]

conv2_block2_1_bn (BatchNormali (None, 38, 38, 128) 512 conv2_block2_1_conv[0][0]

conv2_block2_1_relu (Activation (None, 38, 38, 128) 0 conv2_block2_1_bn[0][0]

conv2_block2_2_conv (Conv2D) (None, 38, 38, 32) 36864 conv2_block2_1_relu[0][0]

conv2_block2_concat (Concatenat (None, 38, 38, 128) 0 conv2_block1_concat[0][0]
conv2_block2_2_conv[0][0]

conv2_block3_0_bn (BatchNormali (None, 38, 38, 128) 512 conv2_block2_concat[0][0]

conv2_block3_0_relu (Activation (None, 38, 38, 128) 0 conv2_block3_0_bn[0][0]

conv2_block3_1_conv (Conv2D) (None, 38, 38, 128) 16384 conv2_block3_0_relu[0][0]

conv2_block3_1_bn (BatchNormali (None, 38, 38, 128) 512 conv2_block3_1_conv[0][0]

conv2_block3_1_relu (Activation (None, 38, 38, 128) 0 conv2_block3_1_bn[0][0]

conv2_block3_2_conv (Conv2D) (None, 38, 38, 32) 36864 conv2_block3_1_relu[0][0]

conv2_block3_concat (Concatenat (None, 38, 38, 160) 0 conv2_block2_concat[0][0]
conv2_block3_2_conv[0][0]

conv2_block4_0_bn (BatchNormali (None, 38, 38, 160) 640 conv2_block3_concat[0][0]

conv2_block4_0_relu (Activation (None, 38, 38, 160) 0 conv2_block4_0_bn[0][0]

conv2_block4_1_conv (Conv2D) (None, 38, 38, 128) 20480 conv2_block4_0_relu[0][0]

conv2_block4_1_bn (BatchNormali (None, 38, 38, 128) 512 conv2_block4_1_conv[0][0]

conv2_block4_1_relu (Activation (None, 38, 38, 128) 0 conv2_block4_1_bn[0][0]

conv2_block4_2_conv (Conv2D) (None, 38, 38, 32) 36864 conv2_block4_1_relu[0][0]

conv2_block4_concat (Concatenat (None, 38, 38, 192) 0 conv2_block3_concat[0][0]
conv2_block4_2_conv[0][0]

conv2_block5_0_bn (BatchNormali (None, 38, 38, 192) 768 conv2_block4_concat[0][0]

conv2_block5_0_relu (Activation (None, 38, 38, 192) 0 conv2_block5_0_bn[0][0]

conv2_block5_1_conv (Conv2D) (None, 38, 38, 128) 24576 conv2_block5_0_relu[0][0]

conv2_block5_1_bn (BatchNormali (None, 38, 38, 128) 512 conv2_block5_1_conv[0][0]

conv2_block5_1_relu (Activation (None, 38, 38, 128) 0 conv2_block5_1_bn[0][0]

conv2_block5_2_conv (Conv2D) (None, 38, 38, 32) 36864 conv2_block5_1_relu[0][0]

conv2_block5_concat (Concatenat (None, 38, 38, 224) 0 conv2_block4_concat[0][0]
conv2_block5_2_conv[0][0]

conv2_block6_0_bn (BatchNormali (None, 38, 38, 224) 896 conv2_block5_concat[0][0]

conv2_block6_0_relu (Activation (None, 38, 38, 224) 0 conv2_block6_0_bn[0][0]

conv2_block6_1_conv (Conv2D) (None, 38, 38, 128) 28672 conv2_block6_0_relu[0][0]

conv2_block6_1_bn (BatchNormali (None, 38, 38, 128) 512 conv2_block6_1_conv[0][0]

conv2_block6_1_relu (Activation (None, 38, 38, 128) 0 conv2_block6_1_bn[0][0]

conv2_block6_2_conv (Conv2D) (None, 38, 38, 32) 36864 conv2_block6_1_relu[0][0]

conv2_block6_concat (Concatenat (None, 38, 38, 256) 0 conv2_block5_concat[0][0]
conv2_block6_2_conv[0][0]

pool2_bn (BatchNormalization) (None, 38, 38, 256) 1024 conv2_block6_concat[0][0]

pool2_relu (Activation) (None, 38, 38, 256) 0 pool2_bn[0][0]

pool2_conv (Conv2D) (None, 38, 38, 128) 32768 pool2_relu[0][0]

pool2_pool (AveragePooling2D) (None, 19, 19, 128) 0 pool2_conv[0][0]

conv3_block1_0_bn (BatchNormali (None, 19, 19, 128) 512 pool2_pool[0][0]

conv3_block1_0_relu (Activation (None, 19, 19, 128) 0 conv3_block1_0_bn[0][0]

conv3_block1_1_conv (Conv2D) (None, 19, 19, 128) 16384 conv3_block1_0_relu[0][0]

conv3_block1_1_bn (BatchNormali (None, 19, 19, 128) 512 conv3_block1_1_conv[0][0]

conv3_block1_1_relu (Activation (None, 19, 19, 128) 0 conv3_block1_1_bn[0][0]

conv3_block1_2_conv (Conv2D) (None, 19, 19, 32) 36864 conv3_block1_1_relu[0][0]

conv3_block1_concat (Concatenat (None, 19, 19, 160) 0 pool2_pool[0][0]
conv3_block1_2_conv[0][0]

conv3_block2_0_bn (BatchNormali (None, 19, 19, 160) 640 conv3_block1_concat[0][0]

conv3_block2_0_relu (Activation (None, 19, 19, 160) 0 conv3_block2_0_bn[0][0]

conv3_block2_1_conv (Conv2D) (None, 19, 19, 128) 20480 conv3_block2_0_relu[0][0]

conv3_block2_1_bn (BatchNormali (None, 19, 19, 128) 512 conv3_block2_1_conv[0][0]

conv3_block2_1_relu (Activation (None, 19, 19, 128) 0 conv3_block2_1_bn[0][0]

conv3_block2_2_conv (Conv2D) (None, 19, 19, 32) 36864 conv3_block2_1_relu[0][0]

conv3_block2_concat (Concatenat (None, 19, 19, 192) 0 conv3_block1_concat[0][0]
conv3_block2_2_conv[0][0]

conv3_block3_0_bn (BatchNormali (None, 19, 19, 192) 768 conv3_block2_concat[0][0]

conv3_block3_0_relu (Activation (None, 19, 19, 192) 0 conv3_block3_0_bn[0][0]

conv3_block3_1_conv (Conv2D) (None, 19, 19, 128) 24576 conv3_block3_0_relu[0][0]

conv3_block3_1_bn (BatchNormali (None, 19, 19, 128) 512 conv3_block3_1_conv[0][0]

conv3_block3_1_relu (Activation (None, 19, 19, 128) 0 conv3_block3_1_bn[0][0]

conv3_block3_2_conv (Conv2D) (None, 19, 19, 32) 36864 conv3_block3_1_relu[0][0]

conv3_block3_concat (Concatenat (None, 19, 19, 224) 0 conv3_block2_concat[0][0]
conv3_block3_2_conv[0][0]

conv3_block4_0_bn (BatchNormali (None, 19, 19, 224) 896 conv3_block3_concat[0][0]

conv3_block4_0_relu (Activation (None, 19, 19, 224) 0 conv3_block4_0_bn[0][0]

conv3_block4_1_conv (Conv2D) (None, 19, 19, 128) 28672 conv3_block4_0_relu[0][0]

conv3_block4_1_bn (BatchNormali (None, 19, 19, 128) 512 conv3_block4_1_conv[0][0]

conv3_block4_1_relu (Activation (None, 19, 19, 128) 0 conv3_block4_1_bn[0][0]

conv3_block4_2_conv (Conv2D) (None, 19, 19, 32) 36864 conv3_block4_1_relu[0][0]

conv3_block4_concat (Concatenat (None, 19, 19, 256) 0 conv3_block3_concat[0][0]
conv3_block4_2_conv[0][0]

conv3_block5_0_bn (BatchNormali (None, 19, 19, 256) 1024 conv3_block4_concat[0][0]

conv3_block5_0_relu (Activation (None, 19, 19, 256) 0 conv3_block5_0_bn[0][0]

conv3_block5_1_conv (Conv2D) (None, 19, 19, 128) 32768 conv3_block5_0_relu[0][0]

conv3_block5_1_bn (BatchNormali (None, 19, 19, 128) 512 conv3_block5_1_conv[0][0]

conv3_block5_1_relu (Activation (None, 19, 19, 128) 0 conv3_block5_1_bn[0][0]

conv3_block5_2_conv (Conv2D)	(None, 19, 19, 32)	36864	conv3_block5_1_relu[0][0]
conv3_block5_concat (Concatenat	(None, 19, 19, 288)	0	conv3_block4_concat[0][0] conv3_block5_2_conv[0][0]
conv3_block6_0_bn (BatchNormali	(None, 19, 19, 288)	1152	conv3_block5_concat[0][0]
conv3_block6_0_relu (Activation	(None, 19, 19, 288)	0	conv3_block6_0_bn[0][0]
conv3_block6_1_conv (Conv2D)	(None, 19, 19, 128)	36864	conv3_block6_0_relu[0][0]
conv3_block6_1_bn (BatchNormali	(None, 19, 19, 128)	512	conv3_block6_1_conv[0][0]
conv3_block6_1_relu (Activation	(None, 19, 19, 128)	0	conv3_block6_1_bn[0][0]
conv3_block6_2_conv (Conv2D)	(None, 19, 19, 32)	36864	conv3_block6_1_relu[0][0]
conv3_block6_concat (Concatenat	(None, 19, 19, 320)	0	conv3_block5_concat[0][0] conv3_block6_2_conv[0][0]
conv3_block7_0_bn (BatchNormali	(None, 19, 19, 320)	1280	conv3_block6_concat[0][0]
conv3_block7_0_relu (Activation	(None, 19, 19, 320)	0	conv3_block7_0_bn[0][0]
conv3_block7_1_conv (Conv2D)	(None, 19, 19, 128)	40960	conv3_block7_0_relu[0][0]
conv3_block7_1_bn (BatchNormali	(None, 19, 19, 128)	512	conv3_block7_1_conv[0][0]
conv3_block7_1_relu (Activation	(None, 19, 19, 128)	0	conv3_block7_1_bn[0][0]
conv3_block7_2_conv (Conv2D)	(None, 19, 19, 32)	36864	conv3_block7_1_relu[0][0]
conv3_block7_concat (Concatenat	(None, 19, 19, 352)	0	conv3_block6_concat[0][0] conv3_block7_2_conv[0][0]
conv3_block8_0_bn (BatchNormali	(None, 19, 19, 352)	1408	conv3_block7_concat[0][0]
conv3_block8_0_relu (Activation	(None, 19, 19, 352)	0	conv3_block8_0_bn[0][0]
conv3_block8_1_conv (Conv2D)	(None, 19, 19, 128)	45056	conv3_block8_0_relu[0][0]
conv3_block8_1_bn (BatchNormali	(None, 19, 19, 128)	512	conv3_block8_1_conv[0][0]
conv3_block8_1_relu (Activation	(None, 19, 19, 128)	0	conv3_block8_1_bn[0][0]
conv3_block8_2_conv (Conv2D)	(None, 19, 19, 32)	36864	conv3_block8_1_relu[0][0]
conv3_block8_concat (Concatenat	(None, 19, 19, 384)	0	conv3_block7_concat[0][0] conv3_block8_2_conv[0][0]
conv3_block9_0_bn (BatchNormali	(None, 19, 19, 384)	1536	conv3_block8_concat[0][0]
conv3_block9_0_relu (Activation	(None, 19, 19, 384)	0	conv3_block9_0_bn[0][0]
conv3_block9_1_conv (Conv2D)	(None, 19, 19, 128)	49152	conv3_block9_0_relu[0][0]
conv3_block9_1_bn (BatchNormali	(None, 19, 19, 128)	512	conv3_block9_1_conv[0][0]
conv3_block9_1_relu (Activation	(None, 19, 19, 128)	0	conv3_block9_1_bn[0][0]
conv3_block9_2_conv (Conv2D)	(None, 19, 19, 32)	36864	conv3_block9_1_relu[0][0]
conv3_block9_concat (Concatenat	(None, 19, 19, 416)	0	conv3_block8_concat[0][0] conv3_block9_2_conv[0][0]
conv3_block10_0_bn (BatchNormal	(None, 19, 19, 416)	1664	conv3_block9_concat[0][0]
conv3_block10_0_relu (Activatio	(None, 19, 19, 416)	0	conv3_block10_0_bn[0][0]
conv3_block10_1_conv (Conv2D)	(None, 19, 19, 128)	53248	conv3_block10_0_relu[0][0]
conv3_block10_1_bn (BatchNormal	(None, 19, 19, 128)	512	conv3_block10_1_conv[0][0]

conv3_block10_1_relu (Activatio	(None, 19, 19, 128)	0	conv3_block10_1_bn[0][0]
conv3_block10_2_conv (Conv2D)	(None, 19, 19, 32)	36864	conv3_block10_1_relu[0][0]
conv3_block10_concat (Concatena	(None, 19, 19, 448)	0	conv3_block9_concat[0][0] conv3_block10_2_conv[0][0]
conv3_block11_0_bn (BatchNormal	(None, 19, 19, 448)	1792	conv3_block10_concat[0][0]
conv3_block11_0_relu (Activatio	(None, 19, 19, 448)	0	conv3_block11_0_bn[0][0]
conv3_block11_1_conv (Conv2D)	(None, 19, 19, 128)	57344	conv3_block11_0_relu[0][0]
conv3_block11_1_bn (BatchNormal	(None, 19, 19, 128)	512	conv3_block11_1_conv[0][0]
conv3_block11_1_relu (Activatio	(None, 19, 19, 128)	0	conv3_block11_1_bn[0][0]
conv3_block11_2_conv (Conv2D)	(None, 19, 19, 32)	36864	conv3_block11_1_relu[0][0]
conv3_block11_concat (Concatena	(None, 19, 19, 480)	0	conv3_block10_concat[0][0] conv3_block11_2_conv[0][0]
conv3_block12_0_bn (BatchNormal	(None, 19, 19, 480)	1920	conv3_block11_concat[0][0]
conv3_block12_0_relu (Activatio	(None, 19, 19, 480)	0	conv3_block12_0_bn[0][0]
conv3_block12_1_conv (Conv2D)	(None, 19, 19, 128)	61440	conv3_block12_0_relu[0][0]
conv3_block12_1_bn (BatchNormal	(None, 19, 19, 128)	512	conv3_block12_1_conv[0][0]
conv3_block12_1_relu (Activatio	(None, 19, 19, 128)	0	conv3_block12_1_bn[0][0]
conv3_block12_2_conv (Conv2D)	(None, 19, 19, 32)	36864	conv3_block12_1_relu[0][0]
conv3_block12_concat (Concatena	(None, 19, 19, 512)	0	conv3_block11_concat[0][0] conv3_block12_2_conv[0][0]
pool3_bn (BatchNormalization)	(None, 19, 19, 512)	2048	conv3_block12_concat[0][0]
pool3_relu (Activation)	(None, 19, 19, 512)	0	pool3_bn[0][0]
pool3_conv (Conv2D)	(None, 19, 19, 256)	131072	pool3_relu[0][0]
pool3_pool (AveragePooling2D)	(None, 9, 9, 256)	0	pool3_conv[0][0]
conv4_block1_0_bn (BatchNormali	(None, 9, 9, 256)	1024	pool3_pool[0][0]
conv4_block1_0_relu (Activation	(None, 9, 9, 256)	0	conv4_block1_0_bn[0][0]
conv4_block1_1_conv (Conv2D)	(None, 9, 9, 128)	32768	conv4_block1_0_relu[0][0]
conv4_block1_1_bn (BatchNormali	(None, 9, 9, 128)	512	conv4_block1_1_conv[0][0]
conv4_block1_1_relu (Activation	(None, 9, 9, 128)	0	conv4_block1_1_bn[0][0]
conv4_block1_2_conv (Conv2D)	(None, 9, 9, 32)	36864	conv4_block1_1_relu[0][0]
conv4_block1_concat (Concatenat	(None, 9, 9, 288)	0	pool3_pool[0][0] conv4_block1_2_conv[0][0]
conv4_block2_0_bn (BatchNormali	(None, 9, 9, 288)	1152	conv4_block1_concat[0][0]
conv4_block2_0_relu (Activation	(None, 9, 9, 288)	0	conv4_block2_0_bn[0][0]
conv4_block2_1_conv (Conv2D)	(None, 9, 9, 128)	36864	conv4_block2_0_relu[0][0]
conv4_block2_1_bn (BatchNormali	(None, 9, 9, 128)	512	conv4_block2_1_conv[0][0]
conv4_block2_1_relu (Activation	(None, 9, 9, 128)	0	conv4_block2_1_bn[0][0]
conv4_block2_2_conv (Conv2D)	(None, 9, 9, 32)	36864	conv4_block2_1_relu[0][0]

conv4_block2_concat (Concatenat (None, 9, 9, 320) 0 conv4_block1_concat[0][0]
conv4_block2_2_conv[0][0]

conv4_block3_0_bn (BatchNormali (None, 9, 9, 320) 1280 conv4_block2_concat[0][0]

conv4_block3_0_relu (Activation (None, 9, 9, 320) 0 conv4_block3_0_bn[0][0]

conv4_block3_1_conv (Conv2D) (None, 9, 9, 128) 40960 conv4_block3_0_relu[0][0]

conv4_block3_1_bn (BatchNormali (None, 9, 9, 128) 512 conv4_block3_1_conv[0][0]

conv4_block3_1_relu (Activation (None, 9, 9, 128) 0 conv4_block3_1_bn[0][0]

conv4_block3_2_conv (Conv2D) (None, 9, 9, 32) 36864 conv4_block3_1_relu[0][0]

conv4_block3_concat (Concatenat (None, 9, 9, 352) 0 conv4_block2_concat[0][0]
conv4_block3_2_conv[0][0]

conv4_block4_0_bn (BatchNormali (None, 9, 9, 352) 1408 conv4_block3_concat[0][0]

conv4_block4_0_relu (Activation (None, 9, 9, 352) 0 conv4_block4_0_bn[0][0]

conv4_block4_1_conv (Conv2D) (None, 9, 9, 128) 45056 conv4_block4_0_relu[0][0]

conv4_block4_1_bn (BatchNormali (None, 9, 9, 128) 512 conv4_block4_1_conv[0][0]

conv4_block4_1_relu (Activation (None, 9, 9, 128) 0 conv4_block4_1_bn[0][0]

conv4_block4_2_conv (Conv2D) (None, 9, 9, 32) 36864 conv4_block4_1_relu[0][0]

conv4_block4_concat (Concatenat (None, 9, 9, 384) 0 conv4_block3_concat[0][0]
conv4_block4_2_conv[0][0]

conv4_block5_0_bn (BatchNormali (None, 9, 9, 384) 1536 conv4_block4_concat[0][0]

conv4_block5_0_relu (Activation (None, 9, 9, 384) 0 conv4_block5_0_bn[0][0]

conv4_block5_1_conv (Conv2D) (None, 9, 9, 128) 49152 conv4_block5_0_relu[0][0]

conv4_block5_1_bn (BatchNormali (None, 9, 9, 128) 512 conv4_block5_1_conv[0][0]

conv4_block5_1_relu (Activation (None, 9, 9, 128) 0 conv4_block5_1_bn[0][0]

conv4_block5_2_conv (Conv2D) (None, 9, 9, 32) 36864 conv4_block5_1_relu[0][0]

conv4_block5_concat (Concatenat (None, 9, 9, 416) 0 conv4_block4_concat[0][0]
conv4_block5_2_conv[0][0]

conv4_block6_0_bn (BatchNormali (None, 9, 9, 416) 1664 conv4_block5_concat[0][0]

conv4_block6_0_relu (Activation (None, 9, 9, 416) 0 conv4_block6_0_bn[0][0]

conv4_block6_1_conv (Conv2D) (None, 9, 9, 128) 53248 conv4_block6_0_relu[0][0]

conv4_block6_1_bn (BatchNormali (None, 9, 9, 128) 512 conv4_block6_1_conv[0][0]

conv4_block6_1_relu (Activation (None, 9, 9, 128) 0 conv4_block6_1_bn[0][0]

conv4_block6_2_conv (Conv2D) (None, 9, 9, 32) 36864 conv4_block6_1_relu[0][0]

conv4_block6_concat (Concatenat (None, 9, 9, 448) 0 conv4_block5_concat[0][0]
conv4_block6_2_conv[0][0]

conv4_block7_0_bn (BatchNormali (None, 9, 9, 448) 1792 conv4_block6_concat[0][0]

conv4_block7_0_relu (Activation (None, 9, 9, 448) 0 conv4_block7_0_bn[0][0]

conv4_block7_1_conv (Conv2D) (None, 9, 9, 128) 57344 conv4_block7_0_relu[0][0]

conv4_block7_1_bn (BatchNormali (None, 9, 9, 128) 512 conv4_block7_1_conv[0][0]

conv4_block7_1_relu (Activation (None, 9, 9, 128) 0 conv4_block7_1_bn[0][0]

conv4_block7_2_conv (Conv2D)	(None, 9, 9, 32)	36864	conv4_block7_1_relu[0][0]
conv4_block7_concat (Concatenat	(None, 9, 9, 480)	0	conv4_block6_concat[0][0] conv4_block7_2_conv[0][0]
conv4_block8_0_bn (BatchNormali	(None, 9, 9, 480)	1920	conv4_block7_concat[0][0]
conv4_block8_0_relu (Activation	(None, 9, 9, 480)	0	conv4_block8_0_bn[0][0]
conv4_block8_1_conv (Conv2D)	(None, 9, 9, 128)	61440	conv4_block8_0_relu[0][0]
conv4_block8_1_bn (BatchNormali	(None, 9, 9, 128)	512	conv4_block8_1_conv[0][0]
conv4_block8_1_relu (Activation	(None, 9, 9, 128)	0	conv4_block8_1_bn[0][0]
conv4_block8_2_conv (Conv2D)	(None, 9, 9, 32)	36864	conv4_block8_1_relu[0][0]
conv4_block8_concat (Concatenat	(None, 9, 9, 512)	0	conv4_block7_concat[0][0] conv4_block8_2_conv[0][0]
conv4_block9_0_bn (BatchNormali	(None, 9, 9, 512)	2048	conv4_block8_concat[0][0]
conv4_block9_0_relu (Activation	(None, 9, 9, 512)	0	conv4_block9_0_bn[0][0]
conv4_block9_1_conv (Conv2D)	(None, 9, 9, 128)	65536	conv4_block9_0_relu[0][0]
conv4_block9_1_bn (BatchNormali	(None, 9, 9, 128)	512	conv4_block9_1_conv[0][0]
conv4_block9_1_relu (Activation	(None, 9, 9, 128)	0	conv4_block9_1_bn[0][0]
conv4_block9_2_conv (Conv2D)	(None, 9, 9, 32)	36864	conv4_block9_1_relu[0][0]
conv4_block9_concat (Concatenat	(None, 9, 9, 544)	0	conv4_block8_concat[0][0] conv4_block9_2_conv[0][0]
conv4_block10_0_bn (BatchNormal	(None, 9, 9, 544)	2176	conv4_block9_concat[0][0]
conv4_block10_0_relu (Activatio	(None, 9, 9, 544)	0	conv4_block10_0_bn[0][0]
conv4_block10_1_conv (Conv2D)	(None, 9, 9, 128)	69632	conv4_block10_0_relu[0][0]
conv4_block10_1_bn (BatchNormal	(None, 9, 9, 128)	512	conv4_block10_1_conv[0][0]
conv4_block10_1_relu (Activatio	(None, 9, 9, 128)	0	conv4_block10_1_bn[0][0]
conv4_block10_2_conv (Conv2D)	(None, 9, 9, 32)	36864	conv4_block10_1_relu[0][0]
conv4_block10_concat (Concatena	(None, 9, 9, 576)	0	conv4_block9_concat[0][0] conv4_block10_2_conv[0][0]
conv4_block11_0_bn (BatchNormal	(None, 9, 9, 576)	2304	conv4_block10_concat[0][0]
conv4_block11_0_relu (Activatio	(None, 9, 9, 576)	0	conv4_block11_0_bn[0][0]
conv4_block11_1_conv (Conv2D)	(None, 9, 9, 128)	73728	conv4_block11_0_relu[0][0]
conv4_block11_1_bn (BatchNormal	(None, 9, 9, 128)	512	conv4_block11_1_conv[0][0]
conv4_block11_1_relu (Activatio	(None, 9, 9, 128)	0	conv4_block11_1_bn[0][0]
conv4_block11_2_conv (Conv2D)	(None, 9, 9, 32)	36864	conv4_block11_1_relu[0][0]
conv4_block11_concat (Concatena	(None, 9, 9, 608)	0	conv4_block10_concat[0][0] conv4_block11_2_conv[0][0]
conv4_block12_0_bn (BatchNormal	(None, 9, 9, 608)	2432	conv4_block11_concat[0][0]
conv4_block12_0_relu (Activatio	(None, 9, 9, 608)	0	conv4_block12_0_bn[0][0]
conv4_block12_1_conv (Conv2D)	(None, 9, 9, 128)	77824	conv4_block12_0_relu[0][0]
conv4_block12_1_bn (BatchNormal	(None, 9, 9, 128)	512	conv4_block12_1_conv[0][0]

conv4_block12_1_relu (Activatio (None, 9, 9, 128) 0 conv4_block12_1_bn[0][0]

conv4_block12_2_conv (Conv2D) (None, 9, 9, 32) 36864 conv4_block12_1_relu[0][0]

conv4_block12_concat (Concatena (None, 9, 9, 640) 0 conv4_block11_concat[0][0]
conv4_block12_2_conv[0][0]

conv4_block13_0_bn (BatchNormal (None, 9, 9, 640) 2560 conv4_block12_concat[0][0]

conv4_block13_0_relu (Activatio (None, 9, 9, 640) 0 conv4_block13_0_bn[0][0]

conv4_block13_1_conv (Conv2D) (None, 9, 9, 128) 81920 conv4_block13_0_relu[0][0]

conv4_block13_1_bn (BatchNormal (None, 9, 9, 128) 512 conv4_block13_1_conv[0][0]

conv4_block13_1_relu (Activatio (None, 9, 9, 128) 0 conv4_block13_1_bn[0][0]

conv4_block13_2_conv (Conv2D) (None, 9, 9, 32) 36864 conv4_block13_1_relu[0][0]

conv4_block13_concat (Concatena (None, 9, 9, 672) 0 conv4_block12_concat[0][0]
conv4_block13_2_conv[0][0]

conv4_block14_0_bn (BatchNormal (None, 9, 9, 672) 2688 conv4_block13_concat[0][0]

conv4_block14_0_relu (Activatio (None, 9, 9, 672) 0 conv4_block14_0_bn[0][0]

conv4_block14_1_conv (Conv2D) (None, 9, 9, 128) 86016 conv4_block14_0_relu[0][0]

conv4_block14_1_bn (BatchNormal (None, 9, 9, 128) 512 conv4_block14_1_conv[0][0]

conv4_block14_1_relu (Activatio (None, 9, 9, 128) 0 conv4_block14_1_bn[0][0]

conv4_block14_2_conv (Conv2D) (None, 9, 9, 32) 36864 conv4_block14_1_relu[0][0]

conv4_block14_concat (Concatena (None, 9, 9, 704) 0 conv4_block13_concat[0][0]
conv4_block14_2_conv[0][0]

conv4_block15_0_bn (BatchNormal (None, 9, 9, 704) 2816 conv4_block14_concat[0][0]

conv4_block15_0_relu (Activatio (None, 9, 9, 704) 0 conv4_block15_0_bn[0][0]

conv4_block15_1_conv (Conv2D) (None, 9, 9, 128) 90112 conv4_block15_0_relu[0][0]

conv4_block15_1_bn (BatchNormal (None, 9, 9, 128) 512 conv4_block15_1_conv[0][0]

conv4_block15_1_relu (Activatio (None, 9, 9, 128) 0 conv4_block15_1_bn[0][0]

conv4_block15_2_conv (Conv2D) (None, 9, 9, 32) 36864 conv4_block15_1_relu[0][0]

conv4_block15_concat (Concatena (None, 9, 9, 736) 0 conv4_block14_concat[0][0]
conv4_block15_2_conv[0][0]

conv4_block16_0_bn (BatchNormal (None, 9, 9, 736) 2944 conv4_block15_concat[0][0]

conv4_block16_0_relu (Activatio (None, 9, 9, 736) 0 conv4_block16_0_bn[0][0]

conv4_block16_1_conv (Conv2D) (None, 9, 9, 128) 94208 conv4_block16_0_relu[0][0]

conv4_block16_1_bn (BatchNormal (None, 9, 9, 128) 512 conv4_block16_1_conv[0][0]

conv4_block16_1_relu (Activatio (None, 9, 9, 128) 0 conv4_block16_1_bn[0][0]

conv4_block16_2_conv (Conv2D) (None, 9, 9, 32) 36864 conv4_block16_1_relu[0][0]

conv4_block16_concat (Concatena (None, 9, 9, 768) 0 conv4_block15_concat[0][0]
conv4_block16_2_conv[0][0]

conv4_block17_0_bn (BatchNormal (None, 9, 9, 768) 3072 conv4_block16_concat[0][0]

conv4_block17_0_relu (Activatio (None, 9, 9, 768) 0 conv4_block17_0_bn[0][0]

conv4_block17_1_conv (Conv2D) (None, 9, 9, 128) 98304 conv4_block17_0_relu[0][0]

conv4_block17_1_bn (BatchNormal (None, 9, 9, 128) 512 conv4_block17_1_conv[0][0]

conv4_block17_1_relu (Activatio (None, 9, 9, 128) 0 conv4_block17_1_bn[0][0]

conv4_block17_2_conv (Conv2D) (None, 9, 9, 32) 36864 conv4_block17_1_relu[0][0]

conv4_block17_concat (Concatena (None, 9, 9, 800) 0 conv4_block16_concat[0][0]
conv4_block17_2_conv[0][0]

conv4_block18_0_bn (BatchNormal (None, 9, 9, 800) 3200 conv4_block17_concat[0][0]

conv4_block18_0_relu (Activatio (None, 9, 9, 800) 0 conv4_block18_0_bn[0][0]

conv4_block18_1_conv (Conv2D) (None, 9, 9, 128) 102400 conv4_block18_0_relu[0][0]

conv4_block18_1_bn (BatchNormal (None, 9, 9, 128) 512 conv4_block18_1_conv[0][0]

conv4_block18_1_relu (Activatio (None, 9, 9, 128) 0 conv4_block18_1_bn[0][0]

conv4_block18_2_conv (Conv2D) (None, 9, 9, 32) 36864 conv4_block18_1_relu[0][0]

conv4_block18_concat (Concatena (None, 9, 9, 832) 0 conv4_block17_concat[0][0]
conv4_block18_2_conv[0][0]

conv4_block19_0_bn (BatchNormal (None, 9, 9, 832) 3328 conv4_block18_concat[0][0]

conv4_block19_0_relu (Activatio (None, 9, 9, 832) 0 conv4_block19_0_bn[0][0]

conv4_block19_1_conv (Conv2D) (None, 9, 9, 128) 106496 conv4_block19_0_relu[0][0]

conv4_block19_1_bn (BatchNormal (None, 9, 9, 128) 512 conv4_block19_1_conv[0][0]

conv4_block19_1_relu (Activatio (None, 9, 9, 128) 0 conv4_block19_1_bn[0][0]

conv4_block19_2_conv (Conv2D) (None, 9, 9, 32) 36864 conv4_block19_1_relu[0][0]

conv4_block19_concat (Concatena (None, 9, 9, 864) 0 conv4_block18_concat[0][0]
conv4_block19_2_conv[0][0]

conv4_block20_0_bn (BatchNormal (None, 9, 9, 864) 3456 conv4_block19_concat[0][0]

conv4_block20_0_relu (Activatio (None, 9, 9, 864) 0 conv4_block20_0_bn[0][0]

conv4_block20_1_conv (Conv2D) (None, 9, 9, 128) 110592 conv4_block20_0_relu[0][0]

conv4_block20_1_bn (BatchNormal (None, 9, 9, 128) 512 conv4_block20_1_conv[0][0]

conv4_block20_1_relu (Activatio (None, 9, 9, 128) 0 conv4_block20_1_bn[0][0]

conv4_block20_2_conv (Conv2D) (None, 9, 9, 32) 36864 conv4_block20_1_relu[0][0]

conv4_block20_concat (Concatena (None, 9, 9, 896) 0 conv4_block19_concat[0][0]
conv4_block20_2_conv[0][0]

conv4_block21_0_bn (BatchNormal (None, 9, 9, 896) 3584 conv4_block20_concat[0][0]

conv4_block21_0_relu (Activatio (None, 9, 9, 896) 0 conv4_block21_0_bn[0][0]

conv4_block21_1_conv (Conv2D) (None, 9, 9, 128) 114688 conv4_block21_0_relu[0][0]

conv4_block21_1_bn (BatchNormal (None, 9, 9, 128) 512 conv4_block21_1_conv[0][0]

conv4_block21_1_relu (Activatio (None, 9, 9, 128) 0 conv4_block21_1_bn[0][0]

conv4_block21_2_conv (Conv2D) (None, 9, 9, 32) 36864 conv4_block21_1_relu[0][0]

conv4_block21_concat (Concatena (None, 9, 9, 928) 0 conv4_block20_concat[0][0]
conv4_block21_2_conv[0][0]

conv4_block22_0_bn (BatchNormal (None, 9, 9, 928) 3712 conv4_block21_concat[0][0]

conv4_block22_0_relu (Activatio (None, 9, 9, 928) 0 conv4_block22_0_bn[0][0]

conv4_block22_1_conv (Conv2D) (None, 9, 9, 128) 118784 conv4_block22_0_relu[0][0]

conv4_block22_1_bn (BatchNormal (None, 9, 9, 128) 512 conv4_block22_1_conv[0][0]

conv4_block22_1_relu (Activatio (None, 9, 9, 128) 0 conv4_block22_1_bn[0][0]

conv4_block22_2_conv (Conv2D) (None, 9, 9, 32) 36864 conv4_block22_1_relu[0][0]

conv4_block22_concat (Concatena (None, 9, 9, 960) 0 conv4_block21_concat[0][0]
conv4_block22_2_conv[0][0]

conv4_block23_0_bn (BatchNormal (None, 9, 9, 960) 3840 conv4_block22_concat[0][0]

conv4_block23_0_relu (Activatio (None, 9, 9, 960) 0 conv4_block23_0_bn[0][0]

conv4_block23_1_conv (Conv2D) (None, 9, 9, 128) 122880 conv4_block23_0_relu[0][0]

conv4_block23_1_bn (BatchNormal (None, 9, 9, 128) 512 conv4_block23_1_conv[0][0]

conv4_block23_1_relu (Activatio (None, 9, 9, 128) 0 conv4_block23_1_bn[0][0]

conv4_block23_2_conv (Conv2D) (None, 9, 9, 32) 36864 conv4_block23_1_relu[0][0]

conv4_block23_concat (Concatena (None, 9, 9, 992) 0 conv4_block22_concat[0][0]
conv4_block23_2_conv[0][0]

conv4_block24_0_bn (BatchNormal (None, 9, 9, 992) 3968 conv4_block23_concat[0][0]

conv4_block24_0_relu (Activatio (None, 9, 9, 992) 0 conv4_block24_0_bn[0][0]

conv4_block24_1_conv (Conv2D) (None, 9, 9, 128) 126976 conv4_block24_0_relu[0][0]

conv4_block24_1_bn (BatchNormal (None, 9, 9, 128) 512 conv4_block24_1_conv[0][0]

conv4_block24_1_relu (Activatio (None, 9, 9, 128) 0 conv4_block24_1_bn[0][0]

conv4_block24_2_conv (Conv2D) (None, 9, 9, 32) 36864 conv4_block24_1_relu[0][0]

conv4_block24_concat (Concatena (None, 9, 9, 1024) 0 conv4_block23_concat[0][0]
conv4_block24_2_conv[0][0]

pool4_bn (BatchNormalization) (None, 9, 9, 1024) 4096 conv4_block24_concat[0][0]

pool4_relu (Activation) (None, 9, 9, 1024) 0 pool4_bn[0][0]

pool4_conv (Conv2D) (None, 9, 9, 512) 524288 pool4_relu[0][0]

pool4_pool (AveragePooling2D) (None, 4, 4, 512) 0 pool4_conv[0][0]

conv5_block1_0_bn (BatchNormali (None, 4, 4, 512) 2048 pool4_pool[0][0]

conv5_block1_0_relu (Activation (None, 4, 4, 512) 0 conv5_block1_0_bn[0][0]

conv5_block1_1_conv (Conv2D) (None, 4, 4, 128) 65536 conv5_block1_0_relu[0][0]

conv5_block1_1_bn (BatchNormali (None, 4, 4, 128) 512 conv5_block1_1_conv[0][0]

conv5_block1_1_relu (Activation (None, 4, 4, 128) 0 conv5_block1_1_bn[0][0]

conv5_block1_2_conv (Conv2D) (None, 4, 4, 32) 36864 conv5_block1_1_relu[0][0]

conv5_block1_concat (Concatenat (None, 4, 4, 544) 0 pool4_pool[0][0]
conv5_block1_2_conv[0][0]

conv5_block2_0_bn (BatchNormali (None, 4, 4, 544) 2176 conv5_block1_concat[0][0]

conv5_block2_0_relu (Activation (None, 4, 4, 544) 0 conv5_block2_0_bn[0][0]

conv5_block2_1_conv (Conv2D) (None, 4, 4, 128) 69632 conv5_block2_0_relu[0][0]

conv5_block2_1_bn (BatchNormali (None, 4, 4, 128) 512 conv5_block2_1_conv[0][0]

conv5_block2_1_relu (Activation (None, 4, 4, 128) 0 conv5_block2_1_bn[0][0]

conv5_block2_2_conv (Conv2D)	(None, 4, 4, 32)	36864	conv5_block2_1_relu[0][0]
conv5_block2_concat (Concatenat	(None, 4, 4, 576)	0	conv5_block1_concat[0][0] conv5_block2_2_conv[0][0]
conv5_block3_0_bn (BatchNormali	(None, 4, 4, 576)	2304	conv5_block2_concat[0][0]
conv5_block3_0_relu (Activation	(None, 4, 4, 576)	0	conv5_block3_0_bn[0][0]
conv5_block3_1_conv (Conv2D)	(None, 4, 4, 128)	73728	conv5_block3_0_relu[0][0]
conv5_block3_1_bn (BatchNormali	(None, 4, 4, 128)	512	conv5_block3_1_conv[0][0]
conv5_block3_1_relu (Activation	(None, 4, 4, 128)	0	conv5_block3_1_bn[0][0]
conv5_block3_2_conv (Conv2D)	(None, 4, 4, 32)	36864	conv5_block3_1_relu[0][0]
conv5_block3_concat (Concatenat	(None, 4, 4, 608)	0	conv5_block2_concat[0][0] conv5_block3_2_conv[0][0]
conv5_block4_0_bn (BatchNormali	(None, 4, 4, 608)	2432	conv5_block3_concat[0][0]
conv5_block4_0_relu (Activation	(None, 4, 4, 608)	0	conv5_block4_0_bn[0][0]
conv5_block4_1_conv (Conv2D)	(None, 4, 4, 128)	77824	conv5_block4_0_relu[0][0]
conv5_block4_1_bn (BatchNormali	(None, 4, 4, 128)	512	conv5_block4_1_conv[0][0]
conv5_block4_1_relu (Activation	(None, 4, 4, 128)	0	conv5_block4_1_bn[0][0]
conv5_block4_2_conv (Conv2D)	(None, 4, 4, 32)	36864	conv5_block4_1_relu[0][0]
conv5_block4_concat (Concatenat	(None, 4, 4, 640)	0	conv5_block3_concat[0][0] conv5_block4_2_conv[0][0]
conv5_block5_0_bn (BatchNormali	(None, 4, 4, 640)	2560	conv5_block4_concat[0][0]
conv5_block5_0_relu (Activation	(None, 4, 4, 640)	0	conv5_block5_0_bn[0][0]
conv5_block5_1_conv (Conv2D)	(None, 4, 4, 128)	81920	conv5_block5_0_relu[0][0]
conv5_block5_1_bn (BatchNormali	(None, 4, 4, 128)	512	conv5_block5_1_conv[0][0]
conv5_block5_1_relu (Activation	(None, 4, 4, 128)	0	conv5_block5_1_bn[0][0]
conv5_block5_2_conv (Conv2D)	(None, 4, 4, 32)	36864	conv5_block5_1_relu[0][0]
conv5_block5_concat (Concatenat	(None, 4, 4, 672)	0	conv5_block4_concat[0][0] conv5_block5_2_conv[0][0]
conv5_block6_0_bn (BatchNormali	(None, 4, 4, 672)	2688	conv5_block5_concat[0][0]
conv5_block6_0_relu (Activation	(None, 4, 4, 672)	0	conv5_block6_0_bn[0][0]
conv5_block6_1_conv (Conv2D)	(None, 4, 4, 128)	86016	conv5_block6_0_relu[0][0]
conv5_block6_1_bn (BatchNormali	(None, 4, 4, 128)	512	conv5_block6_1_conv[0][0]
conv5_block6_1_relu (Activation	(None, 4, 4, 128)	0	conv5_block6_1_bn[0][0]
conv5_block6_2_conv (Conv2D)	(None, 4, 4, 32)	36864	conv5_block6_1_relu[0][0]
conv5_block6_concat (Concatenat	(None, 4, 4, 704)	0	conv5_block5_concat[0][0] conv5_block6_2_conv[0][0]
conv5_block7_0_bn (BatchNormali	(None, 4, 4, 704)	2816	conv5_block6_concat[0][0]
conv5_block7_0_relu (Activation	(None, 4, 4, 704)	0	conv5_block7_0_bn[0][0]
conv5_block7_1_conv (Conv2D)	(None, 4, 4, 128)	90112	conv5_block7_0_relu[0][0]
conv5_block7_1_bn (BatchNormali	(None, 4, 4, 128)	512	conv5_block7_1_conv[0][0]

conv5_block7_1_relu (Activation	(None, 4, 4, 128)	0	conv5_block7_1_bn[0][0]
conv5_block7_2_conv (Conv2D)	(None, 4, 4, 32)	36864	conv5_block7_1_relu[0][0]
conv5_block7_concat (Concatenat	(None, 4, 4, 736)	0	conv5_block6_concat[0][0] conv5_block7_2_conv[0][0]
conv5_block8_0_bn (BatchNormali	(None, 4, 4, 736)	2944	conv5_block7_concat[0][0]
conv5_block8_0_relu (Activation	(None, 4, 4, 736)	0	conv5_block8_0_bn[0][0]
conv5_block8_1_conv (Conv2D)	(None, 4, 4, 128)	94208	conv5_block8_0_relu[0][0]
conv5_block8_1_bn (BatchNormali	(None, 4, 4, 128)	512	conv5_block8_1_conv[0][0]
conv5_block8_1_relu (Activation	(None, 4, 4, 128)	0	conv5_block8_1_bn[0][0]
conv5_block8_2_conv (Conv2D)	(None, 4, 4, 32)	36864	conv5_block8_1_relu[0][0]
conv5_block8_concat (Concatenat	(None, 4, 4, 768)	0	conv5_block7_concat[0][0] conv5_block8_2_conv[0][0]
conv5_block9_0_bn (BatchNormali	(None, 4, 4, 768)	3072	conv5_block8_concat[0][0]
conv5_block9_0_relu (Activation	(None, 4, 4, 768)	0	conv5_block9_0_bn[0][0]
conv5_block9_1_conv (Conv2D)	(None, 4, 4, 128)	98304	conv5_block9_0_relu[0][0]
conv5_block9_1_bn (BatchNormali	(None, 4, 4, 128)	512	conv5_block9_1_conv[0][0]
conv5_block9_1_relu (Activation	(None, 4, 4, 128)	0	conv5_block9_1_bn[0][0]
conv5_block9_2_conv (Conv2D)	(None, 4, 4, 32)	36864	conv5_block9_1_relu[0][0]
conv5_block9_concat (Concatenat	(None, 4, 4, 800)	0	conv5_block8_concat[0][0] conv5_block9_2_conv[0][0]
conv5_block10_0_bn (BatchNormal	(None, 4, 4, 800)	3200	conv5_block9_concat[0][0]
conv5_block10_0_relu (Activatio	(None, 4, 4, 800)	0	conv5_block10_0_bn[0][0]
conv5_block10_1_conv (Conv2D)	(None, 4, 4, 128)	102400	conv5_block10_0_relu[0][0]
conv5_block10_1_bn (BatchNormal	(None, 4, 4, 128)	512	conv5_block10_1_conv[0][0]
conv5_block10_1_relu (Activatio	(None, 4, 4, 128)	0	conv5_block10_1_bn[0][0]
conv5_block10_2_conv (Conv2D)	(None, 4, 4, 32)	36864	conv5_block10_1_relu[0][0]
conv5_block10_concat (Concatena	(None, 4, 4, 832)	0	conv5_block9_concat[0][0] conv5_block10_2_conv[0][0]
conv5_block11_0_bn (BatchNormal	(None, 4, 4, 832)	3328	conv5_block10_concat[0][0]
conv5_block11_0_relu (Activatio	(None, 4, 4, 832)	0	conv5_block11_0_bn[0][0]
conv5_block11_1_conv (Conv2D)	(None, 4, 4, 128)	106496	conv5_block11_0_relu[0][0]
conv5_block11_1_bn (BatchNormal	(None, 4, 4, 128)	512	conv5_block11_1_conv[0][0]
conv5_block11_1_relu (Activatio	(None, 4, 4, 128)	0	conv5_block11_1_bn[0][0]
conv5_block11_2_conv (Conv2D)	(None, 4, 4, 32)	36864	conv5_block11_1_relu[0][0]
conv5_block11_concat (Concatena	(None, 4, 4, 864)	0	conv5_block10_concat[0][0] conv5_block11_2_conv[0][0]
conv5_block12_0_bn (BatchNormal	(None, 4, 4, 864)	3456	conv5_block11_concat[0][0]
conv5_block12_0_relu (Activatio	(None, 4, 4, 864)	0	conv5_block12_0_bn[0][0]
conv5_block12_1_conv (Conv2D)	(None, 4, 4, 128)	110592	conv5_block12_0_relu[0][0]

conv5_block12_1_bn (BatchNormal	(None, 4, 4, 128)	512	conv5_block12_1_conv[0][0]
conv5_block12_1_relu (Activatio	(None, 4, 4, 128)	0	conv5_block12_1_bn[0][0]
conv5_block12_2_conv (Conv2D)	(None, 4, 4, 32)	36864	conv5_block12_1_relu[0][0]
conv5_block12_concat (Concatena	(None, 4, 4, 896)	0	conv5_block11_concat[0][0] conv5_block12_2_conv[0][0]
conv5_block13_0_bn (BatchNormal	(None, 4, 4, 896)	3584	conv5_block12_concat[0][0]
conv5_block13_0_relu (Activatio	(None, 4, 4, 896)	0	conv5_block13_0_bn[0][0]
conv5_block13_1_conv (Conv2D)	(None, 4, 4, 128)	114688	conv5_block13_0_relu[0][0]
conv5_block13_1_bn (BatchNormal	(None, 4, 4, 128)	512	conv5_block13_1_conv[0][0]
conv5_block13_1_relu (Activatio	(None, 4, 4, 128)	0	conv5_block13_1_bn[0][0]
conv5_block13_2_conv (Conv2D)	(None, 4, 4, 32)	36864	conv5_block13_1_relu[0][0]
conv5_block13_concat (Concatena	(None, 4, 4, 928)	0	conv5_block12_concat[0][0] conv5_block13_2_conv[0][0]
conv5_block14_0_bn (BatchNormal	(None, 4, 4, 928)	3712	conv5_block13_concat[0][0]
conv5_block14_0_relu (Activatio	(None, 4, 4, 928)	0	conv5_block14_0_bn[0][0]
conv5_block14_1_conv (Conv2D)	(None, 4, 4, 128)	118784	conv5_block14_0_relu[0][0]
conv5_block14_1_bn (BatchNormal	(None, 4, 4, 128)	512	conv5_block14_1_conv[0][0]
conv5_block14_1_relu (Activatio	(None, 4, 4, 128)	0	conv5_block14_1_bn[0][0]
conv5_block14_2_conv (Conv2D)	(None, 4, 4, 32)	36864	conv5_block14_1_relu[0][0]
conv5_block14_concat (Concatena	(None, 4, 4, 960)	0	conv5_block13_concat[0][0] conv5_block14_2_conv[0][0]
conv5_block15_0_bn (BatchNormal	(None, 4, 4, 960)	3840	conv5_block14_concat[0][0]
conv5_block15_0_relu (Activatio	(None, 4, 4, 960)	0	conv5_block15_0_bn[0][0]
conv5_block15_1_conv (Conv2D)	(None, 4, 4, 128)	122880	conv5_block15_0_relu[0][0]
conv5_block15_1_bn (BatchNormal	(None, 4, 4, 128)	512	conv5_block15_1_conv[0][0]
conv5_block15_1_relu (Activatio	(None, 4, 4, 128)	0	conv5_block15_1_bn[0][0]
conv5_block15_2_conv (Conv2D)	(None, 4, 4, 32)	36864	conv5_block15_1_relu[0][0]
conv5_block15_concat (Concatena	(None, 4, 4, 992)	0	conv5_block14_concat[0][0] conv5_block15_2_conv[0][0]
conv5_block16_0_bn (BatchNormal	(None, 4, 4, 992)	3968	conv5_block15_concat[0][0]
conv5_block16_0_relu (Activatio	(None, 4, 4, 992)	0	conv5_block16_0_bn[0][0]
conv5_block16_1_conv (Conv2D)	(None, 4, 4, 128)	126976	conv5_block16_0_relu[0][0]
conv5_block16_1_bn (BatchNormal	(None, 4, 4, 128)	512	conv5_block16_1_conv[0][0]
conv5_block16_1_relu (Activatio	(None, 4, 4, 128)	0	conv5_block16_1_bn[0][0]
conv5_block16_2_conv (Conv2D)	(None, 4, 4, 32)	36864	conv5_block16_1_relu[0][0]
conv5_block16_concat (Concatena	(None, 4, 4, 1024)	0	conv5_block15_concat[0][0] conv5_block16_2_conv[0][0]
bn (BatchNormalization)	(None, 4, 4, 1024)	4096	conv5_block16_concat[0][0]
relu (Activation)	(None, 4, 4, 1024)	0	bn[0][0]

global_average_pooling2d_1 (Glo	(None, 1024)	0	relu[0][0]
dense_1 (Dense)	(None, 4096)	4198400	global_average_pooling2d_1[0][0]
dense_2 (Dense)	(None, 242)	991474	dense_1[0][0]

Total params: 12,227,378
Trainable params: 12,143,730
Non-trainable params: 83,648

2. Evaluación del modelo entrenado

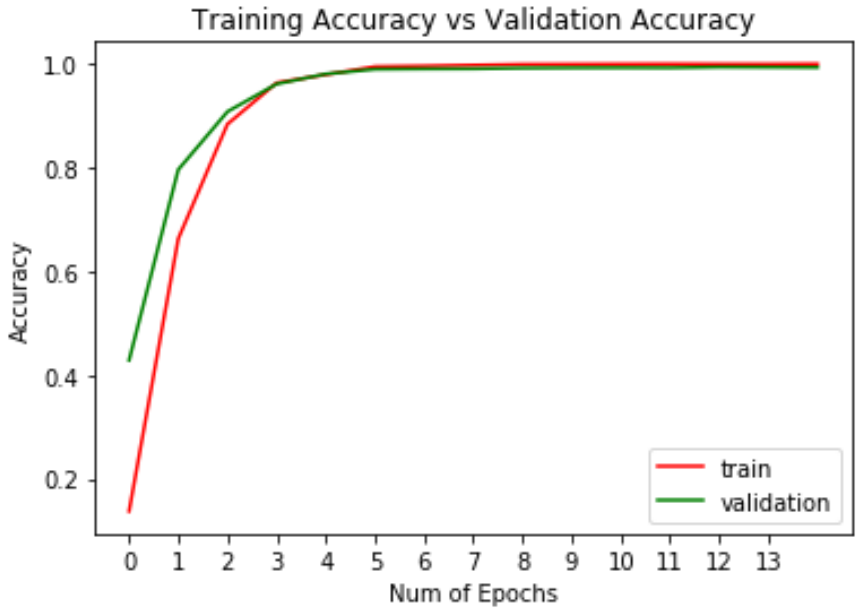
Loss	Acc	Mean_scuared_error
[0.02903933154195422,	**0.9932497589199615**,	0.0013556241169845804]

Comentario: Como se aprecia en la evaluación, el modelo logra converger y alcanza una proporción de **0.9932497589199615** de Precisión (Accurancy), usando 8,296 imágenes de validación a un tamaño de entrada de 150 x 150 píxeles.

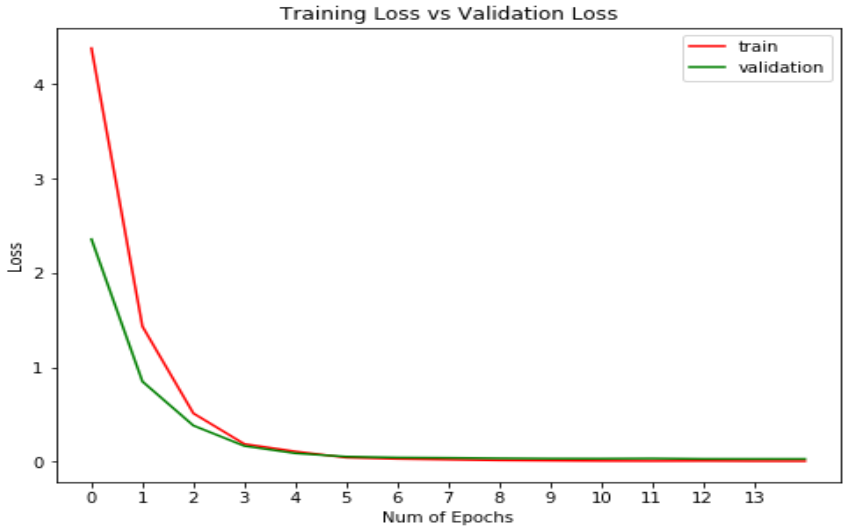
3. Gráficas del proceso de entrenamiento final (etapa II)

Gráficas que muestran el proceso de entrenamiento.

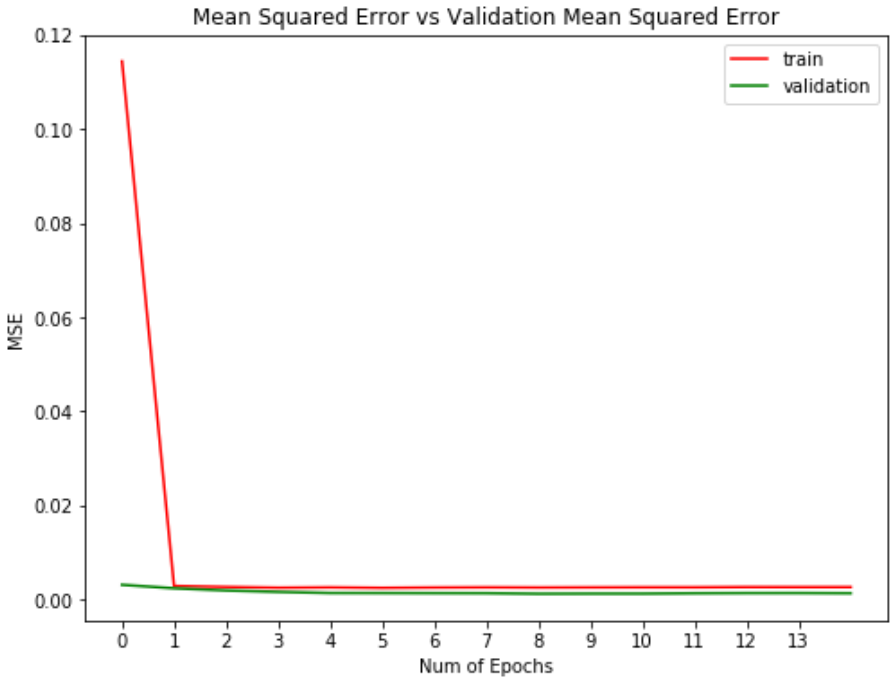
Accurancy (precisión): En la gráfica vemos la precisión del entrenamiento ver-sus la precisión de la validación, hasta lograr un nivel de precisión alto a través de las diferentes épocas.



Loss (pérdida): En la siguiente gráfica mostramos la evolución del entrenamiento visto desde una perspectiva de la perdida de información (Loss), tendiendo a cero en las épocas finales; es decir, en un inicio se tuvo altas tasas de perdida (no se lograba identificar a las personas).

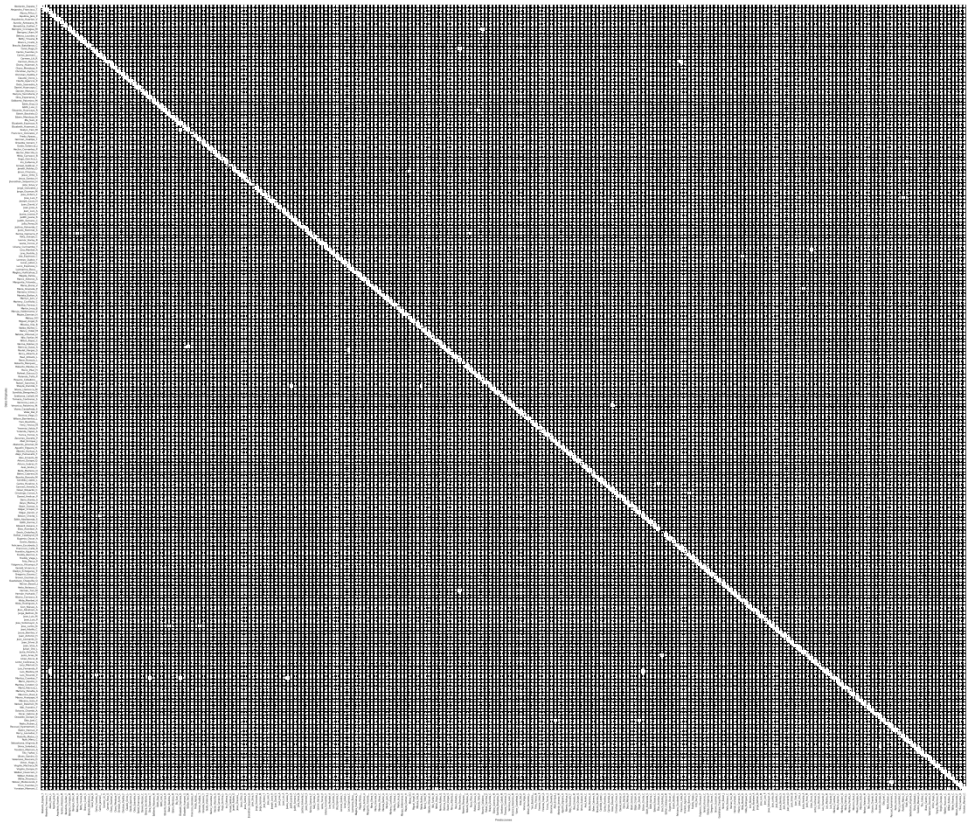


Error cuadrático medio (ECM): Es un estimador que mide el promedio de los errores al cuadrado, en otras palabras, la diferencia entre el estimador y lo que se estima. El ECM es una función de riesgo, correspondiente al valor esperado de la pérdida del error al cuadrado o pérdida cuadrática, donde se aprecia que inicialmente los errores son altos, y al final de la época 01 el error tiende a cero.

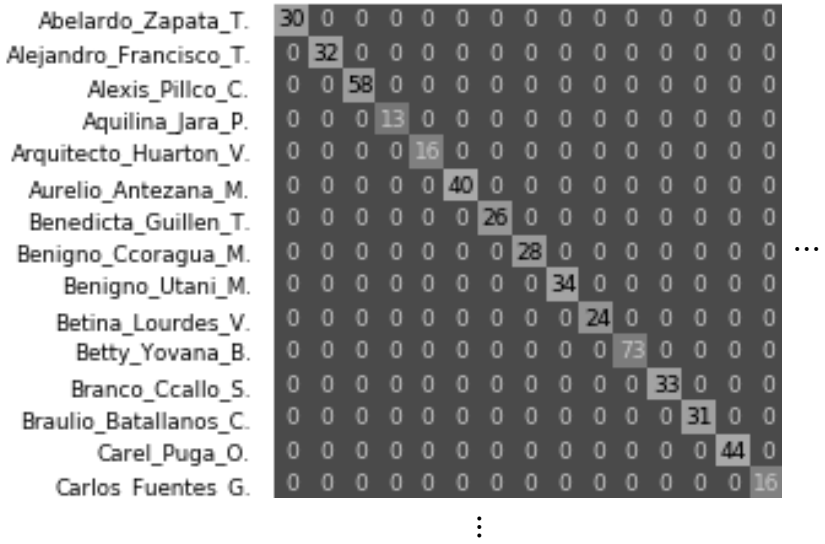


Gráfica de la matriz de confusiones

Una matriz de confusión es una herramienta que nos permite ver el desempeño del modelo en forma general, donde cada columna de la matriz representa la identificación de una persona (clase), mientras que cada fila representa a la persona en la clase real (242 personas). Uno de los beneficios de las matrices de confusión es que facilitan ver si el sistema está confundiendo dos clases o no, por lo que en nuestro modelo apreciamos casi una diagonal perfecta, salvo algunos puntos en donde el modelo se equivocó.



La matriz de confusiones representa a 242 clases, los cuales se ven muy pequeños, por lo que es necesario ver una muestra de la imagen ampliada (parte superior izquierda), como se muestra a continuación:



Comentario: Esta imágen muestra la cantidad de imágenes de entrada de cada clase (persona) y su respectiva predicción por el modelo. En esta sección no existe equivocaciones por el modelo, ya que como se puede ver no existen números fuera de la diagonal principal.

4. Reporte de clasificación del modelo II, propuesto (DenseNet121UNAMBA)

El presente reporte de la matriz de confusión, muestra una cantidad determinada de muestras de imágenes para evaluar el modelo propuesto; asimismo, algunos términos estándar para medir el desempeño del modelo:

N.°	Administrativos y docentes (clases)	Precision (Exactitud)	Recall (Sensibilidad)	F1-Score (Puntaje F1)	Support (Imágenes de entrada correctas)	Numero de prediccio nes del modelo
1	Abelardo_Zapata_T.	1.000	1.000	1.000	30	30
2	Alejandro_Francisco_T.	1.000	1.000	1.000	32	32
3	Alexis_Pillco_C.	0.967	1.000	0.983	58	58
4	Aquilina_Jara_P.	1.000	1.000	1.000	13	13
5	Arquitecto_Huarton_V.	1.000	1.000	1.000	16	16
6	Aurelio_Antezana_M.	1.000	1.000	1.000	40	40
7	Benedicta_Guillen_T.	1.000	1.000	1.000	26	26
8	Benigno_Ccoragua_M.	1.000	0.933	0.966	30	28
9	Benigno_Utani_M.	1.000	1.000	1.000	34	34
10	Betina_Lourdes_V.	0.960	1.000	0.980	24	24
11	Betty_Yovana_B.	1.000	1.000	1.000	73	73
12	Branco_Ccallo_S.	1.000	1.000	1.000	33	33
13	Braulio_Batallanos_C.	1.000	1.000	1.000	31	31
14	Carel_Puga_O.	1.000	0.957	0.978	46	44
15	Carlos_Fuentes_G.	0.941	1.000	0.970	16	16
16	Carlos_Jasmani__.	0.966	1.000	0.982	28	28
17	Carmen_Lili_R.	1.000	1.000	1.000	37	37
18	Carmen_Rosa_O.	1.000	0.947	0.973	38	36
19	Chany_Huaman_A.	1.000	1.000	1.000	19	19
20	Charo_Mendoza_P.	1.000	1.000	1.000	29	29
21	Christian_Aycho_C.	1.000	1.000	1.000	71	71
22	Christian_Padilla_F.	1.000	1.000	1.000	32	32
23	Claudio_Ocros_L.	1.000	1.000	1.000	28	28
24	Cleofe_Aparicio_P.	1.000	1.000	1.000	30	30
25	Cleto_Saavedra_A.	1.000	1.000	1.000	24	24
26	Daniel_Huarcaya_C.	1.000	1.000	1.000	15	15
27	Danien_Monzon_C.	1.000	1.000	1.000	26	26
28	Danyra_Secretaria_V.	1.000	1.000	1.000	26	26
29	Dina_Esperanza_T.	0.929	1.000	0.963	13	13
30	Edilberto_Palomino_M.	1.000	1.000	1.000	29	29
31	Edith_Diaz_Q.	1.000	1.000	1.000	34	34
32	Edith_Linn_C.	1.000	1.000	1.000	31	31
33	Eduardo_Huarcaya_R.	1.000	0.974	0.987	39	38
34	Edwin_Bautista_D.	0.979	1.000	0.989	47	47
35	Edwin_Mendoza_M.	1.000	1.000	1.000	50	50
36	Efa_Suni_Z.	1.000	1.000	1.000	39	39
37	Elizabeth_Espinoza_R.	0.938	1.000	0.968	45	45
38	Elizabeth_Huamani_S.	1.000	1.000	1.000	15	15
39	Evelyn_Yeni_M.	0.974	0.974	0.974	76	74
40	Francisco_Gonzales_V.	1.000	1.000	1.000	78	78
41	Fredy_Apaza__.	1.000	1.000	1.000	57	57
42	German_Huallpa_T.	0.963	1.000	0.981	26	26
43	Griselda_Venero_T.	1.000	1.000	1.000	22	22

44	Guido_Valencia_J.	1.000	1.000	1.000	15	15
45	Hector_Cervantes_P.	1.000	1.000	1.000	29	29
46	Hector_Marcilla_G.	1.000	1.000	1.000	52	52
47	Hilda_Carrasco_B.	1.000	1.000	1.000	17	17
48	Hugo_Quichca_L.	1.000	1.000	1.000	59	59
49	Iris_Eufemia_P.	1.000	0.980	0.990	51	50
50	Ismael_Saldivar_T.	1.000	1.000	1.000	79	79
51	Janeth_Robles_O.	1.000	1.000	1.000	16	16
52	Jesus_Chacara__.	1.000	0.964	0.982	28	27
53	Jesus_Ortiz_S.	1.000	1.000	1.000	70	70
54	Jesus_Santos_H.	1.000	1.000	1.000	24	24
55	Jhonathan_Sequeiros	1.000	1.000	1.000	25	25
56	John_Silva_V.	1.000	1.000	1.000	82	82
57	Jorge_Gonzales_J.	1.000	1.000	1.000	28	28
58	Jorge_Guzman_M.	1.000	1.000	1.000	34	34
59	Jose_Antero_F.	1.000	1.000	1.000	19	19
60	Jose_Luis_F.	1.000	0.971	0.985	68	66
61	Joseph_Louis_H.	1.000	0.972	0.986	36	35
62	Juan_David_V.	1.000	1.000	1.000	25	25
63	Juan_Jose_A.	1.000	1.000	1.000	45	45
64	Juan_Luis_A.	1.000	1.000	1.000	37	37
65	Juana_Loaisa_P.	0.957	0.978	0.967	45	44
66	Judith_Juana_R.	0.977	1.000	0.989	43	43
67	Judith_Serrano_C.	1.000	1.000	1.000	14	14
68	Julia_Perez_N.	1.000	1.000	1.000	15	15
69	Justina_Valverde_C.	1.000	1.000	1.000	56	56
70	Justo_Rommel_S.	1.000	1.000	1.000	41	41
71	Karina_Gamarra_P.	1.000	0.961	0.980	51	49
72	Katy_Quispe_C.	1.000	1.000	1.000	55	55
73	Leonor_Garay_R.	1.000	1.000	1.000	48	48
74	Leslie_Torres_P.	1.000	1.000	1.000	19	19
75	Liliana_Curinambe_T.	1.000	1.000	1.000	35	35
76	Lina_Maribel_A.	0.958	0.958	0.958	24	23
77	Lirio_Portillo_S.	1.000	1.000	1.000	36	36
78	Lita_Espinoza_C.	1.000	0.962	0.980	52	50
79	Lorenzo_Quibio_Y.	1.000	1.000	1.000	28	28
80	Lucas_Julian_L.	1.000	1.000	1.000	19	19
81	Lucio_Espinoza_C.	0.980	1.000	0.990	49	49
82	Luzmarina_Baca_.	1.000	1.000	1.000	16	16
83	Magbis_Huillcahua_S.	1.000	1.000	1.000	12	12
84	Magda_Paliza.	1.000	1.000	1.000	54	54
85	Marco_Antonio_A.	1.000	1.000	1.000	37	37
86	Margarita_Yolanda.	1.000	1.000	1.000	37	37
87	Maria_Elena_V.	1.000	1.000	1.000	34	34
88	Maria_Graciela_P.	1.000	0.978	0.989	46	45
89	Mariana_Vilma_C.	1.000	1.000	1.000	29	29
90	Mariela_Bañon_A.	1.000	1.000	1.000	58	58
91	Marilyn_Juro_V.	1.000	1.000	1.000	33	33
92	Marleny_Ccoñislla_C.	1.000	1.000	1.000	12	12
93	Martha_Teresa_C.	1.000	1.000	1.000	48	48

94	Martin_Inca_B.	1.000	1.000	1.000	31	31
95	Maruja_Valderrama_V.	1.000	1.000	1.000	63	63
96	Mayte_Damian_H.	1.000	1.000	1.000	37	37
97	Melisa_OCI	0.980	1.000	0.990	48	48
98	Miguel_Angel_R.	1.000	1.000	1.000	24	24
99	Miluska_Vila_B.	1.000	1.000	1.000	18	18
100	Nalda_Nuñez_C.	0.960	1.000	0.980	24	24
101	Nancy_Vidal_M.	1.000	1.000	1.000	14	14
102	Natalie_Villaroel_H.	1.000	1.000	1.000	56	56
103	Nilo_Farfan_M.	1.000	1.000	1.000	48	48
104	Nilton_Rojas_C.	0.971	1.000	0.986	34	34
105	Norma_Robles_Q.	1.000	1.000	1.000	25	25
106	Patricia_Quiza_A.	1.000	0.943	0.971	35	33
107	Paulet_Vargas_S.	1.000	0.967	0.983	30	29
108	Percy_Alberto_E.	1.000	1.000	1.000	16	16
109	Raul_Arbieto_L.	1.000	1.000	1.000	21	21
110	Rene_Romulo_L.	0.940	1.000	0.969	63	63
111	Roberto_Marquez_.	1.000	1.000	1.000	82	82
112	Roberto_Mestas_G.	1.000	1.000	1.000	33	33
113	Rocio_Pilar_H.	0.955	1.000	0.977	21	21
114	Rofwel_Quisca_Q.	1.000	1.000	1.000	50	50
115	Rolando_Tello_P.	0.973	1.000	0.986	36	36
116	Rosario_Collabino__.	0.947	1.000	0.973	36	36
117	Ruben_Sanchez_E.	1.000	1.000	1.000	22	22
118	Sheyla_Zunilda_C.	1.000	0.970	0.985	67	65
119	Silano_Llamocca_M.	1.000	1.000	1.000	41	41
120	Sumilda_Bengolea_D.	1.000	1.000	1.000	70	70
121	Svetlania_Callalli_M.	1.000	1.000	1.000	36	36
122	Tomaza_Contreras_S.	1.000	1.000	1.000	46	46
123	Veronica_Leon_C.	1.000	1.000	1.000	26	26
124	Veronica_Palomino_M.	1.000	0.947	0.973	38	36
125	Viane_Castañeda_S.	1.000	1.000	1.000	28	28
126	Vidal_Del_P.	1.000	1.000	1.000	26	26
127	Vinerva_Vega_O.	1.000	1.000	1.000	66	66
128	Wilson_Barrientos_L.	1.000	1.000	1.000	13	13
129	Yeni_Bautista__.	1.000	1.000	1.000	63	63
130	Yeny_Yesica_M.	1.000	1.000	1.000	45	45
131	Yesenia_Ustúa_P.	1.000	1.000	1.000	55	55
132	Yolanda_Yepez_A.	1.000	1.000	1.000	19	19
133	Yurica_Torres_S.	1.000	1.000	1.000	65	65
134	Zacarias_Zavalla_P.	1.000	1.000	1.000	33	33
135	Abel_Enrique_J.	0.947	1.000	0.973	18	18
136	Abelardo_Jimenez_M.	1.000	1.000	1.000	30	30
137	Agustín_Elguera_H.	1.000	1.000	1.000	19	19
138	Alberto_Huihua_A.	1.000	1.000	1.000	50	50
139	Alejo_Pumacallo_F.	1.000	1.000	1.000	12	12
140	Alex_Ernesto_M.	1.000	1.000	1.000	26	26
141	Arturo_Quispe_Q.	1.000	1.000	1.000	23	23
142	Arturo_Suárez_O.	1.000	1.000	1.000	68	68
143	Axel_Andre_C.	1.000	1.000	1.000	54	54

144	Beda_Marlene_O.	1.000	1.000	1.000	28	28
145	Belen_Cabrera_N.	1.000	1.000	1.000	67	67
146	Braulio_Barzola_M.	1.000	1.000	1.000	20	20
147	Candida_Lopez_L.	1.000	1.000	1.000	37	37
148	Carlos_Rivelino_S.	1.000	0.929	0.963	28	26
149	Carmen_Amelia_A.	1.000	1.000	1.000	32	32
150	César_Eduardo_C.	0.921	1.000	0.959	35	35
151	Crisologo_Conza_A.	1.000	0.969	0.984	64	62
152	Daniel_Amilcar_P.	1.000	1.000	1.000	14	14
153	Dario_Dante_S.	1.000	1.000	1.000	15	15
154	Deysi_Melisa_P.	1.000	1.000	1.000	32	32
155	Ebert_Gomez_A.	1.000	1.000	1.000	14	14
156	Edgar_Crispin_H.	1.000	1.000	1.000	20	20
157	Edgar_Zenón_V.	0.968	1.000	0.984	30	30
158	Edison_Chiclla_C.	0.953	0.976	0.965	42	41
159	Edith_Abollaneda_S.	1.000	1.000	1.000	18	18
160	Edith_Karina_C.	1.000	1.000	1.000	16	16
161	Edward_Ilasaca_C.	1.000	1.000	1.000	44	44
162	Elias_Aranibar_A.	0.960	1.000	0.980	24	24
163	Erech_Ordoñez_R.	0.979	1.000	0.989	93	93
164	Esther_Calatayud_M.	1.000	1.000	1.000	15	15
165	Eugenio_Oscar_A.	1.000	1.000	1.000	26	26
166	Evelin_Naida_L.	1.000	1.000	1.000	31	31
167	Feliciano_Escobedo_S.	1.000	1.000	1.000	37	37
168	Francisco_Calle_R.	0.944	1.000	0.971	34	34
169	Franklin_Aguirre_H.	1.000	1.000	1.000	30	30
170	Freddy_Barrios_S.	0.979	1.000	0.989	46	46
171	Freddy_Vega_L.	1.000	1.000	1.000	31	31
172	Fritz_Percy_P.	1.000	1.000	1.000	14	14
173	Fulgencio_Vilcanqui_P.	1.000	1.000	1.000	11	11
174	Gerald_Vicencio_C.	1.000	1.000	1.000	32	32
175	Gladys_Echegaray_P.	0.980	1.000	0.990	48	48
176	Gregorio_Gauna_C.	1.000	1.000	1.000	15	15
177	Grover_Guzman_G.	1.000	1.000	1.000	12	12
178	Guadalupe_Chaquilla_Q	1.000	1.000	1.000	32	32
179	Hector_Basan_J.	1.000	1.000	1.000	36	36
180	Helio_Nolasco_C.	1.000	1.000	1.000	15	15
181	Hernan_Yari_M.	1.000	0.923	0.960	26	24
182	Hernán_Hurtado_T.	1.000	1.000	1.000	42	42
183	Hilario_Carrasco_K.	1.000	1.000	1.000	18	18
184	Hilda_Maribel_H.	0.977	1.000	0.989	43	43
185	Hilda_Rodriguez_A.	1.000	1.000	1.000	21	21
186	Ivon_Nieves_A.	1.000	1.000	1.000	34	34
187	Jhon_Abraham_A.	1.000	1.000	1.000	20	20
188	Jorge_Beltrán_M.	1.000	1.000	1.000	40	40
189	Jose_Luis_M.	1.000	1.000	1.000	12	12
190	Jose_Luis_P.	1.000	1.000	1.000	43	43
191	Jose_Sotomayor_A.	1.000	1.000	1.000	17	17
192	Jose_carlos_N.	1.000	0.961	0.980	51	49
193	José_Adolfo_C.	1.000	1.000	1.000	54	54

194	Josue_Benites_V.	1.000	1.000	1.000	15	15
195	Juan_Antonio_H.	1.000	1.000	1.000	30	30
196	Juan_Leonardo_D.	1.000	1.000	1.000	15	15
197	Juan_Silver_B.	1.000	1.000	1.000	54	54
198	Juan_Viza_A.	1.000	1.000	1.000	43	43
199	Julian_Ore_L.	0.971	1.000	0.986	34	34
200	Justa_Amalia_A.	0.966	1.000	0.982	28	28
201	Justo_Arias_M.	1.000	0.941	0.970	34	32
202	Lilian_Rocio_B.	0.929	1.000	0.963	13	13
203	Lintol_Contreras_S.	1.000	1.000	1.000	33	33
204	Lucy_Marisol_G.	1.000	1.000	1.000	58	58
205	Luis_Fernando_P.	1.000	1.000	1.000	41	41
206	Luis_Medina_M.	1.000	0.918	0.957	49	45
207	Luis_Ricardo_P.	1.000	0.976	0.988	42	41
208	Mariluz_Cuentas_T.	1.000	0.871	0.931	31	27
209	Mario_Aquino_C.	1.000	1.000	1.000	22	22
210	Maritza_Condori_Q.	1.000	1.000	1.000	14	14
211	María_Patricia_L.	1.000	1.000	1.000	45	45
212	Marleny_Peralta_A.	1.000	1.000	1.000	24	24
213	Mauricio_Raul_E.	1.000	1.000	1.000	44	44
214	Mauro_Huayapa_H.	1.000	1.000	1.000	34	34
215	Máximo_Soto_P.	1.000	1.000	1.000	49	49
216	Nelson_Palemón_M.	1.000	1.000	1.000	16	16
217	Niki_Franklin_F.	1.000	1.000	1.000	13	13
218	Octavio_Chambi_A.	1.000	1.000	1.000	51	51
219	Oscar_Sabino_B.	1.000	1.000	1.000	27	27
220	Oswaldo_Quispe_Q.	0.906	1.000	0.951	29	29
221	Otto_Joel_C.	1.000	0.935	0.967	31	29
222	Pablo_Ruben_Z.	1.000	1.000	1.000	46	46
223	Pascual_Ayamamani_C.	0.929	1.000	0.963	26	26
224	Pedro_Hernan_P.	1.000	1.000	1.000	25	25
225	Percy_Leonidas_C.	1.000	1.000	1.000	28	28
226	Rodolfo_Matos_O.	0.968	1.000	0.984	30	30
227	Ruth_Mery_C.	0.969	1.000	0.984	31	31
228	Sebastiana_Virginia_B.	1.000	1.000	1.000	13	13
229	Silvia_Soledad_L.	1.000	0.963	0.981	27	26
230	Teodoro_Mamani_A.	0.975	1.000	0.987	39	39
231	Tito_Yañez_C.	1.000	0.938	0.968	16	15
232	Ulises_Sandro_Q.	1.000	1.000	1.000	37	37
233	Valeriano_Paucara_O.	1.000	1.000	1.000	15	15
234	Victor_Hugo_S.	1.000	1.000	1.000	15	15
235	Virgilio_Machaca_M.	1.000	1.000	1.000	17	17
236	Virgilio_Quispe_D.	1.000	1.000	1.000	16	16
237	Walker_Huaccani_C.	1.000	1.000	1.000	23	23
238	Wilber_Robles_D.	0.960	1.000	0.980	24	24
239	Willie_Alvarez_C.	1.000	1.000	1.000	12	12
240	Wilson_Mollocondo_F.	1.000	0.959	0.979	49	47
241	Yhon_Fuentes_H.	1.000	1.000	1.000	17	17
242	Yonatan_Mamani_C.	1.000	0.882	0.938	17	15
	Macro avg (Promedio)	**0.993364**	**0.993711**	**0.993361**		

TOTAL		**8296**	**8240**
Accuracy (Precisión)	**0.9932498**		

Promedio de muestras	**34.3**
Desviación estándar	**16.6**
Mediana	**31**
Mínima muestra	**11**
Máxima muestra	**93**

Finalmente, podemos concluir que se ha logrado la proporción más alta de precisión (**0.993249759**) en la identificación del personal administrativo y docente de la Universidad Nacional Micaela Bastidas de Apurímac, usando una arquitectura moderna de Redes Convolucionales como DenseNet121 adecuado a un nuevo modelo DenseNet121UNAMBA.

$$p_2 = \frac{\text{Número correcto de predicciones}}{\text{Número total de imágenes}} = \frac{8240}{8296} = 0.9932498$$

Donde p_2 es la segunda proporción alta hallada.

Anexos 8: Obtención y procesamiento de imágenes

1. Técnica de Video Scraping

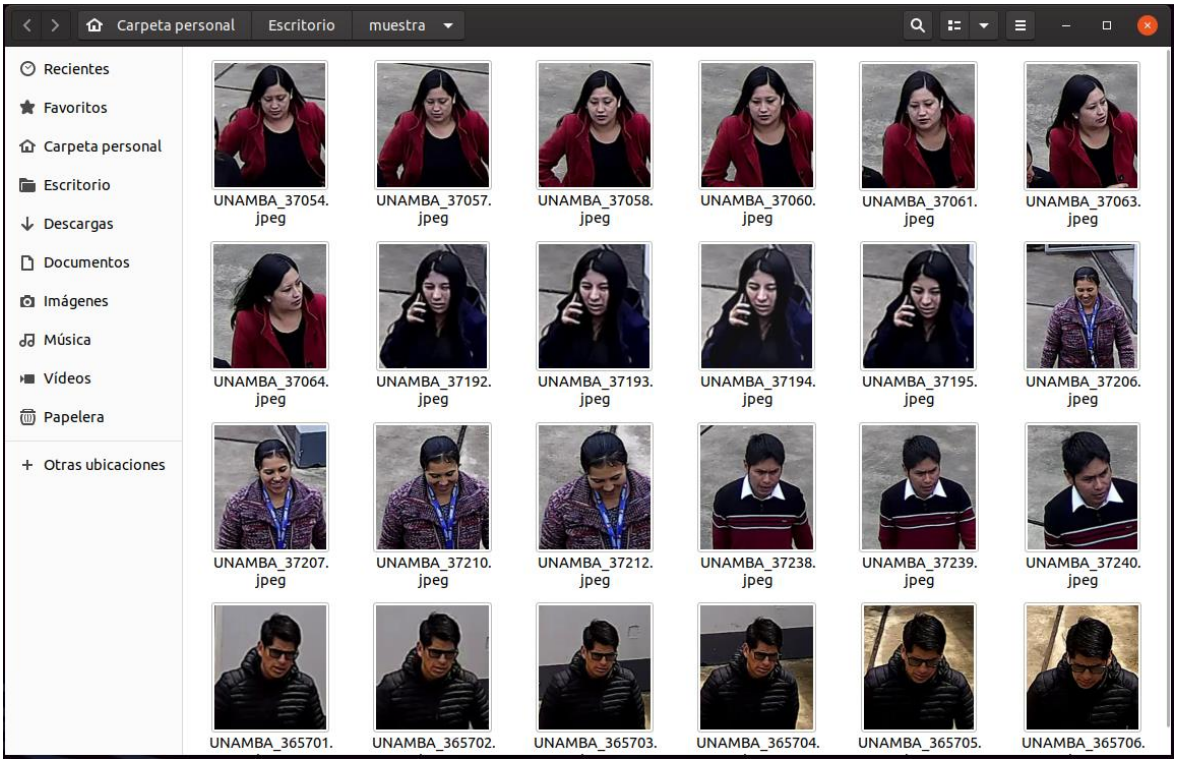
Para obtener imágenes (fotos) de cada persona a través de una cámara de video, se utilizó el siguiente código fuente en python:

```
import cv2

medioCuerpoCascade = cv2.CascadeClassifier('Cascades/haarcascade_mcs_upperbody.xml')
cuerpoEnteroCascade = cv2.CascadeClassifier('Cascades/haarcascade_fullbody.xml')

faceCascade = cv2.CascadeClassifier("Cascades/haarcascade_frontalface_alt2.xml")
captura = cv2.VideoCapture('rtsp://admin:pasword@192.168.0.100:554')
count = 0

if captura.isOpened():
   print("La conexion a la camara fue exitosa")
else:
   print("La conexion a la camara NO fue exitosa")
exit()
while(True):
   try:
      ret, frame = captura.read()
      if(ret==True):
         small_frame = cv2.resize(frame, (0, 0), fx=0.25, fy=0.25)#025=1/4, de tamaño
         grises = cv2.cvtColor(small_frame, cv2.COLOR_BGR2GRAY)
cuerpo = medioCuerpoCascade.detectMultiScale(grises)
         #cuerpo = cuerpoEnteroCascade.detectMultiScale(grises,1.2)
         for(x,y,w,h) in cuerpo:
            img_cuerpo=frame[y*4:y*4+h*4, x*4:x*4+w*4]#La mitad del alto de la imágen
cv2.imwrite("images/UNAMBA_"+str(count)+".jpeg", img_cuerpo)
            cv2.rectangle(small_frame, (x,y), (x+w, y+h), (0,255,0), 1)
count += 1
         cv2.imshow("Verificando rostros en medio cuerpo Cascade Video Pequeño", small_frame)
if cv2.waitKey(1) y 0xFF == ord('q'):
            break
         elif count >= 400000:
            break
      else:
print("no se pudo capturar nada")
         print("la transmision se corto")
         captura = cv2.VideoCapture('rtsp://admin:pasword@192.168.0.100:554')
#break
   except ValueError:
      print("Oops! No era válido. Intente nuevamente..." + ValueError)
# Cuando todo está hecho, liberamos la captura
captura.release()
cv2.destroyAllWindows()
print("La salida del programa fue exitosa")
```



Muestra de imágenes capturadas desde una cámara de videvigilancia

2. Data Augmentation for Deep Learning

Para el proceso de data augmentation se duplico las imágenes pero volteados en sentido horizontal, el código fuente en python fue:

```
import os
from PIL import Image, ImageOps

def imagenAumentada(carpeta,nombreImagen="UNAMBA_",enumerarDe=0,tipo="jpeg"):
    listaArchivos=os.listdir(carpeta)
    cont=enumerarDe
    ruta=carpeta+"/"
    for i in listaArchivos:
        if os.path.isfile(ruta+i):
            imagen=Image.open(ruta+i)
            espejo = ImageOps.mirror(imagen)
            espejo.save(ruta+nombreImagen+str(cont)+"."+tipo)
            cont = cont + 1
            #rotacion=imagen.rotate(45, expand=1)
            #rotacion.save(ruta+nombreImagen+str(cont)+"."+tipo)
            #cont = cont + 1
        if os.path.isdir(ruta+i):
            imagenAumentada(ruta+i,nombreImagen=nombreImagen+i[4:6]+"_")#6 primeras
letras a partir del 4to caracter de la carpeta

imagenAumentada("/media/rey/D2/data/entrenamiento")
print("Termino satisfactoriamente el aumento de datos")
```

3. Division de la información

Para el proceso de entrenamiento es necesario dividir la información en dos carpetas (entrenamiento y validación) a un porcentaje de 70% y 30% respectivamente, para ello se uso el siguiente código python:

```
import os
import os.path
import shutil
import glob
from random import shuffle
from tkinter import filedialog
#os.path.exists("Data")
#os.path.isdir("Tools")
#os.path.isfile("Tools")
#os.system('ls')

def copiaEstructura(ruta,sinArchivos=True,porcentajeAcopiar=1):
    estado=""
    dir_Acopiar=ruta+"/entrenamiento"
    dir_final=ruta+"/validacion"

    if os.path.isdir(dir_Acopiar):
        listaArchivos=os.listdir(dir_Acopiar)
        if(not os.path.isdir(dir_final)):
            os.mkdir(dir_final)
        else:
            estado="Error: Ya existe un Directorio -validacion- con contenido\n (borre el
directorio)"
            return [],estado

        for i in listaArchivos:
            os.mkdir(dir_final+'/'+i)
        estado="Ruta copiada satisfactoriamente sin Archivos"
    else:
        estado="No existe el Directorio: entrenamiento "
        return [],estado

    if(sinArchivos==False):
        print("========================================")
        print("Moviendo el ", porcentajeAcopiar*100, "% de Archivos \nal directorio Validacion")
        print("========================================")
        listaArchivos=os.listdir(dir_Acopiar)
        for subDir in listaArchivos:
            rutaObjetivo=dir_Acopiar+"/"+subDir
            archivos = glob.glob1(rutaObjetivo,"*")
            shuffle(archivos) #desordeno aleatoriamente la lista
            tam=len(archivos)
            porcentaje=int (tam*porcentajeAcopiar)

            for i in range(porcentaje):
                #shutil.move("path/to/current/file.foo", "path/to/new/destination/for/file.foo")
                archivoDestino=dir_final+"/"+subDir+"/"+archivos[i]
                archivoOrigen=dir_Acopiar+"/"+subDir+"/"+archivos[i]
                shutil.move(archivoOrigen,archivoDestino)

        estado="Ruta copiada satisfactoriamente con Archivos"
    return os.listdir(dir_final),estado
```

```
#rutaRaiz=os.getcwd()
#rutaObjetivo=rutaRaiz+"/data"
if (__name__):
   rutaObjetivo=filedialog.askdirectory(title = "Ingrese a la carpeta que contiene el directorio: -
entrenamiento-")

nuevaestruct,estado=copiaEstructura(rutaObjetivo,sinArchivos=False,porcentajeAcopiar=0.3)
   print(estado)
   print(nuevaestruct)
   print("termino la división entre entrenamiento y validación")
```