Clasificación de imágenes de biotita, bornita y crisocola

Leonardo Damián Cázares Trejo
Santiago Escamilla del Angel

Resumen

El objetivo de este proyecto es resolver un problema de clasificación multiclase de tres tipos de roca mediante el uso de una red convolucional (CNN).

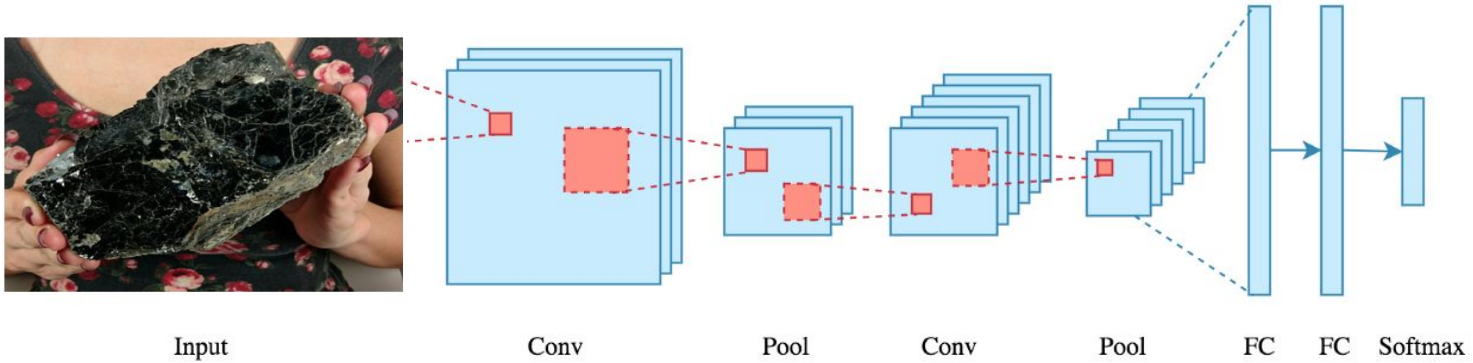
Propondremos dos modelos con diferente arquitectura y un manejo distinto de las imágenes para resolver el problema de clasificación. Presentaremos gráficas del desempeño de cada modelo durante el entrenamiento y la precisión alcanzada con el conjunto de prueba. Así como las modificaciones que se realizaron en cada modelo para mejorar el rendimiento (ajuste de hiper parámetros, uso de *dropout* y *data augmentation*).

Por último presentaremos una solución alternativa con la red pre entrenada VGG16 con *data augmentation* y *fine tunning*.

Introducción

La clasificación de imágenes es el proceso en el cual se asigna una etiqueta a una imagen, con una probabilidad p de que la imagen pertenezca a la clase correspondiente a la etiqueta. Este proceso se puede realizar con una CNN.

Las CNN (Convolutional Neural Network) son redes neuronales conformadas por *convolution layers* con funciones de activación ReLu (u otra), *pooling layers*, y *fully connected layers*.

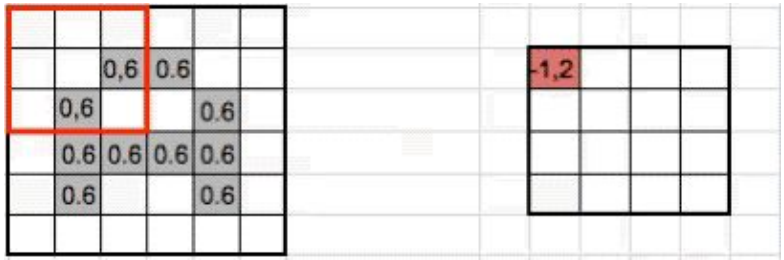


Convolutional neural network

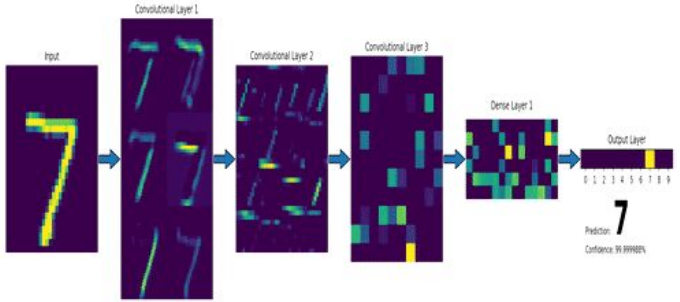
Convolutional Neural Networks

El funcionamiento general de una CNN consiste en recibir una imagen como *input* y extraer información de bloques de pixeles cercanos en las capas de convolución y *pooling* para crear un mapa de detección de características. Entre más capas tenga la red las características que podrá detectar serán más complejas.

Particularmente para la clasificación de imágenes, en la última capa de la CNN se obtiene la probabilidad de que la imagen pertenezca a cierta clase dependiendo de sus características.



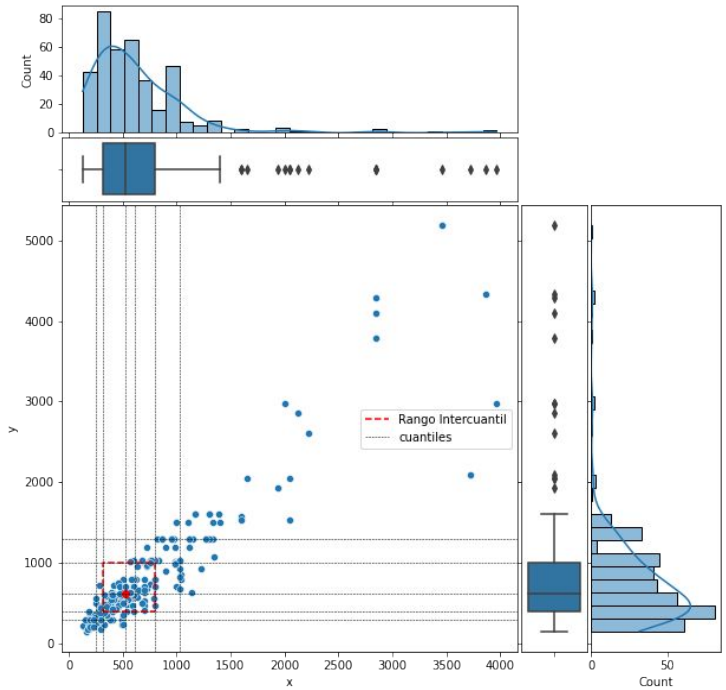
Bloques de pixeles contra el kernel



Ejemplo de mapas de detección de características

Dataset

Minerals Identification Dataset (Kofi Asiedu Brempong en Kera)	
7 Clases selección 3	biotita 68 bornita 173 crisocola 164
Problemas	diferente formato diferente tamaño
reescalado	shape [n,m,3] intensidad [0,255]->[0,1]
codificación	categorical [1,0,0] [0,1,0] [0,0,1]
split data	train 0.72 validation 0.18 test 0.1



Distribución dimensional de las imágenes

medida	x	y
media	648.13	761.58
mediana	530	612
cuantil 0.25	321	400

Manejo imágenes y Dataset split

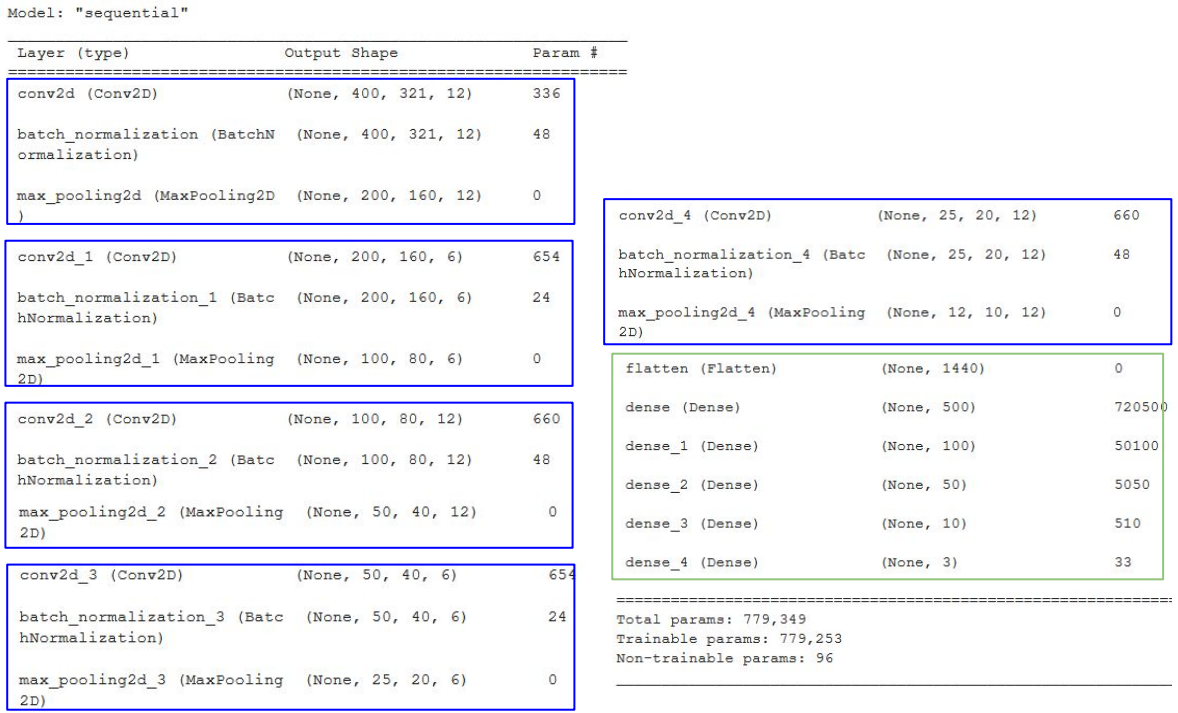
Utilizaremos un *dataset* de Kaggle que contiene 398 imágenes de biotita, bornita y crisocola a color.

Modelo	Dataset split *train*, *validation*, *test*	Reescalado n x m píxeles
1	72 %, 18%, 10%	321 x 400
2	40%, 20% , 40%	180 x 180
3		

Muestras del dataset

Modelo 1

Arquitectura de la red	
Partes	**Componentes**
Input(400,321,3)	
Parte 1	Conv2D(n,,(3,3),ReLu) BatchNormalization() MaxPooling2D(2,2) Bloque x5 n= 12, 6, 12, 6, 12
Parte 2	Flatten()
	Dense(m, ReLu) x4 m=500,100,50,10
	Dense(3, softmax)



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 400, 321, 12)      336

batch_normalization (BatchN  (None, 400, 321, 12)      48
ormalization)

max_pooling2d (MaxPooling2D  (None, 200, 160, 12)      0
)

conv2d_1 (Conv2D)            (None, 200, 160, 6)       654

batch_normalization_1 (Batc  (None, 200, 160, 6)       24
hNormalization)

max_pooling2d_1 (MaxPooling  (None, 100, 80, 6)        0
2D)

conv2d_2 (Conv2D)            (None, 100, 80, 12)       660

batch_normalization_2 (Batc  (None, 100, 80, 12)       48
hNormalization)

max_pooling2d_2 (MaxPooling  (None, 50, 40, 12)        0
2D)

conv2d_3 (Conv2D)            (None, 50, 40, 6)         654

batch_normalization_3 (Batc  (None, 50, 40, 6)         24
hNormalization)

max_pooling2d_3 (MaxPooling  (None, 25, 20, 6)         0
2D)

conv2d_4 (Conv2D)            (None, 25, 20, 12)        660

batch_normalization_4 (Batc  (None, 25, 20, 12)        48
hNormalization)

max_pooling2d_4 (MaxPooling  (None, 12, 10, 12)        0
2D)

flatten (Flatten)            (None, 1440)              0

dense (Dense)                (None, 500)               720500

dense_1 (Dense)              (None, 100)               50100

dense_2 (Dense)              (None, 50)                5050

dense_3 (Dense)              (None, 10)                510

dense_4 (Dense)              (None, 3)                 33

=================================================================
Total params: 779,349
Trainable params: 779,253
Non-trainable params: 96
_________________________________________________________________
```

Entrenamiento Modelo 1

Algoritmo de optimización:
Adam(learning rate = 1e-4, decay=1e-5)

Función de pérdida:
Categorical cross entropy

Métrica: **Accuracy**

Epochs: **50**

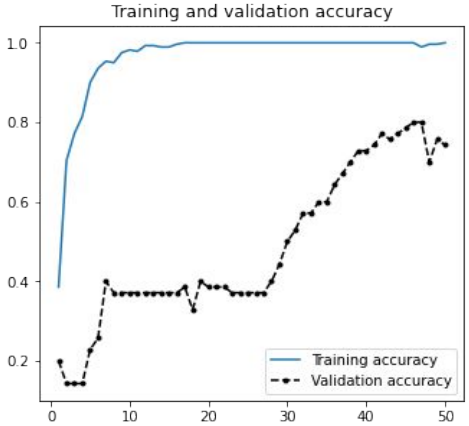
```
optimizador = tf.keras.optimizers.Adam(
    learning_rate=1e-4,
    name="Adam"
    ,decay=1e-5)
```

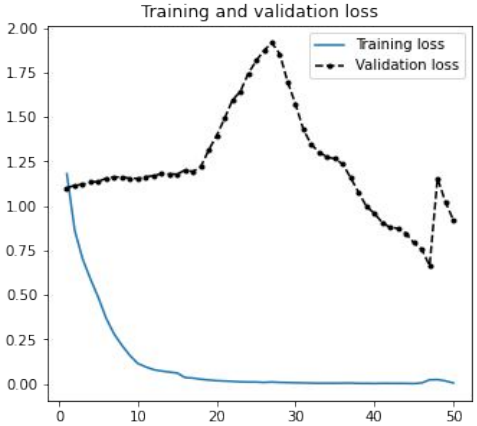
```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="/content/drive/Shareddrives/TSFC-IDL/model0.keras",
        save_best_only=True
       ,monitor="val_loss"
    )]
```

```
model0.compile(loss='categorical_crossentropy',
               optimizer=optimizador,
               metrics=["accuracy"])
```

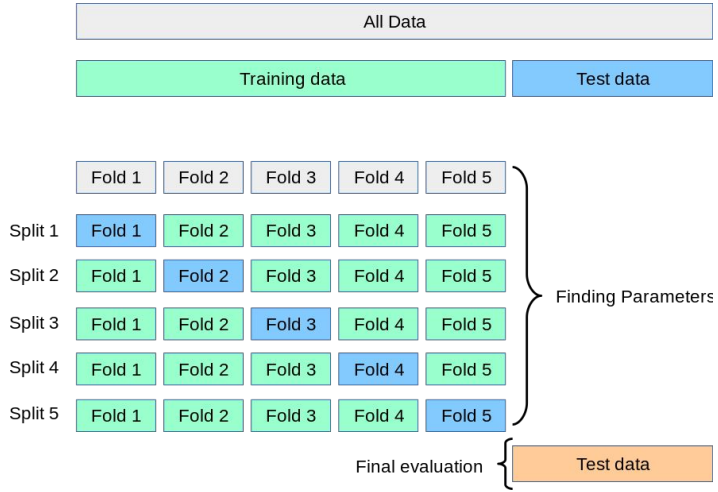
```
history0 = model0.fit(
    train_X,
    train_Y,
    epochs=50,
    batch_size=24,
    validation_data=(validation_X,validation_Y)
   ,callbacks=callbacks )
```

Resultados modelo 1



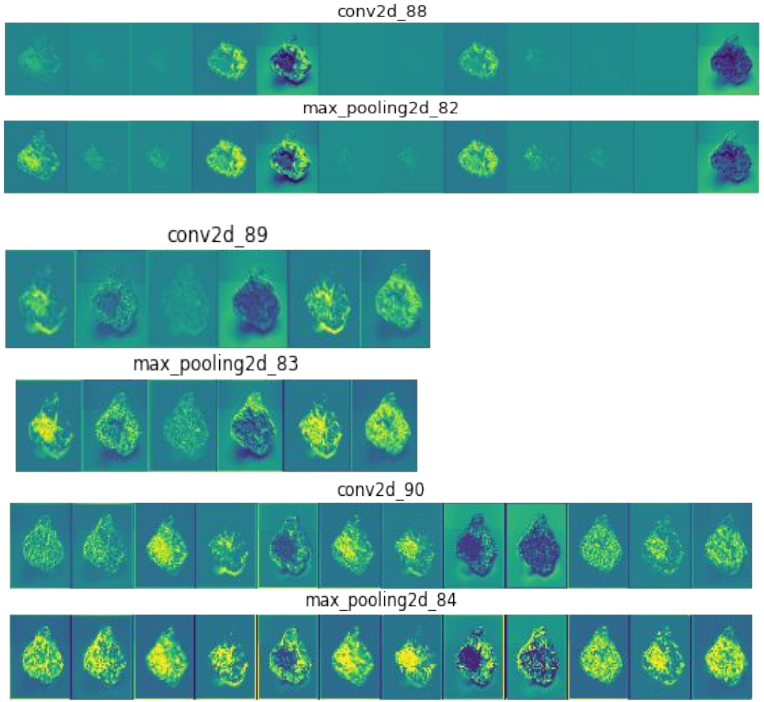


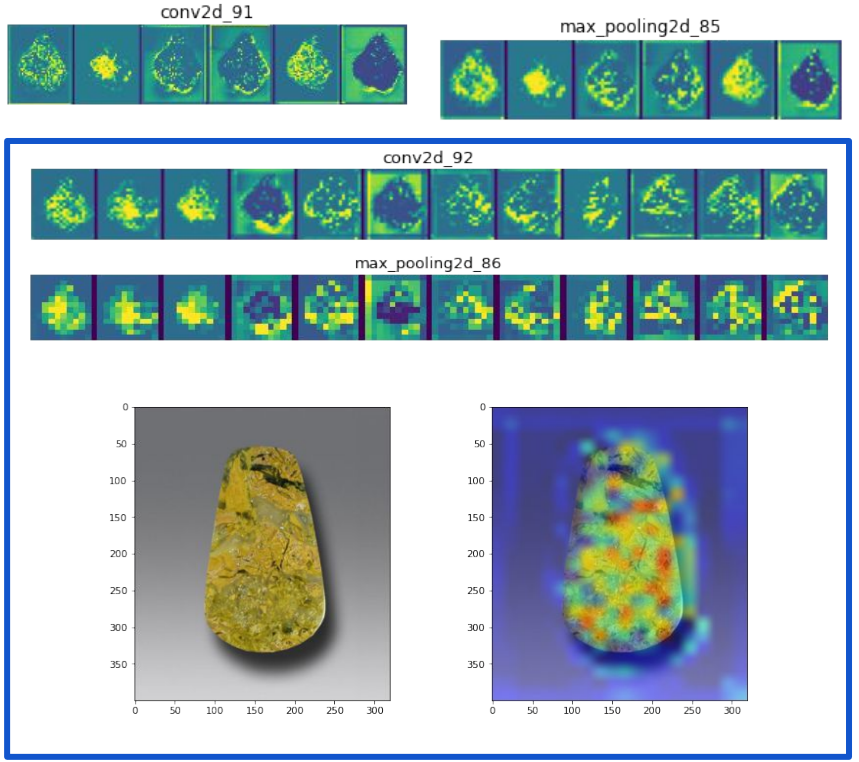

```
loss: 0.5409 - accuracy: 0.8250
```



K-fold validation	
n split	5
accuracy loss	0.68

Mapa de características y heatmap





Las imágenes RGB proporcionan información valiosa sobre los minerales por lo tanto el modelos extrae características seguida de una reducción de la información.

Modelo 2

Arquitectura de la red

- 4 capas Conv2D - MaxPooling2D y función de activación ReLu
- 1 capa Conv2D - Flatten función de activación ReLu
- última capa con 3 nodos y función de activación softmax

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 180, 180, 3)]     0

 rescaling (Rescaling)       (None, 180, 180, 3)       0

 conv2d (Conv2D)             (None, 178, 178, 32)      896

 max_pooling2d (MaxPooling2D  (None, 89, 89, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 87, 87, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 43, 43, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 41, 41, 128)       73856

 max_pooling2d_2 (MaxPooling  (None, 20, 20, 128)      0
 2D)

 conv2d_3 (Conv2D)           (None, 18, 18, 256)       295168

 max_pooling2d_3 (MaxPooling  (None, 9, 9, 256)        0
 2D)

 conv2d_4 (Conv2D)           (None, 7, 7, 256)         590080

 flatten (Flatten)           (None, 12544)             0

 dense (Dense)               (None, 3)                 37635

=================================================================
Total params: 1,016,131
Trainable params: 1,016,131
Non-trainable params: 0
```

Model summary

Entrenamiento del modelo 2

Algoritmo de optimización: **Root mean square propagation** (RMSprop)

Función de pérdida: **Sparse Categorical Crossentropy**

Métrica: **Accuracy**

Entrenamos el modelo con 50 épocas

```
model.compile(optimizer="rmsprop",
    loss="sparse_categorical_crossentropy",
    metrics=["accuracy"])
```

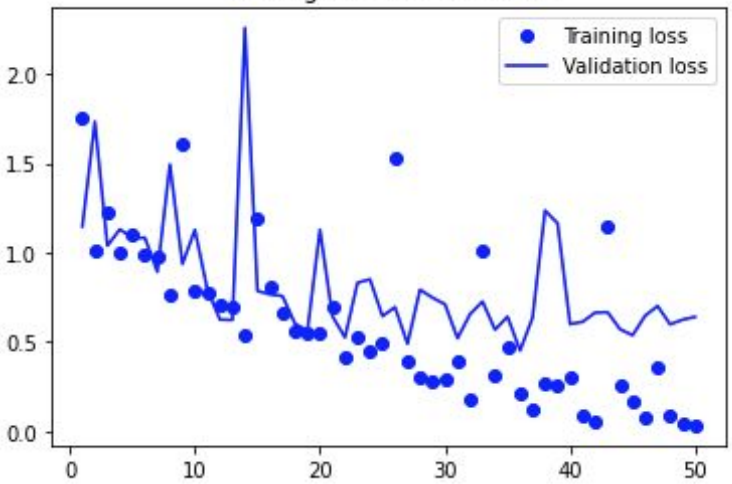
Figura 6. Compilación del modelo

```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch.keras",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
    train_dataset,
    epochs=50,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

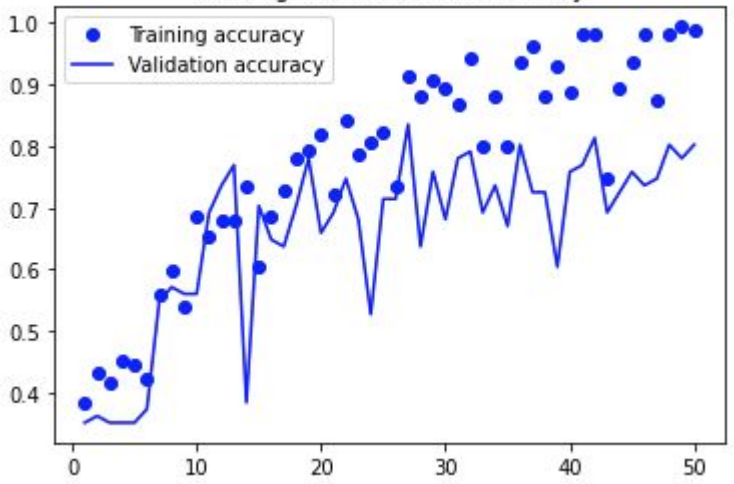
Callbacks y entrenamiento.

Resultados del modelo 2

Gráfica 1. Training and validation loss



Gráfica 2. Training and validation accuracy



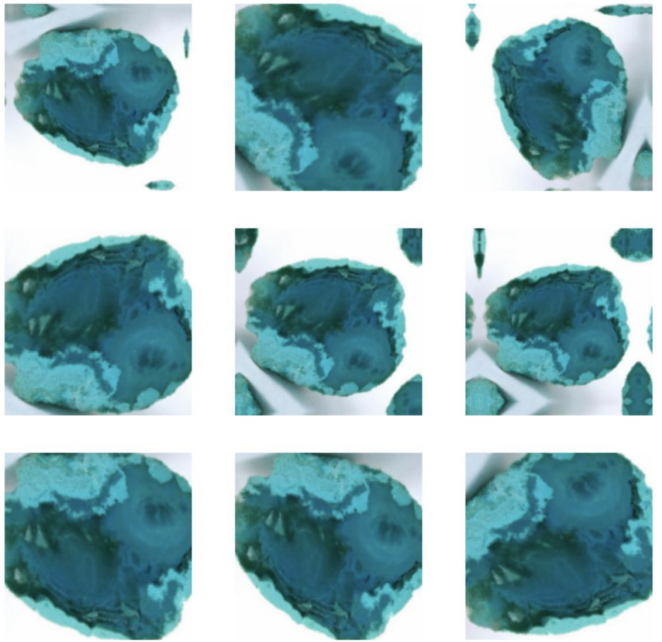

```
loss: 0.3997 - accuracy: 0.8311
```
(test)

Observamos una separación entre la precisión y la pérdida de los conjuntos train y validation a partir de la época 30. Overfitting.

Modelo 2.1: Data augmentation y dropout

```
data_augmentation = keras.Sequential(
    [
        layers.RandomFlip("vertical"),
        layers.RandomRotation(0.25),
        layers.RandomZoom(0.3),
    ]
)
```

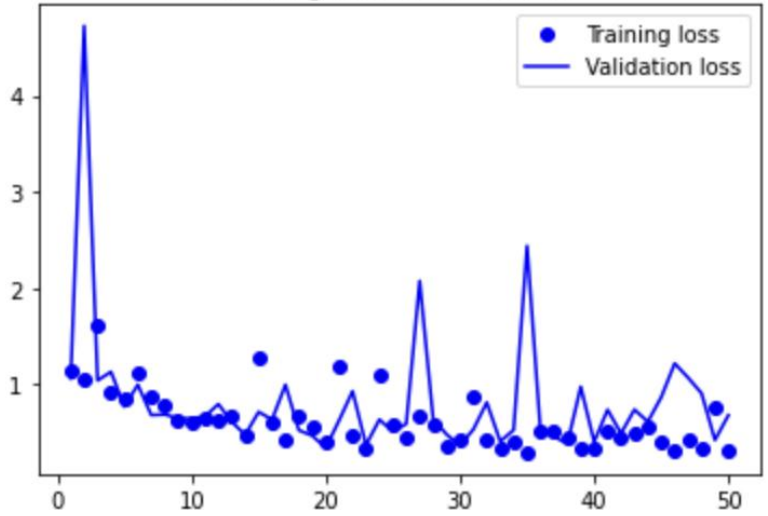
Agregamos dropout de 0.5



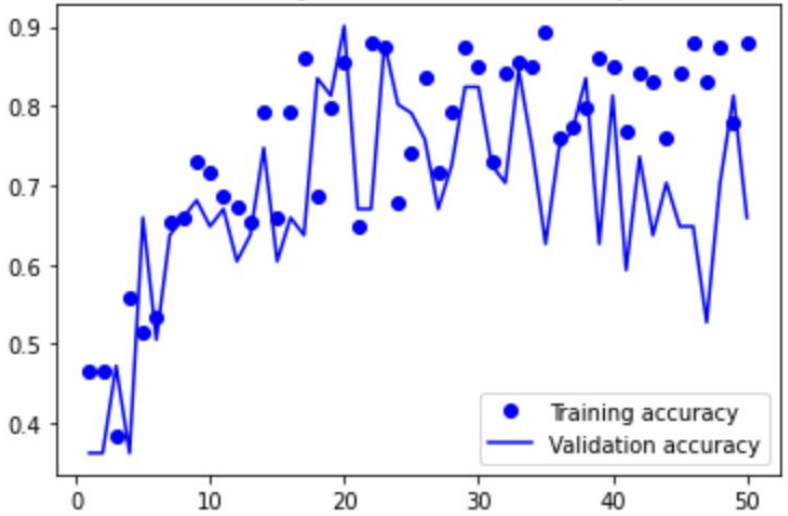
Muestras de una imagen rotada y aumentada.

Resultado modelo 2.1

Gráfica 3. Training and validation loss



Gráfica 4. Training and validation accuracy



`loss: 0.3156 – accuracy: 0.8919` (test)

La precisión es bastante buena, podría mejorar un poco al usar *early stopping*.

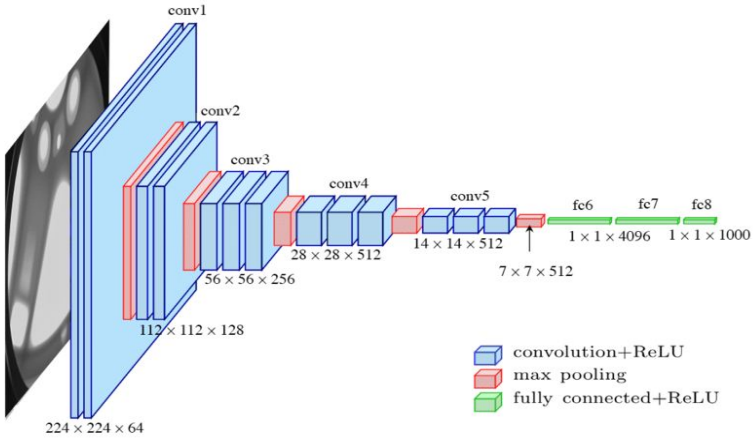
Modelo pre entrenado VGG16

```
conv_base  = keras.applications.vgg16.VGG16(
    weights="imagenet",
    include_top=False)
conv_base.trainable = False
```

Figura 9. Cargando la red VGG16 de Keras

- 13 capas de convolución y 3 capas densas.

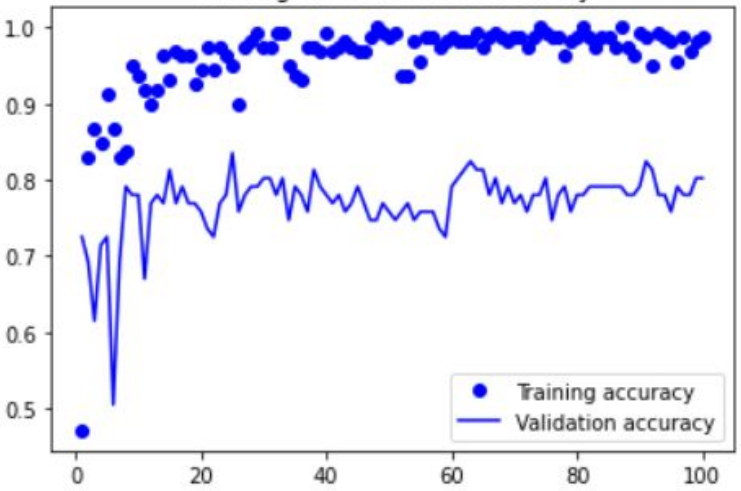
La red VGG16 fue entrenada para resolver el problema de clasificación de 1000 clases de animales en ImageNet, debe en sus pesos codificar información para extraer rasgos de las distintas clases representadas en más de 1.4 millones de fotografías.



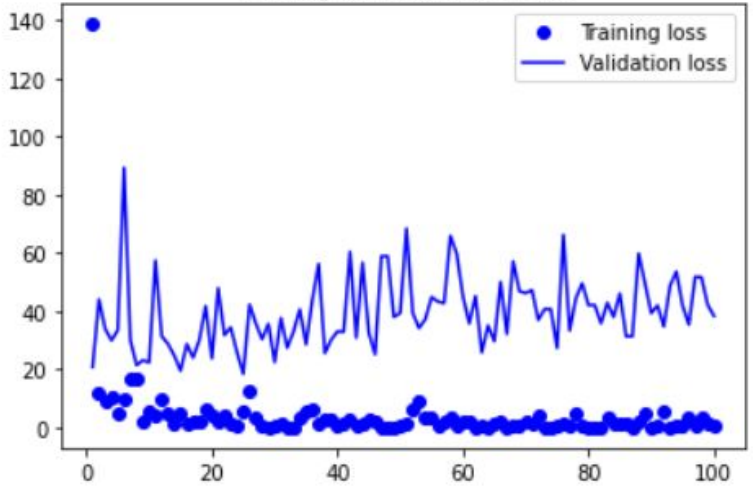
Arquitectura del modelo VGG16

Resultado modelo VGG16

Gráfica 5. Training and validation accuracy



Gráfica 6. Training and validation loss



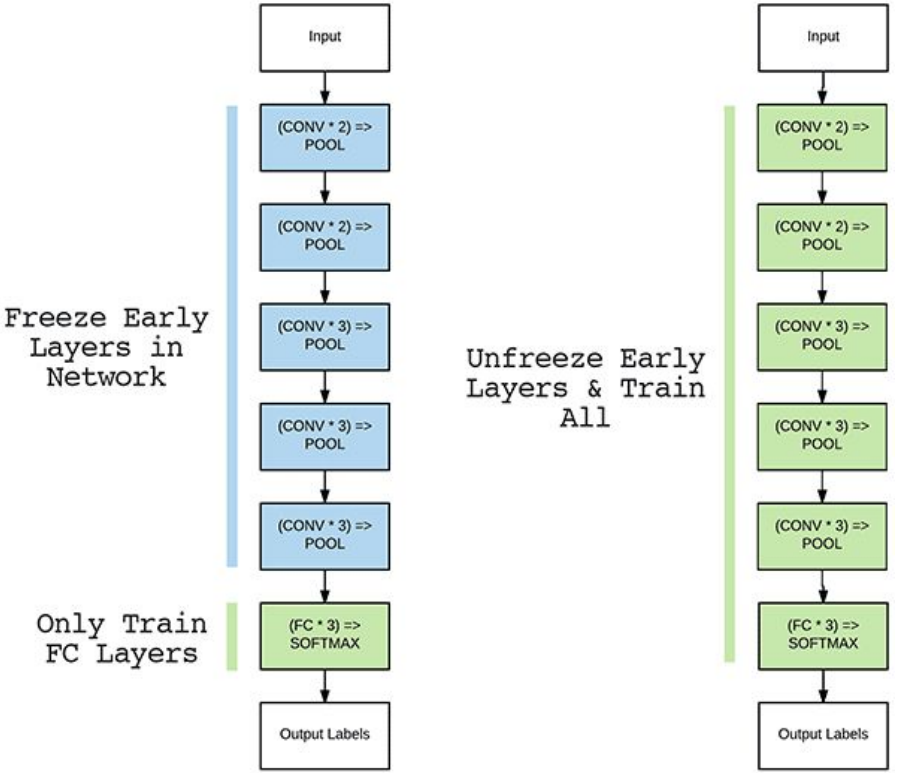
`loss: 32.5925 - accuracy: 0.8446` (test)

La función de pérdida es grande en ambos casos, y la precisión de buena. Lo cual puede significar pocos errores grandes.

Modelo VGG16 con Fine-tuning

```
conv_base.trainable = True
for layer in conv_base.layers[:-4]:
    layer.trainable = False
```

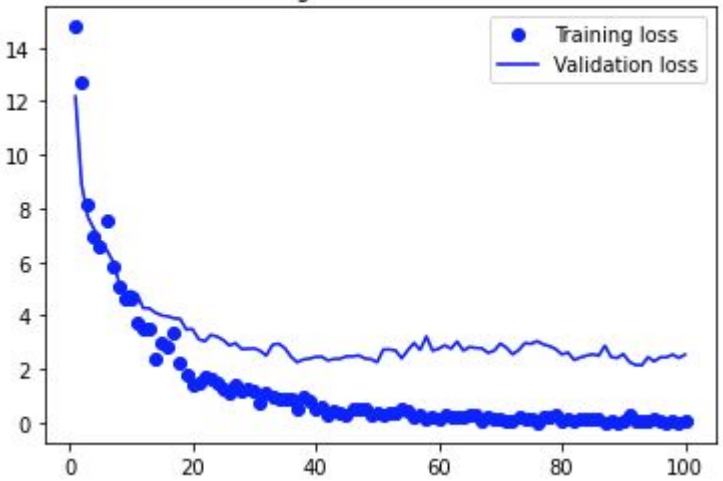
Figura 11. Declaración del Fine tuning



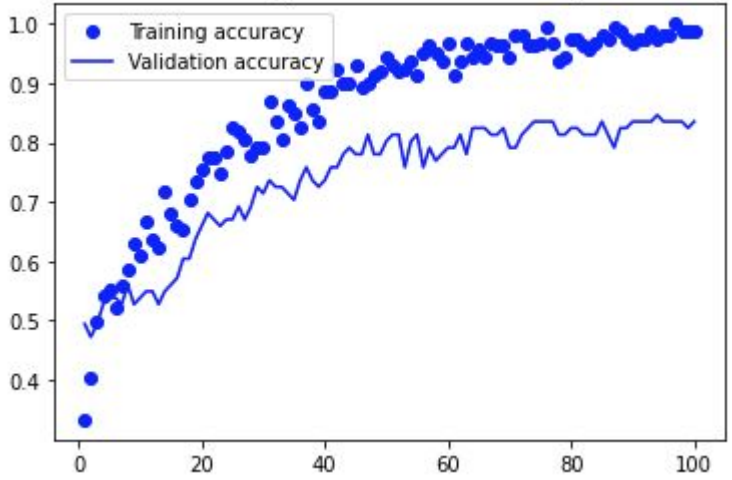
Funcionamiento del Fine tuning

Resultados del modelo VGG16

Gráfica 7. Training and validation loss



Gráfica 8. Training and validation accuracy




```
loss: 0.5826 - accuracy: 0.9054
```

La precisión con el uso de fine tuning es buena, supera a la precisión mínima con VGG16 data augmentation y la función de pérdida es mucho menor.

Resultados

Los resultados para ambos modelos son:

- En el entrenamiento del modelo 1 se obtuvo un precisión mayor a 82 %, pero al realizar un la validación cruzada con 5 splits se obtiene 68 % , es decir, la selección de la partición afecta el resultado.
- Se obseva en el mapa de características y de calor que el modelo 1 en la ultima capa convolucional aprende a identificar ciertas textura, combinación de color y regiones.
- En el modelo 2 la precisión mejoró de 83.1% a 89.1% al agregar *dropout* y *data augmentation*, además la función de pérdida se redujo de 0.399 a 0.315.
- En el modelo pre entrenado VGG16 con 100 épocas se obtuvo lo siguiente: con *data augmentation* una pérdida grande y precisión de 84.46%, mientras que con *fine tuning* una pérdida pequeña y precisión de 90.54%.

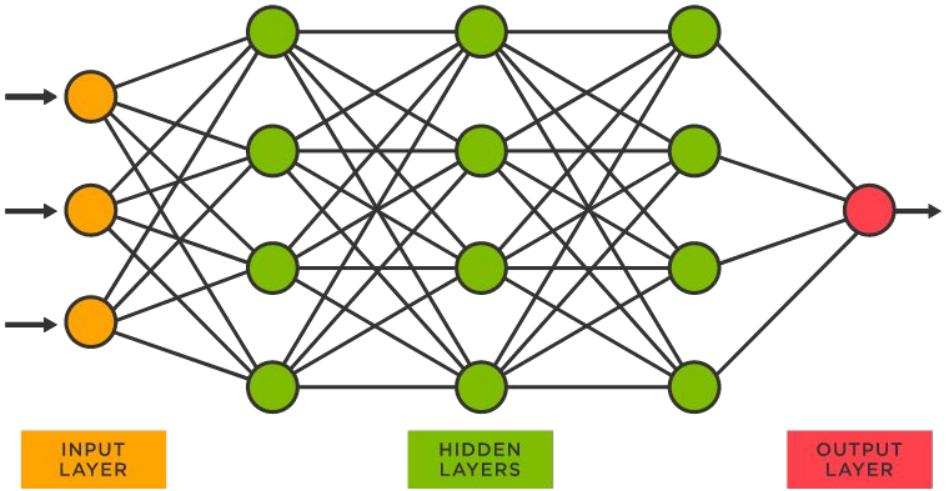
Conclusiones

El uso de capas convolucionales, *Maxpooling* y capas densas resulta un método exitoso para la clasificación de imágenes en distintas categorías.

El dataset consta de aproximadamente 400 imágenes, así que fue necesario la aplicación de K-fold validation para verificar la independencia de la partición de los datos. Del mismo modo recursos como *data augmentation* y *dropout* fueron cruciales para mejorar la precisión de los modelos (recomendable con datasets reducidos).

Métodos convolucionales aún más sofisticados como las redes preentrenadas (por ejemplo la VGG16), demuestran buenos resultados al ser combinados con *data augmentation*, sin embargo, los mejores resultados se alcanzaron al agregar *fine tuning*.

POR SU ATENCIÓN



GRACIAS