Tarea 6 - Redes Neuronales Convolucionales (CNN)

Alumno: Irving Daniel Estrada López

Matrícula: 1739907

Introducción

Según el Centro de Internet (IDC), el volumen total de datos globales alcanzó los 42ZB en 2020. Más del 70% de la información se transmite como fotos o videos. Para extraer información útil de estas imágenes y datos de video, apareció la visión computacional como algo necesario en estos tiempos. Como un importante componente de investigación el aprendizaje automático y el análisis de visión computacional, la clasificación de imágenes es una base teórica importante y un soporte técnico para avanzar en el desarrollo de la IA. La clasificación de imágenes comenzó a fines de 1950 y ha sido ampliamente utilizada en varios campos. La clasificación de imágenes es una de las tareas más conocidas en visión computacional y es un proceso complejo que puede verse afectado por muchos factores. Permite la clasificación de una imagen dada como perteneciente a un conjunto de categorías etiquetadas y predefinidas. En este notebook nos enfocaremos en el estudio de las Redes Neuronales Convolucionales (CNN) como modelo de clasificación supervisado para clasificar imágenes de felinos, comenzando entendiendo que es un Red Neuronal Artificial y sus componentes principales.

Redes Neuronales Artificiales (ANN)

Una red neuronal artificial es la parte de un sistema informático diseñado para simular que el cerebro humano analiza y procesa la información. Resuelve problemas que pueden ser difíciles para los estándares humanos o estadísticos. Las ANN tienen capacidades de autoaprendizaje que les permiten lograr mejores resultados a medida que hay más datos disponibles.

Capas

Estas consisten en tres capas: capa de entrada, capa oculta y capa de salida.

- **Capa de entrada:** La capa de entrada es donde alimentamos la entrada a la red. El número de neuronas en la capa de entrada es el número de entradas que alimentamos a la red. Cada entrada tendrá alguna influencia en la predicción de la salida. Sin embargo, no se realiza ningún cálculo en la capa de entrada; solo se usa para pasar información del mundo exterior a la red.

- **Capa oculta:** Cualquier capa entre la capa de entrada y la capa de salida se denomina capa oculta. Procesa la entrada recibida de la capa de entrada. La capa oculta es responsable de derivar relaciones complejas entre la entrada y la salida. Es decir, la capa oculta identifica el patrón en el conjunto de datos. Es principalmente responsable de aprender la representación de datos y de extraer las características.

- **Capa de salida:** Después de procesar la entrada, la capa oculta envía su resultado a la capa de salida. Como sugiere el nombre, la capa de salida emite la salida. El número de neuronas en la capa de salida se basa en el tipo de problema que queremos que resuelva nuestra red. Si es una clasificación binaria, entonces el número de neuronas en la capa de salida es el que nos dice a qué clase pertenece la entrada. Si se trata de una clasificación multiclase, por ejemplo, con cinco clases, y si queremos obtener la probabilidad de cada clase como salida, entonces el número de neuronas en la capa de salida es cinco, cada una de las cuales emite la probabilidad. Si es un problema de regresión, entonces tenemos una neurona en la capa de salida.

Funciones de Activación

Los modelos modernos de redes neuronales utilizan funciones de activación no lineales. Permite que el modelo cree asignaciones complejas entre las entradas y salidas de la red, que son esenciales para aprender y modelar datos complejos, como imágenes, video, audio y conjuntos de datos no lineales o de gran dimensión. A continuación se presentan algunas funciones de activación más populares, por temas de espacio no se mostrarán sus gráficas:

- **Sigmoide:** La función sigmoide es una de las funciones de activación más utilizadas. Escala el valor entre 0 y 1.

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

- **Tanh:** La función tangente hiperbólica genera el valores entre -1 y 1.

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

- **ReLu:** La Rectified Linear Unit es otra de las funciones de activación más utilizadas. Da como resultado un valor de 0 a infinito. Es básicamente una función por partes.

$$f(x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases}$$

$$f(x) = max(0, x)$$

- **Leaky ReLu:** es una variante de la función ReLU que resuelve el problema de dying de la función ReLU. En lugar de convertir cada entrada negativa a cero, tiene una pequeña pendiente para un valor negativo.

$$f(x) = \begin{cases} \alpha x & x < 0 \\ x & x \geq 0 \end{cases}$$

- **ELU:** La unidad lineal exponencial, como Leaky ReLU, tiene una pendiente pequeña para valores negativos. Pero en lugar de tener una línea recta, tiene una curva logarítmica.

$$f(x) = \begin{cases} \alpha(e^x - 1) & x < 0 \\ x & x \geq 0 \end{cases}$$

- **Swish:** La función Swish es una función de activación recientemente introducida por Google. A diferencia de otras funciones de activación, que son monótonas, Swish es una función no monótona, lo que significa que no siempre es ni creciente ni decreciente. Proporciona un mejor rendimiento que ReLU.

$$f(x) = x\sigma(x)$$

- **Softmax:** La función softmax es básicamente la generalización de la función sigmoide. Por lo general, se aplica a la capa final de la red y al realizar tareas de clasificación de clases múltiples. Da las probabilidades de que cada clase sea de salida y, por lo tanto, la suma de los valores softmax siempre será igual a 1.

$$f(x_i) = \frac{e^{x^i}}{\sum_j e^{x^j}}$$

Propagaciones

- **Forward Propagation:** Es todo el proceso desde la capa de entrada hasta la capa

de salida se conoce como forward propagation. Por lo tanto, para predecir el valor de salida, las entradas se propagan desde la capa de entrada a la capa de salida. Durante esta propagación, se multiplican por sus respectivos pesos en cada capa y se les aplica una función de activación encima.

- **Función de Perdida:** nos dice qué tan bien está funcionando nuestra red neuronal. Hay muchas funciones de costo diferentes.
- **Backpropagation:** Es todo el proceso de retropropagación de la red desde la capa de salida a la capa de entrada y la actualización de los pesos de la red mediante el optimizador para minimizar la pérdida. Existen diferentes tipos de optimizadores como:

  - SGD
  - RMSprop
  - Adam
  - Adadelta
  - Adagrad
  - Adamax
  - Nadam
  - Ftrl

Ventajas

- Almacena información sobre toda la red.
- Puede funcionar con conocimientos incompletos.
- Puede tener una memoria distribuida.
- Puede hacer aprendizaje automático porque aprende eventos y puede tomar decisiones en base a eventos similares.
- Puede trabajar en paralelo: tiene una fuerza numérica que puede realizar más de una tarea al mismo tiempo.

Desventajas

- No existe una regla específica para determinar la estructura. Es necesario apoyarse en metodologías.

- Necesita una gran cantidad de datos.

Redes Neuronales Convolucionales (CNN)

Las redes neuronales convolucionales (CNN) son uno de los algoritmos de aprendizaje profundo más utilizados. Son ampliamente utilizados para tareas relacionadas con imágenes, como reconocimiento de imágenes, detección de objetos, segmentación de imágenes y más. Las aplicaciones de las CNN pueden ser, desde potenciar la visión en autos autónomos hasta el etiquetado automático de amigos en nuestras fotos de Facebook. Aunque las CNN se utilizan ampliamente para conjuntos de datos de imágenes, también se pueden aplicar a conjuntos de datos de texto.

Arquitectura

- **Capa de Entrada:** contiene un conjunto de píxeles que representan la imagen.
- **Capa de Convolución:** La capa convolucional es la primera y principal capa de la CNN. Es uno de los componentes básicos de una CNN y se utiliza para extraer características importantes de la imagen. La operación de convolución nos ayuda a comprender de qué se trata la imagen. Como sabemos, cada imagen de entrada está representada por una matriz de valores de píxeles. Además de la matriz de entrada, también tenemos otra matriz llamada matriz de filtro. La matriz de filtro también se conoce como kernel, del cual obtenemos un mapa de características. Algo a destacas es que el número de píxeles que deslizamos sobre la matriz de entrada por la matriz de filtro se llama stride. Con la operación de convolución, nos deslizamos sobre la matriz de entrada con una matriz de filtro. Pero en algunos casos, el filtro no se ajusta perfectamente a la matriz de entrada. En esos casos se utiliza lo que se conoce como padding, el cual podemos rellenar con ceros simulando que son pixeles faltantes o simplemente omitir esa sección de la imágen
- **Capa de pooling:** Para reducir las dimensiones de los mapas de características, realizamos una operación de agrupación (pooling). Esto reduce las dimensiones de los mapas de características y mantiene solo los detalles necesarios para que se pueda reducir la cantidad de cómputo. Hay diferentes tipos de operaciones de agrupación, incluida la max pooling, average pooling y sum pooling. En la agrupación máxima, nos deslizamos sobre el filtro en la matriz de entrada y simplemente tomamos el valor máximo de la ventana del filtro, cada pooling tiene una operación diferente. La operación de agrupación no cambiará la profundidad de los mapas de características; solo afectará la altura y el ancho.

- **Capa de fully connected:** Una CNN puede tener múltiples capas convolucionales y capas de agrupación. Sin embargo, estas capas solo extraerán características de la imagen de entrada y producirán el mapa de características; es decir, son solo los extractores de características. La capa final de una CNN es la capa de fully connected, estas capas calculan el peso total de la última capa de características. Todos los elementos de todas las características de la capa anterior se pueden utilizar en el cálculo de cada elemento de cada característica de salida, para llevar a cabo la clasifiación o el análisis.

Ventajas

- Simplifica significativamente el cálculo en el proceso de convolución sin perder la esencia de los datos.
- Es excelente en la clasificación de imágenes.
- Reparto de pesos.

Desventajas

- La debilidad de una CNN es la cantidad de datos que le proporcionamos. Si le proporcionamos pocos datos, hay que esperar que le vaya mal. La CNN contienen millones de parámetros y con un pequeño conjunto de datos, se encontrarán con un problema , ya que necesitan una gran cantidad de datos para tener un buen desempeño. Por lo tanto, al momento de proporcionar una gran cantidad de datos, la CNN es más fuertes y está más dispuesta a brindar un mejor rendimiento.
- Problemas de overfitting.
- Tiene un costo computacional muy elevado.

Código

A continuación, se presenta el código donde se prueba una CNN con distintos preprocesados. En la primera CNN se destaca el preprocesado de ajuste de tamaño, así como quitar las imágenes que no son RGB. En la segunda CNN además de lo anteriormente mencionado, se generan datos sintéticos en base a las imágenes que tenemos, las cuales son pocas. La generación de datos sintéticos consiste en hacer modificaciones de nuestras imágenes originales y generando nuevas imágenes con pequeños cambios, con la intención de que nuestra CNN tenga mejor desempeño.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import PIL
import os
```

In [2]:
```
os.chdir('/Users/irvingestrada/Documents/Maestría/9- Procesamiento y Clasifi
```

In [3]:
```
f = os.listdir()[1:]
f.sort()
```

In [4]:
```
data = []
target = []
# new_size = (224,224)
new_size = (224,224)

# iteration by folders
for folder in f:
    os.chdir(folder)
    for file in os.listdir():
        # opening and resizing the image
        img = PIL.Image.open(file)
        img_res = np.array(img.resize(new_size))

        # adding data and target to arrays
        data.append(img_res)
        target.append(folder)
    os.chdir('..')
```

In [5]:
```
try:
    data = np.array(data)
    target = np.array(target)
except:
    print('Problem with broadcast')
```

```
Problem with broadcast
```

```
/var/folders/41/vdjdsv_91l3fnv_9v2c830940000gn/T/ipykernel_5823/3198010212.p
y:2: VisibleDeprecationWarning: Creating an ndarray from ragged nested seque
nces (which is a list-or-tuple of lists-or-tuples-or ndarrays with different
lengths or shapes) is deprecated. If you meant to do this, you must specify
'dtype=object' when creating the ndarray.
  data = np.array(data)
```

In [6]:
```
wrong_imgs_idx = []

# checking if all pics are in RGB and adding incorrect indexes to list
for idx,img in enumerate(data):
    try:
        if img.shape[2] != 3:
            wrong_imgs_idx.append(idx)
    except:
        if len(img.shape) != 3:
            wrong_imgs_idx.append(idx)
```

In [7]:
```
a = 0
for idx in wrong_imgs_idx:
    del data[idx-a]
    del target[idx-a]
    a += 1
```

In [8]:
```
try:
    data = np.array(data)
    target = np.array(target)
except:
    print('Problem with broadcast')
```

In [9]:
```
from sklearn.preprocessing import LabelEncoder

lbl = LabelEncoder()
target_n = lbl.fit_transform(target)
```

In [10]:
```
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical

x_train, x_test, y_train, y_test = train_test_split(data,target_n,test_size=
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_siz

print('x_train shape:', x_train.shape)
print('x_test shape:', x_test.shape)
print('x_val shape:', x_val.shape)
print('y_train shape:', y_train.shape)
print('y_test shape:', y_test.shape)
print('y_val shape:', y_val.shape)
```

```
Init Plugin
Init Graph Optimizer
Init Kernel
x_train shape: (168, 224, 224, 3)
x_test shape: (25, 224, 224, 3)
x_val shape: (48, 224, 224, 3)
y_train shape: (168,)
y_test shape: (25,)
y_val shape: (48,)
```

In [11]:
```
x_train_n = x_train / 255
x_test_n = x_test / 255
y_train_cat = to_categorical(y_train,num_classes=len(set(y_train)))
y_test_cat = to_categorical(y_test,num_classes=len(set(y_test)))
```

In [12]:
```
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras import layers,optimizers
from tensorflow.keras.callbacks import TensorBoard,ModelCheckpoint,EarlyStop
from keras import initializers
```

In [13]:
```
x_train_n[0].shape
```

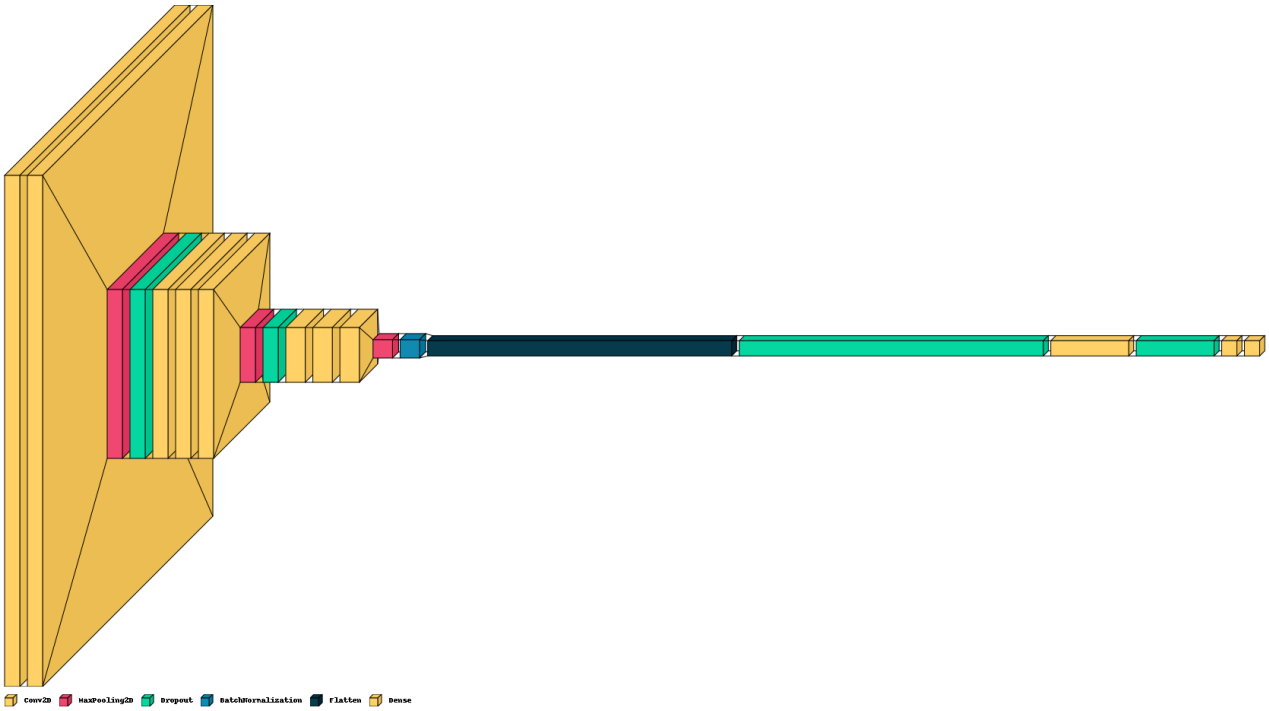
Out[13]:
```
(224, 224, 3)
```

In [14]:
```
def model_create():
    model = Sequential()
    model.add(layers.Conv2D(filters=64,kernel_size=(3,3),activation='relu',i
                                padding='same'))
    model.add(layers.Conv2D(filters=64,kernel_size=(3,3),activation='relu',
    model.add(layers.MaxPool2D(pool_size=(3,3)))
    model.add(layers.Dropout(0.15))

    model.add(layers.Conv2D(filters=128,kernel_size=(3,3),activation='relu',
    model.add(layers.Conv2D(filters=128,kernel_size=(3,3),activation='relu',
    model.add(layers.Conv2D(filters=128,kernel_size=(3,3),activation='relu',
    model.add(layers.MaxPool2D(pool_size=(3,3)))
    model.add(layers.Dropout(0.15))

    model.add(layers.Conv2D(filters=256,kernel_size=(3,3),activation='relu',
    model.add(layers.Conv2D(filters=256,kernel_size=(3,3),activation='relu',
    model.add(layers.Conv2D(filters=256,kernel_size=(3,3),activation='relu',
    model.add(layers.MaxPool2D(pool_size=(3,3)))
    model.add(layers.BatchNormalization())

    model.add(layers.Flatten())
    model.add(layers.Dropout(0.25))
    model.add(layers.Dense(units=1024,activation='sigmoid'))
    model.add(layers.Dropout(0.25))
    model.add(layers.Dense(units=128,activation='sigmoid'))
    model.add(layers.Dense(units=5,activation='softmax'))
    return model

model = model_create()
```

```
Metal device set to: Apple M1
2022-07-06 16:23:20.201286: I tensorflow/core/common_runtime/pluggable_devic
e/pluggable_device_factory.cc:305] Could not identify NUMA node of platform
GPU ID 0, defaulting to 0. Your kernel may not have been built with NUMA sup
port.
2022-07-06 16:23:20.201414: I tensorflow/core/common_runtime/pluggable_devic
e/pluggable_device_factory.cc:271] Created TensorFlow device (/job:localhost
/replica:0/task:0/device:GPU:0 with 0 MB memory) -> physical PluggableDevice
(device: 0, name: METAL, pci bus id: <undefined>)
```

In [15]:
```
# Visualizing our model (Hidden Input)
import visualkeras
visualkeras.layered_view(model, scale_xy=3, legend=True,)
```

Out[15]:



In [16]:
```
model.compile(optimizer='rmsprop',loss='categorical_crossentropy',metrics=['
```

In [17]:
```
es = EarlyStopping(monitor='val_loss',mode='min',verbose=1,patience=25,resto

history = model.fit(x_train_n, y_train_cat,epochs=200,validation_data=(x_tes
```

```
2022-07-06 16:23:20.447002: I tensorflow/compiler/mlir/mlir_graph_optimizati
on_pass.cc:176] None of the MLIR Optimization Passes are enabled (registered
2)
2022-07-06 16:23:20.448971: W tensorflow/core/platform/profile_utils/cpu_uti
ls.cc:128] Failed to get CPU frequency: 0 Hz
Epoch 1/200
2022-07-06 16:23:20.786438: I tensorflow/core/grappler/optimizers/custom_gra
ph_optimizer_registry.cc:112] Plugin optimizer for device_type GPU is enable
d.
6/6 [==============================] - ETA: 0s - loss: 2.1709 - accuracy: 0.
2143
2022-07-06 16:23:26.533616: I tensorflow/core/grappler/optimizers/custom_gra
ph_optimizer_registry.cc:112] Plugin optimizer for device_type GPU is enable
d.
6/6 [==============================] - 6s 957ms/step - loss: 2.1709 - accura
cy: 0.2143 - val_loss: 2.0852 - val_accuracy: 0.2000
Epoch 2/200
6/6 [==============================] - 5s 765ms/step - loss: 1.3380 - accura
cy: 0.3869 - val_loss: 2.1621 - val_accuracy: 0.2000
Epoch 3/200
6/6 [==============================] - 5s 807ms/step - loss: 1.2488 - accura
cy: 0.4524 - val_loss: 2.2478 - val_accuracy: 0.2000
Epoch 4/200
6/6 [==============================] - 5s 811ms/step - loss: 1.2539 - accura
```

```
cy: 0.4881 - val_loss: 2.4075 - val_accuracy: 0.2000
Epoch 5/200
6/6 [==============================] - 5s 827ms/step - loss: 1.1701 - accura
cy: 0.4821 - val_loss: 2.4059 - val_accuracy: 0.2000
Epoch 6/200
6/6 [==============================] - 5s 834ms/step - loss: 1.0393 - accura
cy: 0.5595 - val_loss: 2.2632 - val_accuracy: 0.2400
Epoch 7/200
6/6 [==============================] - 5s 803ms/step - loss: 0.9878 - accura
cy: 0.6131 - val_loss: 2.4669 - val_accuracy: 0.2000
Epoch 8/200
6/6 [==============================] - 5s 798ms/step - loss: 0.9282 - accura
cy: 0.6488 - val_loss: 2.2541 - val_accuracy: 0.1600
Epoch 9/200
6/6 [==============================] - 5s 778ms/step - loss: 1.0335 - accura
cy: 0.5774 - val_loss: 2.2153 - val_accuracy: 0.2800
Epoch 10/200
6/6 [==============================] - 5s 843ms/step - loss: 0.8910 - accura
cy: 0.6369 - val_loss: 2.3979 - val_accuracy: 0.3200
Epoch 11/200
6/6 [==============================] - 5s 806ms/step - loss: 0.7520 - accura
cy: 0.7262 - val_loss: 2.6242 - val_accuracy: 0.2800
Epoch 12/200
6/6 [==============================] - 5s 805ms/step - loss: 0.8080 - accura
cy: 0.6845 - val_loss: 2.4148 - val_accuracy: 0.2400
Epoch 13/200
6/6 [==============================] - 5s 842ms/step - loss: 0.6757 - accura
cy: 0.7500 - val_loss: 2.4396 - val_accuracy: 0.2800
Epoch 14/200
6/6 [==============================] - 5s 785ms/step - loss: 0.5556 - accura
cy: 0.8214 - val_loss: 2.5129 - val_accuracy: 0.2400
Epoch 15/200
6/6 [==============================] - 5s 795ms/step - loss: 0.4736 - accura
cy: 0.8631 - val_loss: 2.0720 - val_accuracy: 0.4000
Epoch 16/200
6/6 [==============================] - 5s 785ms/step - loss: 0.4813 - accura
cy: 0.8333 - val_loss: 1.8647 - val_accuracy: 0.4400
Epoch 17/200
6/6 [==============================] - 5s 804ms/step - loss: 0.3921 - accura
cy: 0.9048 - val_loss: 2.5854 - val_accuracy: 0.1600
Epoch 18/200
6/6 [==============================] - 5s 809ms/step - loss: 0.3376 - accura
cy: 0.8929 - val_loss: 2.5502 - val_accuracy: 0.3600
Epoch 19/200
6/6 [==============================] - 5s 808ms/step - loss: 0.2517 - accura
cy: 0.9345 - val_loss: 2.7880 - val_accuracy: 0.2800
Epoch 20/200
6/6 [==============================] - 5s 797ms/step - loss: 0.6273 - accura
cy: 0.7798 - val_loss: 2.6185 - val_accuracy: 0.2800
Epoch 21/200
6/6 [==============================] - 5s 809ms/step - loss: 0.2101 - accura
cy: 0.9643 - val_loss: 2.7806 - val_accuracy: 0.2800
Epoch 22/200
```

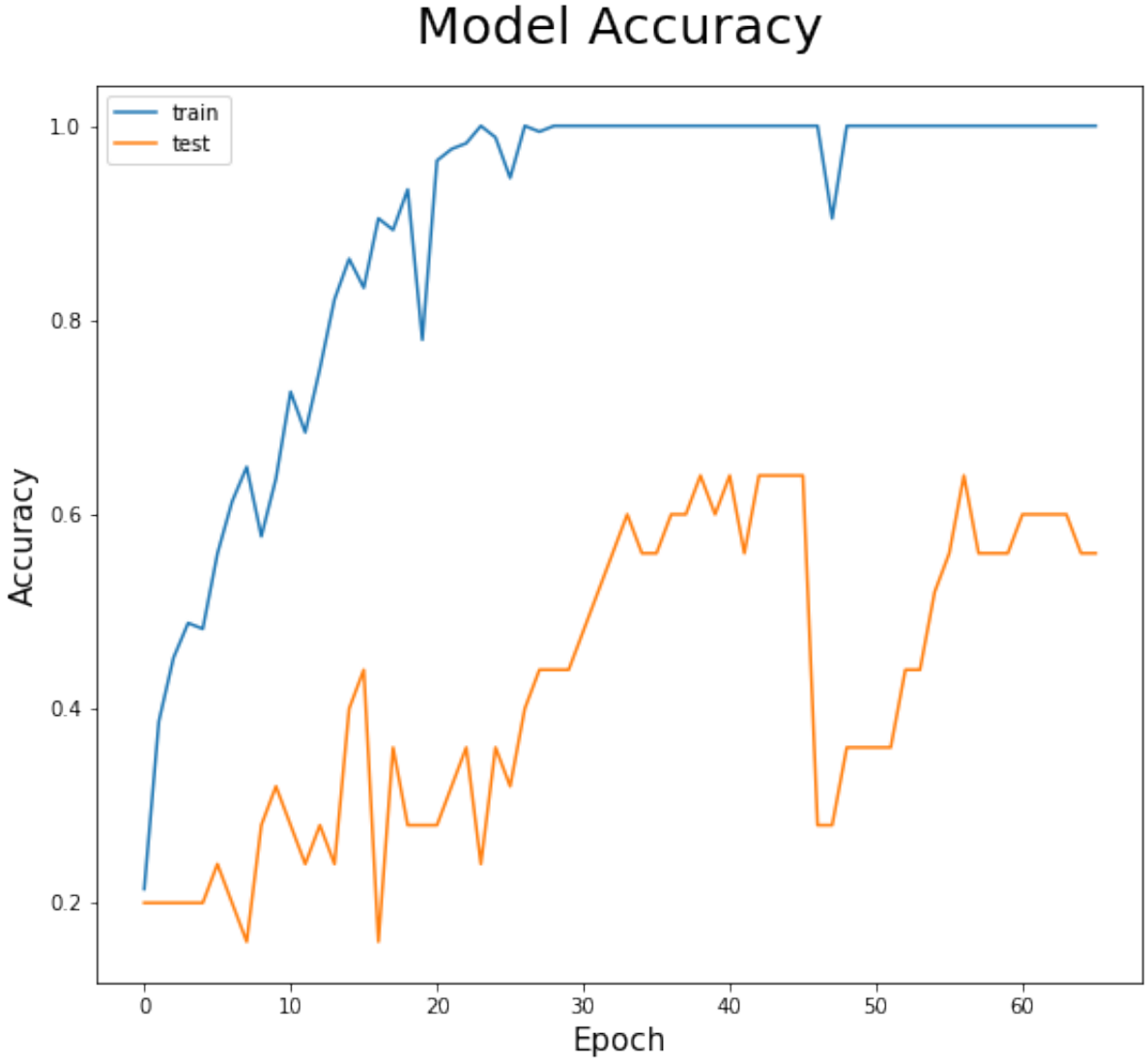
```
6/6 [==============================] - 5s 775ms/step - loss: 0.1244 - accura
cy: 0.9762 - val_loss: 2.3442 - val_accuracy: 0.3200
Epoch 23/200
6/6 [==============================] - 5s 816ms/step - loss: 0.0938 - accura
cy: 0.9821 - val_loss: 2.5609 - val_accuracy: 0.3600
Epoch 24/200
6/6 [==============================] - 5s 767ms/step - loss: 0.0635 - accura
cy: 1.0000 - val_loss: 2.8348 - val_accuracy: 0.2400
Epoch 25/200
6/6 [==============================] - 5s 819ms/step - loss: 0.0771 - accura
cy: 0.9881 - val_loss: 3.1385 - val_accuracy: 0.3600
Epoch 26/200
6/6 [==============================] - 5s 782ms/step - loss: 0.1471 - accura
cy: 0.9464 - val_loss: 2.5829 - val_accuracy: 0.3200
Epoch 27/200
6/6 [==============================] - 5s 797ms/step - loss: 0.0463 - accura
cy: 1.0000 - val_loss: 2.5503 - val_accuracy: 0.4000
Epoch 28/200
6/6 [==============================] - 5s 772ms/step - loss: 0.0518 - accura
cy: 0.9940 - val_loss: 2.3783 - val_accuracy: 0.4400
Epoch 29/200
6/6 [==============================] - 5s 781ms/step - loss: 0.0239 - accura
cy: 1.0000 - val_loss: 2.6402 - val_accuracy: 0.4400
Epoch 30/200
6/6 [==============================] - 5s 761ms/step - loss: 0.0159 - accura
cy: 1.0000 - val_loss: 2.2944 - val_accuracy: 0.4400
Epoch 31/200
6/6 [==============================] - 5s 754ms/step - loss: 0.0130 - accura
cy: 1.0000 - val_loss: 2.4512 - val_accuracy: 0.4800
Epoch 32/200
6/6 [==============================] - 5s 802ms/step - loss: 0.0108 - accura
cy: 1.0000 - val_loss: 2.3466 - val_accuracy: 0.5200
Epoch 33/200
6/6 [==============================] - 5s 778ms/step - loss: 0.0086 - accura
cy: 1.0000 - val_loss: 2.2865 - val_accuracy: 0.5600
Epoch 34/200
6/6 [==============================] - 5s 768ms/step - loss: 0.0060 - accura
cy: 1.0000 - val_loss: 2.1043 - val_accuracy: 0.6000
Epoch 35/200
6/6 [==============================] - 5s 800ms/step - loss: 0.0045 - accura
cy: 1.0000 - val_loss: 1.9700 - val_accuracy: 0.5600
Epoch 36/200
6/6 [==============================] - 5s 798ms/step - loss: 0.0035 - accura
cy: 1.0000 - val_loss: 1.9118 - val_accuracy: 0.5600
Epoch 37/200
6/6 [==============================] - 5s 814ms/step - loss: 0.0031 - accura
cy: 1.0000 - val_loss: 2.1366 - val_accuracy: 0.6000
Epoch 38/200
6/6 [==============================] - 5s 805ms/step - loss: 0.0026 - accura
cy: 1.0000 - val_loss: 2.0399 - val_accuracy: 0.6000
Epoch 39/200
6/6 [==============================] - 5s 839ms/step - loss: 0.0023 - accura
cy: 1.0000 - val_loss: 1.9826 - val_accuracy: 0.6400
```

```
Epoch 40/200
6/6 [==============================] - 5s 907ms/step - loss: 0.0016 - accura
cy: 1.0000 - val_loss: 1.8502 - val_accuracy: 0.6000
Epoch 41/200
6/6 [==============================] - 5s 812ms/step - loss: 0.0011 - accura
cy: 1.0000 - val_loss: 1.7706 - val_accuracy: 0.6400
Epoch 42/200
6/6 [==============================] - 5s 798ms/step - loss: 8.6313e-04 - ac
curacy: 1.0000 - val_loss: 1.8351 - val_accuracy: 0.5600
Epoch 43/200
6/6 [==============================] - 5s 886ms/step - loss: 7.5713e-04 - ac
curacy: 1.0000 - val_loss: 2.0523 - val_accuracy: 0.6400
Epoch 44/200
6/6 [==============================] - 5s 774ms/step - loss: 5.8911e-04 - ac
curacy: 1.0000 - val_loss: 2.3317 - val_accuracy: 0.6400
Epoch 45/200
6/6 [==============================] - 5s 809ms/step - loss: 4.4663e-04 - ac
curacy: 1.0000 - val_loss: 2.5331 - val_accuracy: 0.6400
Epoch 46/200
6/6 [==============================] - 5s 781ms/step - loss: 3.4570e-04 - ac
curacy: 1.0000 - val_loss: 2.1611 - val_accuracy: 0.6400
Epoch 47/200
6/6 [==============================] - 5s 755ms/step - loss: 4.3148e-04 - ac
curacy: 1.0000 - val_loss: 3.8050 - val_accuracy: 0.2800
Epoch 48/200
6/6 [==============================] - 5s 761ms/step - loss: 0.3671 - accura
cy: 0.9048 - val_loss: 3.7801 - val_accuracy: 0.2800
Epoch 49/200
6/6 [==============================] - 5s 778ms/step - loss: 0.0023 - accura
cy: 1.0000 - val_loss: 3.6936 - val_accuracy: 0.3600
Epoch 50/200
6/6 [==============================] - 5s 758ms/step - loss: 0.0018 - accura
cy: 1.0000 - val_loss: 3.5256 - val_accuracy: 0.3600
Epoch 51/200
6/6 [==============================] - 5s 762ms/step - loss: 0.0011 - accura
cy: 1.0000 - val_loss: 3.4247 - val_accuracy: 0.3600
Epoch 52/200
6/6 [==============================] - 5s 762ms/step - loss: 8.7457e-04 - ac
curacy: 1.0000 - val_loss: 3.2752 - val_accuracy: 0.3600
Epoch 53/200
6/6 [==============================] - 5s 801ms/step - loss: 9.8593e-04 - ac
curacy: 1.0000 - val_loss: 2.8732 - val_accuracy: 0.4400
Epoch 54/200
6/6 [==============================] - 5s 758ms/step - loss: 5.2479e-04 - ac
curacy: 1.0000 - val_loss: 2.8671 - val_accuracy: 0.4400
Epoch 55/200
6/6 [==============================] - 5s 794ms/step - loss: 4.7262e-04 - ac
curacy: 1.0000 - val_loss: 2.7908 - val_accuracy: 0.5200
Epoch 56/200
6/6 [==============================] - 5s 779ms/step - loss: 4.0569e-04 - ac
curacy: 1.0000 - val_loss: 2.7275 - val_accuracy: 0.5600
Epoch 57/200
6/6 [==============================] - 5s 772ms/step - loss: 3.5709e-04 - ac
```

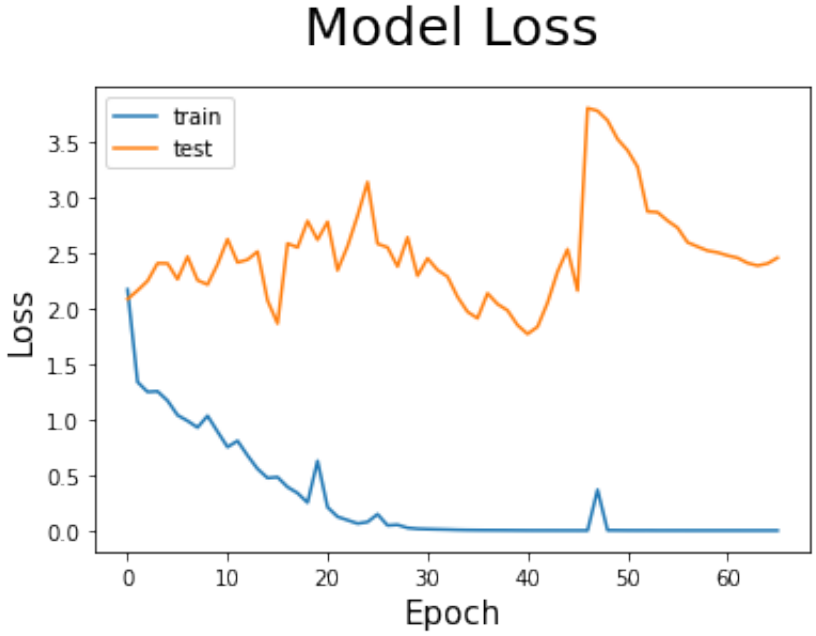
```
curacy: 1.0000 - val_loss: 2.5958 - val_accuracy: 0.6400
Epoch 58/200
6/6 [==============================] - 5s 791ms/step - loss: 3.2437e-04 - ac
curacy: 1.0000 - val_loss: 2.5571 - val_accuracy: 0.5600
Epoch 59/200
6/6 [==============================] - 5s 803ms/step - loss: 2.7748e-04 - ac
curacy: 1.0000 - val_loss: 2.5207 - val_accuracy: 0.5600
Epoch 60/200
6/6 [==============================] - 5s 795ms/step - loss: 2.5501e-04 - ac
curacy: 1.0000 - val_loss: 2.5043 - val_accuracy: 0.5600
Epoch 61/200
6/6 [==============================] - 5s 805ms/step - loss: 2.4713e-04 - ac
curacy: 1.0000 - val_loss: 2.4771 - val_accuracy: 0.6000
Epoch 62/200
6/6 [==============================] - 5s 784ms/step - loss: 2.0151e-04 - ac
curacy: 1.0000 - val_loss: 2.4563 - val_accuracy: 0.6000
Epoch 63/200
6/6 [==============================] - 5s 874ms/step - loss: 1.9180e-04 - ac
curacy: 1.0000 - val_loss: 2.4095 - val_accuracy: 0.6000
Epoch 64/200
6/6 [==============================] - 5s 813ms/step - loss: 1.6900e-04 - ac
curacy: 1.0000 - val_loss: 2.3862 - val_accuracy: 0.6000
Epoch 65/200
6/6 [==============================] - 5s 846ms/step - loss: 1.4719e-04 - ac
curacy: 1.0000 - val_loss: 2.4048 - val_accuracy: 0.5600
Epoch 66/200
6/6 [==============================] - 5s 802ms/step - loss: 1.3051e-04 - ac
curacy: 1.0000 - val_loss: 2.4568 - val_accuracy: 0.5600
Restoring model weights from the end of the best epoch.
Epoch 00066: early stopping
```

In [18]:
```
# Plotting the Model Accuracy & Model Loss vs Epochs (Hidden Input)
plt.figure(figsize=[20,8])

# summarize history for accuracy
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy', size=25, pad=20)
plt.ylabel('Accuracy', size=15)
plt.xlabel('Epoch', size=15)
plt.legend(['train', 'test'], loc='upper left')
# summarize history for loss
```

Out[18]:
```
<matplotlib.legend.Legend at 0x11ee0ce80>
```



In [19]:
```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss', size=25, pad=20)
plt.ylabel('Loss', size=15)
plt.xlabel('Epoch', size=15)
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



In [20]:
```
y_pred = model.predict(x_test_n).argmax(1)
y_true = y_test
```

```
2022-07-06 16:28:42.218770: I tensorflow/core/grappler/optimizers/custom_gra
ph_optimizer_registry.cc:112] Plugin optimizer for device_type GPU is enable
d.
```

In [21]:
```
import seaborn as sns
from sklearn.metrics import accuracy_score,f1_score,confusion_matrix,classif
set(lbl.inverse_transform(y_test))
```

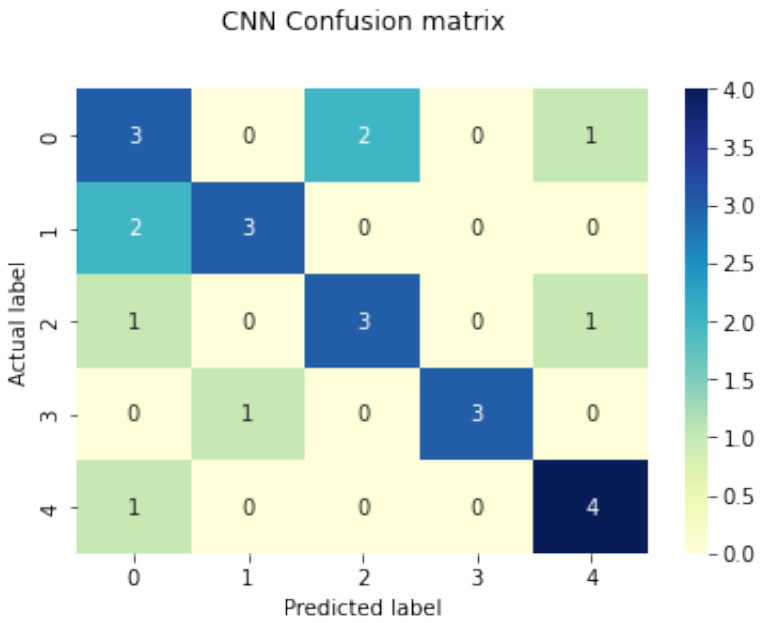
Out[21]:
```
{'Cheetah', 'Leopard', 'Lion', 'Puma', 'Tiger'}
```

In [22]:
```
cm = confusion_matrix(y_true,y_pred)
```

Como podemos ver en la matriz de confusión de nuestra CNN confunde con mayor frecuencia el leopardo con el chita seguramente por sus manchas y el chita con el león. También podemos identificar otros errores alrededor de la matriz, sin embargo, no pareciera que el modelo esté sesgado de algun modo.

In [23]:
```
p = sns.heatmap(pd.DataFrame(cm), annot=True, cmap="YlGnBu" ,fmt='g')
plt.title('CNN Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

Out[23]:
```
Text(0.5, 15.0, 'Predicted label')
```



En el reporte de clasificación podemos destacar que lo mejor que identifica nuestro modelo es la clase 3 que es perteneciente al puma, este no es tan parecido en cierto grado a los demás,los cuales tienen características destacables, como manchas, rayas o melena. Tuvo problemas con las clases anteriormente mencionadas, debido a sus manchas en su piel, específicamente con el chita. El modelo alcanza un 64% de accuracy el cual es aceptable ya que está por encima de 50% que este representaría una moneda al aire.

In [24]:
```
print(classification_report(y_true,y_pred))
print(accuracy_score(y_true,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.43      0.50      0.46         6
           1       0.75      0.60      0.67         5
           2       0.60      0.60      0.60         5
           3       1.00      0.75      0.86         4
           4       0.67      0.80      0.73         5

    accuracy                           0.64        25
   macro avg       0.69      0.65      0.66        25
weighted avg       0.67      0.64      0.65        25

0.64
```

CNN Validation

In [25]:
```
y_pred = model.predict(x_val).argmax(1)
y_true = y_val
```

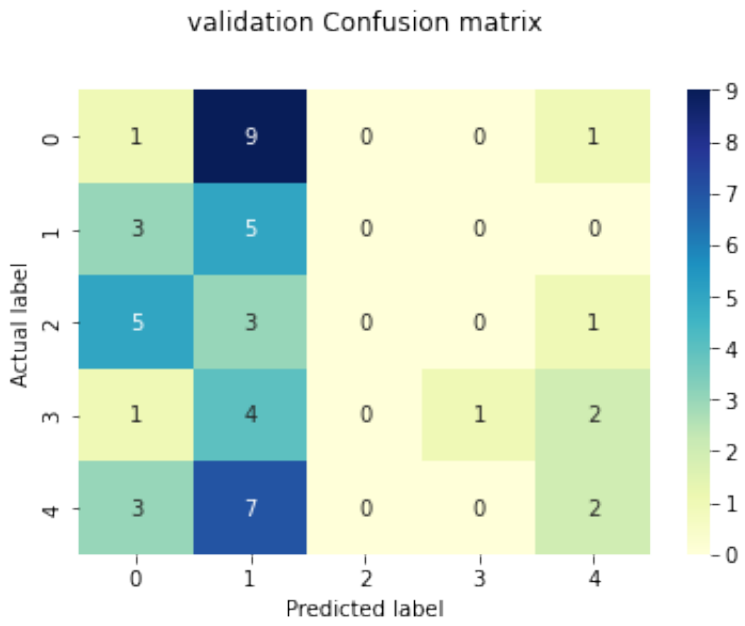
```
2022-07-06 16:28:42.630850: I tensorflow/core/grappler/optimizers/custom_gra
ph_optimizer_registry.cc:112] Plugin optimizer for device_type GPU is enable
d.
```

In [26]:
```
cm = confusion_matrix(y_true,y_pred)
```

La matriz de confusión de validación aparentemente pasó lo mismo que en el modelo anterior que confunde al chita con el leopardo, sin embargo, el modelo lo que mejor clasifica es el leopardo. Aparentemente nuestro modelo está sesgado en la parte de confundir al chita y al leopardo con los demás felinos, nos podemos dar cuenta viendo la primera y segunda columna.

In [27]:
```
p = sns.heatmap(pd.DataFrame(cm), annot=True, cmap="YlGnBu" ,fmt='g')
plt.title('validation Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

Out[27]:
```
Text(0.5, 15.0, 'Predicted label')
```



El reporte de los datos de validación nos indica el pésimo desempeño que se obtuvo clasificando los felinos de forma general, especialmente el león y el chita, es importante tomar en cuenta la cantidad de datos con la que se está trabajando ya que este es el punto débil del Deep Learning, el tener pocos datos.

In [28]:
```
print(classification_report(y_true,y_pred))
print(accuracy_score(y_true,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.08      0.09      0.08        11
           1       0.18      0.62      0.28         8
           2       0.00      0.00      0.00         9
           3       1.00      0.12      0.22         8
           4       0.33      0.17      0.22        12

    accuracy                           0.19        48
   macro avg       0.32      0.20      0.16        48
weighted avg       0.30      0.19      0.16        48

0.1875
/Users/irvingestrada/miniforge3/envs/tensorflow/lib/python3.9/site-packages/
sklearn/metrics/_classification.py:1327: UndefinedMetricWarning: Precision a
nd F-score are ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/irvingestrada/miniforge3/envs/tensorflow/lib/python3.9/site-packages/
sklearn/metrics/_classification.py:1327: UndefinedMetricWarning: Precision a
nd F-score are ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/irvingestrada/miniforge3/envs/tensorflow/lib/python3.9/site-packages/
sklearn/metrics/_classification.py:1327: UndefinedMetricWarning: Precision a
nd F-score are ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

CNN con aumentage data

In [29]:
```
os.chdir('..')
```

In [30]:
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=30,
    width_shift_range=0.15,
    height_shift_range=0.15,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    vertical_flip=True,
    fill_mode='nearest',
    validation_split = .25
)

valid_datagen = ImageDataGenerator(
    rescale=1./255,
    validation_split = .25
)

data_dir = 'Felidae'

train_data = train_datagen.flow_from_directory(data_dir, target_size = new_s
                                               subset = 'training')

val_data = valid_datagen.flow_from_directory(data_dir, target_size = new_siz
                                               subset = 'validation')
```

```
Found 183 images belonging to 5 classes.
Found 60 images belonging to 5 classes.
```

In [31]:
```
model2 = model_create()
```

In [32]:
```
model2.compile(optimizer='rmsprop',loss='categorical_crossentropy',metrics=[

history = model2.fit_generator(
    generator=train_data,
    validation_data=val_data,
    epochs=200,
    callbacks=es
)
```

```
/Users/irvingestrada/miniforge3/envs/tensorflow/lib/python3.9/site-packages/
tensorflow/python/keras/engine/training.py:1940: UserWarning: `Model.fit_gen
erator` is deprecated and will be removed in a future version. Please use `M
odel.fit`, which supports generators.
  warnings.warn('`Model.fit_generator` is deprecated and '
Epoch 1/200
2022-07-06 16:28:44.281050: I tensorflow/core/grappler/optimizers/custom_gra
ph_optimizer_registry.cc:112] Plugin optimizer for device_type GPU is enable
d.
```

```
6/6 [==============================] - ETA: 0s - loss: 1.9942 - accuracy: 0.
2459
2022-07-06 16:28:51.001329: I tensorflow/core/grappler/optimizers/custom_gra
ph_optimizer_registry.cc:112] Plugin optimizer for device_type GPU is enable
d.
6/6 [==============================] - 9s 1s/step - loss: 1.9942 - accuracy:
0.2459 - val_loss: 1.7040 - val_accuracy: 0.2000
Epoch 2/200
6/6 [==============================] - 8s 1s/step - loss: 1.6729 - accuracy:
0.2514 - val_loss: 1.6934 - val_accuracy: 0.2000
Epoch 3/200
6/6 [==============================] - 9s 2s/step - loss: 1.6973 - accuracy:
0.2350 - val_loss: 1.7159 - val_accuracy: 0.2000
Epoch 4/200
6/6 [==============================] - 10s 2s/step - loss: 1.5962 - accuracy
: 0.2732 - val_loss: 1.7479 - val_accuracy: 0.2000
Epoch 5/200
6/6 [==============================] - 11s 2s/step - loss: 1.5848 - accuracy
: 0.2787 - val_loss: 1.7362 - val_accuracy: 0.2000
Epoch 6/200
6/6 [==============================] - 12s 2s/step - loss: 1.5248 - accuracy
: 0.3169 - val_loss: 1.7599 - val_accuracy: 0.2000
Epoch 7/200
6/6 [==============================] - 13s 2s/step - loss: 1.4460 - accuracy
: 0.3552 - val_loss: 1.8289 - val_accuracy: 0.2000
Epoch 8/200
6/6 [==============================] - 16s 3s/step - loss: 1.4300 - accuracy
: 0.3716 - val_loss: 1.9035 - val_accuracy: 0.2000
Epoch 9/200
6/6 [==============================] - 17s 3s/step - loss: 1.4018 - accuracy
: 0.3388 - val_loss: 2.0209 - val_accuracy: 0.2000
Epoch 10/200
6/6 [==============================] - 17s 3s/step - loss: 1.3688 - accuracy
: 0.4481 - val_loss: 1.7630 - val_accuracy: 0.2667
Epoch 11/200
6/6 [==============================] - 15s 3s/step - loss: 1.2870 - accuracy
: 0.3880 - val_loss: 1.2963 - val_accuracy: 0.3833
Epoch 12/200
6/6 [==============================] - 14s 2s/step - loss: 1.4137 - accuracy
: 0.4153 - val_loss: 2.0451 - val_accuracy: 0.2000
Epoch 13/200
6/6 [==============================] - 13s 2s/step - loss: 1.3521 - accuracy
: 0.4262 - val_loss: 2.1437 - val_accuracy: 0.2000
Epoch 14/200
6/6 [==============================] - 12s 2s/step - loss: 1.2744 - accuracy
: 0.4426 - val_loss: 2.3453 - val_accuracy: 0.2000
Epoch 15/200
6/6 [==============================] - 11s 2s/step - loss: 1.2854 - accuracy
: 0.3607 - val_loss: 2.1355 - val_accuracy: 0.2000
Epoch 16/200
6/6 [==============================] - 11s 2s/step - loss: 1.3024 - accuracy
: 0.4044 - val_loss: 2.1079 - val_accuracy: 0.2333
Epoch 17/200
```

```
6/6 [==============================] - 10s 2s/step - loss: 1.2982 - accuracy
: 0.3989 - val_loss: 1.2787 - val_accuracy: 0.4167
Epoch 18/200
6/6 [==============================] - 10s 2s/step - loss: 1.3181 - accuracy
: 0.3607 - val_loss: 2.3409 - val_accuracy: 0.2000
Epoch 19/200
6/6 [==============================] - 10s 2s/step - loss: 1.2387 - accuracy
: 0.4044 - val_loss: 2.1232 - val_accuracy: 0.2333
Epoch 20/200
6/6 [==============================] - 10s 2s/step - loss: 1.2262 - accuracy
: 0.4536 - val_loss: 2.4870 - val_accuracy: 0.2000
Epoch 21/200
6/6 [==============================] - 10s 2s/step - loss: 1.2519 - accuracy
: 0.4754 - val_loss: 2.0685 - val_accuracy: 0.2667
Epoch 22/200
6/6 [==============================] - 9s 2s/step - loss: 1.1912 - accuracy:
0.4590 - val_loss: 2.4717 - val_accuracy: 0.2000
Epoch 23/200
6/6 [==============================] - 10s 2s/step - loss: 1.2802 - accuracy
: 0.4208 - val_loss: 2.5080 - val_accuracy: 0.2000
Epoch 24/200
6/6 [==============================] - 9s 2s/step - loss: 1.2361 - accuracy:
0.4153 - val_loss: 2.4756 - val_accuracy: 0.2000
Epoch 25/200
6/6 [==============================] - 9s 2s/step - loss: 1.2253 - accuracy:
0.4536 - val_loss: 1.2550 - val_accuracy: 0.3500
Epoch 26/200
6/6 [==============================] - 10s 2s/step - loss: 1.2562 - accuracy
: 0.4426 - val_loss: 2.5534 - val_accuracy: 0.2000
Epoch 27/200
6/6 [==============================] - 9s 1s/step - loss: 1.1748 - accuracy:
0.4262 - val_loss: 2.8362 - val_accuracy: 0.2000
Epoch 28/200
6/6 [==============================] - 9s 2s/step - loss: 1.2018 - accuracy:
0.4372 - val_loss: 2.9602 - val_accuracy: 0.2000
Epoch 29/200
6/6 [==============================] - 9s 1s/step - loss: 1.1827 - accuracy:
0.4481 - val_loss: 1.7692 - val_accuracy: 0.2833
Epoch 30/200
6/6 [==============================] - 9s 1s/step - loss: 1.1987 - accuracy:
0.4208 - val_loss: 2.5607 - val_accuracy: 0.2333
Epoch 31/200
6/6 [==============================] - 9s 1s/step - loss: 1.1811 - accuracy:
0.4481 - val_loss: 1.9539 - val_accuracy: 0.2500
Epoch 32/200
6/6 [==============================] - 9s 2s/step - loss: 1.1756 - accuracy:
0.4426 - val_loss: 1.2102 - val_accuracy: 0.5000
Epoch 33/200
6/6 [==============================] - 8s 1s/step - loss: 1.1739 - accuracy:
0.4754 - val_loss: 2.1949 - val_accuracy: 0.2167
Epoch 34/200
6/6 [==============================] - 9s 1s/step - loss: 1.1405 - accuracy:
0.4699 - val_loss: 1.5684 - val_accuracy: 0.3167
```

```
Epoch 35/200
6/6 [==============================] - 8s 1s/step - loss: 1.2419 - accuracy:
0.3934 - val_loss: 2.6223 - val_accuracy: 0.2000
Epoch 36/200
6/6 [==============================] - 8s 1s/step - loss: 1.1214 - accuracy:
0.4481 - val_loss: 1.1349 - val_accuracy: 0.4167
Epoch 37/200
6/6 [==============================] - 8s 1s/step - loss: 1.1178 - accuracy:
0.4809 - val_loss: 3.1311 - val_accuracy: 0.2000
Epoch 38/200
6/6 [==============================] - 8s 1s/step - loss: 1.1231 - accuracy:
0.4262 - val_loss: 1.9074 - val_accuracy: 0.2833
Epoch 39/200
6/6 [==============================] - 8s 1s/step - loss: 1.1106 - accuracy:
0.5301 - val_loss: 2.4341 - val_accuracy: 0.2167
Epoch 40/200
6/6 [==============================] - 8s 1s/step - loss: 1.2685 - accuracy:
0.3989 - val_loss: 3.0385 - val_accuracy: 0.2000
Epoch 41/200
6/6 [==============================] - 8s 1s/step - loss: 1.1593 - accuracy:
0.4590 - val_loss: 3.1102 - val_accuracy: 0.2000
Epoch 42/200
6/6 [==============================] - 8s 1s/step - loss: 1.1008 - accuracy:
0.5137 - val_loss: 2.5557 - val_accuracy: 0.2167
Epoch 43/200
6/6 [==============================] - 9s 1s/step - loss: 1.0330 - accuracy:
0.5191 - val_loss: 1.6723 - val_accuracy: 0.3500
Epoch 44/200
6/6 [==============================] - 8s 1s/step - loss: 1.0852 - accuracy:
0.4645 - val_loss: 0.9111 - val_accuracy: 0.5667
Epoch 45/200
6/6 [==============================] - 8s 1s/step - loss: 1.0588 - accuracy:
0.5519 - val_loss: 1.8204 - val_accuracy: 0.3500
Epoch 46/200
6/6 [==============================] - 8s 1s/step - loss: 1.0887 - accuracy:
0.5355 - val_loss: 1.9235 - val_accuracy: 0.2833
Epoch 47/200
6/6 [==============================] - 8s 1s/step - loss: 1.0621 - accuracy:
0.5082 - val_loss: 2.3125 - val_accuracy: 0.2667
Epoch 48/200
6/6 [==============================] - 8s 1s/step - loss: 1.0315 - accuracy:
0.5137 - val_loss: 1.0991 - val_accuracy: 0.5500
Epoch 49/200
6/6 [==============================] - 8s 1s/step - loss: 1.1065 - accuracy:
0.4973 - val_loss: 3.4817 - val_accuracy: 0.2000
Epoch 50/200
6/6 [==============================] - 8s 1s/step - loss: 1.1183 - accuracy:
0.4754 - val_loss: 3.5328 - val_accuracy: 0.2000
Epoch 51/200
6/6 [==============================] - 8s 1s/step - loss: 1.0728 - accuracy:
0.5191 - val_loss: 2.2441 - val_accuracy: 0.3167
Epoch 52/200
6/6 [==============================] - 8s 1s/step - loss: 1.1030 - accuracy:
```

```
0.4973 - val_loss: 1.2034 - val_accuracy: 0.4167
Epoch 53/200
6/6 [==============================] - 8s 1s/step - loss: 0.9892 - accuracy:
0.5519 - val_loss: 1.2709 - val_accuracy: 0.3667
Epoch 54/200
6/6 [==============================] - 8s 1s/step - loss: 0.9857 - accuracy:
0.5519 - val_loss: 1.0078 - val_accuracy: 0.4667
Epoch 55/200
6/6 [==============================] - 8s 1s/step - loss: 1.1081 - accuracy:
0.4809 - val_loss: 1.0801 - val_accuracy: 0.5000
Epoch 56/200
6/6 [==============================] - 8s 1s/step - loss: 1.0906 - accuracy:
0.4918 - val_loss: 1.9342 - val_accuracy: 0.3000
Epoch 57/200
6/6 [==============================] - 8s 1s/step - loss: 1.0325 - accuracy:
0.4918 - val_loss: 2.0018 - val_accuracy: 0.3333
Epoch 58/200
6/6 [==============================] - 8s 1s/step - loss: 1.0189 - accuracy:
0.5137 - val_loss: 1.4183 - val_accuracy: 0.3833
Epoch 59/200
6/6 [==============================] - 8s 1s/step - loss: 1.0184 - accuracy:
0.4863 - val_loss: 1.4608 - val_accuracy: 0.4333
Epoch 60/200
6/6 [==============================] - 8s 1s/step - loss: 1.0638 - accuracy:
0.4809 - val_loss: 3.4493 - val_accuracy: 0.2000
Epoch 61/200
6/6 [==============================] - 7s 1s/step - loss: 1.0049 - accuracy:
0.5246 - val_loss: 3.2860 - val_accuracy: 0.2000
Epoch 62/200
6/6 [==============================] - 8s 1s/step - loss: 1.0519 - accuracy:
0.5355 - val_loss: 1.3578 - val_accuracy: 0.4167
Epoch 63/200
6/6 [==============================] - 7s 1s/step - loss: 0.9373 - accuracy:
0.5519 - val_loss: 1.7200 - val_accuracy: 0.3167
Epoch 64/200
6/6 [==============================] - 8s 1s/step - loss: 0.9913 - accuracy:
0.5628 - val_loss: 1.3110 - val_accuracy: 0.3500
Epoch 65/200
6/6 [==============================] - 7s 1s/step - loss: 0.9505 - accuracy:
0.6011 - val_loss: 0.9883 - val_accuracy: 0.4667
Epoch 66/200
6/6 [==============================] - 7s 1s/step - loss: 1.0202 - accuracy:
0.5246 - val_loss: 0.9007 - val_accuracy: 0.5333
Epoch 67/200
6/6 [==============================] - 8s 1s/step - loss: 0.9653 - accuracy:
0.5683 - val_loss: 1.9068 - val_accuracy: 0.2833
Epoch 68/200
6/6 [==============================] - 8s 1s/step - loss: 1.0250 - accuracy:
0.4754 - val_loss: 1.1280 - val_accuracy: 0.4500
Epoch 69/200
6/6 [==============================] - 8s 1s/step - loss: 1.0345 - accuracy:
0.5082 - val_loss: 0.9521 - val_accuracy: 0.5333
Epoch 70/200
```

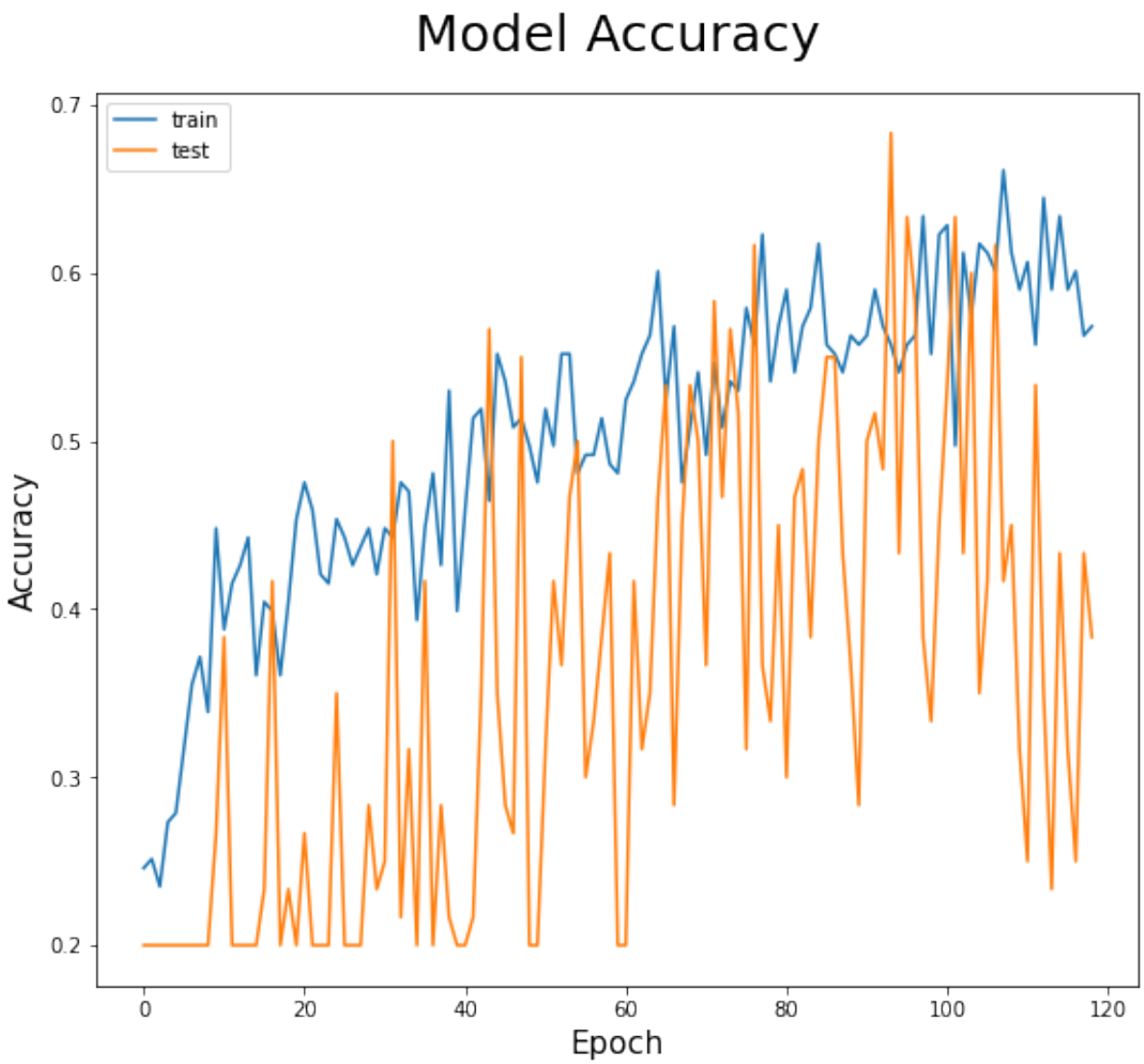
```
6/6 [==============================] - 8s 1s/step - loss: 0.9061 - accuracy:
0.5410 - val_loss: 0.9969 - val_accuracy: 0.5000
Epoch 71/200
6/6 [==============================] - 8s 1s/step - loss: 1.0424 - accuracy:
0.4918 - val_loss: 1.3569 - val_accuracy: 0.3667
Epoch 72/200
6/6 [==============================] - 7s 1s/step - loss: 0.9712 - accuracy:
0.5464 - val_loss: 0.9111 - val_accuracy: 0.5833
Epoch 73/200
6/6 [==============================] - 7s 1s/step - loss: 1.0069 - accuracy:
0.5082 - val_loss: 1.8778 - val_accuracy: 0.4667
Epoch 74/200
6/6 [==============================] - 7s 1s/step - loss: 1.0113 - accuracy:
0.5355 - val_loss: 1.0162 - val_accuracy: 0.5667
Epoch 75/200
6/6 [==============================] - 7s 1s/step - loss: 1.0106 - accuracy:
0.5301 - val_loss: 1.0611 - val_accuracy: 0.5167
Epoch 76/200
6/6 [==============================] - 7s 1s/step - loss: 0.9208 - accuracy:
0.5792 - val_loss: 2.3277 - val_accuracy: 0.3167
Epoch 77/200
6/6 [==============================] - 7s 1s/step - loss: 0.9455 - accuracy:
0.5574 - val_loss: 0.8715 - val_accuracy: 0.6167
Epoch 78/200
6/6 [==============================] - 7s 1s/step - loss: 0.9291 - accuracy:
0.6230 - val_loss: 1.9756 - val_accuracy: 0.3667
Epoch 79/200
6/6 [==============================] - 8s 1s/step - loss: 0.9448 - accuracy:
0.5355 - val_loss: 1.7134 - val_accuracy: 0.3333
Epoch 80/200
6/6 [==============================] - 7s 1s/step - loss: 0.9590 - accuracy:
0.5683 - val_loss: 1.4292 - val_accuracy: 0.4500
Epoch 81/200
6/6 [==============================] - 8s 1s/step - loss: 0.9945 - accuracy:
0.5902 - val_loss: 1.4903 - val_accuracy: 0.3000
Epoch 82/200
6/6 [==============================] - 8s 1s/step - loss: 1.0017 - accuracy:
0.5410 - val_loss: 1.1019 - val_accuracy: 0.4667
Epoch 83/200
6/6 [==============================] - 7s 1s/step - loss: 0.9086 - accuracy:
0.5683 - val_loss: 1.5588 - val_accuracy: 0.4833
Epoch 84/200
6/6 [==============================] - 7s 1s/step - loss: 0.9429 - accuracy:
0.5792 - val_loss: 1.7936 - val_accuracy: 0.3833
Epoch 85/200
6/6 [==============================] - 7s 1s/step - loss: 0.9081 - accuracy:
0.6175 - val_loss: 1.1585 - val_accuracy: 0.5000
Epoch 86/200
6/6 [==============================] - 7s 1s/step - loss: 0.9030 - accuracy:
0.5574 - val_loss: 1.0911 - val_accuracy: 0.5500
Epoch 87/200
6/6 [==============================] - 7s 1s/step - loss: 0.9169 - accuracy:
0.5519 - val_loss: 0.8879 - val_accuracy: 0.5500
```

```
Epoch 88/200
6/6 [==============================] - 7s 1s/step - loss: 0.9682 - accuracy:
0.5410 - val_loss: 1.6896 - val_accuracy: 0.4333
Epoch 89/200
6/6 [==============================] - 7s 1s/step - loss: 0.9225 - accuracy:
0.5628 - val_loss: 1.5772 - val_accuracy: 0.3667
Epoch 90/200
6/6 [==============================] - 7s 1s/step - loss: 0.9233 - accuracy:
0.5574 - val_loss: 3.0644 - val_accuracy: 0.2833
Epoch 91/200
6/6 [==============================] - 7s 1s/step - loss: 0.9098 - accuracy:
0.5628 - val_loss: 0.9314 - val_accuracy: 0.5000
Epoch 92/200
6/6 [==============================] - 7s 1s/step - loss: 0.9145 - accuracy:
0.5902 - val_loss: 1.3534 - val_accuracy: 0.5167
Epoch 93/200
6/6 [==============================] - 7s 1s/step - loss: 0.9050 - accuracy:
0.5683 - val_loss: 0.9078 - val_accuracy: 0.4833
Epoch 94/200
6/6 [==============================] - 7s 1s/step - loss: 0.9935 - accuracy:
0.5574 - val_loss: 0.8512 - val_accuracy: 0.6833
Epoch 95/200
6/6 [==============================] - 7s 1s/step - loss: 1.0280 - accuracy:
0.5410 - val_loss: 1.4386 - val_accuracy: 0.4333
Epoch 96/200
6/6 [==============================] - 8s 1s/step - loss: 0.9155 - accuracy:
0.5574 - val_loss: 0.9250 - val_accuracy: 0.6333
Epoch 97/200
6/6 [==============================] - 7s 1s/step - loss: 1.0176 - accuracy:
0.5628 - val_loss: 0.9279 - val_accuracy: 0.5833
Epoch 98/200
6/6 [==============================] - 8s 1s/step - loss: 0.8264 - accuracy:
0.6339 - val_loss: 1.9215 - val_accuracy: 0.3833
Epoch 99/200
6/6 [==============================] - 7s 1s/step - loss: 0.9878 - accuracy:
0.5519 - val_loss: 2.1428 - val_accuracy: 0.3333
Epoch 100/200
6/6 [==============================] - 7s 1s/step - loss: 0.8821 - accuracy:
0.6230 - val_loss: 1.8522 - val_accuracy: 0.4500
Epoch 101/200
6/6 [==============================] - 8s 1s/step - loss: 0.8960 - accuracy:
0.6284 - val_loss: 0.8563 - val_accuracy: 0.5333
Epoch 102/200
6/6 [==============================] - 7s 1s/step - loss: 1.0613 - accuracy:
0.4973 - val_loss: 0.8765 - val_accuracy: 0.6333
Epoch 103/200
6/6 [==============================] - 7s 1s/step - loss: 0.9281 - accuracy:
0.6120 - val_loss: 1.4968 - val_accuracy: 0.4333
Epoch 104/200
6/6 [==============================] - 7s 1s/step - loss: 0.9164 - accuracy:
0.5738 - val_loss: 0.9247 - val_accuracy: 0.6000
Epoch 105/200
6/6 [==============================] - 8s 1s/step - loss: 0.7998 - accuracy:
```

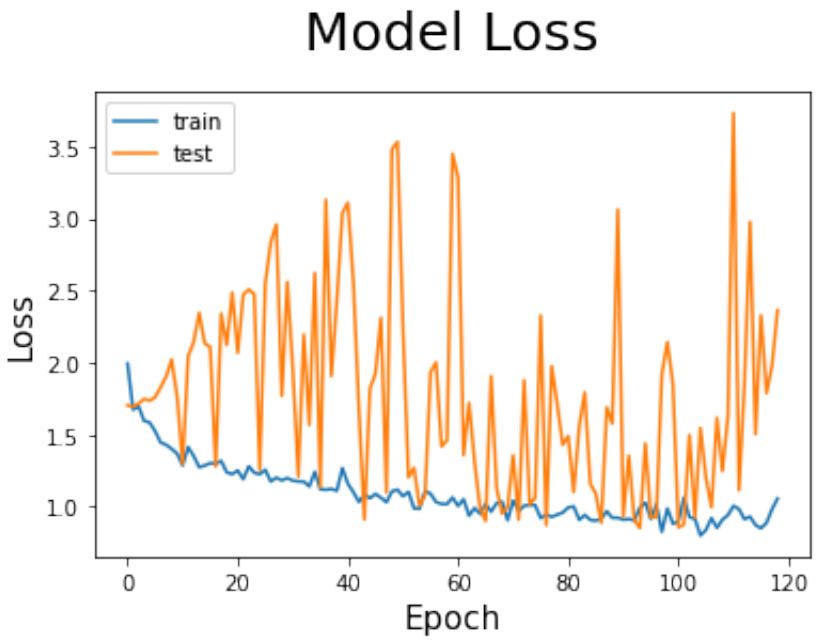
```
0.6175 - val_loss: 1.5456 - val_accuracy: 0.3500
Epoch 106/200
6/6 [==============================] - 7s 1s/step - loss: 0.8385 - accuracy:
0.6120 - val_loss: 1.2011 - val_accuracy: 0.4167
Epoch 107/200
6/6 [==============================] - 7s 1s/step - loss: 0.9209 - accuracy:
0.6011 - val_loss: 0.9964 - val_accuracy: 0.6167
Epoch 108/200
6/6 [==============================] - 7s 1s/step - loss: 0.8533 - accuracy:
0.6612 - val_loss: 1.6166 - val_accuracy: 0.4167
Epoch 109/200
6/6 [==============================] - 7s 1s/step - loss: 0.9103 - accuracy:
0.6120 - val_loss: 1.2475 - val_accuracy: 0.4500
Epoch 110/200
6/6 [==============================] - 7s 1s/step - loss: 0.9415 - accuracy:
0.5902 - val_loss: 1.6310 - val_accuracy: 0.3167
Epoch 111/200
6/6 [==============================] - 7s 1s/step - loss: 1.0054 - accuracy:
0.6066 - val_loss: 3.7331 - val_accuracy: 0.2500
Epoch 112/200
6/6 [==============================] - 7s 1s/step - loss: 0.9840 - accuracy:
0.5574 - val_loss: 1.1143 - val_accuracy: 0.5333
Epoch 113/200
6/6 [==============================] - 7s 1s/step - loss: 0.9133 - accuracy:
0.6448 - val_loss: 1.8612 - val_accuracy: 0.3500
Epoch 114/200
6/6 [==============================] - 7s 1s/step - loss: 0.9313 - accuracy:
0.5902 - val_loss: 2.9777 - val_accuracy: 0.2333
Epoch 115/200
6/6 [==============================] - 7s 1s/step - loss: 0.8731 - accuracy:
0.6339 - val_loss: 1.5033 - val_accuracy: 0.4333
Epoch 116/200
6/6 [==============================] - 7s 1s/step - loss: 0.8503 - accuracy:
0.5902 - val_loss: 2.3290 - val_accuracy: 0.3167
Epoch 117/200
6/6 [==============================] - 7s 1s/step - loss: 0.8881 - accuracy:
0.6011 - val_loss: 1.7887 - val_accuracy: 0.2500
Epoch 118/200
6/6 [==============================] - 7s 1s/step - loss: 0.9827 - accuracy:
0.5628 - val_loss: 1.9814 - val_accuracy: 0.4333
Epoch 119/200
6/6 [==============================] - 7s 1s/step - loss: 1.0562 - accuracy:
0.5683 - val_loss: 2.3648 - val_accuracy: 0.3833
Restoring model weights from the end of the best epoch.
Epoch 00119: early stopping
```

In [33]:
```
# Plotting the Model Accuracy & Model Loss vs Epochs (Hidden Input)
plt.figure(figsize=[20,8])

# summarize history for accuracy
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy', size=25, pad=20)
plt.ylabel('Accuracy', size=15)
plt.xlabel('Epoch', size=15)
plt.legend(['train', 'test'], loc='upper left')
# summarize history for loss
```

Out[33]:
```
<matplotlib.legend.Legend at 0x11ea5bfa0>
```



In [34]:
```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss', size=25, pad=20)
plt.ylabel('Loss', size=15)
plt.xlabel('Epoch', size=15)
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



In [35]:
```
y_pred2 = model2.predict(x_test_n).argmax(1)
y_true = y_test
```

```
2022-07-06 16:45:30.853070: I tensorflow/core/grappler/optimizers/custom_gra
ph_optimizer_registry.cc:112] Plugin optimizer for device_type GPU is enable
d.
```

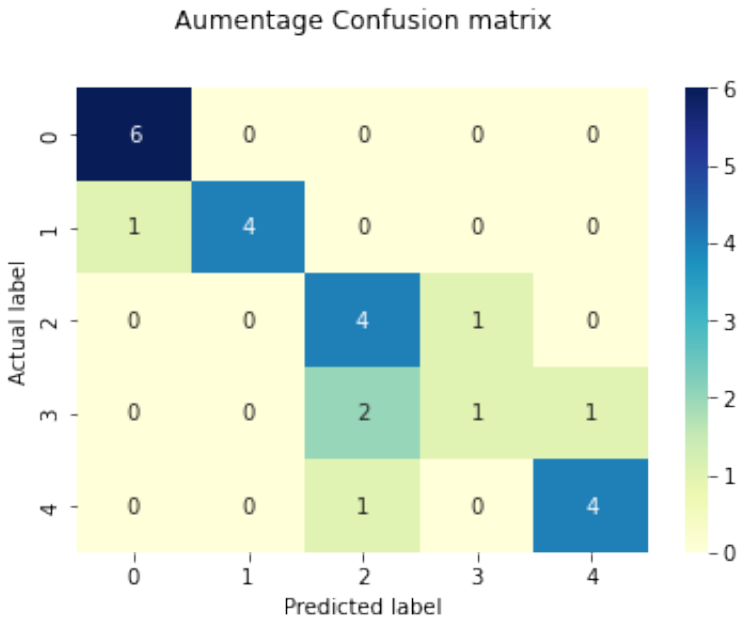
In [36]:
```
classes = ['Cheetah', 'Leopard', 'Lion', 'Puma', 'Tiger']
```

In [37]:
```
cm = confusion_matrix(y_true,y_pred2)
```

En la matriz de confusión de nuestro modelo ya entrenado con datos sintéticos, podemos apreciar a simple vista que aparentemente obtuvo un desempeño aceptable, destacando que tiene problemas clasificando al puma. Sin embargo, hay que apoyarnos en el reporte de clasificación para verificar lo anteriormente mencionado.

In [38]:
```
p = sns.heatmap(pd.DataFrame(cm), annot=True, cmap="YlGnBu" ,fmt='g')
plt.title('Aumentage Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

Out[38]:
```
Text(0.5, 15.0, 'Predicted label')
```



En nuestro reporte de clasificación confirmamos que al menos obtuvo un mejor desempeño que el modelo anterior, confirmando que la creación de datos sintéticos si apoyó al modelo al tener un mejor resultado. Confirmamos que tuvo problemas clasificando al puma, los demás felinos tienen un buen f1-score, específicamente el chita. La prueba verdadera serán los datos de validación.

In [39]:
```
print(classification_report(y_true,y_pred2))
print(accuracy_score(y_true,y_pred2))
```

```
              precision    recall  f1-score   support

           0       0.86      1.00      0.92         6
           1       1.00      0.80      0.89         5
           2       0.57      0.80      0.67         5
           3       0.50      0.25      0.33         4
           4       0.80      0.80      0.80         5

    accuracy                           0.76        25
   macro avg       0.75      0.73      0.72        25
weighted avg       0.76      0.76      0.75        25

0.76
```

Aumentage Validation

In [40]:
```
y_pred = model2.predict(x_val).argmax(1)
y_true = y_val
```

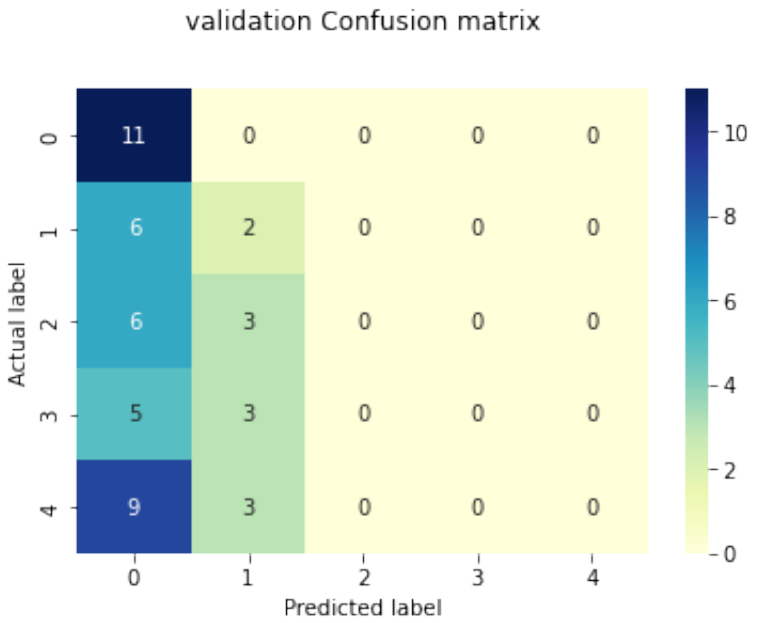
```
2022-07-06 16:45:31.238102: I tensorflow/core/grappler/optimizers/custom_gra
ph_optimizer_registry.cc:112] Plugin optimizer for device_type GPU is enable
d.
```

In [41]:
```
cm = confusion_matrix(y_true,y_pred)
```

En los datos de validación podemos destacar que la matriz de confusión está sesgada en la parte del chita y el leopardo. De la misma forma que el modelo pasado.

In [42]:
```
p = sns.heatmap(pd.DataFrame(cm), annot=True, cmap="YlGnBu" ,fmt='g')
plt.title('validation Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

Out[42]:
```
Text(0.5, 15.0, 'Predicted label')
```



El reporte de clasificación nos confirma lo anteriormente mencionado, obtuvo mejores resultados que sin los datos sintéticos sin embargo, no alcanza el resultado esperado. Confirmando su sesgo con las dos primeras clases.

In [43]:
```
print(classification_report(y_true,y_pred))
print(accuracy_score(y_true,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.30      1.00      0.46        11
           1       0.18      0.25      0.21         8
           2       0.00      0.00      0.00         9
           3       0.00      0.00      0.00         8
           4       0.00      0.00      0.00        12

    accuracy                           0.27        48
   macro avg       0.10      0.25      0.13        48
weighted avg       0.10      0.27      0.14        48

0.2708333333333333
/Users/irvingestrada/miniforge3/envs/tensorflow/lib/python3.9/site-packages/
sklearn/metrics/_classification.py:1327: UndefinedMetricWarning: Precision a
nd F-score are ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/irvingestrada/miniforge3/envs/tensorflow/lib/python3.9/site-packages/
sklearn/metrics/_classification.py:1327: UndefinedMetricWarning: Precision a
nd F-score are ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/irvingestrada/miniforge3/envs/tensorflow/lib/python3.9/site-packages/
sklearn/metrics/_classification.py:1327: UndefinedMetricWarning: Precision a
nd F-score are ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Conlcusión

La visión computacional es una de las áreas más relevantes en la actualidad. Como pudimos darnos cuenta en este notebook el trabajo con imágenes no es fácil, contiene muchos parámetros y no existe una metodología tal cual que nos guíe hacia un resultado ideal. El modelo construido en este notebook fue deficiente ya que alcanza muy bajos niveles de certeza. Esto es debido a que se cuenta con pocas imágenes para entrenar el modelo. Nuestro equipo para esta fase de imágenes se propuso el hacer distintas combinaciones para probar parámetros, no se designaron parámetros a estudiar de forma individual ya que queríamos abarcar el estudio de la mayoría de ellos para alcanzar una buena comprensión de las CNN y abriendo todas las posibilidades a los resultados. Además, consideramos que si solo nos enfocábamos en un parámetro cada uno, el resultado se sesgaría. En cuanto a mi desarrollo, hice distintos preprocesados para tratar de encontrar una mejoría o algún apoyo para nuestra CNN pero no se tuvo éxito. También intenté agregar capas de convolución, utilizar otro tipo de pooling, modificar la capa de fully connected, entre otras cosas. Probé con distintas funciones de activación, pero las que mejor me dieron resultados en la fully connected fueros la sigmoide y la ReLu. Sin embargo, la sigmoide superó a la ReLu. Algo a destacar es que son 5 categorías de felinos que de forma general tienen características similares, esto es común con animales que incluso un humano podría confundir, como un ratón y un hámster. Aparentemente el modelo no alcanza a identificar bien cada una de estas clases. El agregar datos aumentados al modelo mejora su validation accuracy sin embargo, nuestro modelo sigue siendo deficiente al probarlo con nuevas imágenes de los felinos las cuales no fueron utilizadas para entrenar el modelo, por eso es importante apartar un grupo de imágenes o datos para probarlos al final de cada modelo. Una de las soluciones para mejorar el modelo es obtener nuevas imágenes para tener un conjunto de datos de mayor tamaño, ya que generar imágenes en base a las que tenemos no nos trajo resultados óptimos. Otra solución es el transfer learning, que nos ayudaría en esta parte de pocos datos, dichas redes ya están entrenadas en base a un conjunto de datos sumamente grande.

Referencias

- JULIEN CALENGE. (Mayo 12 del 2022). Felidae | Cat species recognition. 30 de junio del 2022, de Kaggle Sitio web: https://www.kaggle.com/datasets/juliencalenge/felidae-tiger-lion-cheetah-leopard-puma
- Irving Estrada. Github. 2022,Sitio web: https://github.com/Irving-Estrada/Procesamiento
- François Chollet. (2022). Keras Documentation. 6 de Julio 2022, de Keras Sitio web: https://keras.io