# Tarea 7 - Transfer Learning

Alumno: Irving Daniel Estrada López

Matrícula: 1739907

## Introducción

El aprendizaje profundo ha recibido recientemente una atención cada vez mayor por parte de los investigadores y se ha aplicado con éxito a numerosas aplicaciones del mundo real. Los algoritmos de aprendizaje profundo intentan aprender funciones de alto nivel a partir de datos masivos, lo que hace que el aprendizaje profundo vaya más allá del aprendizaje automático tradicional. Se puede decir que el aprendizaje profundo es un algoritmo de aprendizaje de representación basado en datos a gran escala en el aprendizaje automático. Sin embargo, el aprendizaje profundo tiene algunos inconvenientes.

- La dependencia de datos es uno de los problemas más serios en el aprendizaje profundo. El aprendizaje profundo tiene una dependencia muy fuerte en los datos masivos de entrenamiento en comparación con los métodos tradicionales de aprendizaje automático, porque necesita una gran cantidad de datos para comprender los patrones latentes de los datos.

- Los datos de entrenamiento insuficientes son un problema inevitable en algunos dominios especiales. La recopilación de datos es compleja y costosa, lo que hace que sea extremadamente difícil crear un conjunto de datos anotados a gran escala y de alta calidad.

El transfer learning nos ayuda en la parte de que los datos de entrenamiento deben ser independientes e idénticamente distribuidos con los datos de prueba, el cual da solución al problema de tener datos de entrenamiento insuficientes.

# IMAGENET

Es una base de datos de imágenes organizada según la jerarquía WordNet, hay más de 100,000 synsets en la WordNet, la mayoría son sustantivos (más de 80,000), en el que cada nodo de la jerarquía está representado por cientos y miles de imágenes. El proyecto ha sido fundamental en el avance de la visión computacional y la investigación de aprendizaje profundo. Los datos están disponibles de forma gratuita para los investigadores para uso no comercial.

# Código

Como vimos en la tarea 6 nuestra CNN no obtuvo un buen desempeño y en las conclusiones de dicho notebook existía la sospecha de que la deficiencia de nuestra CNN era debido a la cantidad de datos que teníamos. El transfer learning debería ser una solución a dicho problema, obteniendo mejores resultados. En este se probarán distintos modelos de transfer learning para identificar el modelo óptimo para la clasificación de felinos.

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)


# 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_2639/2932696935.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]:
```
wrong_imgs_idx
```

Out[7]:
```
[115, 173]
```

In [8]:
```
a = 0
for idx in wrong_imgs_idx:
    del data[idx-a]
    del target[idx-a]
    a += 1
```

In [9]:
```
try:
    data = np.array(data)
    target = np.array(target)
except:
    print('Problem with broadcast')
```

In [10]:
```
print(data.shape)
print(target.shape)
```

```
(241, 224, 224, 3)
(241,)
```

In [11]:
```
from sklearn.preprocessing import LabelEncoder
lbl = LabelEncoder()
target_n = lbl.fit_transform(target)
```

In [12]:
```
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 [13]:
```
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 [14]:
```
os.chdir('..')
```

In [15]:
```
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 = .30
)

valid_datagen = ImageDataGenerator(
    rescale=1./255,
    validation_split = .30
)

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 172 images belonging to 5 classes.
Found 71 images belonging to 5 classes.
```

# VGG16

In [16]:
```
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.models import Model
from tensorflow.keras.applications.vgg16 import preprocess_input
vgg16 = VGG16(input_shape=x_train_n[0].shape, weights='imagenet',include_top
for layer in vgg16.layers:
    layer.trainable = False
```

```
Metal device set to: Apple M1
```

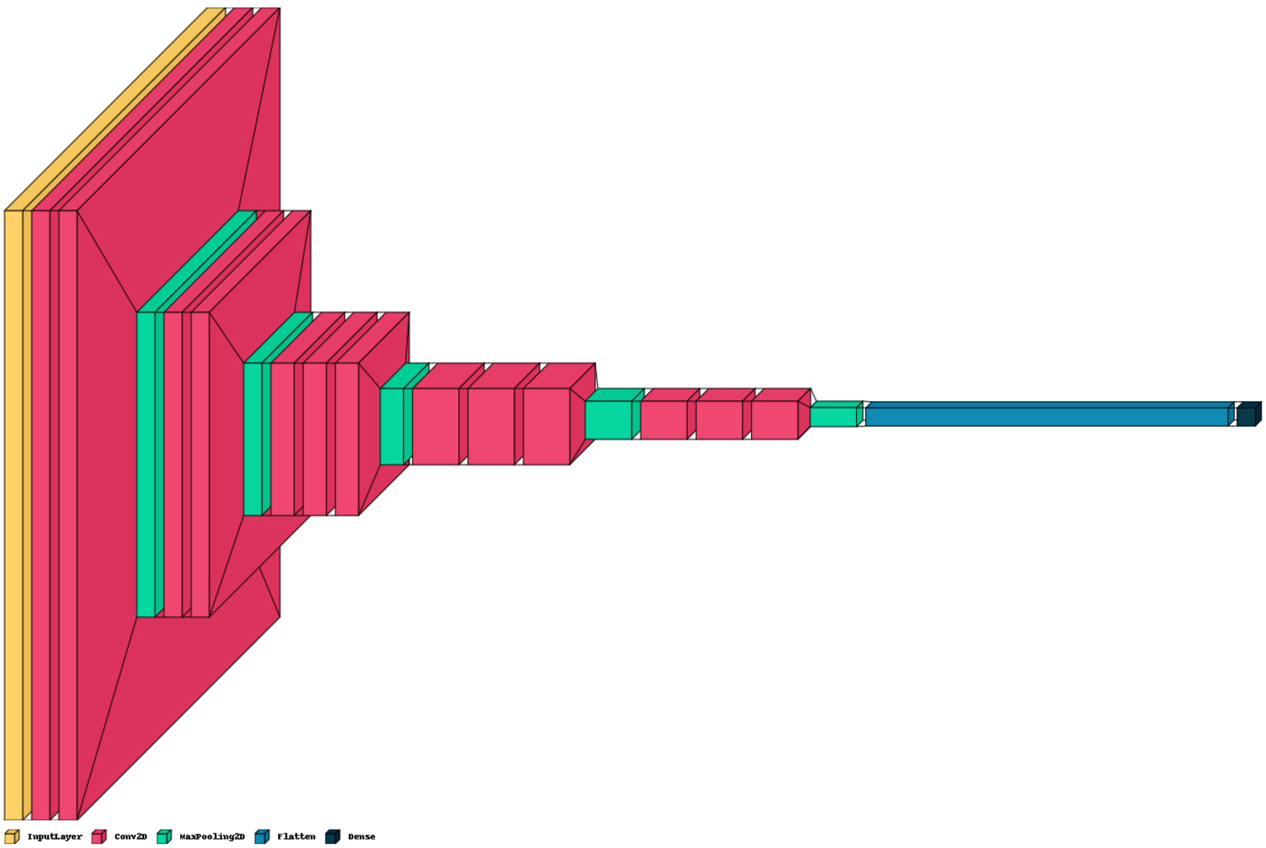
```
2022-07-06 08:40:41.994757: I tensorflow/core/common_runtime/pluggable_devic
e/pluggable_device_factory.cc:305] Could not identify NUMA node of platform
GPU ID 0, defaulting to 0. Your kernel may not have been built with NUMA sup
port.
2022-07-06 08:40:41.994928: I tensorflow/core/common_runtime/pluggable_devic
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 [17]:
```
from tensorflow.keras import layers,optimizers
x1 = layers.Flatten()(vgg16.output)
x2 = layers.Dense(5,activation='softmax')(x1)
model = Model(inputs=vgg16.input,outputs=x2)
```

In [18]:
```
model.compile(optimizer='rmsprop',loss='categorical_crossentropy',metrics=['
```

In [19]:
```
# Visualizing our model (Hidden Input)
import visualkeras
visualkeras.layered_view(model, scale_xy=3, legend=True,)
```

Out[19]:




In [20]:
```
from tensorflow.keras.callbacks import TensorBoard,ModelCheckpoint,EarlyStop
es = EarlyStopping(monitor='val_loss',mode='min',verbose=1,patience=10,restc
```

In [21]:
```
history = model.fit_generator(
    generator=train_data,
    validation_data=val_data,
    epochs=100,
    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 '
2022-07-06 08:40:43.199765: I tensorflow/compiler/mlir/mlir_graph_optimizati
on_pass.cc:176] None of the MLIR Optimization Passes are enabled (registered
2)
2022-07-06 08:40:43.199881: W tensorflow/core/platform/profile_utils/cpu_uti
ls.cc:128] Failed to get CPU frequency: 0 Hz
2022-07-06 08:40:43.356530: I tensorflow/core/grappler/optimizers/custom_gra
ph_optimizer_registry.cc:112] Plugin optimizer for device_type GPU is enable
d.
Epoch 1/100
6/6 [==============================] - ETA: 0s - loss: 6.5962 - accuracy: 0.
2442
2022-07-06 08:40:48.435456: I tensorflow/core/grappler/optimizers/custom_gra
ph_optimizer_registry.cc:112] Plugin optimizer for device_type GPU is enable
d.
6/6 [==============================] - 7s 1s/step - loss: 6.5962 - accuracy:
0.2442 - val_loss: 3.0895 - val_accuracy: 0.2113
Epoch 2/100
6/6 [==============================] - 6s 1s/step - loss: 2.4401 - accuracy:
0.3953 - val_loss: 2.8175 - val_accuracy: 0.3944
Epoch 3/100
6/6 [==============================] - 6s 959ms/step - loss: 2.0554 - accura
cy: 0.4128 - val_loss: 2.0602 - val_accuracy: 0.3380
Epoch 4/100
6/6 [==============================] - 6s 1s/step - loss: 1.6231 - accuracy:
0.5058 - val_loss: 0.8294 - val_accuracy: 0.6197
Epoch 5/100
6/6 [==============================] - 6s 981ms/step - loss: 1.5010 - accura
cy: 0.5116 - val_loss: 2.3221 - val_accuracy: 0.5493
Epoch 6/100
6/6 [==============================] - 6s 1s/step - loss: 1.2178 - accuracy:
0.5930 - val_loss: 1.2983 - val_accuracy: 0.5493
Epoch 7/100
6/6 [==============================] - 6s 1s/step - loss: 1.4973 - accuracy:
0.5116 - val_loss: 1.4392 - val_accuracy: 0.5634
Epoch 8/100
6/6 [==============================] - 6s 1s/step - loss: 1.3900 - accuracy:
0.5523 - val_loss: 1.7684 - val_accuracy: 0.5634
Epoch 9/100
6/6 [==============================] - 6s 1s/step - loss: 1.7311 - accuracy:
0.5174 - val_loss: 0.9787 - val_accuracy: 0.6620
Epoch 10/100
6/6 [==============================] - 6s 963ms/step - loss: 0.9635 - accura
cy: 0.6919 - val_loss: 1.9844 - val_accuracy: 0.5352
Epoch 11/100
6/6 [==============================] - 6s 1s/step - loss: 1.8046 - accuracy:
0.5058 - val_loss: 1.3763 - val_accuracy: 0.5915
Epoch 12/100
```

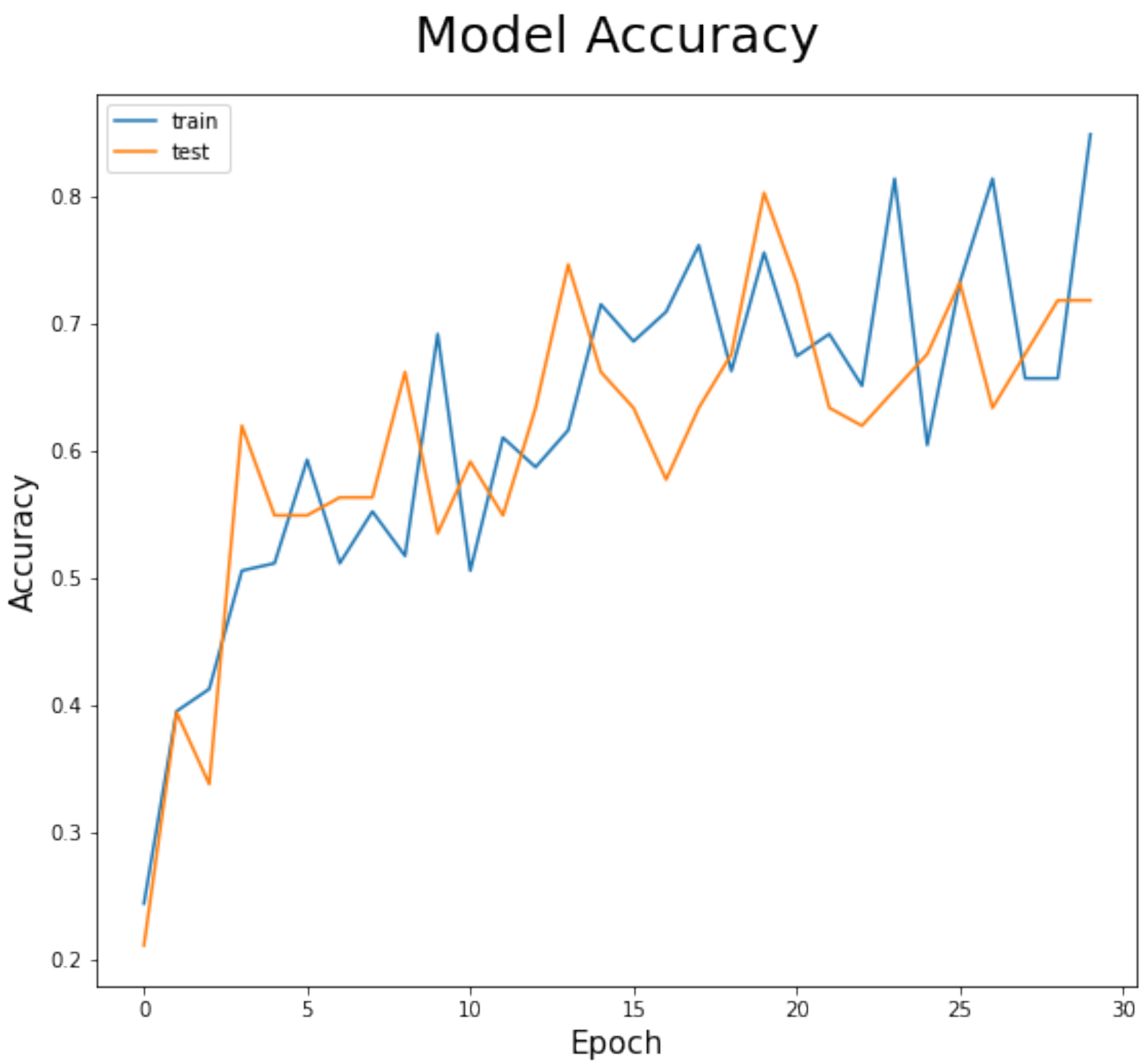
```
6/6 [==============================] - 6s 1s/step - loss: 1.1058 - accuracy:
0.6105 - val_loss: 1.7961 - val_accuracy: 0.5493
Epoch 13/100
6/6 [==============================] - 6s 1s/step - loss: 1.1970 - accuracy:
0.5872 - val_loss: 1.0809 - val_accuracy: 0.6338
Epoch 14/100
6/6 [==============================] - 7s 1s/step - loss: 1.3591 - accuracy:
0.6163 - val_loss: 0.8223 - val_accuracy: 0.7465
Epoch 15/100
6/6 [==============================] - 6s 1s/step - loss: 0.9361 - accuracy:
0.7151 - val_loss: 0.8341 - val_accuracy: 0.6620
Epoch 16/100
6/6 [==============================] - 7s 1s/step - loss: 0.9320 - accuracy:
0.6860 - val_loss: 1.3102 - val_accuracy: 0.6338
Epoch 17/100
6/6 [==============================] - 6s 1s/step - loss: 0.9263 - accuracy:
0.7093 - val_loss: 2.0784 - val_accuracy: 0.5775
Epoch 18/100
6/6 [==============================] - 6s 1s/step - loss: 0.7598 - accuracy:
0.7616 - val_loss: 1.6698 - val_accuracy: 0.6338
Epoch 19/100
6/6 [==============================] - 6s 1s/step - loss: 1.2653 - accuracy:
0.6628 - val_loss: 1.1513 - val_accuracy: 0.6761
Epoch 20/100
6/6 [==============================] - 6s 924ms/step - loss: 0.7566 - accura
cy: 0.7558 - val_loss: 0.5679 - val_accuracy: 0.8028
Epoch 21/100
6/6 [==============================] - 6s 1s/step - loss: 0.9826 - accuracy:
0.6744 - val_loss: 0.8749 - val_accuracy: 0.7324
Epoch 22/100
6/6 [==============================] - 6s 983ms/step - loss: 0.9948 - accura
cy: 0.6919 - val_loss: 1.0465 - val_accuracy: 0.6338
Epoch 23/100
6/6 [==============================] - 6s 1s/step - loss: 1.1778 - accuracy:
0.6512 - val_loss: 1.3793 - val_accuracy: 0.6197
Epoch 24/100
6/6 [==============================] - 6s 1s/step - loss: 0.5719 - accuracy:
0.8140 - val_loss: 0.9164 - val_accuracy: 0.6479
Epoch 25/100
6/6 [==============================] - 6s 1s/step - loss: 1.2022 - accuracy:
0.6047 - val_loss: 1.8075 - val_accuracy: 0.6761
Epoch 26/100
6/6 [==============================] - 6s 961ms/step - loss: 0.8709 - accura
cy: 0.7326 - val_loss: 0.8190 - val_accuracy: 0.7324
Epoch 27/100
6/6 [==============================] - 6s 1s/step - loss: 0.6480 - accuracy:
0.8140 - val_loss: 1.4130 - val_accuracy: 0.6338
Epoch 28/100
6/6 [==============================] - 6s 1s/step - loss: 1.0576 - accuracy:
0.6570 - val_loss: 1.6253 - val_accuracy: 0.6761
Epoch 29/100
6/6 [==============================] - 6s 994ms/step - loss: 1.0979 - accura
cy: 0.6570 - val_loss: 1.1227 - val_accuracy: 0.7183
```

```
Epoch 30/100
6/6 [==============================] - 7s 1s/step - loss: 0.4371 - accuracy:
0.8488 - val_loss: 0.7137 - val_accuracy: 0.7183
Restoring model weights from the end of the best epoch.
Epoch 00030: early stopping
```

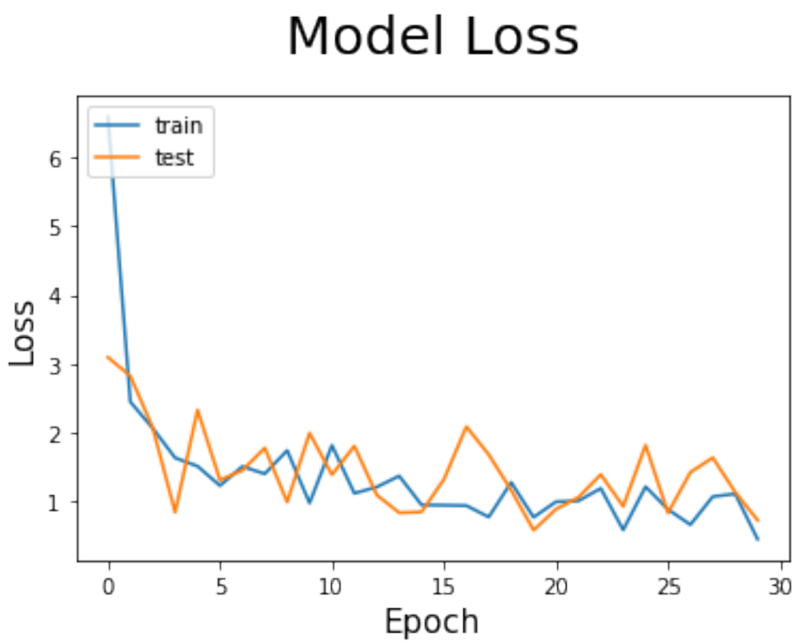
In [22]:

```
# 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[22]: <matplotlib.legend.Legend at 0x294ab66a0>



In [23]:
```
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 [24]:
```
y_pred3 = model.predict(x_test_n).argmax(1)
y_true = y_test
```

```
2022-07-06 08:43:52.126740: I tensorflow/core/grappler/optimizers/custom_gra
ph_optimizer_registry.cc:112] Plugin optimizer for device_type GPU is enable
d.
```

In [25]:
```
classes = ['Cheetah', 'Leopard', 'Lion', 'Puma', 'Tiger']
```

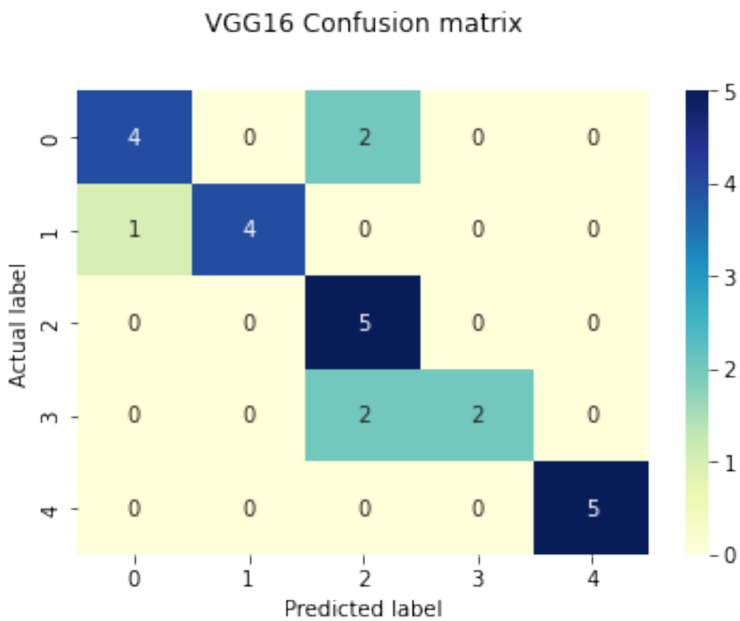
In [26]:
```
import seaborn as sns
from sklearn.metrics import accuracy_score,f1_score,confusion_matrix,classif
```

In [27]:
```
cm = confusion_matrix(y_true,y_pred3)
```

El modelo aparentemente tiene buenos resultados, algo a destacar es que tuvo problemas al predecir correctamente al Chita, confundiéndolo con el león. El modelo obtuvo buenos resultados con el tigre y el león.

In [28]:
```python
p = sns.heatmap(pd.DataFrame(cm), annot=True, cmap="YlGnBu" ,fmt='g')
plt.title('VGG16 Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

Out[28]:
```
Text(0.5, 15.0, 'Predicted label')
```




El reporte de clasificación nos confirma lo anteriormente mencionado, obteniendo un 80% de accuarcy. Los F1-Scores son buenos, todos superando el 50% que era uno de los inconvenientes de la tarea 6 ya que esto representaría una moneda al aire. El tigre fue el que obtuvo resultados perfectos, sin embargo, estos son los datos que ya vió el modelo. Hay que probarlo con los datos de validación los cuales no ha visto.

In [29]:
```python
print(classification_report(y_true,y_pred3))
print(accuracy_score(y_true,y_pred3))
```

```
              precision    recall  f1-score   support

           0       0.80      0.67      0.73         6
           1       1.00      0.80      0.89         5
           2       0.56      1.00      0.71         5
           3       1.00      0.50      0.67         4
           4       1.00      1.00      1.00         5

    accuracy                           0.80        25
   macro avg       0.87      0.79      0.80        25
weighted avg       0.86      0.80      0.80        25

0.8
```

# Validation VGG16

In [30]:
```
y_pred3 = model.predict(x_val).argmax(1)
y_true = y_val
```

```
2022-07-06 08:43:52.851450: I tensorflow/core/grappler/optimizers/custom_gra
ph_optimizer_registry.cc:112] Plugin optimizer for device_type GPU is enable
d.
```

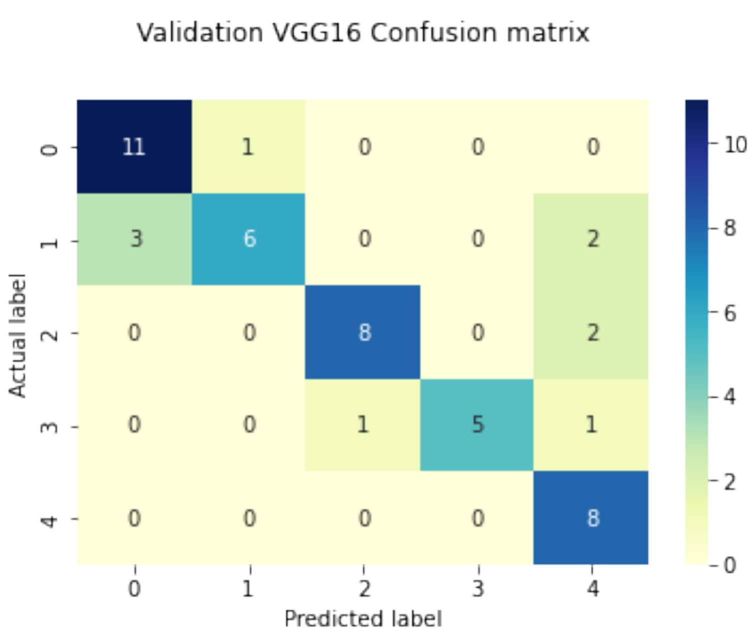
In [31]:
```
classes = ['Cheetah', 'Leopard', 'Lion', 'Puma', 'Tiger']
```

In [32]:
```
cm = confusion_matrix(y_true,y_pred3)
```

A diferencia de un modelo tradicional el transfer learning aparentemente nos ayuda a que no se sesgue. Si recordamos en la tarea 6 nuestros modelos se sesgaban y más con los datos de validación. Algo a destacar es que con el que más tuvo problemas la VGG16 fue con el leopardo confundiéndolo con un chita y un tigre.

In [33]:
```
p = sns.heatmap(pd.DataFrame(cm), annot=True, cmap="YlGnBu" ,fmt='g')
plt.title('Validation VGG16 Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

Out[33]:
```
Text(0.5, 15.0, 'Predicted label')
```



La VGG16 obtuvo excelente resultado comparándola con los modelos de la tarea 6, no está sesgado y alcanza casi el 80% en accuracy con los datos de validación es una mejora impresionante. El reporte nos ayuda a confirmar que el modelo tiene problemas clasificando el leopardo.

In [34]:
```
print(classification_report(y_true,y_pred3))
print(accuracy_score(y_true,y_pred3))
```

```
              precision    recall  f1-score   support

           0       0.79      0.92      0.85        12
           1       0.86      0.55      0.67        11
           2       0.89      0.80      0.84        10
           3       1.00      0.71      0.83         7
           4       0.62      1.00      0.76         8

    accuracy                           0.79        48
   macro avg       0.83      0.80      0.79        48
weighted avg       0.83      0.79      0.79        48

0.7916666666666666
```

In [ ]:

# VGG19

La principal diferencia entre la VGG16 y la VGG19 son las capas que tiene, el numero indica las capas que tienen.

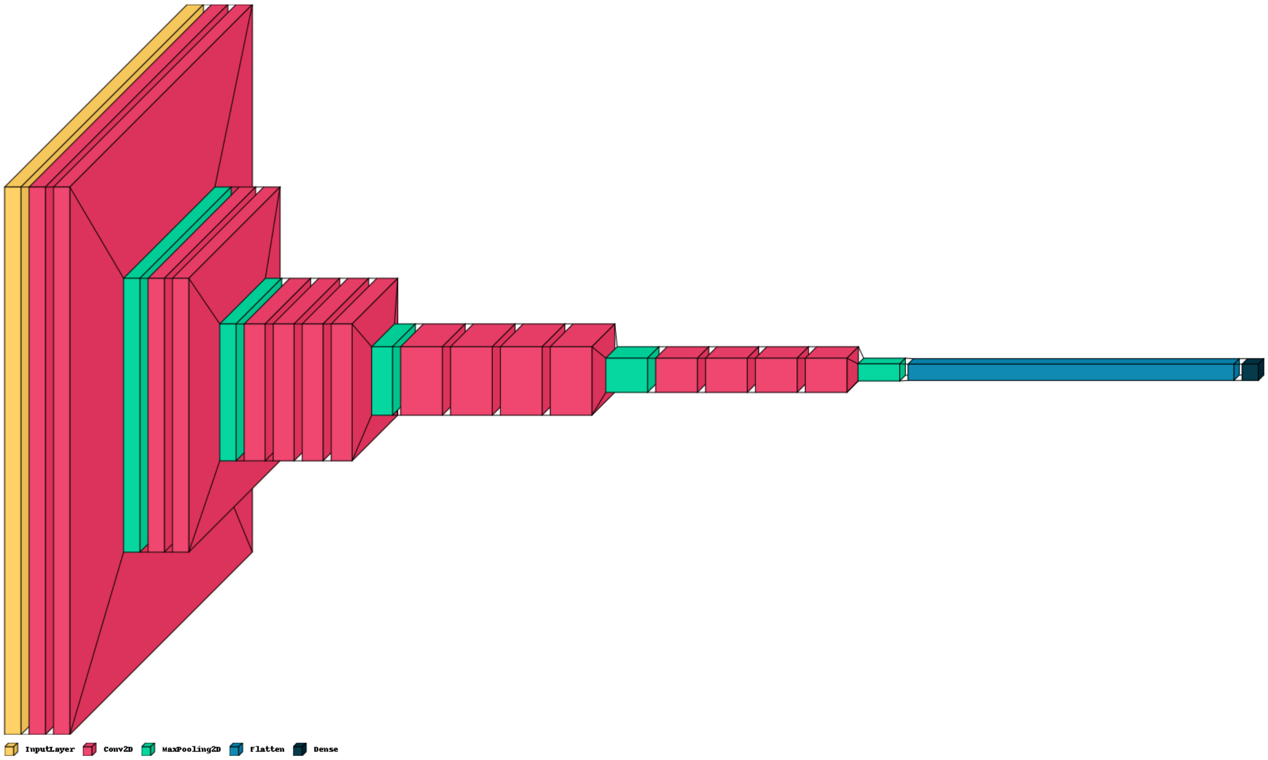
In [35]:
```
from tensorflow.keras.applications.vgg19 import VGG19
from tensorflow.keras.models import Model
vgg19 = VGG19(input_shape=x_train_n[0].shape, weights='imagenet',include_top
for layer in vgg19.layers:
    layer.trainable = False
```

In [36]:
```
from tensorflow.keras import layers,optimizers
x1 = layers.Flatten()(vgg19.output)
x2 = layers.Dense(5,activation='softmax')(x1)
model = Model(inputs=vgg19.input,outputs=x2)
```

In [37]:
```
model.compile(optimizer='rmsprop',loss='categorical_crossentropy',metrics=['
```

In [38]:
```
# Visualizing our model (Hidden Input)
import visualkeras
visualkeras.layered_view(model, scale_xy=3, legend=True,)
```

Out[38]:




In [39]:
```
from tensorflow.keras.callbacks import TensorBoard,ModelCheckpoint,EarlyStop
es = EarlyStopping(monitor='val_loss',mode='min',verbose=1,patience=10,restc
```

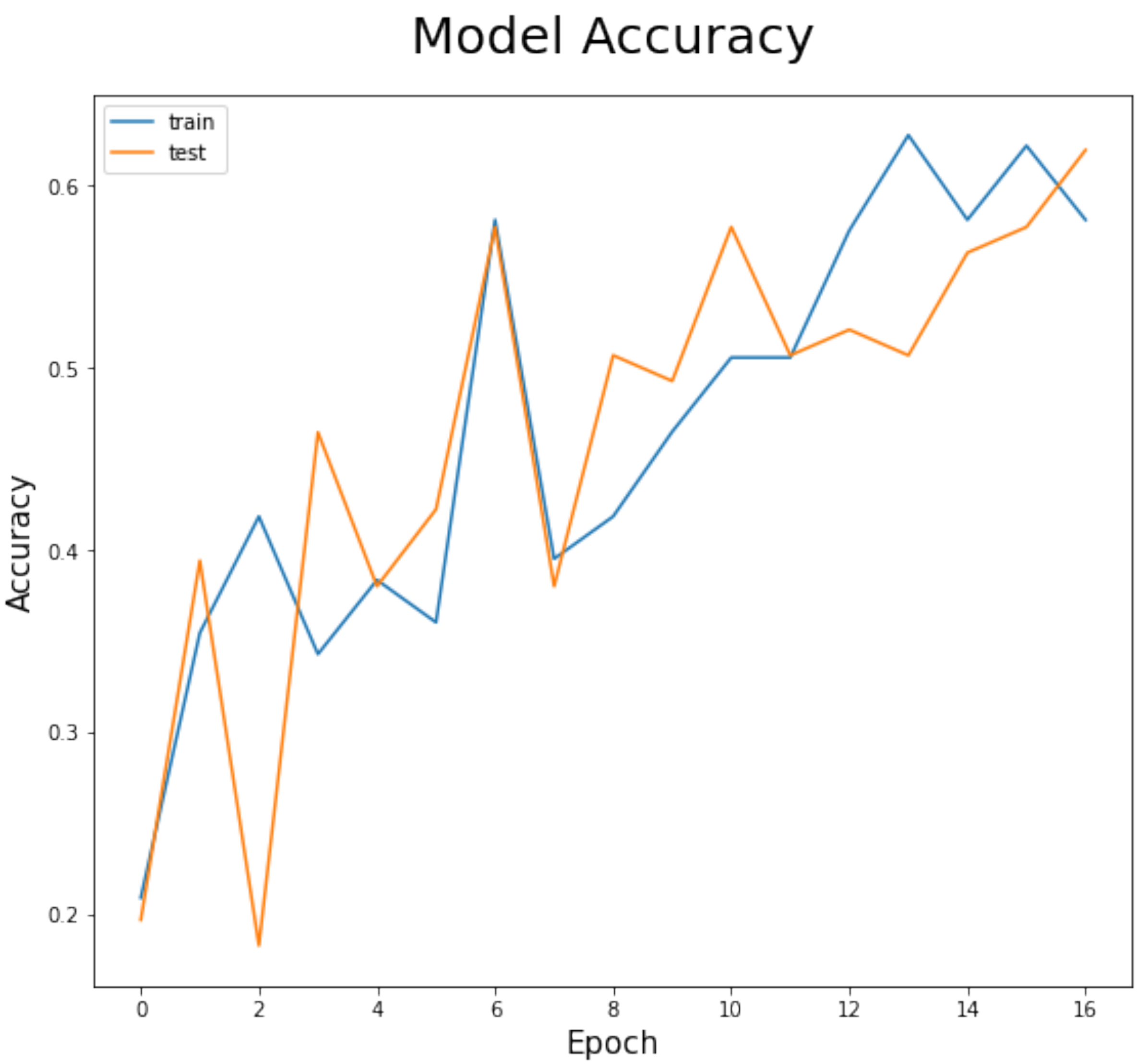
In [40]:
```
history = model.fit_generator(
    generator=train_data,
    validation_data=val_data,
    epochs=100,
    callbacks=es )
```

```
/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 1/100
2022-07-06 08:43:54.725461: 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: 7.9506 - accuracy: 0.
2093
2022-07-06 08:43:59.886222: I tensorflow/core/grappler/optimizers/custom_gra
ph_optimizer_registry.cc:112] Plugin optimizer for device_type GPU is enable
d.
```

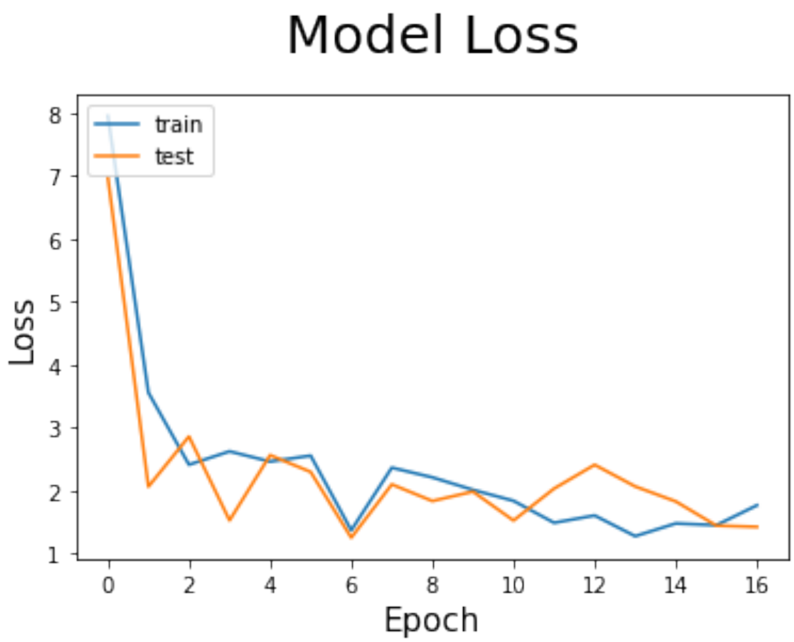
```
6/6 [==============================] - 8s 1s/step - loss: 7.9506 - accuracy: 0.2093 - val_loss: 6.9741 - val_accuracy: 0.1972
Epoch 2/100
6/6 [==============================] - 7s 1s/step - loss: 3.5555 - accuracy: 0.3547 - val_loss: 2.0615 - val_accuracy: 0.3944
Epoch 3/100
6/6 [==============================] - 7s 1s/step - loss: 2.4101 - accuracy: 0.4186 - val_loss: 2.8583 - val_accuracy: 0.1831
Epoch 4/100
6/6 [==============================] - 7s 1s/step - loss: 2.6217 - accuracy: 0.3430 - val_loss: 1.5251 - val_accuracy: 0.4648
Epoch 5/100
6/6 [==============================] - 7s 1s/step - loss: 2.4603 - accuracy: 0.3837 - val_loss: 2.5613 - val_accuracy: 0.3803
Epoch 6/100
6/6 [==============================] - 7s 1s/step - loss: 2.5538 - accuracy: 0.3605 - val_loss: 2.2953 - val_accuracy: 0.4225
Epoch 7/100
6/6 [==============================] - 6s 1s/step - loss: 1.3683 - accuracy: 0.5814 - val_loss: 1.2502 - val_accuracy: 0.5775
Epoch 8/100
6/6 [==============================] - 7s 1s/step - loss: 2.3626 - accuracy: 0.3953 - val_loss: 2.0972 - val_accuracy: 0.3803
Epoch 9/100
6/6 [==============================] - 7s 1s/step - loss: 2.2088 - accuracy: 0.4186 - val_loss: 1.8333 - val_accuracy: 0.5070
Epoch 10/100
6/6 [==============================] - 7s 1s/step - loss: 2.0106 - accuracy: 0.4651 - val_loss: 1.9829 - val_accuracy: 0.4930
Epoch 11/100
6/6 [==============================] - 7s 1s/step - loss: 1.8363 - accuracy: 0.5058 - val_loss: 1.5236 - val_accuracy: 0.5775
Epoch 12/100
6/6 [==============================] - 7s 1s/step - loss: 1.4878 - accuracy: 0.5058 - val_loss: 2.0306 - val_accuracy: 0.5070
Epoch 13/100
6/6 [==============================] - 7s 1s/step - loss: 1.6034 - accuracy: 0.5756 - val_loss: 2.4086 - val_accuracy: 0.5211
Epoch 14/100
6/6 [==============================] - 7s 1s/step - loss: 1.2747 - accuracy: 0.6279 - val_loss: 2.0645 - val_accuracy: 0.5070
Epoch 15/100
6/6 [==============================] - 7s 1s/step - loss: 1.4768 - accuracy: 0.5814 - val_loss: 1.8265 - val_accuracy: 0.5634
Epoch 16/100
6/6 [==============================] - 7s 1s/step - loss: 1.4490 - accuracy: 0.6221 - val_loss: 1.4426 - val_accuracy: 0.5775
Epoch 17/100
6/6 [==============================] - 7s 1s/step - loss: 1.7647 - accuracy: 0.5814 - val_loss: 1.4222 - val_accuracy: 0.6197
Restoring model weights from the end of the best epoch.
Epoch 00017: early stopping
```

In [41]:
```
# 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[41]:
```
<matplotlib.legend.Legend at 0x161c358b0>
```



In [42]:
```
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 [43]:
```
y_pred3 = model.predict(x_test_n).argmax(1)
y_true = y_test
```

```
2022-07-06 08:45:52.516950: I tensorflow/core/grappler/optimizers/custom_gra
ph_optimizer_registry.cc:112] Plugin optimizer for device_type GPU is enable
d.
```

In [44]:
```
classes = ['Cheetah', 'Leopard', 'Lion', 'Puma', 'Tiger']
```

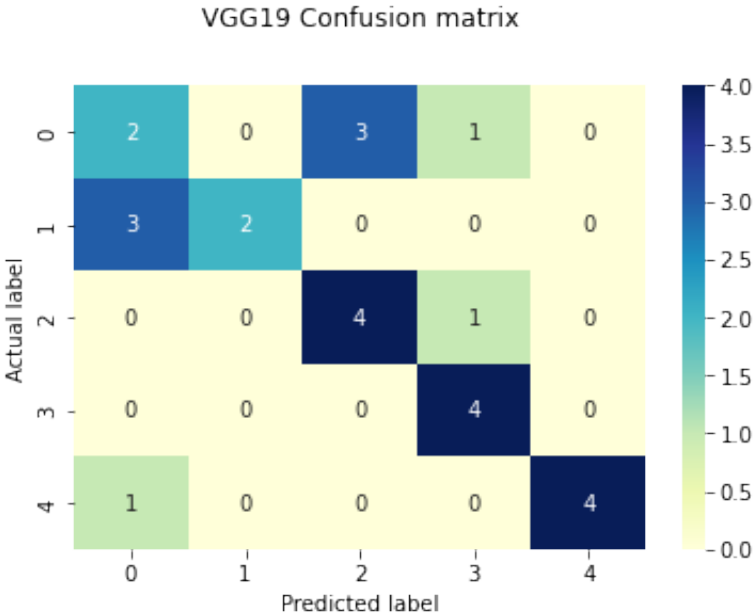
In [45]:
```
import seaborn as sns
from sklearn.metrics import accuracy_score,f1_score,confusion_matrix,classif
```

In [46]:
```
cm = confusion_matrix(y_true,y_pred3)
```

A la VGG19 aparentemente no le fue tan bien como a la VGG16, lo podemos ver en su matriz de confusión, esta obtuvo más errores con el chita y el leopardo, sin embargo, no parece estar segada de algún modo.

In [47]:
```
p = sns.heatmap(pd.DataFrame(cm), annot=True, cmap="YlGnBu" ,fmt='g')
plt.title('VGG19 Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

Out[47]:
```
Text(0.5, 15.0, 'Predicted label')
```




El reporte de clasificación confirma lo anteriormente mencionado, no le fue bien clasificando chitas, las dos últimas clases que son puma y tigre fueron las más altas. La VGG19 es prueba de que no por agregar más capas se ajustaría mejor a nuestros datos.

In [48]:
```
print(classification_report(y_true,y_pred3))
print(accuracy_score(y_true,y_pred3))
```

```
              precision    recall  f1-score   support

           0       0.33      0.33      0.33         6
           1       1.00      0.40      0.57         5
           2       0.57      0.80      0.67         5
           3       0.67      1.00      0.80         4
           4       1.00      0.80      0.89         5

    accuracy                           0.64        25
   macro avg       0.71      0.67      0.65        25
weighted avg       0.70      0.64      0.63        25

0.64
```

# Validation VGG19

In [49]:
```
y_pred3 = model.predict(x_val).argmax(1)
y_true = y_val
```

```
2022-07-06 08:45:53.259076: I tensorflow/core/grappler/optimizers/custom_gra
ph_optimizer_registry.cc:112] Plugin optimizer for device_type GPU is enable
d.
```

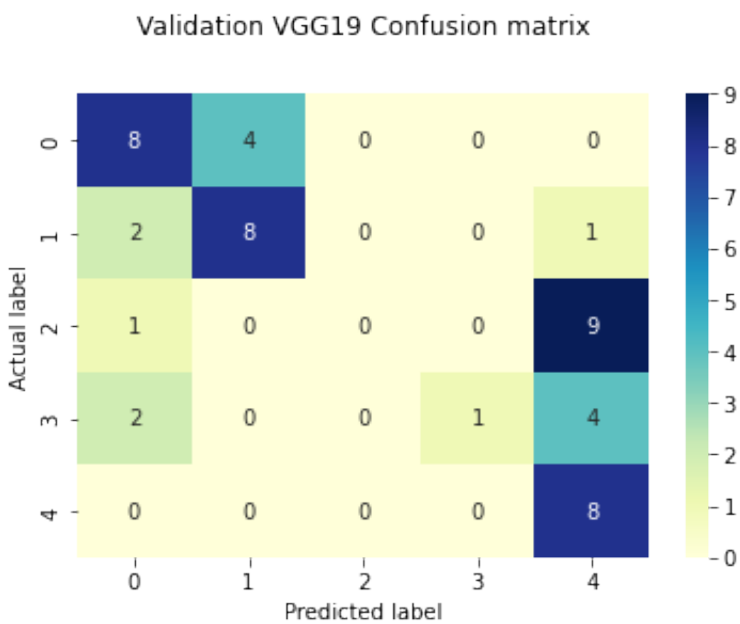
In [50]:
```
classes = ['Cheetah', 'Leopard', 'Lion', 'Puma', 'Tiger']
```

In [51]:
```
cm = confusion_matrix(y_true,y_pred3)
```

Con los datos de validación pareciera que nos indica que está sesgado, está identificando los Leones como si fueran tigres, de igual forma con los pumas. Puede que tanga que ver con su complexión robusta de estos tres felinos, a diferencia del leopardo y el chita. Sin embargo, también tiene problemas clasificando al chita y al leopardo.

In [52]:
```
p = sns.heatmap(pd.DataFrame(cm), annot=True, cmap="YlGnBu" ,fmt='g')
plt.title('Validation VGG19 Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

Out[52]:
```
Text(0.5, 15.0, 'Predicted label')
```



En el reporte podemos identificar como de recall en la última clase, tiene el 100% sin embargo, en precisión tiene 36%, a pesar de que acertó todas las imágenes que en realidad eran leones confundió algunos otros felinos con leones, esta es la importancia de las métricas. Al final son indicadores que nos dan información del modelo. La VGG19 alcanza un 52% de accuracy el cual al igual que en la tarea 6 es una moneda al aire con los datos de validación.

In [53]:
```
print(classification_report(y_true,y_pred3))
print(accuracy_score(y_true,y_pred3))
```

```
              precision    recall  f1-score   support

           0       0.62      0.67      0.64        12
           1       0.67      0.73      0.70        11
           2       0.00      0.00      0.00        10
           3       1.00      0.14      0.25         7
           4       0.36      1.00      0.53         8

    accuracy                           0.52        48
   macro avg       0.53      0.51      0.42        48
weighted avg       0.51      0.52      0.44        48

0.5208333333333334
```

```
/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))
```

# Xception

In [54]:
```
os.chdir('/Users/irvingestrada/Documents/Maestría/9- Procesamiento y Clasifi
```

In [55]:
```
f = os.listdir()[1:]
f.sort()
```

In [56]:
```
data = []
target = []
new_size = (299,299)


# 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 [57]:
```
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_2639/2932696935.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 [58]:
```
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 [59]:
```
wrong_imgs_idx
```

Out[59]:
```
[115, 173]
```

In [60]:
```
a = 0
for idx in wrong_imgs_idx:
    del data[idx-a]
    del target[idx-a]
    a += 1
```

In [61]:
```
try:
    data = np.array(data)
    target = np.array(target)
except:
    print('Problem with broadcast')
```

In [62]:
```
print(data.shape)
print(target.shape)
```

```
(241, 299, 299, 3)
(241,)
```

In [63]:
```
from sklearn.preprocessing import LabelEncoder
lbl = LabelEncoder()
target_n = lbl.fit_transform(target)
```

In [64]:
```
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)
```

```
x_train shape: (168, 299, 299, 3)
x_test shape: (25, 299, 299, 3)
x_val shape: (48, 299, 299, 3)
y_train shape: (168,)
y_test shape: (25,)
y_val shape: (48,)
```

In [65]:
```
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 [66]:
```
os.chdir('..')
```

In [67]:
```
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 = .30
)

valid_datagen = ImageDataGenerator(
    rescale=1./255,
    validation_split = .30
)

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 172 images belonging to 5 classes.
Found 71 images belonging to 5 classes.
```

In [68]:
```
from tensorflow.keras.applications.xception import Xception
```

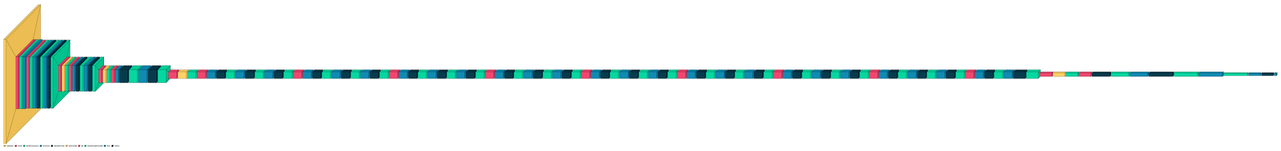
In [69]:
```
from tensorflow.keras.models import Model
xce = Xception(weights='imagenet')
for layer in xce.layers:
    layer.trainable = False
```

In [70]:
```
from tensorflow.keras import layers,optimizers
x1 = layers.Flatten()(xce.output)
x2 = layers.Dense(5,activation='softmax')(x1)
model = Model(inputs=xce.input,outputs=x2)
```

In [71]:
```
model.compile(optimizer='rmsprop',loss='categorical_crossentropy',metrics=['
```

In [72]:
```
# Visualizing our model (Hidden Input)
import visualkeras
visualkeras.layered_view(model, scale_xy=3, legend=True,)
```

Out[72]:



In [73]:

```
from tensorflow.keras.callbacks import TensorBoard,ModelCheckpoint,EarlyStop
es = EarlyStopping(monitor='val_loss',mode='min',verbose=1,patience=10,restc
```

In [74]:

```
history = model.fit_generator(
    generator=train_data,
    validation_data=val_data,
    epochs=100,
    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 1/100
2022-07-06 08:46:03.239622: 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: 1.6197 - accuracy: 0.
0116
2022-07-06 08:46:09.980022: I tensorflow/core/grappler/optimizers/custom_gra
ph_optimizer_registry.cc:112] Plugin optimizer for device_type GPU is enable
d.
6/6 [==============================] - 10s 2s/step - loss: 1.6197 - accuracy
: 0.0116 - val_loss: 1.6141 - val_accuracy: 0.0000e+00
Epoch 2/100
6/6 [==============================] - 8s 1s/step - loss: 1.6109 - accuracy:
0.0407 - val_loss: 1.6046 - val_accuracy: 0.0845
Epoch 3/100
6/6 [==============================] - 8s 2s/step - loss: 1.6032 - accuracy:
0.1919 - val_loss: 1.5961 - val_accuracy: 0.2113
Epoch 4/100
6/6 [==============================] - 9s 1s/step - loss: 1.5964 - accuracy:
0.2791 - val_loss: 1.5878 - val_accuracy: 0.2394
Epoch 5/100
6/6 [==============================] - 8s 2s/step - loss: 1.5928 - accuracy:
0.3779 - val_loss: 1.5806 - val_accuracy: 0.6479
Epoch 6/100
6/6 [==============================] - 8s 1s/step - loss: 1.5855 - accuracy:
0.6221 - val_loss: 1.5728 - val_accuracy: 0.8169
Epoch 7/100
6/6 [==============================] - 9s 1s/step - loss: 1.5781 - accuracy:
0.7209 - val_loss: 1.5647 - val_accuracy: 0.8169
Epoch 8/100
6/6 [==============================] - 8s 1s/step - loss: 1.5732 - accuracy:
0.7209 - val_loss: 1.5575 - val_accuracy: 0.8310
Epoch 9/100
```

```
6/6 [==============================] - 9s 1s/step - loss: 1.5666 - accuracy:
0.7500 - val_loss: 1.5500 - val_accuracy: 0.8310
Epoch 10/100
6/6 [==============================] - 8s 1s/step - loss: 1.5612 - accuracy:
0.7674 - val_loss: 1.5424 - val_accuracy: 0.8310
Epoch 11/100
6/6 [==============================] - 8s 2s/step - loss: 1.5564 - accuracy:
0.7326 - val_loss: 1.5353 - val_accuracy: 0.8310
Epoch 12/100
6/6 [==============================] - 9s 1s/step - loss: 1.5496 - accuracy:
0.7558 - val_loss: 1.5279 - val_accuracy: 0.8310
Epoch 13/100
6/6 [==============================] - 8s 1s/step - loss: 1.5462 - accuracy:
0.7326 - val_loss: 1.5209 - val_accuracy: 0.8310
Epoch 14/100
6/6 [==============================] - 9s 1s/step - loss: 1.5369 - accuracy:
0.7674 - val_loss: 1.5131 - val_accuracy: 0.8310
Epoch 15/100
6/6 [==============================] - 8s 1s/step - loss: 1.5280 - accuracy:
0.7791 - val_loss: 1.5057 - val_accuracy: 0.8310
Epoch 16/100
6/6 [==============================] - 8s 1s/step - loss: 1.5245 - accuracy:
0.7558 - val_loss: 1.4980 - val_accuracy: 0.8310
Epoch 17/100
6/6 [==============================] - 8s 1s/step - loss: 1.5194 - accuracy:
0.7442 - val_loss: 1.4905 - val_accuracy: 0.8310
Epoch 18/100
6/6 [==============================] - 8s 1s/step - loss: 1.5186 - accuracy:
0.7558 - val_loss: 1.4841 - val_accuracy: 0.8310
Epoch 19/100
6/6 [==============================] - 9s 1s/step - loss: 1.5052 - accuracy:
0.7616 - val_loss: 1.4766 - val_accuracy: 0.8873
Epoch 20/100
6/6 [==============================] - 9s 1s/step - loss: 1.5003 - accuracy:
0.8140 - val_loss: 1.4692 - val_accuracy: 0.9437
Epoch 21/100
6/6 [==============================] - 8s 1s/step - loss: 1.4934 - accuracy:
0.8721 - val_loss: 1.4615 - val_accuracy: 0.9718
Epoch 22/100
6/6 [==============================] - 9s 1s/step - loss: 1.4936 - accuracy:
0.8547 - val_loss: 1.4545 - val_accuracy: 0.9718
Epoch 23/100
6/6 [==============================] - 8s 1s/step - loss: 1.4901 - accuracy:
0.8140 - val_loss: 1.4475 - val_accuracy: 0.9718
Epoch 24/100
6/6 [==============================] - 8s 1s/step - loss: 1.4832 - accuracy:
0.8547 - val_loss: 1.4404 - val_accuracy: 0.9718
Epoch 25/100
6/6 [==============================] - 9s 1s/step - loss: 1.4789 - accuracy:
0.8023 - val_loss: 1.4330 - val_accuracy: 0.9859
Epoch 26/100
6/6 [==============================] - 9s 2s/step - loss: 1.4714 - accuracy:
0.8837 - val_loss: 1.4259 - val_accuracy: 0.9859
```

```
Epoch 27/100
6/6 [==============================] - 9s 2s/step - loss: 1.4599 - accuracy:
0.9070 - val_loss: 1.4186 - val_accuracy: 0.9859
Epoch 28/100
6/6 [==============================] - 9s 2s/step - loss: 1.4625 - accuracy:
0.8372 - val_loss: 1.4117 - val_accuracy: 0.9859
Epoch 29/100
6/6 [==============================] - 9s 2s/step - loss: 1.4584 - accuracy:
0.8256 - val_loss: 1.4048 - val_accuracy: 0.9859
Epoch 30/100
6/6 [==============================] - 9s 2s/step - loss: 1.4483 - accuracy:
0.8430 - val_loss: 1.3973 - val_accuracy: 0.9859
Epoch 31/100
6/6 [==============================] - 10s 2s/step - loss: 1.4500 - accuracy
: 0.8605 - val_loss: 1.3905 - val_accuracy: 0.9859
Epoch 32/100
6/6 [==============================] - 9s 2s/step - loss: 1.4387 - accuracy:
0.8430 - val_loss: 1.3834 - val_accuracy: 0.9859
Epoch 33/100
6/6 [==============================] - 9s 1s/step - loss: 1.4344 - accuracy:
0.8721 - val_loss: 1.3764 - val_accuracy: 0.9859
Epoch 34/100
6/6 [==============================] - 9s 1s/step - loss: 1.4256 - accuracy:
0.8488 - val_loss: 1.3691 - val_accuracy: 0.9859
Epoch 35/100
6/6 [==============================] - 9s 1s/step - loss: 1.4223 - accuracy:
0.8779 - val_loss: 1.3625 - val_accuracy: 0.9859
Epoch 36/100
6/6 [==============================] - 9s 1s/step - loss: 1.4170 - accuracy:
0.8605 - val_loss: 1.3554 - val_accuracy: 0.9859
Epoch 37/100
6/6 [==============================] - 9s 1s/step - loss: 1.4100 - accuracy:
0.8837 - val_loss: 1.3483 - val_accuracy: 0.9859
Epoch 38/100
6/6 [==============================] - 8s 1s/step - loss: 1.4057 - accuracy:
0.8663 - val_loss: 1.3414 - val_accuracy: 0.9859
Epoch 39/100
6/6 [==============================] - 8s 2s/step - loss: 1.3937 - accuracy:
0.8779 - val_loss: 1.3339 - val_accuracy: 0.9859
Epoch 40/100
6/6 [==============================] - 9s 1s/step - loss: 1.3894 - accuracy:
0.8837 - val_loss: 1.3271 - val_accuracy: 0.9859
Epoch 41/100
6/6 [==============================] - 9s 1s/step - loss: 1.3876 - accuracy:
0.8779 - val_loss: 1.3201 - val_accuracy: 0.9859
Epoch 42/100
6/6 [==============================] - 9s 1s/step - loss: 1.3828 - accuracy:
0.8837 - val_loss: 1.3131 - val_accuracy: 0.9859
Epoch 43/100
6/6 [==============================] - 8s 1s/step - loss: 1.3747 - accuracy:
0.8779 - val_loss: 1.3062 - val_accuracy: 0.9859
Epoch 44/100
6/6 [==============================] - 9s 1s/step - loss: 1.3763 - accuracy:
```

```
0.8605 - val_loss: 1.2998 - val_accuracy: 0.9859
Epoch 45/100
6/6 [==============================] - 8s 1s/step - loss: 1.3636 - accuracy:
0.8895 - val_loss: 1.2929 - val_accuracy: 1.0000
Epoch 46/100
6/6 [==============================] - 8s 1s/step - loss: 1.3468 - accuracy:
0.8895 - val_loss: 1.2858 - val_accuracy: 1.0000
Epoch 47/100
6/6 [==============================] - 8s 1s/step - loss: 1.3556 - accuracy:
0.8605 - val_loss: 1.2793 - val_accuracy: 1.0000
Epoch 48/100
6/6 [==============================] - 8s 1s/step - loss: 1.3574 - accuracy:
0.8430 - val_loss: 1.2727 - val_accuracy: 1.0000
Epoch 49/100
6/6 [==============================] - 8s 2s/step - loss: 1.3453 - accuracy:
0.8430 - val_loss: 1.2659 - val_accuracy: 1.0000
Epoch 50/100
6/6 [==============================] - 8s 1s/step - loss: 1.3442 - accuracy:
0.8605 - val_loss: 1.2591 - val_accuracy: 1.0000
Epoch 51/100
6/6 [==============================] - 8s 1s/step - loss: 1.3404 - accuracy:
0.8837 - val_loss: 1.2527 - val_accuracy: 1.0000
Epoch 52/100
6/6 [==============================] - 9s 1s/step - loss: 1.3309 - accuracy:
0.8779 - val_loss: 1.2460 - val_accuracy: 1.0000
Epoch 53/100
6/6 [==============================] - 8s 2s/step - loss: 1.3321 - accuracy:
0.8663 - val_loss: 1.2395 - val_accuracy: 1.0000
Epoch 54/100
6/6 [==============================] - 9s 1s/step - loss: 1.3303 - accuracy:
0.8488 - val_loss: 1.2332 - val_accuracy: 1.0000
Epoch 55/100
6/6 [==============================] - 8s 1s/step - loss: 1.3077 - accuracy:
0.9128 - val_loss: 1.2265 - val_accuracy: 1.0000
Epoch 56/100
6/6 [==============================] - 8s 1s/step - loss: 1.3073 - accuracy:
0.8837 - val_loss: 1.2201 - val_accuracy: 1.0000
Epoch 57/100
6/6 [==============================] - 8s 1s/step - loss: 1.3125 - accuracy:
0.8605 - val_loss: 1.2137 - val_accuracy: 1.0000
Epoch 58/100
6/6 [==============================] - 8s 1s/step - loss: 1.2940 - accuracy:
0.8895 - val_loss: 1.2070 - val_accuracy: 1.0000
Epoch 59/100
6/6 [==============================] - 9s 1s/step - loss: 1.2775 - accuracy:
0.9360 - val_loss: 1.2002 - val_accuracy: 1.0000
Epoch 60/100
6/6 [==============================] - 9s 1s/step - loss: 1.2971 - accuracy:
0.9070 - val_loss: 1.1942 - val_accuracy: 1.0000
Epoch 61/100
6/6 [==============================] - 9s 1s/step - loss: 1.2844 - accuracy:
0.8721 - val_loss: 1.1875 - val_accuracy: 1.0000
Epoch 62/100
```

```
6/6 [==============================] - 9s 1s/step - loss: 1.2771 - accuracy:
0.8895 - val_loss: 1.1808 - val_accuracy: 1.0000
Epoch 63/100
6/6 [==============================] - 8s 1s/step - loss: 1.2655 - accuracy:
0.8895 - val_loss: 1.1743 - val_accuracy: 1.0000
Epoch 64/100
6/6 [==============================] - 8s 2s/step - loss: 1.2727 - accuracy:
0.8605 - val_loss: 1.1681 - val_accuracy: 1.0000
Epoch 65/100
6/6 [==============================] - 8s 1s/step - loss: 1.2543 - accuracy:
0.9012 - val_loss: 1.1615 - val_accuracy: 1.0000
Epoch 66/100
6/6 [==============================] - 8s 1s/step - loss: 1.2544 - accuracy:
0.8895 - val_loss: 1.1552 - val_accuracy: 1.0000
Epoch 67/100
6/6 [==============================] - 9s 1s/step - loss: 1.2589 - accuracy:
0.8721 - val_loss: 1.1488 - val_accuracy: 1.0000
Epoch 68/100
6/6 [==============================] - 8s 1s/step - loss: 1.2587 - accuracy:
0.8605 - val_loss: 1.1430 - val_accuracy: 1.0000
Epoch 69/100
6/6 [==============================] - 8s 1s/step - loss: 1.2384 - accuracy:
0.8895 - val_loss: 1.1366 - val_accuracy: 1.0000
Epoch 70/100
6/6 [==============================] - 9s 1s/step - loss: 1.2567 - accuracy:
0.8547 - val_loss: 1.1308 - val_accuracy: 1.0000
Epoch 71/100
6/6 [==============================] - 8s 1s/step - loss: 1.2296 - accuracy:
0.8953 - val_loss: 1.1246 - val_accuracy: 1.0000
Epoch 72/100
6/6 [==============================] - 8s 1s/step - loss: 1.2467 - accuracy:
0.8547 - val_loss: 1.1189 - val_accuracy: 1.0000
Epoch 73/100
6/6 [==============================] - 9s 1s/step - loss: 1.2290 - accuracy:
0.8895 - val_loss: 1.1127 - val_accuracy: 1.0000
Epoch 74/100
6/6 [==============================] - 9s 1s/step - loss: 1.2254 - accuracy:
0.8837 - val_loss: 1.1065 - val_accuracy: 1.0000
Epoch 75/100
6/6 [==============================] - 8s 1s/step - loss: 1.2300 - accuracy:
0.8837 - val_loss: 1.1004 - val_accuracy: 1.0000
Epoch 76/100
6/6 [==============================] - 8s 1s/step - loss: 1.2295 - accuracy:
0.8895 - val_loss: 1.0943 - val_accuracy: 1.0000
Epoch 77/100
6/6 [==============================] - 8s 1s/step - loss: 1.1964 - accuracy:
0.8721 - val_loss: 1.0876 - val_accuracy: 1.0000
Epoch 78/100
6/6 [==============================] - 8s 1s/step - loss: 1.2048 - accuracy:
0.8779 - val_loss: 1.0814 - val_accuracy: 1.0000
Epoch 79/100
6/6 [==============================] - 9s 1s/step - loss: 1.1952 - accuracy:
0.8779 - val_loss: 1.0753 - val_accuracy: 1.0000
```

```
Epoch 80/100
6/6 [==============================] - 8s 1s/step - loss: 1.2062 - accuracy:
0.8547 - val_loss: 1.0695 - val_accuracy: 1.0000
Epoch 81/100
6/6 [==============================] - 9s 1s/step - loss: 1.1811 - accuracy:
0.8953 - val_loss: 1.0630 - val_accuracy: 1.0000
Epoch 82/100
6/6 [==============================] - 8s 1s/step - loss: 1.1802 - accuracy:
0.8547 - val_loss: 1.0568 - val_accuracy: 1.0000
Epoch 83/100
6/6 [==============================] - 9s 1s/step - loss: 1.1870 - accuracy:
0.8953 - val_loss: 1.0510 - val_accuracy: 1.0000
Epoch 84/100
6/6 [==============================] - 9s 1s/step - loss: 1.1721 - accuracy:
0.8779 - val_loss: 1.0449 - val_accuracy: 1.0000
Epoch 85/100
6/6 [==============================] - 8s 1s/step - loss: 1.1716 - accuracy:
0.9012 - val_loss: 1.0389 - val_accuracy: 1.0000
Epoch 86/100
6/6 [==============================] - 9s 1s/step - loss: 1.1553 - accuracy:
0.9128 - val_loss: 1.0327 - val_accuracy: 1.0000
Epoch 87/100
6/6 [==============================] - 9s 1s/step - loss: 1.1670 - accuracy:
0.8605 - val_loss: 1.0266 - val_accuracy: 1.0000
Epoch 88/100
6/6 [==============================] - 8s 1s/step - loss: 1.1527 - accuracy:
0.8779 - val_loss: 1.0205 - val_accuracy: 1.0000
Epoch 89/100
6/6 [==============================] - 8s 1s/step - loss: 1.1558 - accuracy:
0.8953 - val_loss: 1.0147 - val_accuracy: 1.0000
Epoch 90/100
6/6 [==============================] - 8s 1s/step - loss: 1.1603 - accuracy:
0.8779 - val_loss: 1.0091 - val_accuracy: 1.0000
Epoch 91/100
6/6 [==============================] - 8s 1s/step - loss: 1.1419 - accuracy:
0.8779 - val_loss: 1.0032 - val_accuracy: 1.0000
Epoch 92/100
6/6 [==============================] - 9s 1s/step - loss: 1.1394 - accuracy:
0.9070 - val_loss: 0.9972 - val_accuracy: 1.0000
Epoch 93/100
6/6 [==============================] - 8s 1s/step - loss: 1.1286 - accuracy:
0.9012 - val_loss: 0.9913 - val_accuracy: 1.0000
Epoch 94/100
6/6 [==============================] - 8s 1s/step - loss: 1.1278 - accuracy:
0.8779 - val_loss: 0.9852 - val_accuracy: 1.0000
Epoch 95/100
6/6 [==============================] - 8s 1s/step - loss: 1.1300 - accuracy:
0.8547 - val_loss: 0.9793 - val_accuracy: 1.0000
Epoch 96/100
6/6 [==============================] - 8s 1s/step - loss: 1.1197 - accuracy:
0.8953 - val_loss: 0.9735 - val_accuracy: 1.0000
Epoch 97/100
6/6 [==============================] - 9s 1s/step - loss: 1.1065 - accuracy:
```

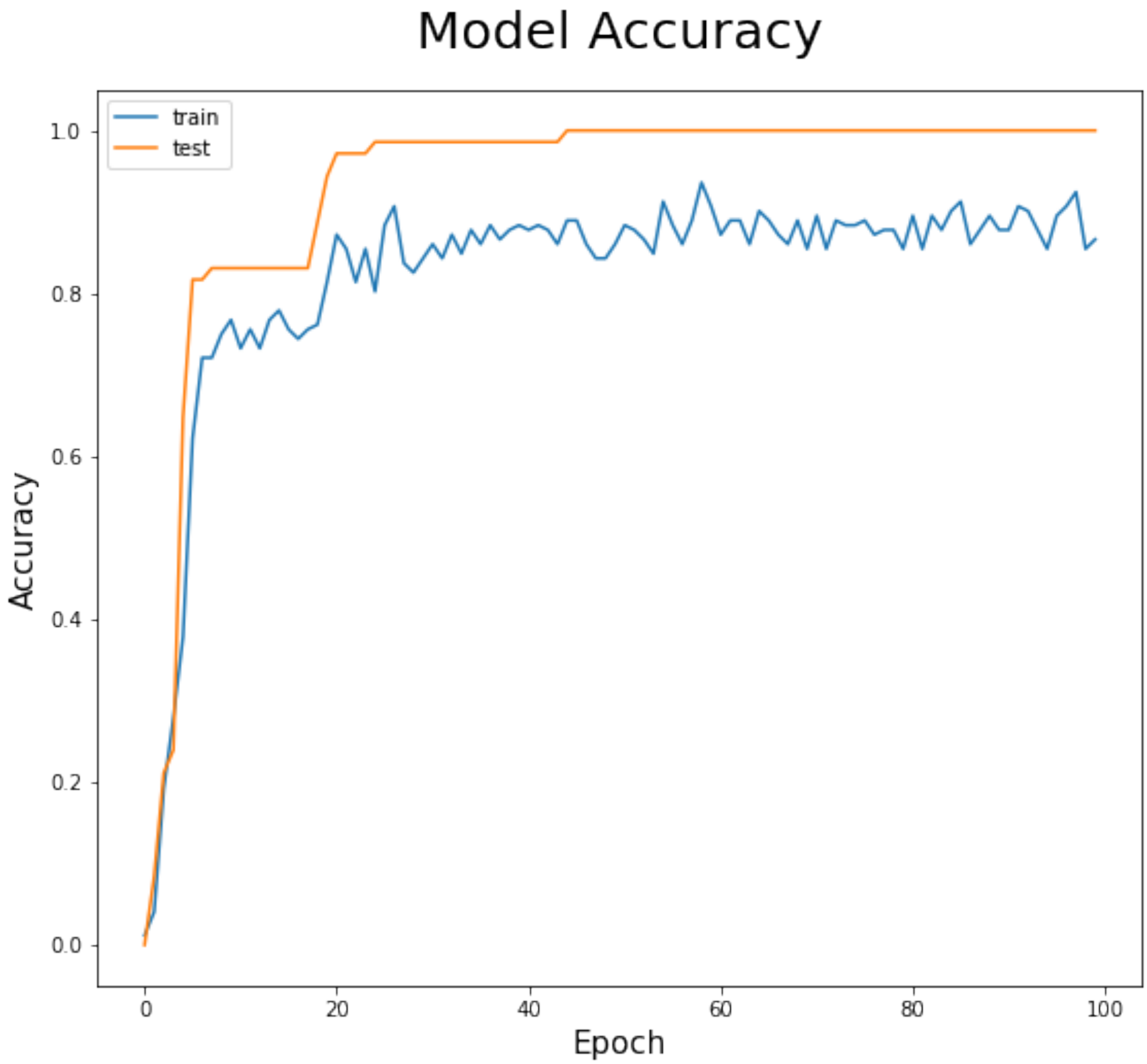
```
0.9070 - val_loss: 0.9676 - val_accuracy: 1.0000
Epoch 98/100
6/6 [==============================] - 9s 1s/step - loss: 1.1066 - accuracy:
0.9244 - val_loss: 0.9617 - val_accuracy: 1.0000
Epoch 99/100
6/6 [==============================] - 8s 1s/step - loss: 1.1070 - accuracy:
0.8547 - val_loss: 0.9560 - val_accuracy: 1.0000
Epoch 100/100
6/6 [==============================] - 9s 1s/step - loss: 1.1140 - accuracy:
0.8663 - val_loss: 0.9505 - val_accuracy: 1.0000
```

In [75]:

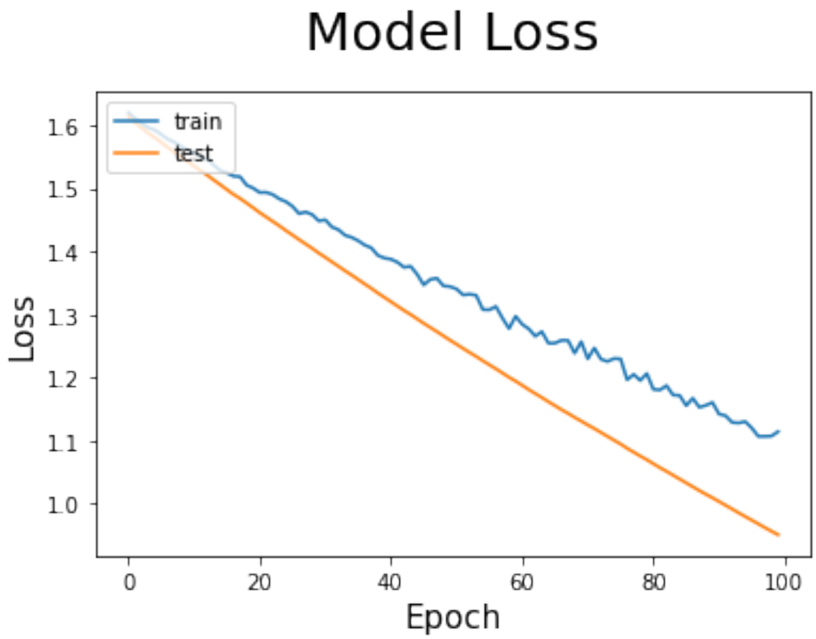
```
# Plotting the Model Accuracy & Model Loss vs Epochs (Hidden Input)
plt.figure(figsize=[20,8])

# 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[75]:

```
<matplotlib.legend.Legend at 0x2b88dcee0>
```




In [76]:
```
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 [77]:
```
y_pred3 = model.predict(x_test_n).argmax(1)
y_true = y_test
```

```
2022-07-06 09:00:20.559243: I tensorflow/core/grappler/optimizers/custom_gra
ph_optimizer_registry.cc:112] Plugin optimizer for device_type GPU is enable
d.
WARNING:tensorflow:5 out of the last 7 calls to <function Model.make_predict
_function.<locals>.predict_function at 0x1608ff280> triggered tf.function re
tracing. Tracing is expensive and the excessive number of tracings could be
due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors w
ith different shapes, (3) passing Python objects instead of tensors. For (1)
, please define your @tf.function outside of the loop. For (2), @tf.function
has experimental_relax_shapes=True option that relaxes argument shapes that
can avoid unnecessary retracing. For (3), please refer to https://www.tensor
flow.org/guide/function#controlling_retracing and https://www.tensorflow.org
/api_docs/python/tf/function for  more details.
```

In [78]:
```
classes = ['Cheetah', 'Leopard', 'Lion', 'Puma', 'Tiger']
```

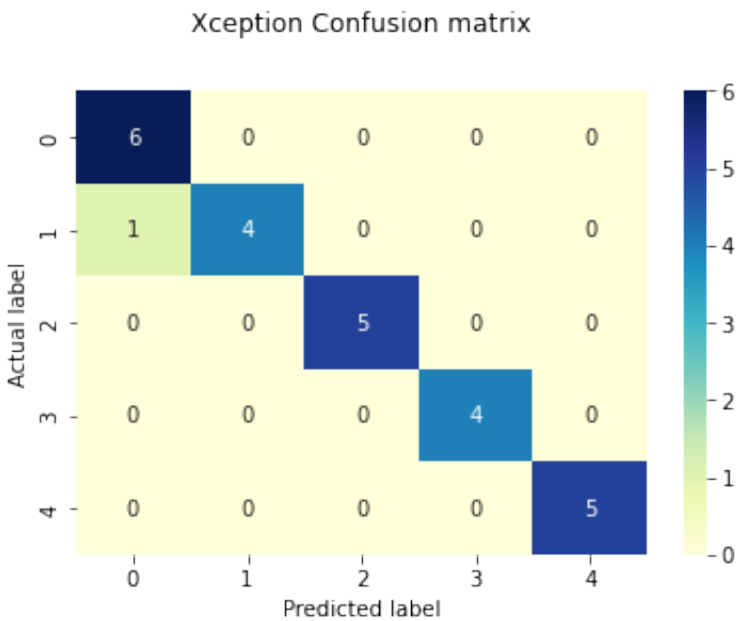
In [79]:
```
import seaborn as sns
from sklearn.metrics import accuracy_score,f1_score,confusion_matrix,classif
```

In [80]:
```
cm = confusion_matrix(y_true,y_pred3)
```

El modelo con Xception parece ser uno de los mejores, almenos con los datos que ya ha visto nuestro modelo, solo tuvo un error el cual fue confundir un leopardo con un chita. Lo demás estuvo perfecto.

In [81]:
```
p = sns.heatmap(pd.DataFrame(cm), annot=True, cmap="YlGnBu" ,fmt='g')
plt.title('Xception Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

Out[81]:
```
Text(0.5, 15.0, 'Predicted label')
```




El reporte de clasificación confirma lo anteriormente mencionado. Obtiene un 95% de accuracy y los valores más bajos de F1-Score fueron los del chita y el leopardo.

In [82]:
```
print(classification_report(y_true,y_pred3))
print(accuracy_score(y_true,y_pred3))
```

```
              precision    recall  f1-score   support

           0       0.86      1.00      0.92         6
           1       1.00      0.80      0.89         5
           2       1.00      1.00      1.00         5
           3       1.00      1.00      1.00         4
           4       1.00      1.00      1.00         5

    accuracy                           0.96        25
   macro avg       0.97      0.96      0.96        25
weighted avg       0.97      0.96      0.96        25

0.96
```

# Validation Xception

In [83]:
```
y_pred3 = model.predict(x_val).argmax(1)
y_true = y_val
```

```
2022-07-06 09:00:21.619177: I tensorflow/core/grappler/optimizers/custom_gra
ph_optimizer_registry.cc:112] Plugin optimizer for device_type GPU is enable
d.
WARNING:tensorflow:6 out of the last 8 calls to <function Model.make_predict
_function.<locals>.predict_function at 0x1608ff280> triggered tf.function re
tracing. Tracing is expensive and the excessive number of tracings could be
due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors w
ith different shapes, (3) passing Python objects instead of tensors. For (1)
, please define your @tf.function outside of the loop. For (2), @tf.function
has experimental_relax_shapes=True option that relaxes argument shapes that
can avoid unnecessary retracing. For (3), please refer to https://www.tensor
flow.org/guide/function#controlling_retracing and https://www.tensorflow.org
/api_docs/python/tf/function for  more details.
```

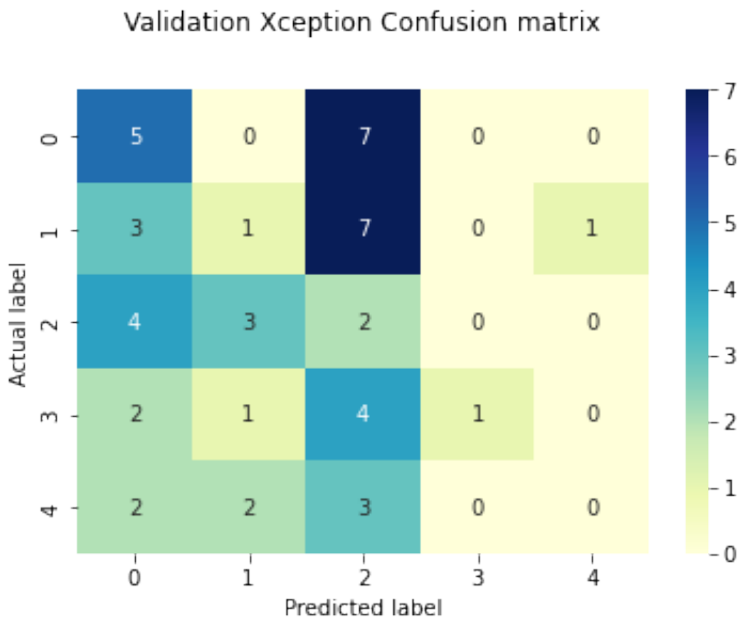
In [84]:
```
classes = ['Cheetah', 'Leopard', 'Lion', 'Puma', 'Tiger']
```

In [85]:
```
cm = confusion_matrix(y_true,y_pred3)
```

Sorprendentemente el Xception obtuvo pésimos resultados clasificando datos que no había visto. Esta es la importancia de guardar unos cuantos datos para evaluar el modelo final. Aparentemente se sesga clasificando chita, leopardo y león. Trata de ubicar los demás felinos en esas categorías.

In [86]:
```
p = sns.heatmap(pd.DataFrame(cm), annot=True, cmap="YlGnBu" ,fmt='g')
plt.title('Validation Xception Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

Out[86]:
```
Text(0.5, 15.0, 'Predicted label')
```




Su reporte de clasificación nos indica lo anteriormente mencionado, aquí pasa lo contrario en la clase de puma, tiene 100% de precisión pero un f1-score muy bajo.

In [88]:
```
print(classification_report(y_true,y_pred3))
print(accuracy_score(y_true,y_pred3))
```

```
              precision    recall  f1-score   support

           0       0.31      0.42      0.36        12
           1       0.14      0.08      0.11        12
           2       0.09      0.22      0.12         9
           3       1.00      0.12      0.22         8
           4       0.00      0.00      0.00         7

    accuracy                           0.19        48
   macro avg       0.31      0.17      0.16        48
weighted avg       0.30      0.19      0.18        48

0.1875
```

# Conclusión

La clasificación de imágenes definitivamente no es una tarea fácil, hay una infinidad de combinaciones las cuales probar. Para poder trabajar con imágenes de una forma adecuada se debe de estar bien documentado, un punto importante a destacar es que la experiencia nos da un norte hacia donde ir en esta parte de las imágenes. Comparado con los análisis de clasificación de datos estructurados o de texto, en las imágenes se tiene una gran cantidad de posibilidades. Algo a destacar en este notebook es la importancia de tener un conjunto de validación el cual no ha visto el modelo, al tener este podemos evaluar nuestro modelo realmente. Como en el caso de la Xception el cual aparentemente tenía unos resultados maravillosos cuando en realidad estaba muy sesgado. El mejor modelo fue el de la VGG16 el cual obtuvo buenos resultados, los cuales no estaban sesgados debido a que funcionaba tanto en el conjunto de prueba como con los de validación. El transfer learning es una herramienta bastante útil y es una solución que nos saca de apuros hablando de la cantidad de datos recolectados y necesarios para que nuestro modelo se desempeñe de la mejor manera. También otra punto a destacar es que la VGG19 a pesar de tener más capas y tener una estructura más “robusta” obtuvo peores resultados que la VGG16, el tener una estructura o arquitectura más compleja no es garantía para un buen resultado. Es importante comparar resultados entre las distintas arquitecturas de transfer learning para de este modo seleccionar los mejores resultados para el miniproyecto de esta fase, en este notebook comprobamos que a pesar de tener la misma fuente de transfer learning, cada uno de los modelos tiene un desempeño diferente para la clasificación de felinos.

# 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
- Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. (2018). A Survey on Deep Transfer Learning. pringer Nature ,270-279.

In [ ]: