Importing the libraries

```
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Input
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import TerminateOnNaN
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
import tensorflow as tf
import pandas as pd
import numpy as np
import cv2 as cv
import os
```

Activating GPU

```
gpus = tf.config.list_physical_devices('GPU')
if gpus:
  try:
    # Currently, memory growth needs to be the same across GPUs
    for gpu in gpus:
      tf.config.experimental.set_memory_growth(gpu, True)
    logical_gpus = tf.config.list_logical_devices('GPU')
    print(len(gpus), "Physical GPUs,", len(logical_gpus), "Logical GPUs")
  except RuntimeError as e:
    # Memory growth must be set before GPUs have been initialized
    print(e)
```

```
1 Physical GPUs, 1 Logical GPUs
```

```
# Reads the image from the given path and store data in the apporpriate lists

def read_image(path, images, labels, filenames):

    for root, dirs, files in os.walk(path):

        for name in dirs:

            direct = os.path.join(path, name)

            for filename in os.listdir(direct):

                img = cv.imread(os.path.join(path + "/" + name, filename))
                labels.append(name)
                img = cv.resize(img, Image_Size)
                images.append(img)
                filenames.append(name + '/' + filename)
```

```
# Declaring image size

Image_Size = (224,224)
```

```
# Reading images from the storage

path = './animals/'
labels = []
filenames = []
images = []
```

```
read_image(path=path, images=images, labels=labels, filenames=filenames)
```

```
# Making df from the images read
df = pd.DataFrame({
    'filename' : filenames,
    'category' : labels
    })
```

```
# Deleting the list to save space as they aren't needed

del labels
del filenames
del images


# List of the categories

print(df['category'].unique())
```

```
['antelope' 'badger' 'bat' 'bear' 'bee' 'beetle' 'bison' 'boar'
 'butterfly' 'cat' 'caterpillar' 'chimpanzee' 'cockroach' 'cow' 'coyote'
 'crab' 'crow' 'deer' 'dog' 'dolphin' 'donkey' 'dragonfly' 'duck' 'eagle'
 'elephant' 'flamingo' 'fly' 'fox' 'goat' 'goldfish' 'goose' 'gorilla'
 'grasshopper' 'hamster' 'hare' 'hedgehog' 'hippopotamus' 'hornbill'
 'horse' 'hummingbird' 'hyena' 'jellyfish' 'kangaroo' 'koala' 'ladybugs'
 'leopard' 'lion' 'lizard' 'lobster' 'mosquito' 'moth' 'mouse' 'octopus'
 'okapi' 'orangutan' 'otter' 'owl' 'ox' 'oyster' 'panda' 'parrot'
 'pelecaniformes' 'penguin' 'pig' 'pigeon' 'porcupine' 'possum' 'raccoon'
 'rat' 'reindeer' 'rhinoceros' 'sandpiper' 'seahorse' 'seal' 'shark'
 'sheep' 'snake' 'sparrow' 'squid' 'squirrel' 'starfish' 'swan' 'tiger'
 'turkey' 'turtle' 'whale' 'wolf' 'wombat' 'woodpecker' 'zebra']
```

```
# Spliting the dataset to train and val

train, val = train_test_split(df, test_size=0.3)


# Create an instance of the ImageDataGenerator with desired augmentation parameters
datagen = ImageDataGenerator(
    rotation_range=20,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    rescale=1./255,
    preprocessing_function=lambda image: tf.image.resize(image, Image_Size)
)

# Apply data augmentation to your training data
augmented_images = datagen.flow_from_dataframe(
                            dataframe=train, directory=path,
                            x_col='filename',
                            y_col='category',
                            target_size=Image_Size
)

test_gen = ImageDataGenerator(
    rescale=1./255,
    preprocessing_function=lambda image: tf.image.resize(image, (224, 224))
)

test_images = test_gen.flow_from_dataframe(
        dataframe=val,
        directory=path,
        x_col='filename',
        y_col='category',
        target_size=Image_Size
)
```

```
Found 3780 validated image filenames belonging to 90 classes.
Found 1620 validated image filenames belonging to 90 classes.
```

```
# list of classes
```

```
test_images.class_indices
```

```
{'antelope': 0,
 'badger': 1,
 'bat': 2,
 'bear': 3,
 'bee': 4,
 'beetle': 5,
 'bison': 6,
 'boar': 7,
 'butterfly': 8,
 'cat': 9,
 'caterpillar': 10,
 'chimpanzee': 11,
 'cockroach': 12,
 'cow': 13,
 'coyote': 14,
 'crab': 15,
 'crow': 16,
 'deer': 17,
 'dog': 18,
 'dolphin': 19,
 'donkey': 20,
 'dragonfly': 21,
 'duck': 22,
 'eagle': 23,
 'elephant': 24,
 'flamingo': 25,
 'fly': 26,
 'fox': 27,
 'goat': 28,
 'goldfish': 29,
 'goose': 30,
 'gorilla': 31,
 'grasshopper': 32,
 'hamster': 33,
 'hare': 34,
 'hedgehog': 35,
 'hippopotamus': 36,
 'hornbill': 37,
 'horse': 38,
 'hummingbird': 39,
 'hyena': 40,
 'jellyfish': 41,
 'kangaroo': 42,
 'koala': 43,
 'ladybugs': 44,
 'leopard': 45,
 'lion': 46,
 'lizard': 47,
 'lobster': 48,
 'mosquito': 49,
 'moth': 50,
 'mouse': 51,
 'octopus': 52,
 'okapi': 53,
 'orangutan': 54,
 'otter': 55,
 'owl': 56,
 'ox': 57,
```

```
# declaring output layer unit

num_classes = len(df['category'].unique())
```

```
# Model creation
model = Sequential()
model.add(Input(shape=(224,224,3)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))
```

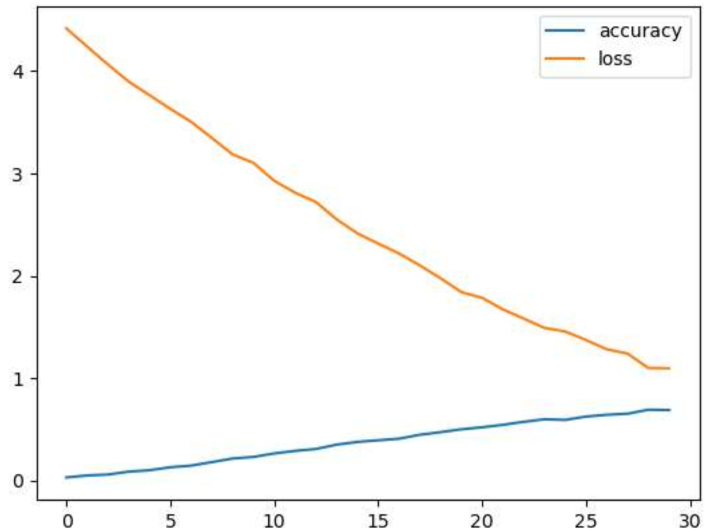
```
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])



# training

history = model.fit(augmented_images, epochs=30, validation_data=test_images, callbacks=[TerminateOnNaN()])
```

```
Epoch 1/30
119/119 [==============================] - 69s 517ms/step - loss: 4.4167 - accuracy: 0.0280 - val_loss: 4.3483 - val_accuracy: 0.0309
Epoch 2/30
119/119 [==============================] - 58s 490ms/step - loss: 4.2387 - accuracy: 0.0476 - val_loss: 4.2159 - val_accuracy: 0.0481
Epoch 3/30
119/119 [==============================] - 61s 511ms/step - loss: 4.0641 - accuracy: 0.0566 - val_loss: 4.0958 - val_accuracy: 0.0574
Epoch 4/30
119/119 [==============================] - 60s 502ms/step - loss: 3.8962 - accuracy: 0.0849 - val_loss: 3.9905 - val_accuracy: 0.0722
Epoch 5/30
119/119 [==============================] - 57s 475ms/step - loss: 3.7637 - accuracy: 0.0987 - val_loss: 3.9312 - val_accuracy: 0.0815
Epoch 6/30
119/119 [==============================] - 61s 509ms/step - loss: 3.6302 - accuracy: 0.1278 - val_loss: 3.8141 - val_accuracy: 0.1093
Epoch 7/30
119/119 [==============================] - 61s 508ms/step - loss: 3.5035 - accuracy: 0.1437 - val_loss: 3.9009 - val_accuracy: 0.1037
Epoch 8/30
119/119 [==============================] - 59s 492ms/step - loss: 3.3456 - accuracy: 0.1783 - val_loss: 3.7654 - val_accuracy: 0.1204
Epoch 9/30
119/119 [==============================] - 59s 494ms/step - loss: 3.1847 - accuracy: 0.2132 - val_loss: 3.7641 - val_accuracy: 0.1451
Epoch 10/30
119/119 [==============================] - 70s 589ms/step - loss: 3.1026 - accuracy: 0.2286 - val_loss: 3.6689 - val_accuracy: 0.1605
Epoch 11/30
119/119 [==============================] - 56s 465ms/step - loss: 2.9276 - accuracy: 0.2630 - val_loss: 3.7253 - val_accuracy: 0.1617
Epoch 12/30
119/119 [==============================] - 59s 498ms/step - loss: 2.8105 - accuracy: 0.2876 - val_loss: 3.5686 - val_accuracy: 0.1840
Epoch 13/30
119/119 [==============================] - 56s 469ms/step - loss: 2.7207 - accuracy: 0.3063 - val_loss: 3.6481 - val_accuracy: 0.1815
Epoch 14/30
119/119 [==============================] - 54s 450ms/step - loss: 2.5513 - accuracy: 0.3484 - val_loss: 3.7047 - val_accuracy: 0.2056
Epoch 15/30
```

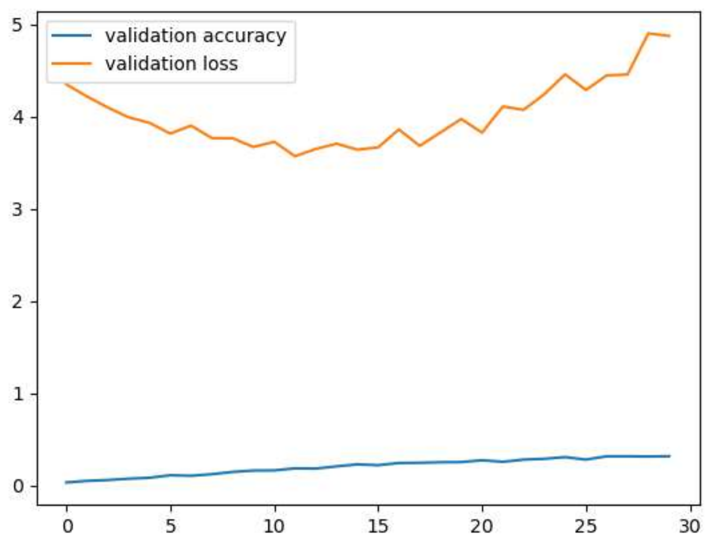
```
# Evaluation

test_loss, test_accuracy = model.evaluate(test_images)

print(f'Test Loss: {test_loss:.4f}')
print(f'Test Accuracy: {test_accuracy:.4f}')
```

```
51/51 [==============================] - 12s 230ms/step - loss: 4.8733 - accuracy: 0.3154
Test Loss: 4.8733
Test Accuracy: 0.3154
```

```
# visualization of accuracy and loss

from matplotlib import pyplot as plt
plt.plot(history.history['accuracy'])
plt.plot(history.history['loss'])
plt.legend(['accuracy', 'loss'])
plt.show()
```




```
from matplotlib import pyplot as plt
plt.plot(history.history['val_accuracy'])
plt.plot(history.history['val_loss'])
plt.legend(['validation accuracy', 'validation loss'])
plt.show()
```



Conclusions

- The loss and validation loss seems to be higher
- The accuracy and the validation accuracy is lower
- We need more amount of dataset to improve the accuracy and reduce the loss