```
import os
import zipfile


# Directory with our training horse pictures
train_horse_dir = os.path.join('/content/drive/My Drive/dataset/train/elephant')
print(train_horse_dir)
# Directory with our training human pictures
train_human_dir = os.path.join('/content/drive/My Drive/dataset/train/no_elephant')

# Directory with our training horse pictures
validation_horse_dir = os.path.join('/content/drive/My Drive/dataset/test/elephant')

# Directory with our training human pictures
validation_human_dir = os.path.join('/content/drive/My Drive/dataset/test/no_elephant')

print('yes')
```

```
/content/drive/My Drive/dataset/train/elephant
yes
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

# Building a Small Model from Scratch

But before we continue, let's start defining the model:

Step 1 will be to import tensorflow.

```
import tensorflow as tf
```

We then add convolutional layers as in the previous example, and flatten the final result to feed into the densely connected layers.

Finally we add the densely connected layers.

Note that because we are facing a two-class classification problem, i.e. a *binary classification problem*, we will end our network with a *sigmoid* activation, so that the output of our network will be a single scalar between 0 and 1, encoding the probability that the current image is class 1 (as opposed to class 0).

```
model = tf.keras.models.Sequential([
    # Note the input shape is the desired size of the image 300x300 with 3 bytes color
    # This is the first convolution
    tf.
    tf.keras.layers.Conv2D(16, (3,3), activation='relu', input_shape=(150,150, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),
    # The second convolution
    tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    # The third convolution
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    # The fourth convolution
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    # The fifth convolution
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    # Flatten the results to feed into a DNN
    tf.keras.layers.Flatten(),
    # 512 neuron hidden layer
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    # Only 1 output neuron. It will contain a value from 0-1 where 0 for 1 class ('horses') a
    tf.keras.layers.Dense(2, activation='sigmoid')
])
```

```
!pip install netron
```

```
     Requirement already satisfied: netron in /usr/local/lib/python3.6/dist-packages (4.7.2)
```

```
netron('/content/drive/My Drive/dataset/elephant_cnn.h5')
```

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_5 (Conv2D)            (None, 148, 148, 16)      448       
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 74, 74, 16)        0         
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 72, 72, 32)        4640      
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 36, 36, 32)        0         
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 34, 34, 64)        18496     
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 17, 17, 64)        0         
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 15, 15, 64)        36928     
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 7, 7, 64)          0         
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 5, 5, 64)          36928     
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 2, 2, 64)          0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 256)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 512)               131584    
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 1026      
=================================================================
Total params: 230,050
Trainable params: 230,050
Non-trainable params: 0
_________________________________________________________________
```

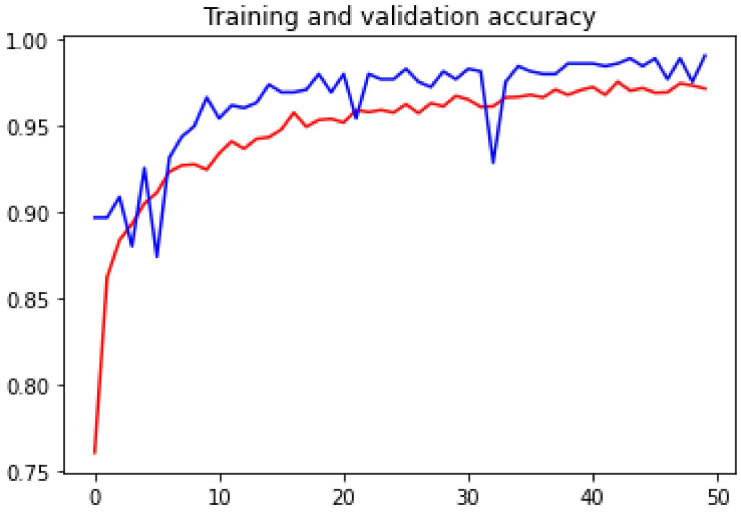
```
from tensorflow.keras.optimizers import RMSprop

model.compile(loss='categorical_crossentropy',
              optimizer=RMSprop(lr=1e-4),
              metrics=['accuracy'])
```

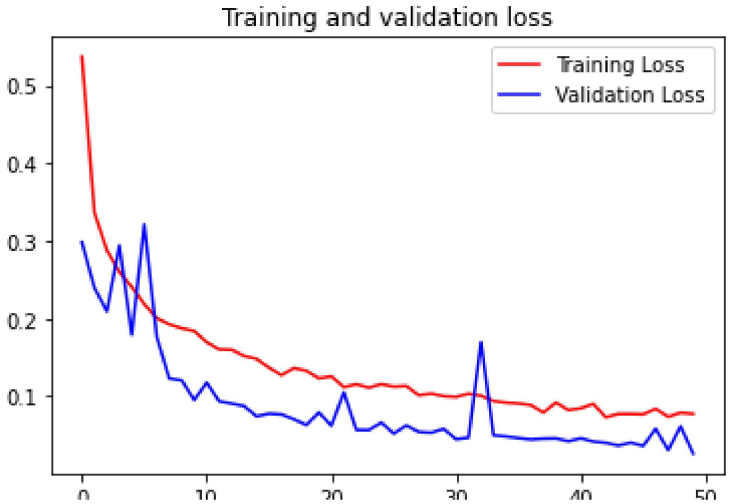
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

import os

# Define our example directories and files
base_dir = '/content/drive/My Drive/dataset'
```

```
train_dir = os.path.join( base_dir, 'train')
validation_dir = os.path.join( base_dir, 'test')


train_cats_dir = os.path.join(train_dir, 'elephant') # Directory with our training cat pictur
train_dogs_dir = os.path.join(train_dir, 'no_elephant') # Directory with our training dog pic
validation_cats_dir = os.path.join(validation_dir, 'elephant') # Directory with our validatic
validation_dogs_dir = os.path.join(validation_dir, 'no_elephant')# Directory with our validat

train_cat_fnames = os.listdir(train_cats_dir)
train_dog_fnames = os.listdir(train_dogs_dir)

# Add our data-augmentation parameters to ImageDataGenerator
train_datagen = ImageDataGenerator(rescale = 1./255.,
                                   rotation_range = 40,
                                   width_shift_range = 0.2,
                                   height_shift_range = 0.2,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)

# Note that the validation data should not be augmented!
test_datagen = ImageDataGenerator( rescale = 1.0/255. ,validation_split=0.2)

# Flow training images in batches of 20 using train_datagen generator
train_generator = train_datagen.flow_from_directory(train_dir,
                                                    batch_size = 32,
                                                    class_mode = 'categorical',
                                                    target_size = (150, 150), subset='trainir

# Flow validation images in batches of 20 using test_datagen generator
validation_generator =  test_datagen.flow_from_directory( train_dir,
                                                          batch_size  = 32,
                                                          class_mode  = 'categorical',
                                                          target_size = (150, 150), subset='v


     Found 3301 images belonging to 2 classes.
     Found 660 images belonging to 2 classes.


history = model.fit(
            train_generator,
            validation_data = validation_generator,
                        epochs = 50, verbose = 1)
model.save('/content/drive/My Drive/dataset/elephant_cnn.h5')
import matplotlib.pyplot as plt
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
```

```
epochs = range(len(acc))

plt.plot(epochs, acc, 'r', label='Training accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')

plt.figure()

plt.plot(epochs, loss, 'r', label='Training Loss')
plt.plot(epochs, val_loss, 'b', label='Validation Loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```

```
104/104 [==============================] - 256s 2s/step - loss: 0.0827 - accuracy: 0.97
Epoch 38/50
104/104 [==============================] - 256s 2s/step - loss: 0.0785 - accuracy: 0.97
Epoch 39/50
104/104 [==============================] - 258s 2s/step - loss: 0.0923 - accuracy: 0.97
Epoch 40/50
104/104 [==============================] - 257s 2s/step - loss: 0.0732 - accuracy: 0.97
Epoch 41/50
104/104 [==============================] - 259s 2s/step - loss: 0.0773 - accuracy: 0.97
Epoch 42/50
104/104 [==============================] - 256s 2s/step - loss: 0.0907 - accuracy: 0.96
Epoch 43/50
104/104 [==============================] - 258s 2s/step - loss: 0.0734 - accuracy: 0.97
Epoch 44/50
104/104 [==============================] - 256s 2s/step - loss: 0.0672 - accuracy: 0.97
Epoch 45/50
104/104 [==============================] - 256s 2s/step - loss: 0.0869 - accuracy: 0.97
Epoch 46/50
104/104 [==============================] - 258s 2s/step - loss: 0.0858 - accuracy: 0.96
Epoch 47/50
104/104 [==============================] - 256s 2s/step - loss: 0.0892 - accuracy: 0.97
Epoch 48/50
104/104 [==============================] - 258s 2s/step - loss: 0.0723 - accuracy: 0.97
Epoch 49/50
104/104 [==============================] - 256s 2s/step - loss: 0.0767 - accuracy: 0.97
Epoch 50/50
104/104 [==============================] - 256s 2s/step - loss: 0.0676 - accuracy: 0.97
```



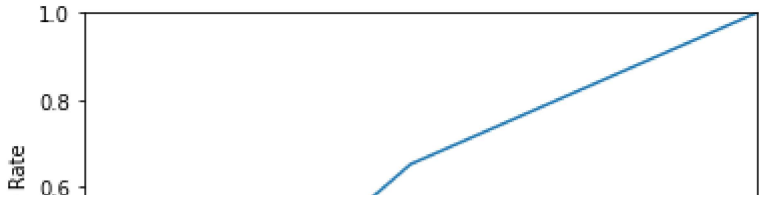
```
from keras.models import load_model
from keras_preprocessing import image
from keras_preprocessing.image import ImageDataGenerator
model = load_model('/content/drive/My Drive/dataset/elephant_cnn.h5')

import numpy as np
import pandas as pd
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_roc_curve
from sklearn.metrics import roc_curve,roc_auc_score



TESTING_DIR = "/content/drive/MyDrive/dataset/test/"
```

```
TESTING_DIR = '/content/drive/MyDrive/dataset/test/'
image_generator = ImageDataGenerator(rescale=1/255)
test_generator =  image_generator.flow_from_directory(batch_size=32,
                                                      directory=TESTING_DIR,
                                                      shuffle=False,
                                                      target_size=(150,150), class_mode='categoric
#test_generator.reset()
no=test_generator.samples


Y_pred = model.predict(test_generator,verbose=1, steps=(round(no/32))+1)
y_pred = np.argmax(Y_pred, axis=1)
print('Confusion Matrix')
print(confusion_matrix(test_generator.classes, y_pred))
print('Classification Report')
target_names = ['elephant','non']
print(classification_report(test_generator.classes, y_pred, target_names=target_names))

fpr , tpr , thresholds = roc_curve ( y_pred , test_generator.classes)

def plot_roc_curve(fpr,tpr):
  plt.plot(fpr,tpr)
  plt.axis([0,1,0,1])
  plt.xlabel('False Positive Rate')
  plt.ylabel('True Positive Rate')
  plt.show()

plot_roc_curve (fpr,tpr)
```

```
Found 1056 images belonging to 2 classes.
33/34 [============================>.] - ETA: 8s WARNING:tensorflow:Your input ran out
34/34 [==============================] - 278s 8s/step
Confusion Matrix
[[504   1]
 [  4 547]]
Classification Report
              precision    recall  f1-score   support
```

```
TESTING_DIR = "/content/drive/MyDrive/dataset/new/"
image_generator = ImageDataGenerator(rescale=1/255)
test_generator =  image_generator.flow_from_directory(batch_size=32,
                                                      directory=TESTING_DIR,
                                                      shuffle=False,
                                                      target_size=(150,150), class_mode='categoric
#test_generator.reset()
no=test_generator.samples


Y_pred = model.predict(test_generator,verbose=1, steps=(round(no/32))+1)

y_pred = np.argmax(Y_pred, axis=1)
print('Confusion Matrix')
print(confusion_matrix(test_generator.classes, y_pred))
print('Classification Report')
target_names = ['elephant','non']
print(classification_report(test_generator.classes, y_pred, target_names=target_names))

fpr , tpr , thresholds = roc_curve ( y_pred , test_generator.classes)

def plot_roc_curve(fpr,tpr):
  plt.plot(fpr,tpr)
  plt.axis([0,1,0,1])
  plt.xlabel('False Positive Rate')
  plt.ylabel('True Positive Rate')
  plt.show()

plot_roc_curve (fpr,tpr)
```

```
Found 56 images belonging to 2 classes.
2/3 [==================>.....] - ETA: 0sWARNING:tensorflow:Your input ran out of
3/3 [==============================] - 0s 69ms/step
Confusion Matrix
[[17  8]
 [16 15]]
Classification Report
              precision    recall  f1-score   support

    elephant       0.52      0.68      0.59        25
         non       0.65      0.48      0.56        31

    accuracy                           0.57        56
   macro avg       0.58      0.58      0.57        56
weighted avg       0.59      0.57      0.57        56
```




```
from keras.models import load_model
from keras_preprocessing import image
import matplotlib.pyplot as plt
from keras_preprocessing.image import ImageDataGenerator
#model = load_model('/content/drive/My Drive/MicheDataset/rps.h5')
model = load_model('/content/drive/My Drive/dataset/elephant_TL.h5')
import numpy as np
import pandas as pd
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import confusion_matrix
TESTING_DIR = "/content/drive/My Drive/dataset/new/"
image_generator = ImageDataGenerator(rescale=1/255)
test_generator =  image_generator.flow_from_directory(batch_size=32,
                                                      directory=TESTING_DIR,
                                                      shuffle=False,
                                                      target_size=(150, 150),
                                                      class_mode='categorical')
test_generator.reset()

test_data_path = TESTING_DIR
# Get the filenames from the generator
fnames = test_generator.filenames

# Get the ground truth from generator
ground_truth = test_generator.classes

# Get the label to class mapping from the generator
label2index = test_generator.class_indices

# Getting the mapping from class index to class label
```

```
idx2label = dict((v,k) for k,v in label2index.items())

# Get the predictions from the model using the generator
predictions = model.predict(test_generator, steps=test_generator.samples/test_generator.batch
predicted_classes = np.argmax(predictions,axis=1)

errors = np.where(predicted_classes != ground_truth)[0]
print("No of errors = {}/{}".format(len(errors),test_generator.samples))

# Show the errors
for i in range(len(errors)):
    pred_class = np.argmax(predictions[errors[i]])
    pred_label = idx2label[pred_class]

    title = 'Original label:{}, Prediction :{}, confidence : {:.3f}'.format(
        fnames[errors[i]].split('/')[0],
        pred_label,
        predictions[errors[i]][pred_class])

    original = image.load_img('{}/{}'.format(test_data_path,fnames[errors[i]]))
    plt.figure(figsize=[7,7])
    plt.axis('off')
    plt.title(title)
    plt.imshow(original)
    plt.show()
correct = np.where(predicted_classes == ground_truth)[0]
print("No of correct classifications = {}/{}".format(len(correct),test_generator.samples))

# Show the correct ones
for i in range(len(correct)):
    pred_class = np.argmax(predictions[correct[i]])
    pred_label = idx2label[pred_class]

    title = 'Original label:{}, Prediction :{}, confidence : {:.3f}'.format(
        fnames[correct[i]].split('/')[0],
        pred_label,
        predictions[correct[i]][pred_class])

    original = image.load_img('{}/{}'.format(test_data_path,fnames[correct[i]]))
    plt.figure(figsize=[7,7])
    plt.axis('off')
    plt.title(title)
    plt.imshow(original)
    plt.show()
cm = confusion_matrix(ground_truth, predicted_classes)
cm

TruePositive = np.diag(cm)
print("TP is" + str(TruePositive) )

FalsePositive = []
for i in range(2):
```

```
for i in range(12):
    FalsePositive.append(sum(cm[:,i]) - cm[i,i])

print("FP is" + str(FalsePositive) )

FalseNegative = []
for i in range(12):
    FalseNegative.append(sum(cm[i,:]) - cm[i,i])

print("FN is" + str(FalseNegative) )

TrueNegative = []
for i in range(12):
    temp = np.delete(cm, i, 0)   # delete ith row
    temp = np.delete(temp, i, 1)  # delete ith column
    TrueNegative.append(sum(sum(temp)))

print("TN is" + str(TrueNegative) )
```