Persian Gulf University
Deep Learning
Spring Semester 2022-2023

Teacher: Dr. Rostami

Ali Behrouzi

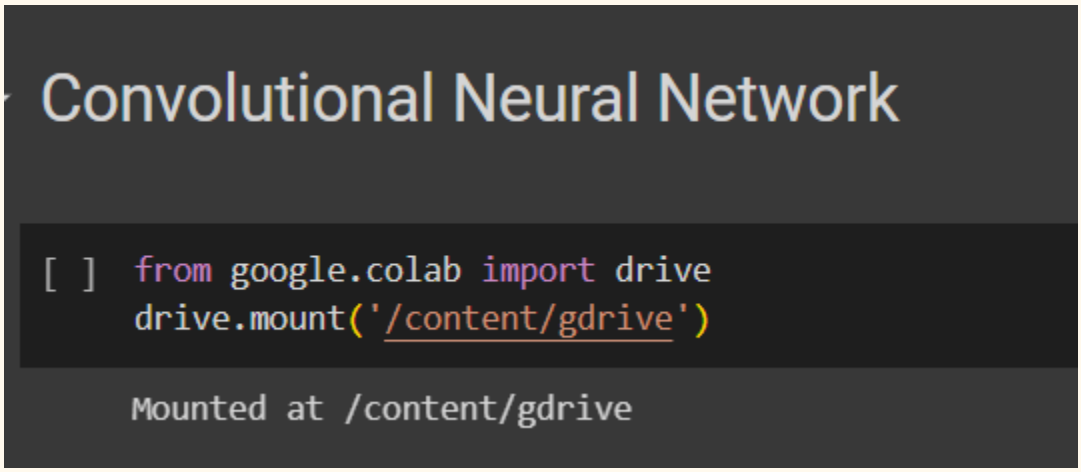
Stdn: 4010724002
Email: alibehrouzi97@gmail.com

Assignment 6
- CNN , cat and dog dataset

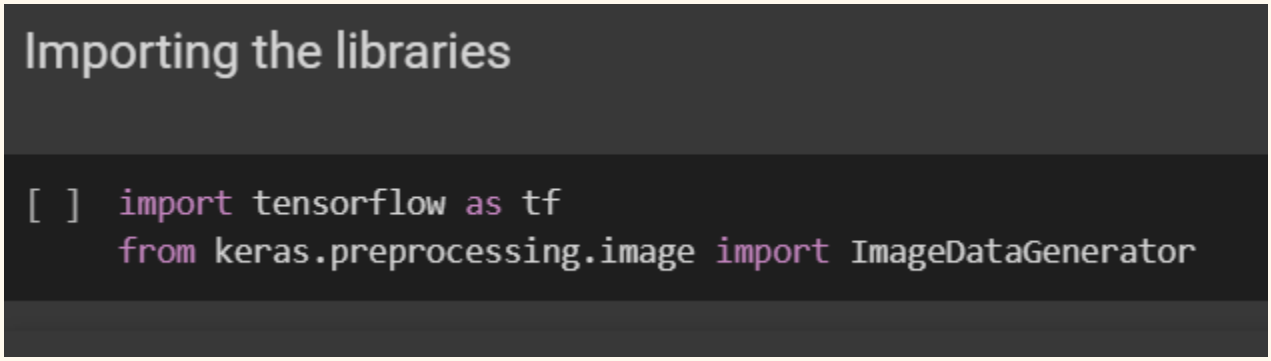
I used a 200 mb database , include training and test folders, each include dog and cat folders. It has been drive from kaggle dataset.

At the end I ended up with 2 piece of code , first doesn't include dropout and just have a slight augmentation , but the second includes dropout layer and more complex augmentation.

I uploaded my dataset to google drive so I needed to mount my drive to access my dataset :

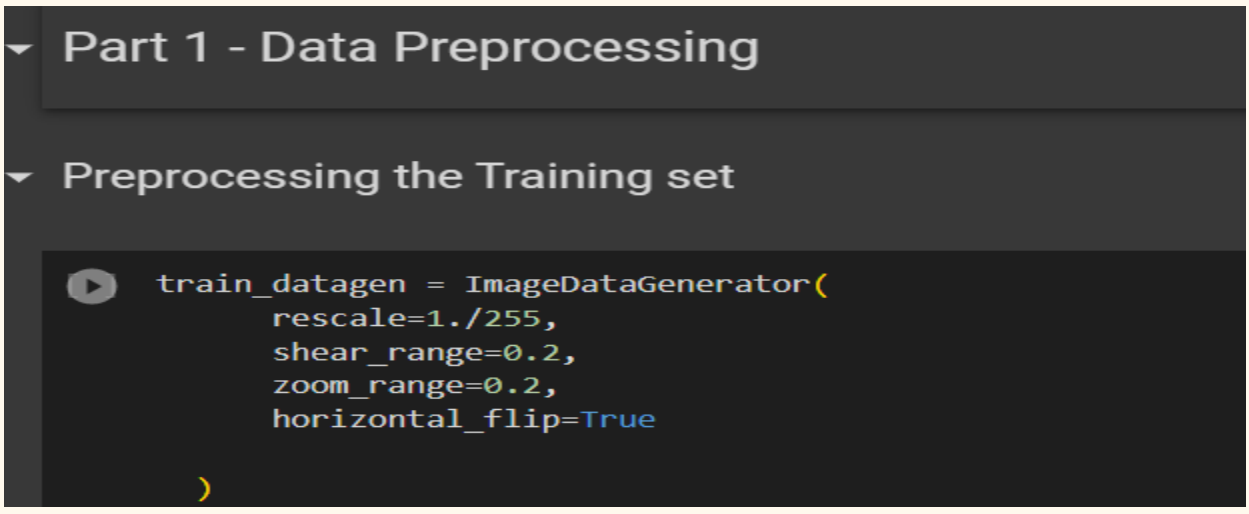


Then import libraries :

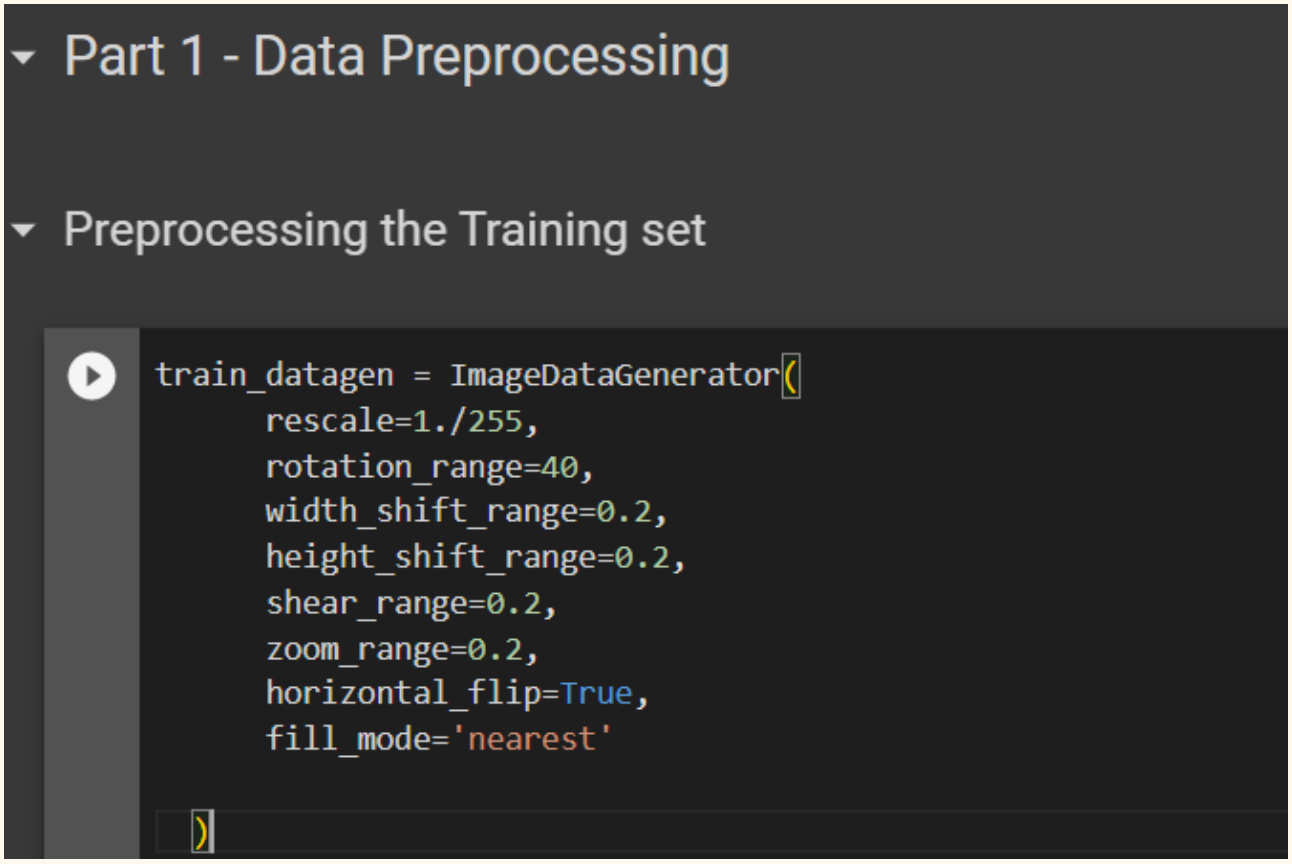


Then we should preprocess the data :

In first code A : I just used normalization and 3 option for data augmentation :



But in code B : I added 4 other augmentation option to the code and the rest is the same :



The we have to tell the program from which source it should get the training images, and also the batch size and if the class mode is binary or categorical:

```
training_set= train_datagen.flow_from_directory(
    '/content/gdrive/My Drive/dataset/training_set',
    target_size=(150,150),
    batch_size=32,
    class_mode='binary'
)
```

```
Found 8000 images belonging to 2 classes.
```

Do the same preprocessing for test data but it just need to be normalized ：

Preprocessing the Test set

```
test_datagen = ImageDataGenerator(
      rescale=1./255)
test_set=test_datagen.flow_from_directory(
    '/content/gdrive/My Drive/dataset/test_set',
    target_size=(150,150),
    batch_size=32,
    class_mode='binary'
)
```

```
Found 2000 images belonging to 2 classes.
```

Start building the cnn : create a sequential model and add a convolutional layer to it the apply maxpooling . you should tell it how many filters you want, and the size of filters , activation function and input size of this layer .stride is (1,1) by default. the size of window for maxpooling should be given to it :

Part 2 - Building the CNN

Initialising the CNN

```
[ ]  cnn =tf.keras.models.Sequential()
```

Step 1 - Convolution

```
[ ]  cnn.add(tf.keras.layers.Conv2D(filters=32,kernel_size=3,activation='relu',input_shape=[150,150,3]))
```

Step 2 - Pooling

```
[ ]  cnn.add(tf.keras.layers.MaxPooling2D((2,2)))
```

We add 3 more conv layers :

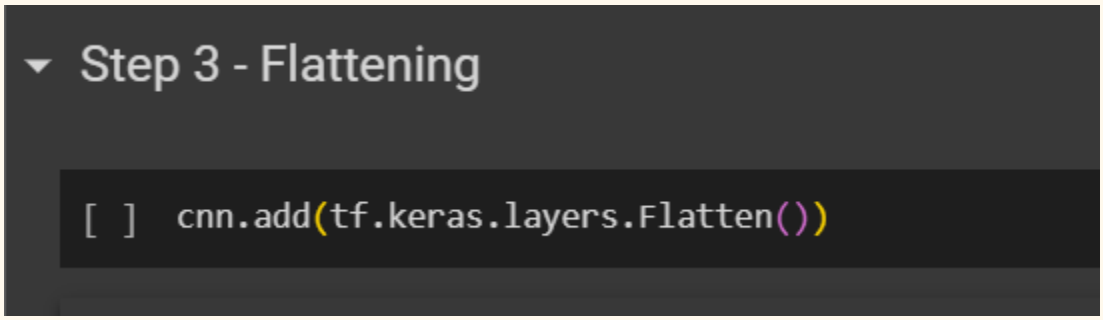
Adding three more convolutional layers

```
[ ]  cnn.add(tf.keras.layers.Conv2D(filters=64,kernel_size=3,activation='relu'))
     cnn.add(tf.keras.layers.MaxPooling2D((2,2)))
```

```
[ ]  cnn.add(tf.keras.layers.Conv2D(filters=128,kernel_size=3,activation='relu'))
     cnn.add(tf.keras.layers.MaxPooling2D((2,2)))
```

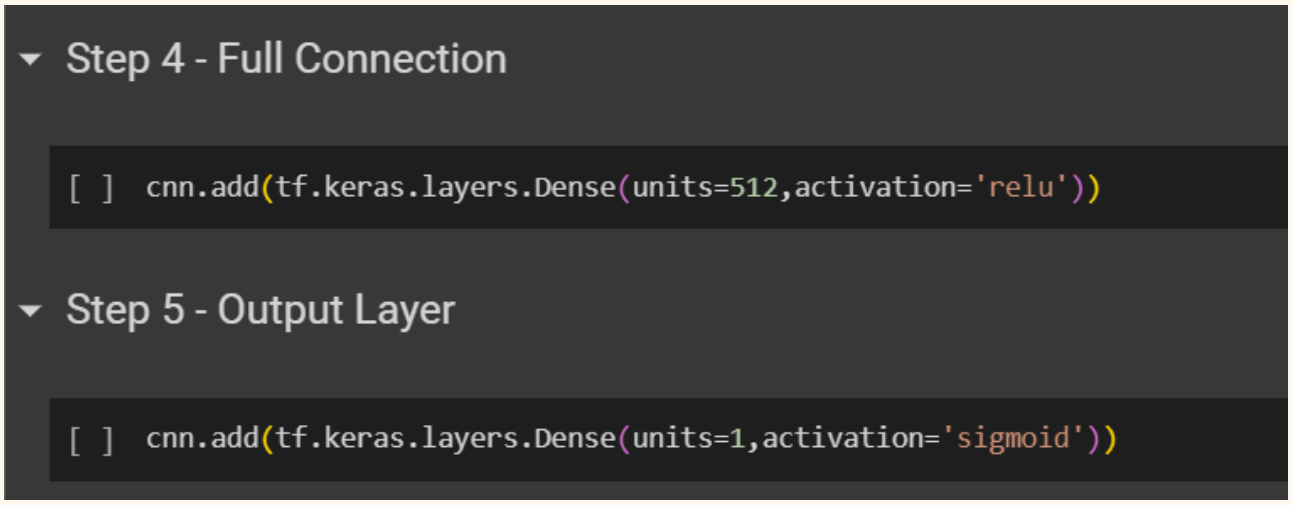
```
[ ]  cnn.add(tf.keras.layers.Conv2D(filters=128,kernel_size=3,activation='relu'))
     cnn.add(tf.keras.layers.MaxPooling2D((2,2)))
```

Then before giving the output of conv layers to dense layer, you should flatten the result :

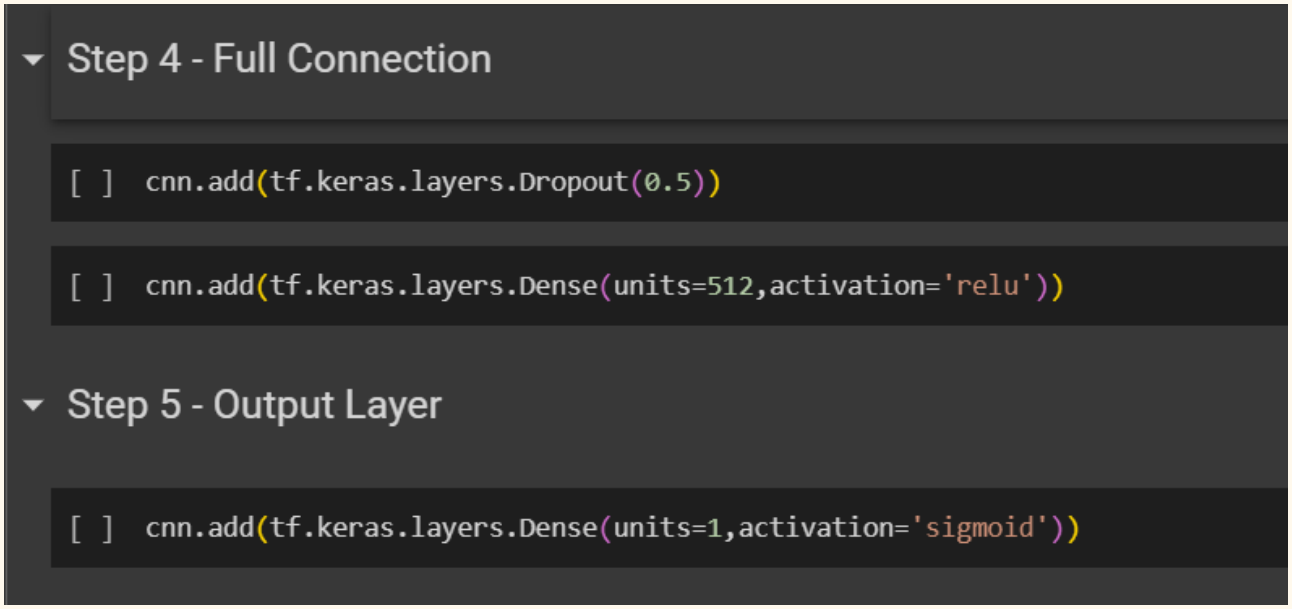


Then we should add dense layer :

In code A we didn't use dropout layer :



But in code B , we added dropout layer for reducing overfitting . we used sigmoid for last layer because it's just binary classification:



Then in training stage we first compile the model with adam optimizer and binary_crossentropy loss function and then call fit function with 30 epochs :

This is the result for code A from epoch 17, it seems from epoch 16 , the val_acc doesn't change significantly but train_acc increases (overfitting) :

```
250/250 [==============================] - 88s 351ms/step - loss: 0.2179 - accuracy: 0.9051 - val_loss: 0.2857 - val_accuracy: 0.8890
Epoch 17/30
250/250 [==============================] - 90s 359ms/step - loss: 0.2030 - accuracy: 0.9145 - val_loss: 0.3195 - val_accuracy: 0.8760
Epoch 18/30
250/250 [==============================] - 88s 351ms/step - loss: 0.1919 - accuracy: 0.9178 - val_loss: 0.3184 - val_accuracy: 0.8780
Epoch 19/30
250/250 [==============================] - 87s 347ms/step - loss: 0.1842 - accuracy: 0.9246 - val_loss: 0.2958 - val_accuracy: 0.8880
Epoch 20/30
250/250 [==============================] - 90s 358ms/step - loss: 0.1667 - accuracy: 0.9340 - val_loss: 0.3244 - val_accuracy: 0.8865
Epoch 21/30
250/250 [==============================] - 87s 349ms/step - loss: 0.1618 - accuracy: 0.9335 - val_loss: 0.3619 - val_accuracy: 0.8705
Epoch 22/30
250/250 [==============================] - 89s 355ms/step - loss: 0.1382 - accuracy: 0.9449 - val_loss: 0.3553 - val_accuracy: 0.8840
Epoch 23/30
250/250 [==============================] - 90s 362ms/step - loss: 0.1302 - accuracy: 0.9488 - val_loss: 0.3663 - val_accuracy: 0.8915
Epoch 24/30
250/250 [==============================] - 86s 346ms/step - loss: 0.1191 - accuracy: 0.9534 - val_loss: 0.3383 - val_accuracy: 0.8850
Epoch 25/30
250/250 [==============================] - 87s 350ms/step - loss: 0.1162 - accuracy: 0.9559 - val_loss: 0.3703 - val_accuracy: 0.8820
Epoch 26/30
250/250 [==============================] - 89s 355ms/step - loss: 0.1049 - accuracy: 0.9607 - val_loss: 0.3567 - val_accuracy: 0.8985
Epoch 27/30
250/250 [==============================] - 92s 368ms/step - loss: 0.0985 - accuracy: 0.9605 - val_loss: 0.4422 - val_accuracy: 0.8810
Epoch 28/30
250/250 [==============================] - 89s 358ms/step - loss: 0.0987 - accuracy: 0.9619 - val_loss: 0.3861 - val_accuracy: 0.8950
Epoch 29/30
250/250 [==============================] - 90s 360ms/step - loss: 0.0911 - accuracy: 0.9672 - val_loss: 0.4002 - val_accuracy: 0.8880
Epoch 30/30
250/250 [==============================] - 87s 349ms/step - loss: 0.0862 - accuracy: 0.9686 - val_loss: 0.4878 - val_accuracy: 0.8750
```

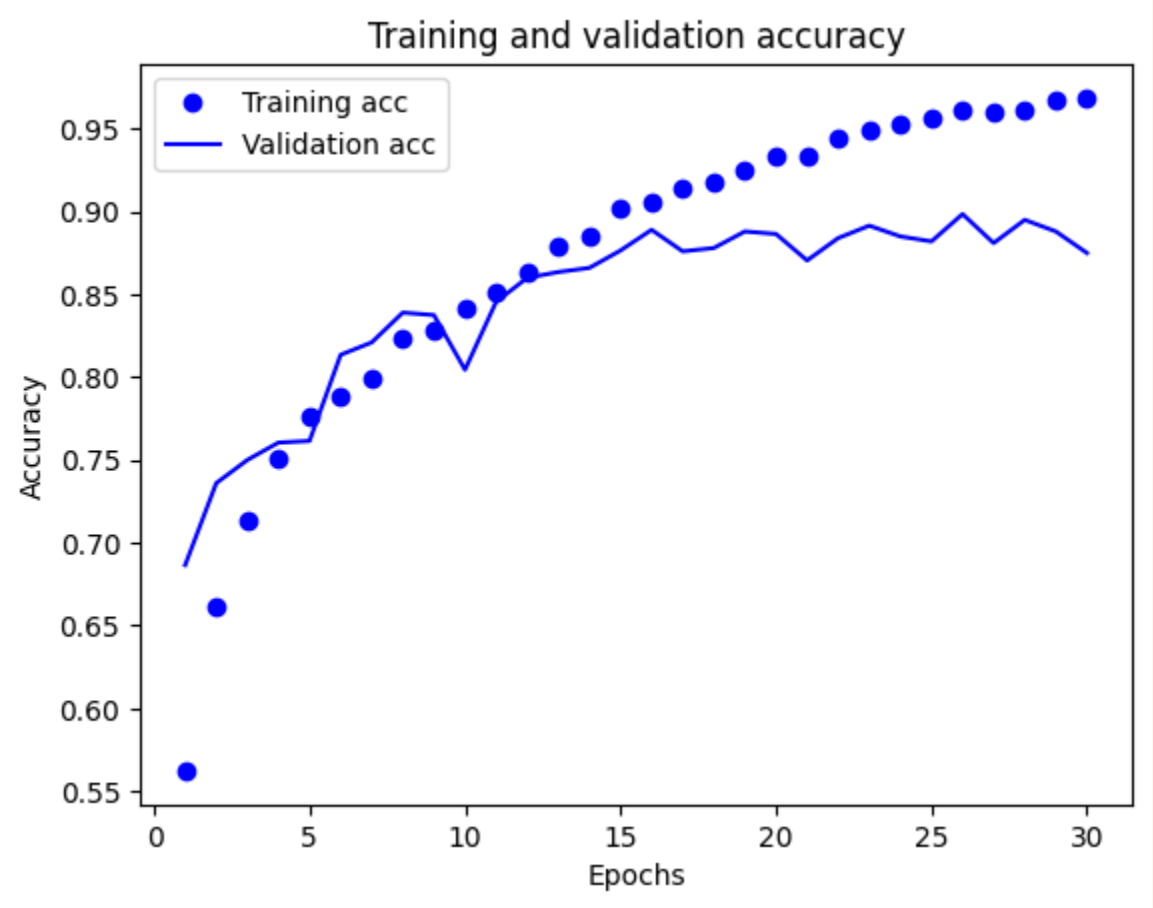
And this is the result of code B from epoch 16 :

```
250/250 [==============================] - 672s 3s/step - loss: 0.4952 - accuracy: 0.7586 - val_loss: 0.4114 - val_accuracy: 0.8145
Epoch 16/30
250/250 [==============================] - 706s 3s/step - loss: 0.4806 - accuracy: 0.7755 - val_loss: 0.4154 - val_accuracy: 0.8120
Epoch 17/30
250/250 [==============================] - 695s 3s/step - loss: 0.4782 - accuracy: 0.7707 - val_loss: 0.3855 - val_accuracy: 0.8260
Epoch 18/30
250/250 [==============================] - 699s 3s/step - loss: 0.4658 - accuracy: 0.7824 - val_loss: 0.3559 - val_accuracy: 0.8520
Epoch 19/30
250/250 [==============================] - 660s 3s/step - loss: 0.4446 - accuracy: 0.7904 - val_loss: 0.3768 - val_accuracy: 0.8340
Epoch 20/30
250/250 [==============================] - 672s 3s/step - loss: 0.4349 - accuracy: 0.7991 - val_loss: 0.3481 - val_accuracy: 0.8540
Epoch 21/30
250/250 [==============================] - 709s 3s/step - loss: 0.4352 - accuracy: 0.7991 - val_loss: 0.3348 - val_accuracy: 0.8575
Epoch 22/30
250/250 [==============================] - 658s 3s/step - loss: 0.4220 - accuracy: 0.8062 - val_loss: 0.3797 - val_accuracy: 0.8275
Epoch 23/30
250/250 [==============================] - 694s 3s/step - loss: 0.4228 - accuracy: 0.8050 - val_loss: 0.3277 - val_accuracy: 0.8570
Epoch 24/30
250/250 [==============================] - 677s 3s/step - loss: 0.4059 - accuracy: 0.8141 - val_loss: 0.3135 - val_accuracy: 0.8680
Epoch 25/30
250/250 [==============================] - 624s 2s/step - loss: 0.4070 - accuracy: 0.8175 - val_loss: 0.3105 - val_accuracy: 0.8620
Epoch 26/30
250/250 [==============================] - 670s 3s/step - loss: 0.3967 - accuracy: 0.8164 - val_loss: 0.3377 - val_accuracy: 0.8490
Epoch 27/30
250/250 [==============================] - 680s 3s/step - loss: 0.3925 - accuracy: 0.8220 - val_loss: 0.3170 - val_accuracy: 0.8745
Epoch 28/30
250/250 [==============================] - 699s 3s/step - loss: 0.3813 - accuracy: 0.8290 - val_loss: 0.3301 - val_accuracy: 0.8510
Epoch 29/30
250/250 [==============================] - 662s 3s/step - loss: 0.3776 - accuracy: 0.8296 - val_loss: 0.2951 - val_accuracy: 0.8720
Epoch 30/30
250/250 [==============================] - 635s 3s/step - loss: 0.3696 - accuracy: 0.8321 - val_loss: 0.3148 - val_accuracy: 0.8675
```

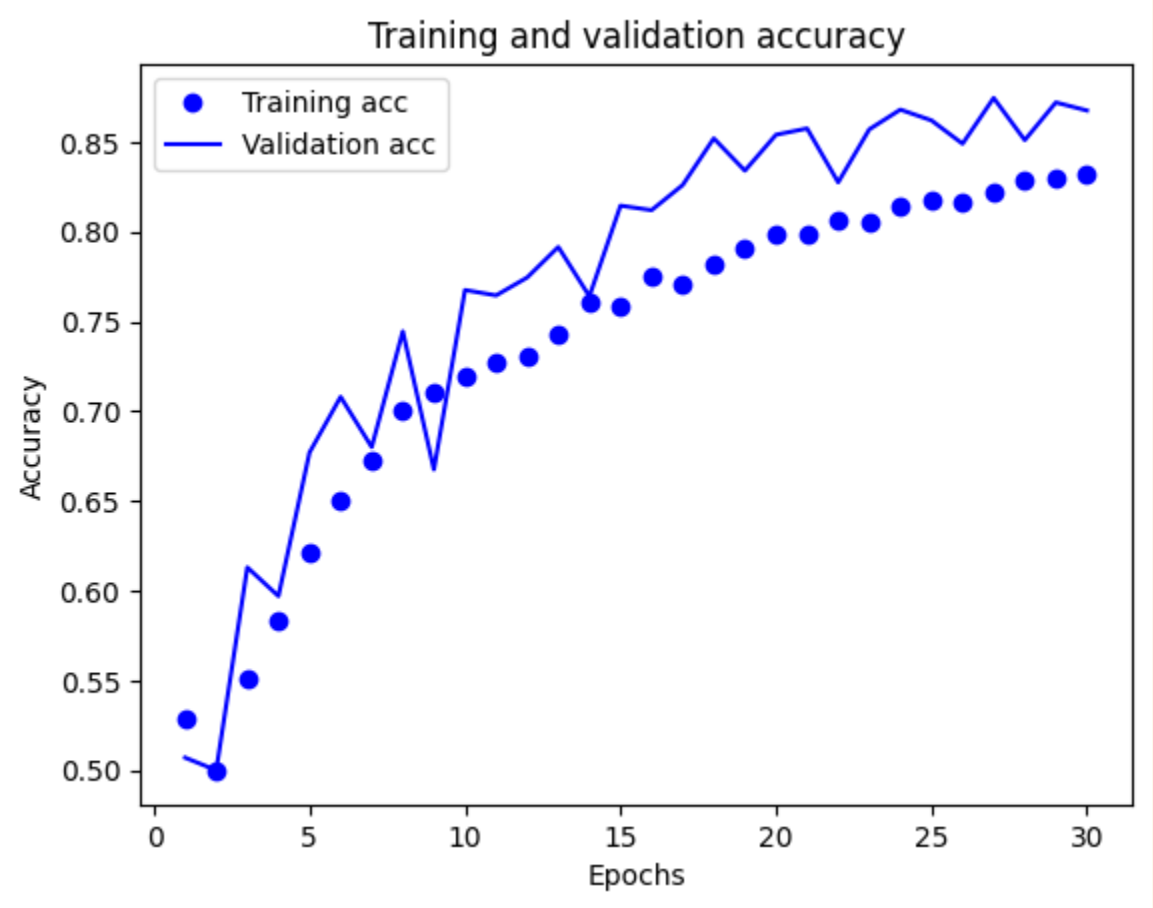
Then I tried to plot the result in all epochs :

```
import matplotlib.pyplot as plt

history_dict = history.history

epochs = range(1, 30 + 1)

acc_values = history_dict['accuracy']
val_acc_values = history_dict['val_accuracy']

plt.plot(epochs,acc_values,'bo',label='Training acc')
plt.plot(epochs,val_acc_values,'b',label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```

This is the result for code A , it seems from epoch 16 the model start overfitting :



This is the result for code B :



Then I tried to find the best epoch(based on val_acc) and val_acc and train_acc of that epoch :

This is the result for code A :

```
[ ]  # Find the epoch with the best validation accuracy
     best_epoch = np.argmax(val_acc_values)

     # Print the training and validation accuracy for the best epoch
     print(f"Best Epoch: {best_epoch + 1}")
     print(f"Training Accuracy: {acc_values[best_epoch]}")
     print(f"Validation Accuracy: {val_acc_values[best_epoch]}")
```

```
Best Epoch: 26
Training Accuracy: 0.9607499837875366
Validation Accuracy: 0.8985000252723694
```

This is the result for code B :

```
# Find the epoch with the best validation accuracy
best_epoch = np.argmax(val_acc_values)

# Print the training and validation accuracy for the best epoch
print(f"Best Epoch: {best_epoch + 1}")
print(f"Training Accuracy: {acc_values[best_epoch]}")
print(f"Validation Accuracy: {val_acc_values[best_epoch]}")
```

```
Best Epoch: 27
Training Accuracy: 0.8220000267028809
Validation Accuracy: 0.8744999766349792
```

Then I just try to predict one single image, and the model predicted correctly :

Part 4 - Making a single prediction

```
from tensorflow.keras.preprocessing import image
test_image = image.load_img('/content/gdrive/My Drive/dataset/single_prediction/cat_or_dog_1.jpg', target_size = (150, 150)) # this is a dog picture
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis = 0)
result = cnn.predict(test_image)
training_set.class_indices
if result[0][0] == 1:
  prediction = 'dog'
else:
  prediction = 'cat'

print(prediction)
```

```
1/1 [==============================] - 0s 279ms/step
dog
```