Assignment 1 – Practical Deep Learning Workshop

1) Present an exploratory data analysis of the dataset that you selected.
   a. What is the size of the data?

      The data consists of 50,000 pictures 614 MB total

   b. What data does each sample contain? (dimensions, channels, how many classes? Should we preprocess the data? Or is it ready for use? Can we use augmentation and what kind of augmentation would be valid?) Is the data balanced? (How many examples there are per class?)

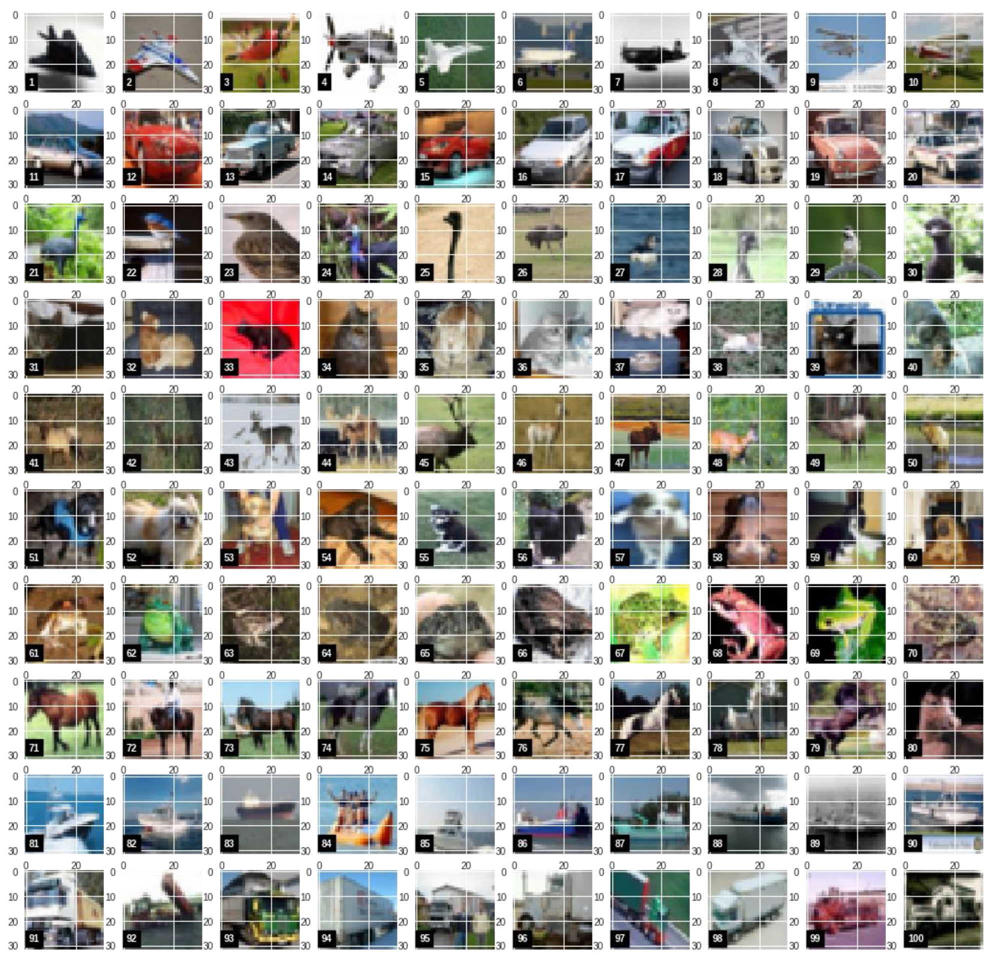
      The picture size is 32X32, and three channels (R,G,B) with 10 categories. Each category contains 5,000 pictures.

      This data set is perfectly balanced between the categories, the pictures seems to be good to go without preprocessing. We can use data augmentation and normalization. We can shift the picture, rotate it, flip it horizontaly (but not vertically – because a ship or a car upside down doesn't make any sense)

   c. Are there any benchmark results for different methods used on this data?

      This data set is featured in [Kaggle] and the leading team got 95.5 percent precision.

   d. Show some samples from each label (if there are many categories try and present examples of easily separable ones vs. harder more similar categories)



here we can see 10 pictures per category.

2) Form a neural network graph based on the components we used in the walkthroughs in class.

   a. Decide your validation strategy for training your model

   We decided to take the test data and split it to train and validation with sklearn train test split. This simple split should be enough due to the perfect balance and large samples per category. We used 20% of the test data to validation. To test our model, we used the test data provided from **Kaggle** and not the test we got from keras.

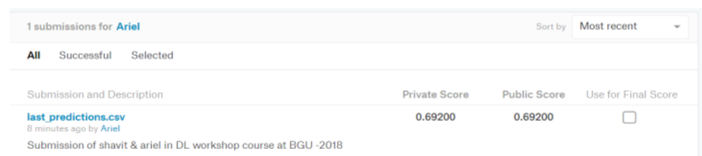
   b. Fit your model to the data and analyze the results (Use visualizations to present your loss and other metrics you find relevant, show examples for good and bad classification with high probability, and refer to the uncertain predictions.

   Compare the results you got on the training data vs. your results for the validation/test data)

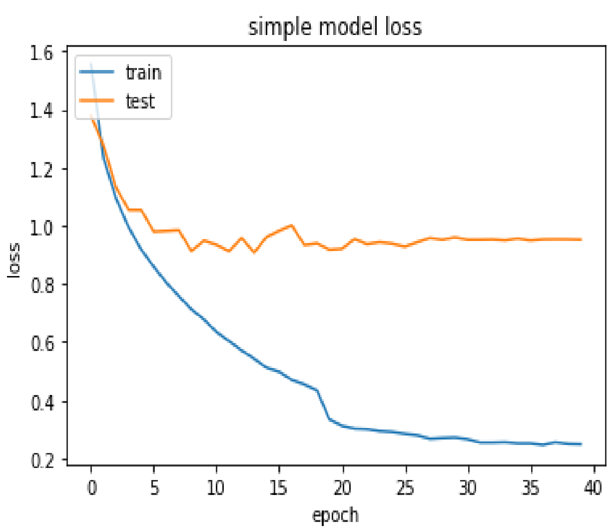
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_1 (Conv2D)            (None, 30, 30, 32)        896       
_________________________________________________________________
activation_1 (Activation)    (None, 30, 30, 32)        0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 30, 30, 32)        0         
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 15, 15, 32)        0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 13, 13, 32)        9248      
_________________________________________________________________
activation_2 (Activation)    (None, 13, 13, 32)        0         
_________________________________________________________________
dropout_2 (Dropout)          (None, 13, 13, 32)        0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 5408)              0         
_________________________________________________________________
dense_1 (Dense)              (None, 128)               692352    
_________________________________________________________________
activation_3 (Activation)    (None, 128)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290      
=================================================================
Total params: 703,786
Trainable params: 703,786
Non-trainable params: 0
```

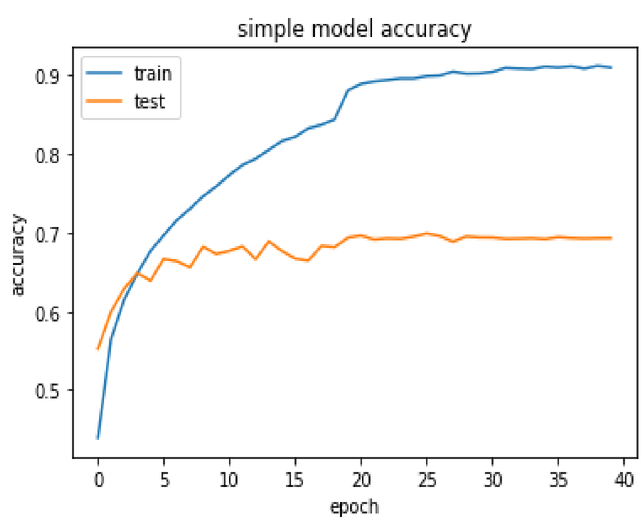
We started with a simple model shown in Lecture 2 and ran it for 40 epochs with ReduceLRateOnPlatou (patience 6) we normalized the data but didn't augmented the data .

We got 0.69 accuracy.



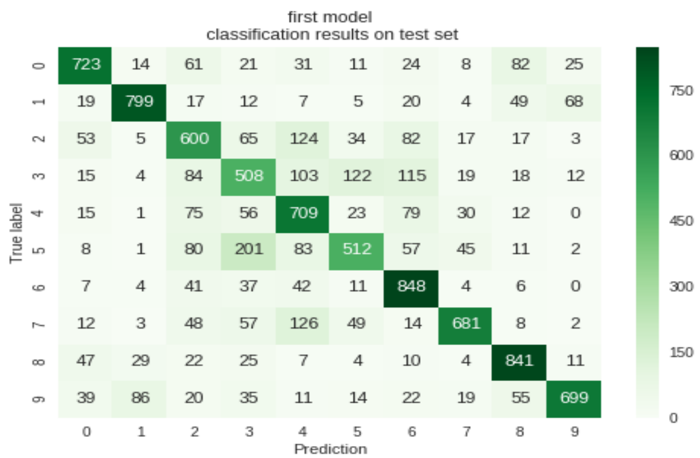

```
Epoch 40/40
40000/40000 [==============================] - 7s 186us/step - loss: 0.2488 - acc: 0.9102 - val_loss: 0.9528 - val_acc: 0.6931
```





As we can see the difference between the train accuracy (0.91) and the validation accuracy (0.69) is big.

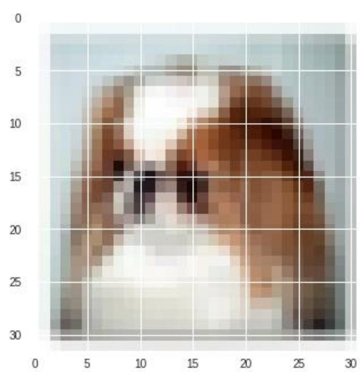
This means that the model overfits the train data but the actual score isn't high .



Our model predicted 211 dogs (cat 5) to be cats (cat 3) .

The frog (cat 6) has a good prediction rate, but the model predicted "frog" on another 423 pictures

The model predicted ship(cat 8) very good.



One of the dogs we labeled as a cat

c. Try to figure out where & why is the model misclassifying and suggest at least 3 ways to improve the results

After our experience with the right whale dataset we normalized the data prior to our firs attempt.

We can augment the test data in order to add some noise to the test in order to prevent the overfit on the data.

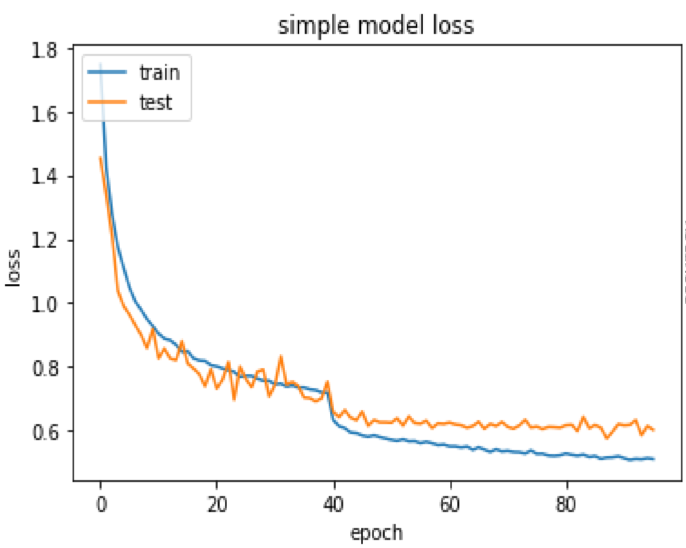
We can deepen the model in order to get more degrees of freedom in order to learn new features that the current model didn't succeed.

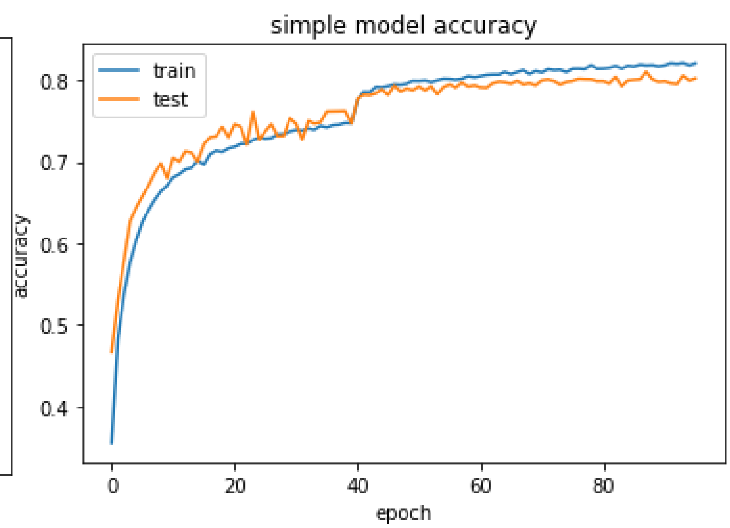
We can make less steps per epoch - this will update the weights faster and should decrease the gap between the train accuracy and the validation accurecy

d. Prioritize the list of suggestions for improvements and implement the first 2 suggestions and repeat section b.

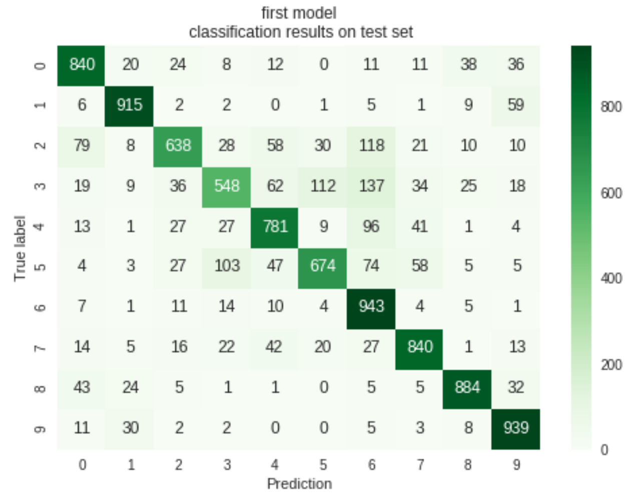
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_60 (Conv2D)           (None, 30, 30, 64)        1792      
_________________________________________________________________
conv2d_61 (Conv2D)           (None, 28, 28, 64)        36928     
_________________________________________________________________
dropout_21 (Dropout)         (None, 28, 28, 64)        0         
_________________________________________________________________
conv2d_62 (Conv2D)           (None, 26, 26, 64)        36928     
_________________________________________________________________
conv2d_63 (Conv2D)           (None, 24, 24, 64)        36928     
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 12, 12, 64)        0         
_________________________________________________________________
conv2d_64 (Conv2D)           (None, 10, 10, 32)        18464     
_________________________________________________________________
conv2d_65 (Conv2D)           (None, 8, 8, 32)          9248      
_________________________________________________________________
dropout_22 (Dropout)         (None, 8, 8, 32)          0         
_________________________________________________________________
flatten_4 (Flatten)          (None, 2048)              0         
_________________________________________________________________
dense_6 (Dense)              (None, 128)               262272    
_________________________________________________________________
activation_17 (Activation)   (None, 128)               0         
_________________________________________________________________
dense_7 (Dense)              (None, 10)                1290      
=================================================================
Total params: 403,850
Trainable params: 403,850
Non-trainable params: 0
```

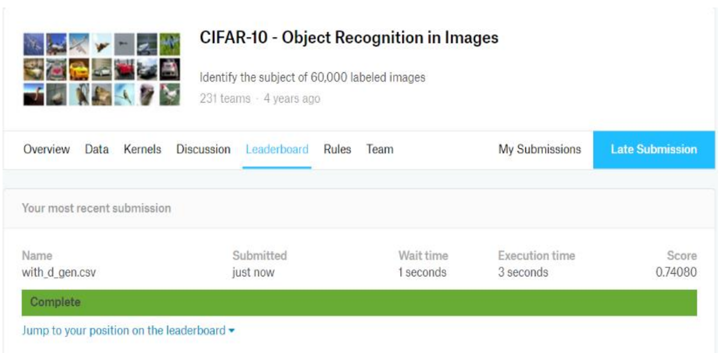
This model uses more layers. And its score is much better.






```
Epoch 96/96
1250/1250 [==============================] - 37s 30ms/step - loss: 0.5110 - acc: 0.8203 - val_loss: 0.6023 - val_acc: 0.8019
```


first model
classification results on test set
True label
Prediction
0 1 2 3 4 5 6 7 8 9
840 20 24 8 12 0 11 11 38 36
6 915 2 2 0 1 5 1 9 59
79 8 638 28 58 30 118 21 10 10
19 9 36 548 62 112 137 34 25 18
13 1 27 27 781 9 96 41 1 4
4 3 27 103 47 674 74 58 5 5
7 1 11 14 10 4 943 4 5 1
14 5 16 22 42 20 27 840 1 13
43 24 5 1 1 0 5 5 884 32
11 30 2 2 0 0 5 3 8 939
800
600
400
200
0


CIFAR-10 - Object Recognition in Images
Identify the subject of 60,000 labeled images
231 teams · 4 years ago
Overview
Data
Kernels
Discussion
Leaderboard
Rules
Team
My Submissions
Late Submission
Your most recent submission
Name
with_d_gen.csv
Submitted
just now
Wait time
1 seconds
Execution time
3 seconds
Score
0.74080
Complete
Jump to your position on the leaderboard

3) Select a trained model architecture (you can use the ones available in the Keras library under applications, or any other model with pretrained weights you find online) **40 pts**

a. Change the last layer to correspond to the task at hand.

In colab.

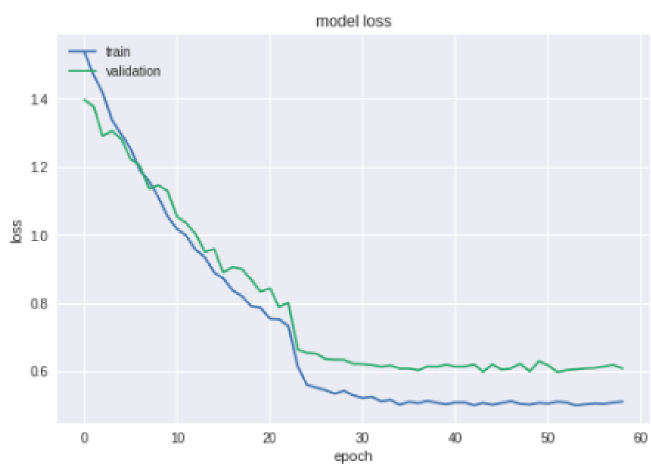
b. Perform the relevant preprocessing steps.

- We load the train and validation data (same as Question2), this data is already normalized and y vector is already converted to categorial.
- We change the size of an image to be 48X48 same as input size in our VGG16 model.

c. Repeat section 2b

Fitting the model:

```
Epoch 00040: val_acc did not improve from 0.80880
Epoch 41/100
1600/1600 [==============================] - 78s 49ms/step - loss: 0.5089 - acc: 0.8284 - val_loss: 0.6137 - val_acc: 0.8032

Epoch 00041: val_acc did not improve from 0.80880

Epoch 00041: ReduceLROnPlateau reducing learning rate to 1.000000082740371e-08.
Epoch 42/100
1600/1600 [==============================] - 79s 49ms/step - loss: 0.5089 - acc: 0.8296 - val_loss: 0.6139 - val_acc: 0.8063

Epoch 00042: val_acc did not improve from 0.80880
Epoch 43/100
1600/1600 [==============================] - 79s 50ms/step - loss: 0.5006 - acc: 0.8292 - val_loss: 0.6200 - val_acc: 0.8077

Epoch 00043: val_acc did not improve from 0.80880

Epoch 00043: ReduceLROnPlateau reducing learning rate to 1.000000082740371e-09.
Epoch 44/100
1600/1600 [==============================] - 79s 49ms/step - loss: 0.5077 - acc: 0.8285 - val_loss: 0.5981 - val_acc: 0.8127

Epoch 00044: val_acc improved from 0.80880 to 0.81270, saving model to ./gdrive/My Drive/Work-1/CIFAR/try_models/weights-improvement2-44-0.81.hdf5
Epoch 45/100
1600/1600 [==============================] - 79s 49ms/step - loss: 0.5021 - acc: 0.8282 - val_loss: 0.6203 - val_acc: 0.8044

Epoch 00045: val_acc did not improve from 0.81270
```



- as we can see the model trained almost 60 epoches, after 30 epoches the model was fitted to the train data set (~80%) with low loss rate (~0.5).
- we can also see that as the epoches progress and the model fit to the train data, it also fitted the validation data, and there is no point where the val_loss start to increase, from this we can conclude that the model was not get to an 'overfitting' point.
- one more interesting thing we can notice is that in epoch 22 the acc steeply increase (and loss decrease), this is because we defined callback that decrease the learning rate.

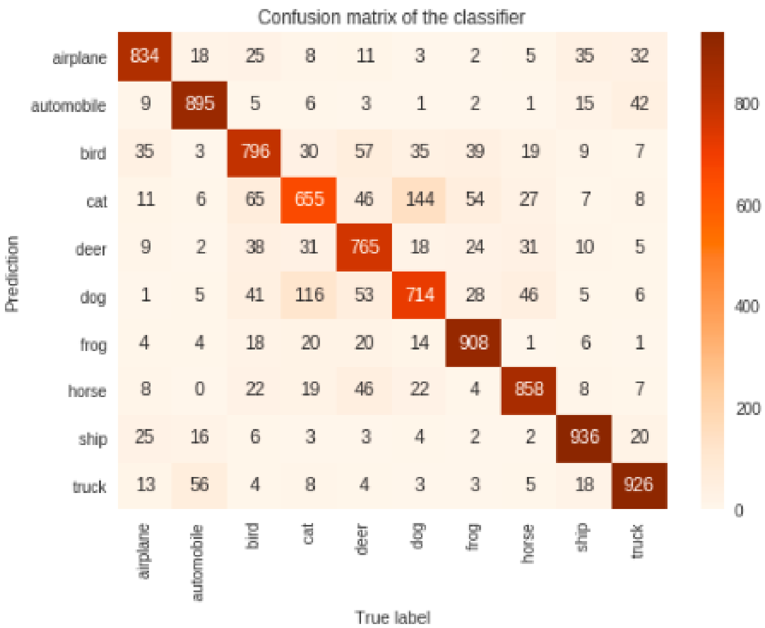
The accuracy:

Name	Submitted	Wait time	Execution time	Score
result.csv	a day ago	87 seconds	3 seconds	0.82900

Complete

We ranked 46/231 in keras leaderboard with this score.

We analyze the wrong prediction by plot confusion matrics:



the main diagonal is darker, which shows that in most predictions we were right.
but there is also place to improve, as we can see the most significant issue is that the model doesn't distinguish good enough between dog and cat (on both of the sides). about this problem we discussed in Q2.

we can also see that automobile as truck and truck as automobile (~50 samples for each sides got wrong.), let's take a look about wrong predicted image of truck:

as we can notice in some of the images such as 10,3,4,11,13 , it's cars that predicted as trucks, we think that the reason is that in these images the cars are very close to the camera and take a big size of the frame, such as trucks that are big vehicles. the problem that both of them have many patterns in common like the wheel's shape, windows and etc.

in image 9 the car and the building with the same color , and it looks like it's one big object, very similar to truck.

in general, these mistakes are acceptable because we know to explain them, and this is not out of context. in compare to question one we can notice that the prediction is better now. so probably we have to try to learn more patterns about the data. or another solution can be to use another model to detect special characteristics of a truck that differ it from a car.

d. Use the trained model you got in 3c as a "feature extractor" (omit the last layer, and then use predicted values as features for a classical ML algorithm of your choice. How does your results for this combination compare to your previous results?

This is our model architecture from 3c:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_8 (InputLayer)         (None, 48, 48, 3)         0         
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 48, 48, 64)        1792      
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 48, 48, 64)        36928     
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 24, 24, 64)        0         
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 24, 24, 128)       73856     
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 24, 24, 128)       147584    
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 12, 12, 128)       0         
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 12, 12, 256)       295168    
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 12, 12, 256)       590080    
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 12, 12, 256)       590080    
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 6, 6, 256)         0         
_________________________________________________________________
flatten_3 (Flatten)          (None, 9216)              0         
_________________________________________________________________
batch_normalization_3 (Batch (None, 9216)              36864     
_________________________________________________________________
dense_3 (Dense)              (None, 10)                92170     
=================================================================
```

we decide to make globalaveragepooling after the forth layer from the end in order to get 256 features. (instead of 9216 if we omit only the last layer)

we were asked to use this model as a feature extraction to classic model, as we can see ***all the parameters that above the last layer are already trained by imagenet dataset***. so probably it will not fit enough to our task (samples of our task doesn't change the weights).
but because we asked to do that with **this model** we tried, And saw that our hypothesis about the result is correct.

We tried different models, the "best" score was with random forest 0.61. but we have got same result or even poor than that, but we are not surprised, and we expect that we will get these results.

three fingers rule when something get wrong:

with this model our only choice to get better results is to train our model again, but how to train it?

- first option is to add more layers to the exist model and train the model only on these layers, after the model trained, we can use these layers for feature extraction.
- second option is instead of adding new layers, just to train some of the existing one so the result we will get will be more fitable to our data set.
- we can also use another pretrained model such as the model from question B that pretrained on our data set.

probably we will take the last option and try it in order to break a record of ourselves, and will update on our drive later.