DEEP NEURAL NETWORK - FALL-2018 PROJECT1

Azad Almasov (1511417)

The University of Tulsa

November 6, 2018

1 STATEMENT OF THE PROBLEMS

1.Please make a folder and name it using your first and last names, applying the camel notation, for example haniGirgis. Save all your solutions under this directory.

2.Download the data from: https://www.kaggle.com/c/leaf-classification

3.Divide the imagesinto three sets. Make sure that each data set is balanced—roughly equal number of each class. Use the three sets as the training set (50

4.Follow the method outlined in Chapter 5 of the Deep Learning with Python book. Try different architectures from the one listed in the book, i.e. try different number of layers with different number of masks or hidden unites as well as different techniques to minimize/avoid over-fitting. Please report the best performing model. You must train your classifier using each of the following techniques:
a.Train a dense network directly on these data sets,
b.Train a deep network directly on these data sets,
c.Train a deep network directly on these data sets with data augmentation (minimize the use of excessive transformations),
d.Use fine tuning with dense network, and
e.Use fine tuning with dense network with the last convolutional block.

5.Write few sentences on the performance of each model.

6.Visualize the activations due to any two classes of your choice (Section 5.4.3). Write few sentences commenting on which part(s) of an input image is/are related to a specific class.

7.Python is interactive. Please use your creativity to make me enjoy reproducing your work.

8.Save your code as hw3.py

2 SOLUTIONS

Steps **1**;**2**;**3** have been completed successfully. At first, I shuffeled data. Then using excell file of *train.csv*, all images are classified under the folder named by their classes under each train, validation and test folders. All these folders have been saved under the folder "*leaves*". Then after all, I have vectorized labels of each training, validation and test data.

Before having started step 4, I have to mention things that I will not modify throughout each different models:

Since it is multiclass classification, I have used *softmax* activation function for last layer, and for hidden layers *relu* is used as usual.
In compilation of the model as a loss function I have used *categorical crossentropy* since my data is multiclass. And in data generation I haves used *categorical* as my *class mode*

4a. First I started from very simple linear model, which is 1 layer model. Since I ll have a lot of features when i flatten picture (150*150), to prevent bottleneck effect I have used hidden units more than 256. First I have flattened data then feed it into 1 dense layer with 1024 hidden units. As s result, both validation and training data give accuracy around 0.02 which is very low (fig.1).

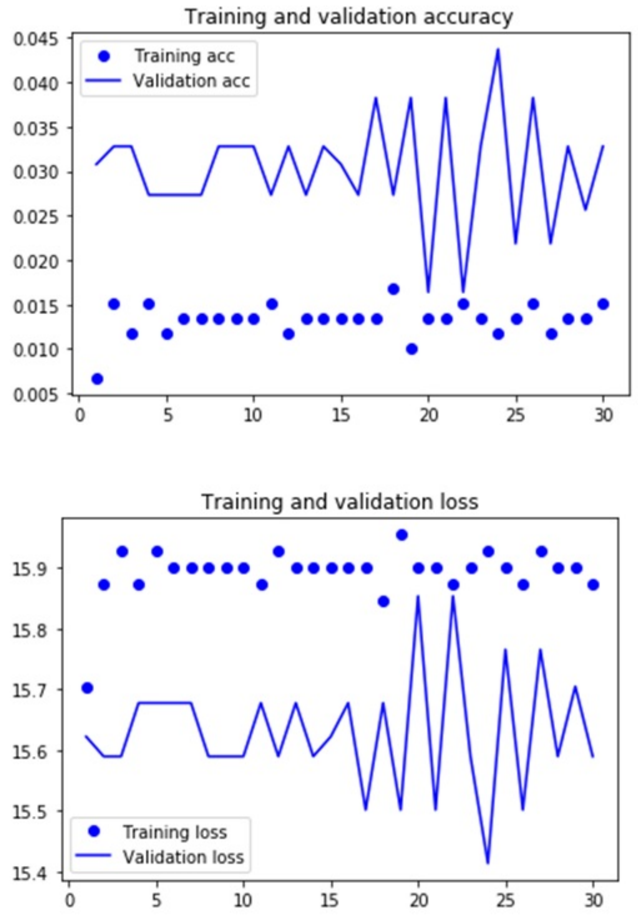


Figure 1: *Simple dense network with 1 layer*

I have used regularizer (with l2(0.001)) and it didnt help so much. It is obvious that linear model didnt work and it gives under fitting.
Later I applied two dense layers(512-256) and with dropouts after each dense layer. Results improved dramatically compared with previous model. Even though I have used regulirizers and dropout there was still overfitting. Then I have used 1 convolutional layer before dense layer and double maxpooling to decrease number of features, and then applied the same double dense layer with dropouts and regulirizers. I have shown the results in the Figure 2. As you see accuracy improved but it is not enough. Furthermore, there is still overfitting.

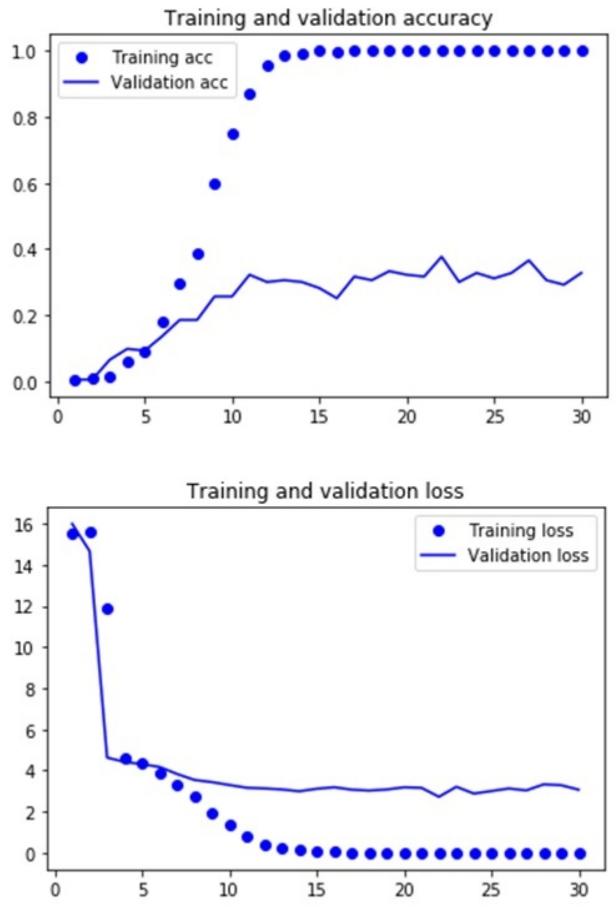


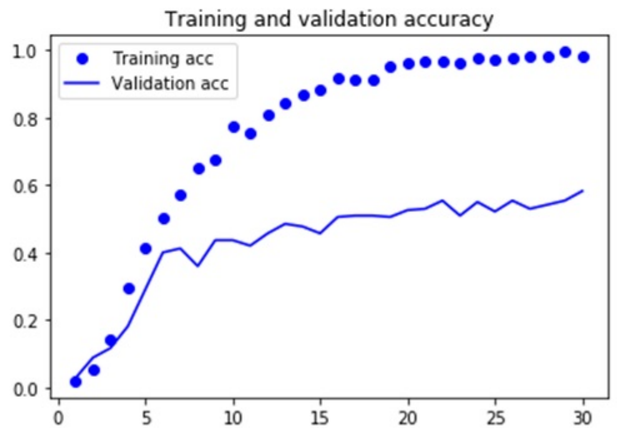
Figure 2: *Simple dense network with 2 layer and 1 conv layer*

4b. Last result from previous section showed that dense layer solely is not enough for the better result. It is because we need better features. Other important reason for us to need feature engineering is that number of classes is a lot and our total data is very few (≈ 960). Even some classes there is only 5 images. So we really need feature engineering, therefore I have used convolutional network in this section. To build better architecture I have tried different architectures:
Throughout all models except Augmented model, I used $batchsize = 6$ since our data is very few. I also devided $training - validation - test$ data as $3 - 1 - 1$ portion so that I can have reasonable amount of training data,

since I have few data.

mp - means Maxpooling (I used only (2,2) max-pooling); conv - means convolutional layers (I used (3,3) ocnvolutional layers); dense - means dense layers; dp - means dropout; reg(l2(0.001)) - means regulirizer L1 with coefficient $= 0.001$
Now lets look at results of each models and reasons why I have modified them:

Model 1: 32-mp-64-mp-128-mp-128-mp conv and 1 dense 512:
This model is the model used in the cats and dogs binary classification problem. It gave 70 percent accuracy or cats and dogs classification, but here in our problem it gave 40 percent accuracy. This is reasonable because we have 99 classes and few data. Furthermore, in the dog and cat classes we have texture as well as edges etc., but in our images we have only edges, thus it leads us bad accuracy. And there is overfitting (fig 3)
Then I changed dense layer architecture to 2 dense layer



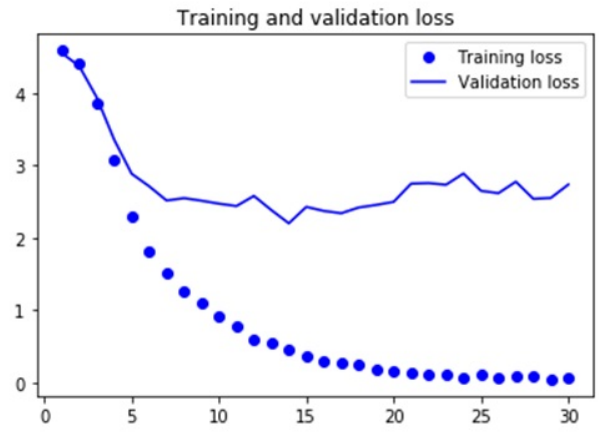
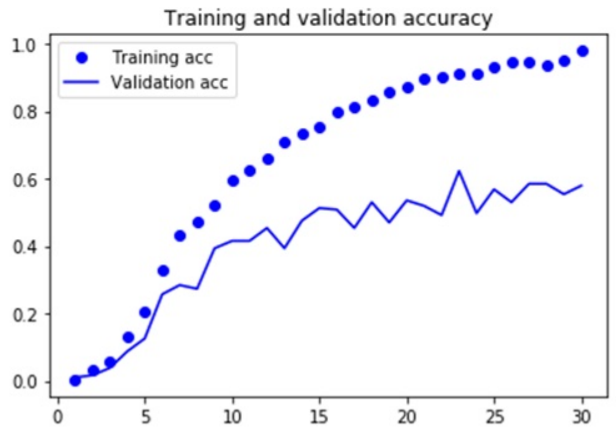


Figure 3: *Model 1*

Model 2: 32-mp-64-mp-128-mp-128-mp conv and 3 dense 512-256-256 with regulirizers:
This model showed better results compared to previous model, and regulirizers decreased overfitting little bit but not totally. Accuracy improved to 45 poercent (Figure 4).
As you can see modification of dense layer only didn't



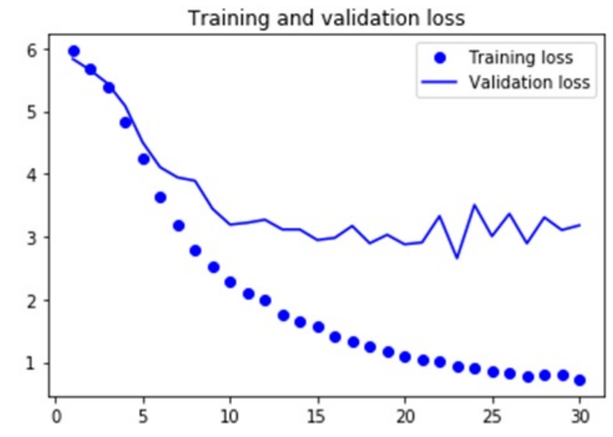


Figure 4: *Model 2*

affect that much. I believe that main problem here is about inappropriate feature extraction. Since I have 99 classes I believe I should use more convolutional layers. I took VGG16 already exist model, as a analogue to create similar model for our problem, since VG16 was also for multiclass problem:

Model 3: 64-64-mp-128-128-mp-256-256-256-mp conv and 2 dense 512-dp-512 with regulirizers:
Here I have also used dropout. Results showed little improvement and accuracy improved to 50 percent, but since model paramaters a lot it was very slow (Figure5).
I believe that since our images have only edges as a feature, instead of using plenty of convolutional layers

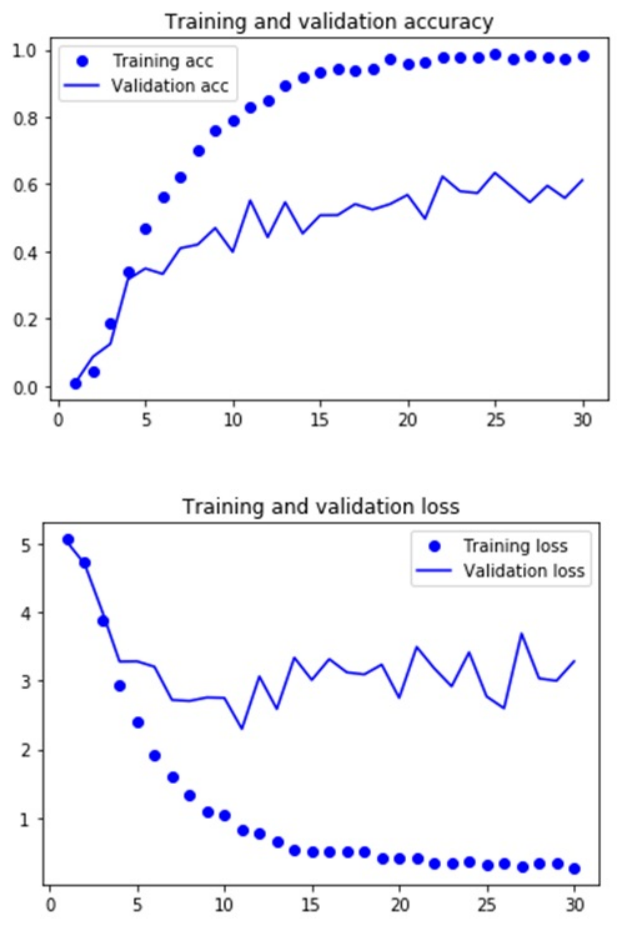


Figure 5: *Model 3*

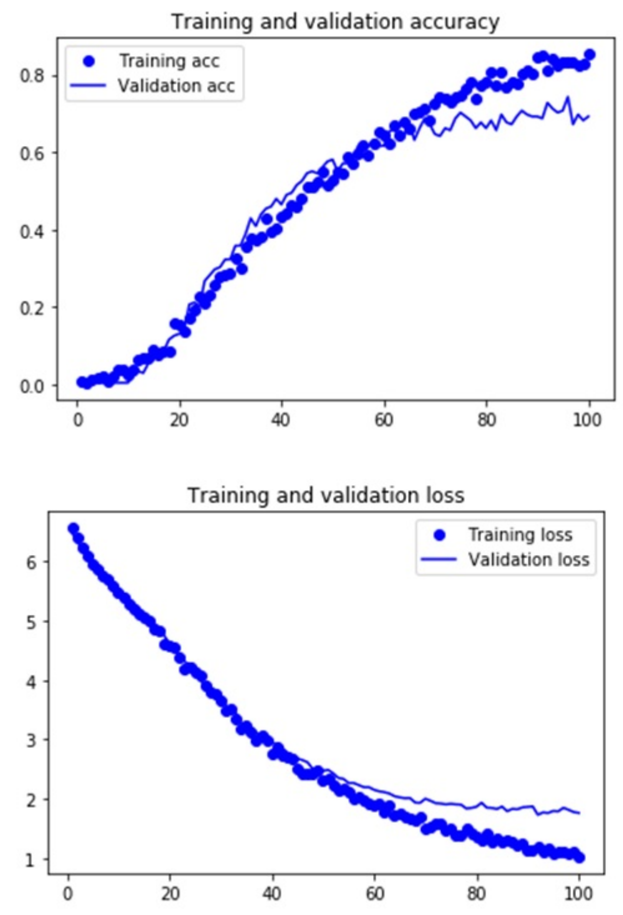


Figure 6: *Model 4*

to extract combination of features, it is better to use maxpooling to sharpen already extracted output features from each convolution layer. Lets look at the results of this kind of model:

Model 4: 64-64-mp-mp-128-128-mp-mp-mp-dp-1024-dp-512-dp with regulirizers
By doing this, number of model parameters decreased a lot (3MM parameters, in previous model we had 14 MM paramters). I have used dropouts and regulirizers as well. Model became faster and results improved pretty good (Figure 6). Here I had to increase number of epochs to see the overfitting seperation. And we see overfitting after 66 epochs. Accuracy of the model improved to **66 percent**. I believe that this accuracy for the 99 class problem with few data is reasonably good. I have to mention that when I evaluated my test data I took $step = len(testdata)/batchsize$ so that generator generate all test data.

However, since simplification of the model in this way improved my accuracy, I would like to try more simpler model believing in that it will decrease model parameters and it will overcome overfitting for longer time. I do so since I dont have so many visual patterns to be combined to create more reasonable features except different kind of edges and lines. Even though I believe that after some simplification my model result can lead unreasonable results as we had observed in only dense layer model. Now lets try the following simpler model:

Model 5: 64-64-mp-mp-128-128-mp-mp-mp-dp-512-dp-256-dp with regulirizers
In this model our resulted model parameters of course were less than previous model (about 2MM). This is good for efficiency. Lets see accuracy results. Accuracy improved to 70 percent. Overfitting started very late. More epochs need to reach 70 percent accuracy (Figure 7).

Even though we reached 70 percent accuracy, I think 100 epoch is a lot to be efficient. Therefore, I will try more simplify my model:

Model 6: 64-64-mp-mp-128-128-mp-mp-mp-dp-256-dp with regulirizers

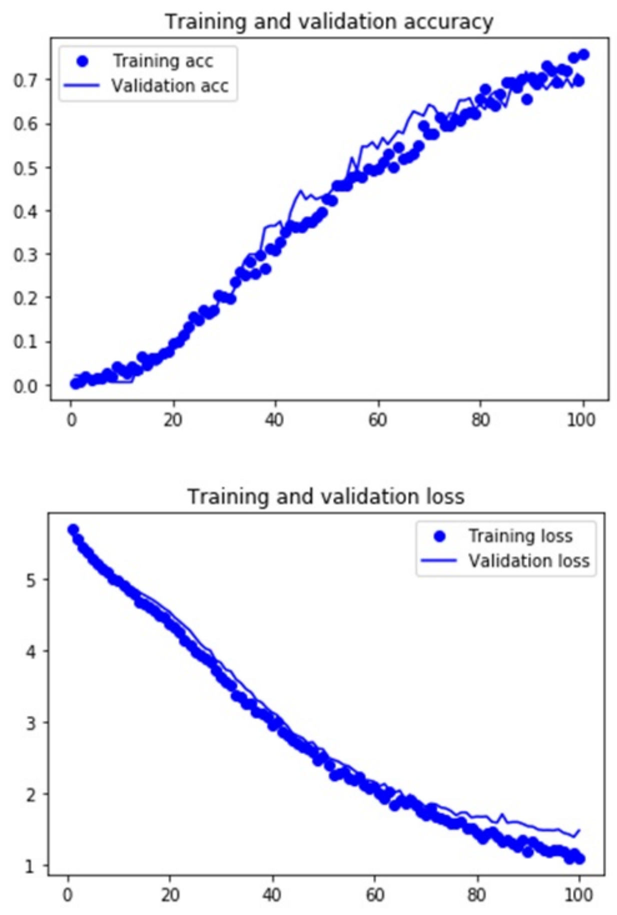


Figure 7: *Model 5*

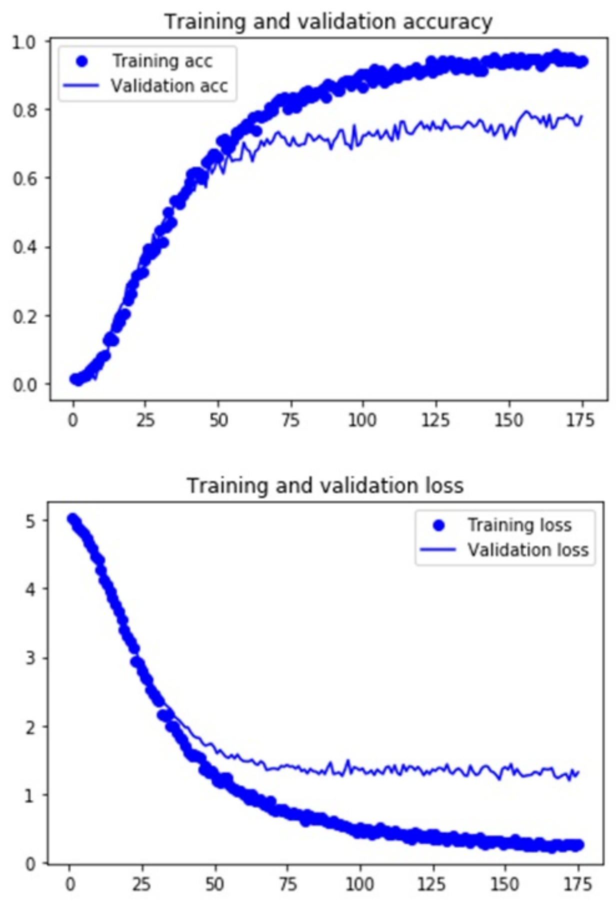


Figure 8: *Model 6*

In this, model I only put 1 dense layer believing that my convolutional layers gave reasonable outputs. And thus my model parameters decreased to around 900K. We see overfitting after 50 epochs (Figure 8). And our accuracy reached to 74 percent. This model is more efficient and accurate.
However, I want to see what more simplified model will give me. Let's test more simplified model's results:

Model 7: 32-32-mp-mp-64-64-mp-mp-128-mp-dp-256-dp with regulirizers
Model parameters decreased to 460k. I have to mention here that in my all models above, my features in last convolutional layer were 4x4 matrices, but in only this model I have 3x3 matrices. This model resulted late stabilization of validation accuracy which is not efficient, requires more than 100 epochs. However, previous model (model 5) required 75 epochs to reach 75 percent accuracy. And accuracy of this model was 68 percent (Figure 9).

Here, I want to say couple sentences about why stabilization changes as our model changes. This is because, for example in previous model, since we have more parameters, which means we will have more punishment by regulirizers, and, thus it will make both training and validation plots to approach each other, and as a result, stabilization will take more epochs. I believe that this stabilization time delays due to regularization. However, it gives us better accuracy. So as it can be seen from the performance of the models, my choice is **Model 6**. However, it takes 70 epochs to get that accuracy. I believe that Augmentation will require more than 300 epochs to overcome overfitting and reach better accuracy. In the following section using this model and data augmentation I showed results.

4c. In data augmentation, I did not use all kind of transformations, because if we see the images of different calsses of leaves we can understand that for example shearing, width shifting, height shifting of one class can resemble to another class. Therefore, I skipped those transformation options and I did only *rotation, zoom* and *horizontal flip*. Even though, augmentation got over overfitting, unfortunately, it didnt stabilized

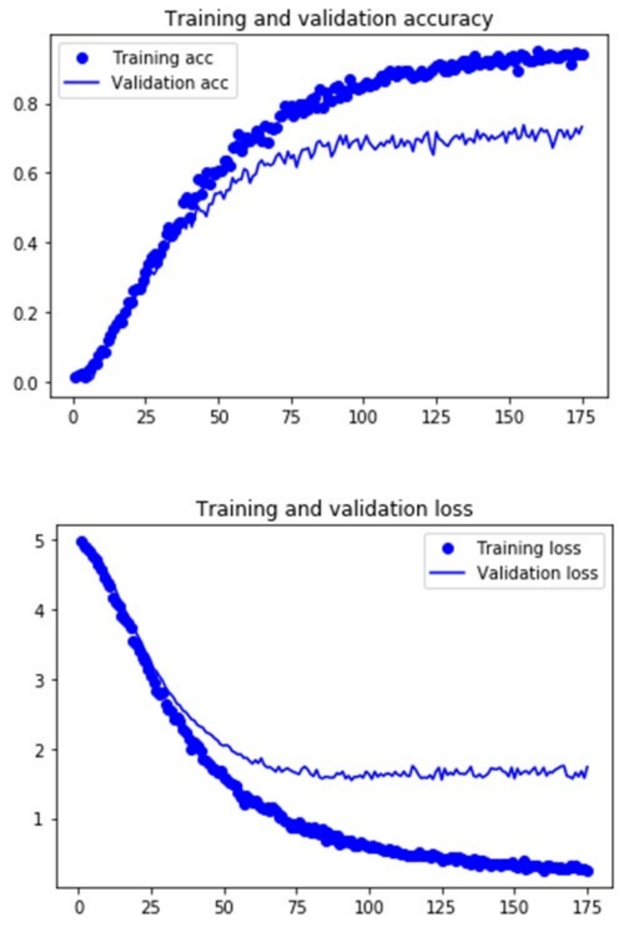


Figure 9: *Model 7*

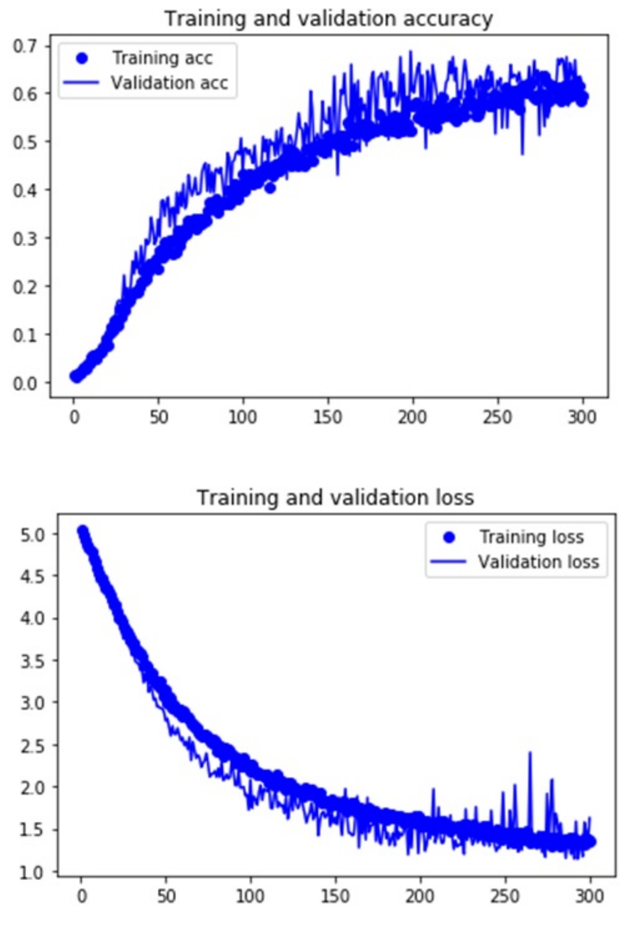


Figure 10: *Model 6 with augmentation*

even after 300 epochs. In my opinion this is because my model 6 itself start overfitting very late after 50 epochs and stabilizes late. Therefore, in augmentation model, validation and training accuracy stabilizes very late and I need more than 300 epochs (Figure 10). For me it is not efficient model.

4d. I have used VGG16 as my convolutional model and extracted features and using dense layer(256 nodes) I fine tuned my model with dense layer only. Results were surprising. I got **84** percent accuracy (Figure 11). However, this accuracy changed to 82 percent when I reshuffled my data again. So it depends how it shuffled, but accuracy doesn't change that much. However, I freezed command of shuffling since data given in excel file has already shuffled. In moy opinion, it is better not to shuffle.

I increased hidden units of last layer to 512 from 256 (modelPT-512), and results are given in Figure 12. Compared to previous model it stabilizes faster with accuracy equal to 85 percent. Taking into consideration that this model reaches this accuracy in short epochs, I choose this model as my best model.
I also tried two dense layers (512-256), but accuracy decreases and it was not efficient.

4e. Here I repurposed my last convolutional layer and trained it with dense layer. Results showed 80 percent accuracy (Figure 13) which is not bad but previous model gave more accuracy and were faster.

k-fold validation (EXTRA). Taking my base model as modelPT-512, which is transfer learning with 1 dense layer (512), I performed k fold validation on this model to further improve my model. To do so I have chosen 80 percent of my data as training set and k equal to 4 because for some classes I have only 4 samples only. I compared average validation accuracy and average training accuracy (Figure 14). Then after all I trained whole training data (modelPT-WHOLE-512) and tested it on test data and got **88** percent accuracy.

5. I have written my comments on each model in the

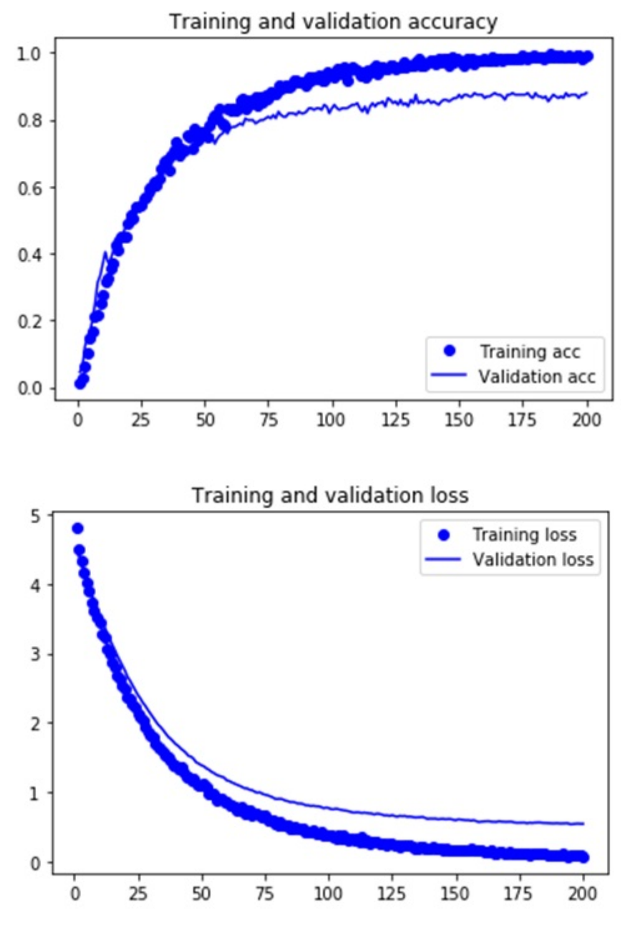


Figure 11: *Pretrained Model - modelPT-256*

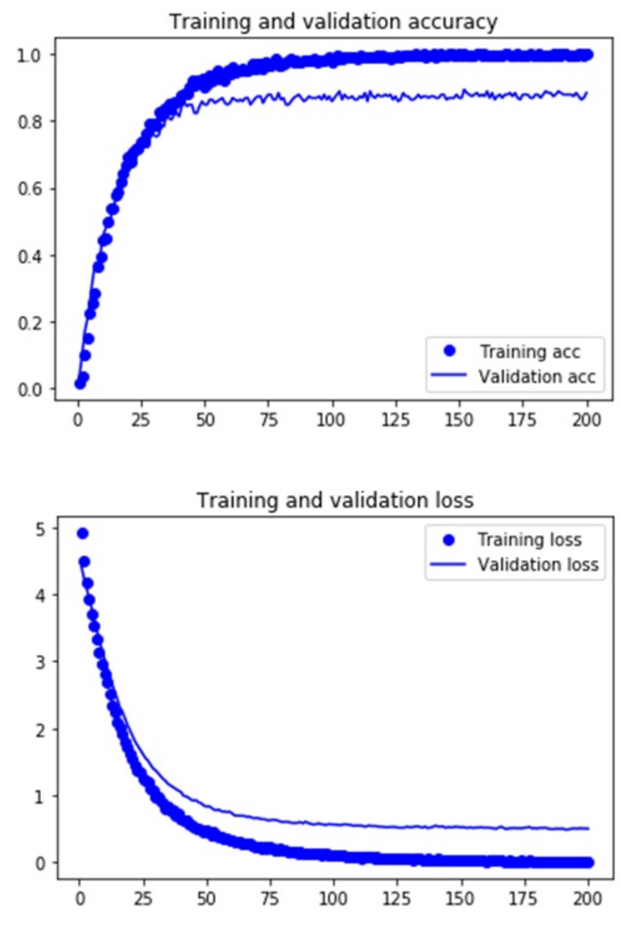


Figure 12: *Pretrained Model - modelPT-512*

sections above already

6. I visualized activations of my model6 model for first and second convolutional layers(Figures 15, 16 and 17).

I visualized filters as well (Figures 18

However, to visualize heatmap I used my transfer model, which is best model for me (Figure 18). As it can be seen from heatplot, mostly, edges of left wide part, begining of the leaf is important feature for class "Acer Capillipes" (Figure 19).
I have to mention that since in transfer learning we use VGG16 convolutional part, I dont think they trained VGG16 for all leaves classes. Therefore, we may get some unreasonable features.

7 I have run almost 10 models to find best one. I have improved my model using better architecture, using regularizations, using dropouts, and validated using k-fold validation.

8 Python code with best model only has been saved under "hw3.py" file. However, I have saved my all models, that I have mentioned in this report, as well. My "hw3.py python file with best model and k-fold validation", all "other models" and this "Report" all have been saved under the folder "azadAlmasov"

3 CONCLUSIONS

•I have run several models with several methods as well as I modified paramters architectures and layers to overcome overfitting •Best model was Transfer learning with 4-fold validation (modelPT-WHOLE-512), giving 88 percent accuracy. In this transfer learning VGG16 is used as convolutional base network.

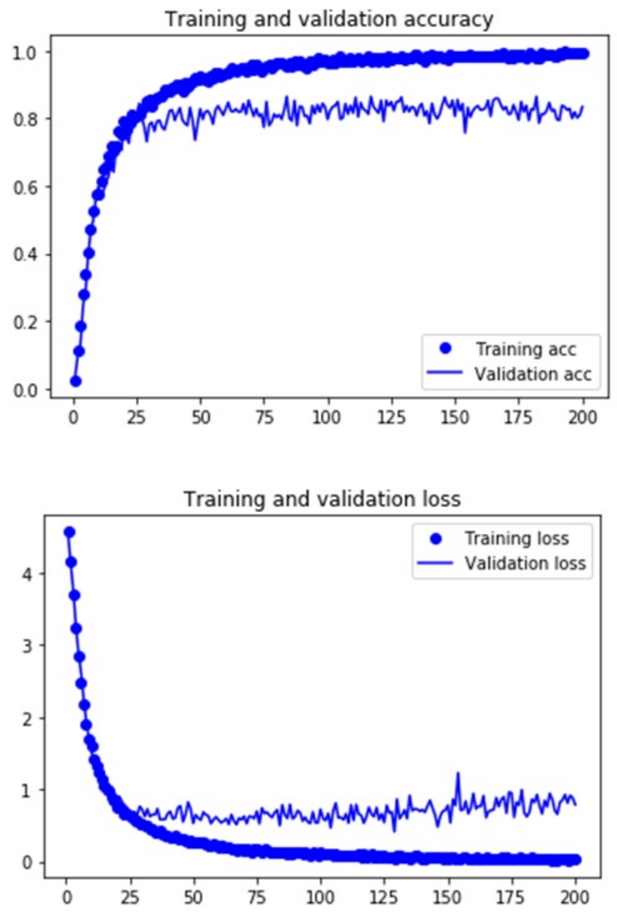


Figure 13: *Pretrained Model with last convolutional layer - modelPTC - 512*

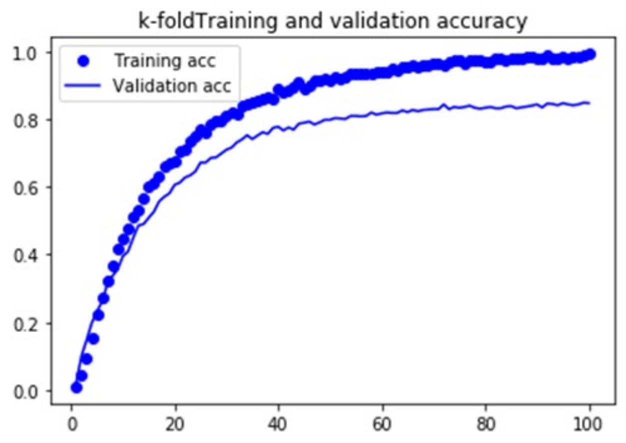


Figure 14: *Pretrained Model with last convolutional layer with 4 fold validation - modelPTC-WHOLE-512*

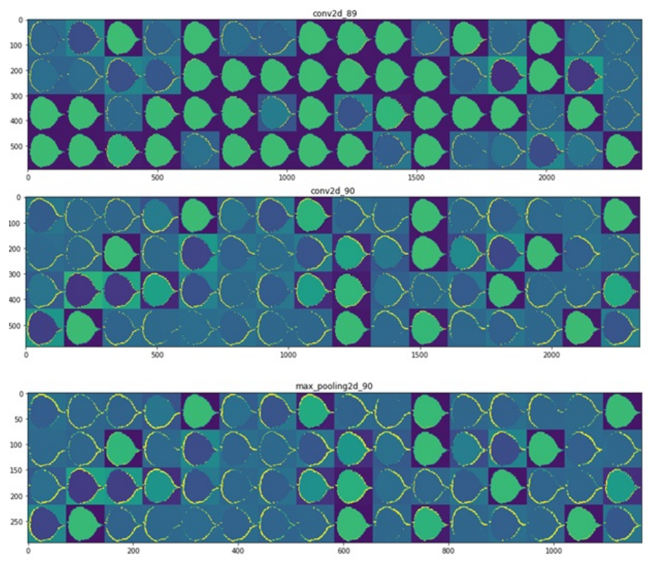


Figure 15: *Activations*

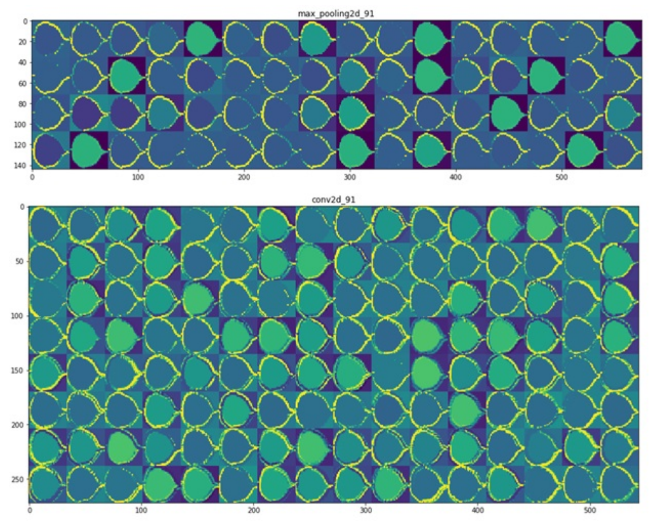


Figure 16: *Activations*

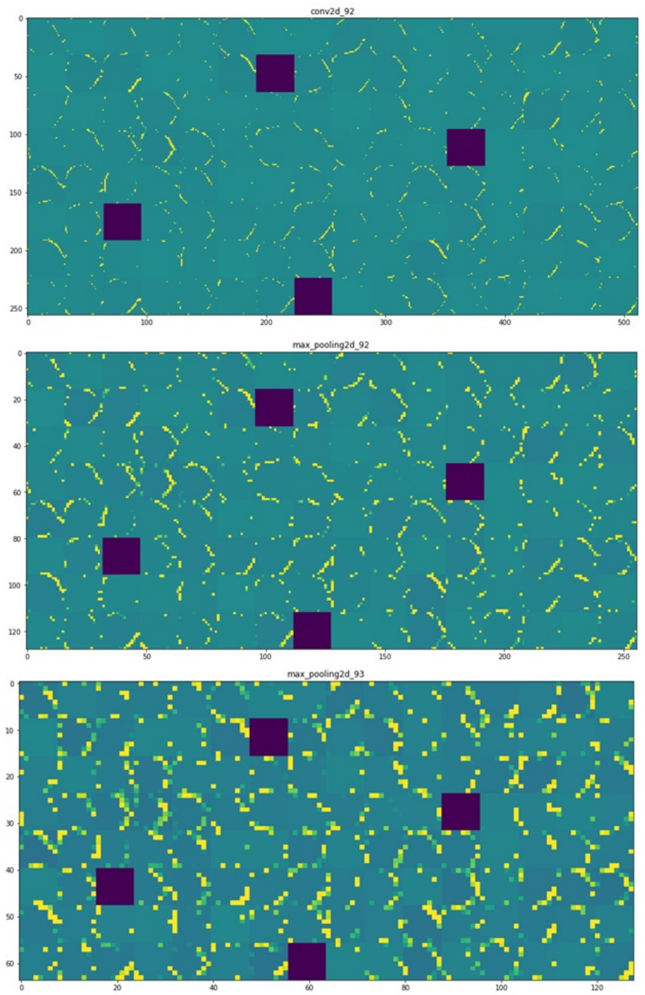


Figure 17: *Activations*

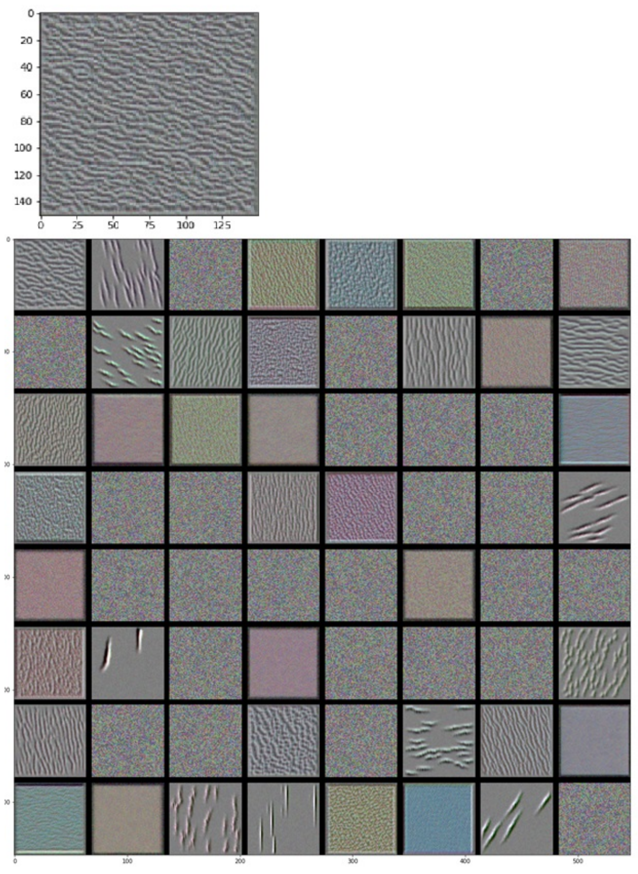


Figure 18: *Filters*

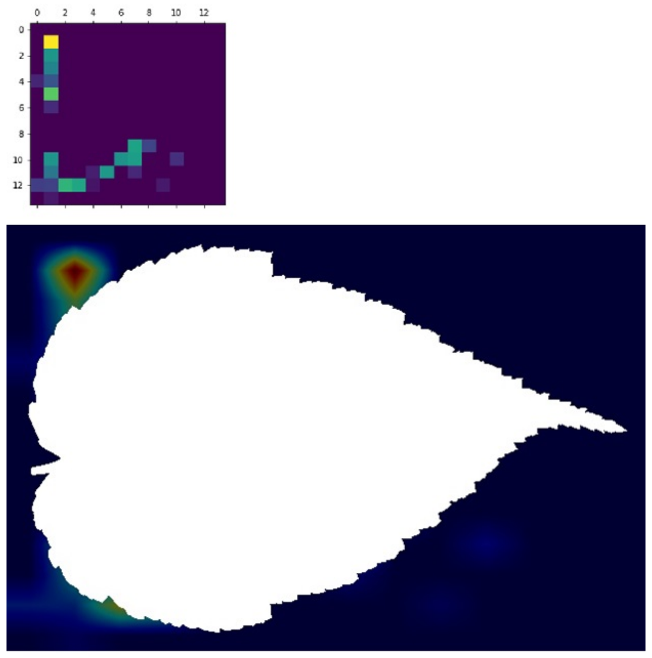


Figure 19: *Heatmap*