Software Development for Algorithmic Problems
Project 2 - Experimental Evaluation

Autoencoder Evaluation

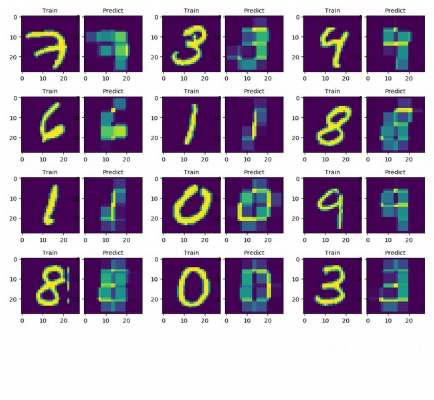
Introduction

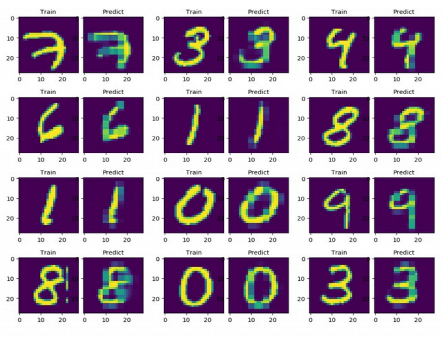
Firstly, before proceeding into our assumptions and experimental results we should firstly define autoencoder's hyperparameters. Autoencoder's hyperparameters are the following:

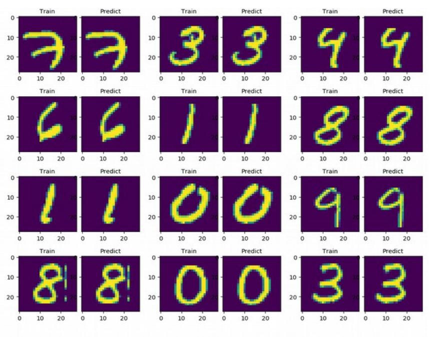
- The number of encoder's convolutional layers and the number of filters in each one of them. Also, the user can define if these layers will have afterwards a max pooling layer and a dropout layer.
- The kernel size of all the convolutional layers.
- The number of training epochs, which is how many times the neural network will read all the training set through its training.
- The batch size, which is the number of samples that the network will read before updating its weights once.

Afterwards, we do the following assumptions about some of our hyperparameters. Firstly, like MNIST we have 28x28 images and we mostly take 3x3 window as kernel size as it is recommended on all tutorials, but we will also test the 5x5 kernel. Secondly, because we want to see the training progress of our autoencoder the number of epochs will be constant because if we train a model with a large number of epochs(will be 50 epochs) we can see the proper number of epochs because we will observe when the overfitting is happening. Furthermore, the convolutional layers will adopt a pyramid scheme where the first layer will have a power of two filters(base) and the next layers will have consecutive powers of two layers. For example if we have an encoder of 3 layers and a base of 16 the 3 convolutional layers will have 16, 32 and 64 filters, respectively. Finally, the metric that is used to evaluate our model is the R-squared method, although in our training process, Mean Squared Error is used with RMSProp optimizer.

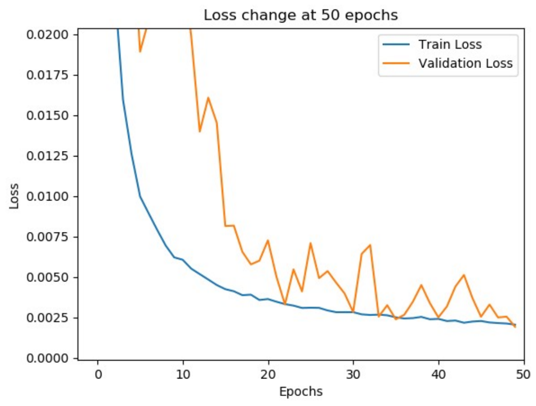
In order to provide some information and intuition to the reader in the figures below, we illustrate what each R-Squared score means in practical terms.


Autoencoder Result for R^2 equals 0.5

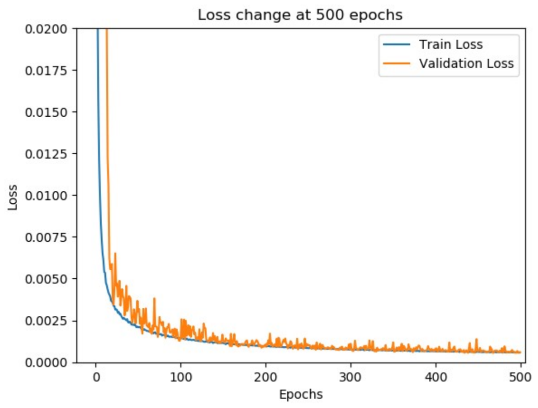

Autoencoder Result for R^2 equals 0.8


Autoencoder Result for R^2 above 0.95

Thus, we can clearly see that when R-Squared is equal to 0.5, then the autoencoder can represent in some way the original input but the picture is unclear. When R-Squared equals 0.8 the original input is better represented, but it has some points that are not represented too precisely, like when R-Squared is around 0.95 where the resulting pictures are close to the original ones. Consequently, we observe that R-Squared scores greater equal 0.95 are acceptable. Note that, the better the autoencoder is, the better dimensionality reduction the encoder does because our model is more capable from the reduced representation to output the original image. Also, from the various experiments that are conducted we saw that our autoencoder never overfits and even when it is done it is fixed at future epochs. Also, the loss continues to reduce through the course of the epochs. Therefore, when we observe that at 50 epochs one model performs worse than another, actually we assume that this model needs more epochs to train in order to perform the same. This is proven in the below figures where we train our worst model in terms of batch size for 50 and 500 epochs. The worst model was a 3 layer with base 16, two 2x2 pools in second and third layer and batch size 2048. Note that this model wasn't actually the worst but it was a model that gave decent results(around 0.98 R-Squared score) in order to perform better at less extra epochs than an even worse model.



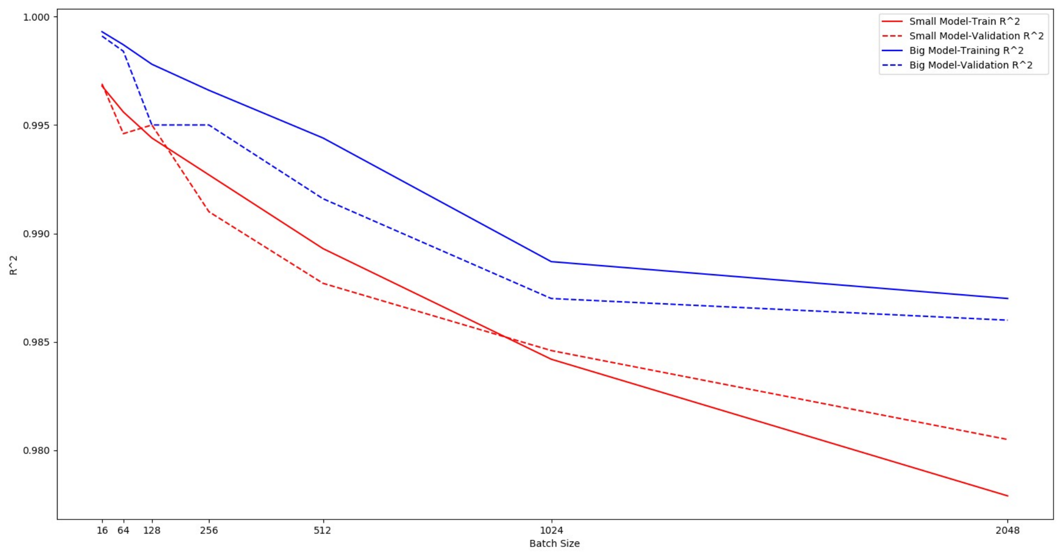
Loss Change of Worst Model after 50 epochs



Loss Change of Worst Model after 500 epochs

Batch Size

In this and the future experiments we test some hyperparameter using two models. The former has 3 layers(is called Small Model) and the latter 6 layers(is called Big Model). Both do not have Dropout layers because it is seem that with dropout the model performs worse. In order to test our model in respect to batch size the models have two consecutive 2x2 pooling layers with the first one being in the encoder's middle layer because this seem to have better performance(we will tell more about it afterwards). The change of R-Squared score through batch size is shown on the following graph.

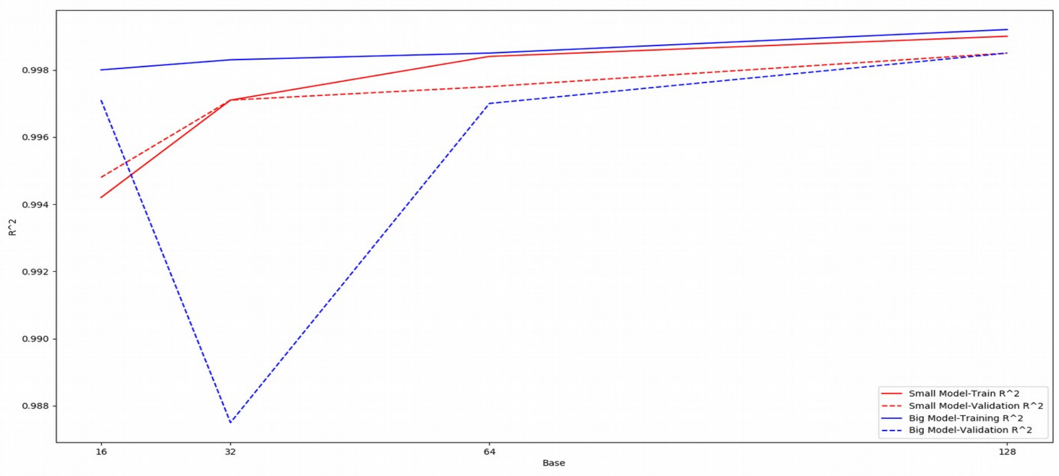


Change of R^2 in terms of batch size of Big and Small model

From the above graph we observe that Big Model converges faster and even it overfits more that Small Model still on the validation set performs far better. Thus, we can conclude that Big Model is generally better than Small Model. In terms of batch size it is clear that as the batch size increases, the model performs worse. Because as batch size increases the training time is decreased, in future experiments we use batch size equal to 128 as it trains in reasonable time and has a decent performance.

Model Base

In this experimental section we evaluate the Model's Base on Big and Small Model, with the same pooling layers, and batch size of 128, but with different base(until 1024, because with 2048 filters on a convolutional layer our model took too much time to train). In order for our model to not become too large we trained our models for base equal to 16, 32, 64 and 128. The results are the following.



From the above graph we can clearly observe that the results become better as base increases. However, in Big Model does not have much change, instead of the Small Model, where when it has a big base it reaches the performance of the Big Model and in that case it is better to use the Small Model because it overfits less and takes the same time to train as the Big Model with base 16.

Dropout

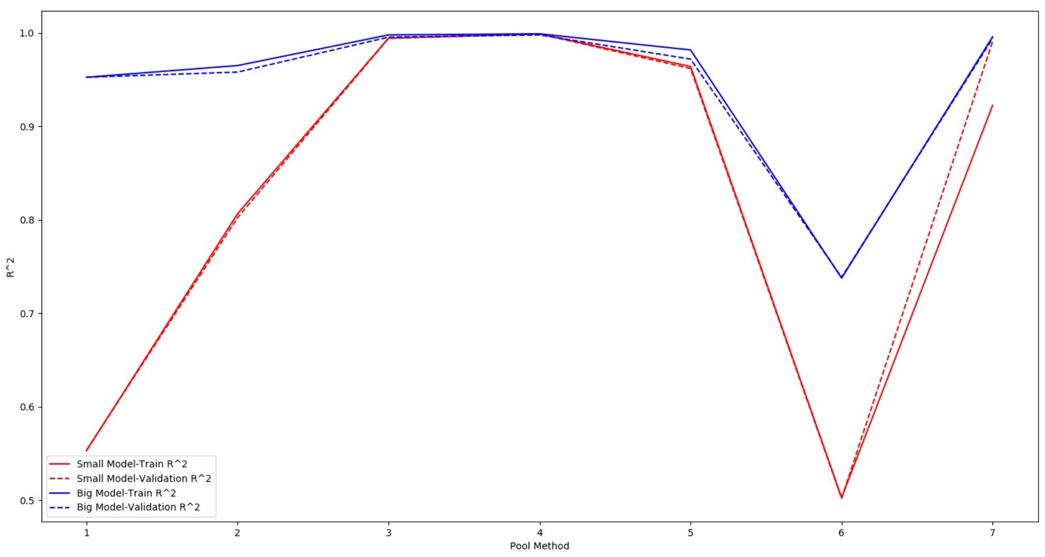
Dropout generally should be applied when the previous layer have too many neurons, because otherwise the results will be corrupted much. However, because our autoencoder performs already good and from some tests we saw that when we have dropout, autoencoder performs worse we thought that there is no point to do experiments with it.

Max Pooling

Because, $28=2^2 4$ and the pictures have width and height equal to 28, we can have the following max pooling layers(the numbers on the enumeration are the same number on the results graph)

1. one 7x7 and two 2x2 max pooling layers
2. one 7x7 and one 2x2 max pooling layer
3. two 2x2 max pooling layers
4. one 2x2 max pooling layer
5. one 7x7 max pooling layer
6. one 7x7 and one 4x4 max pooling layer
7. one 4x4 max pooling layer

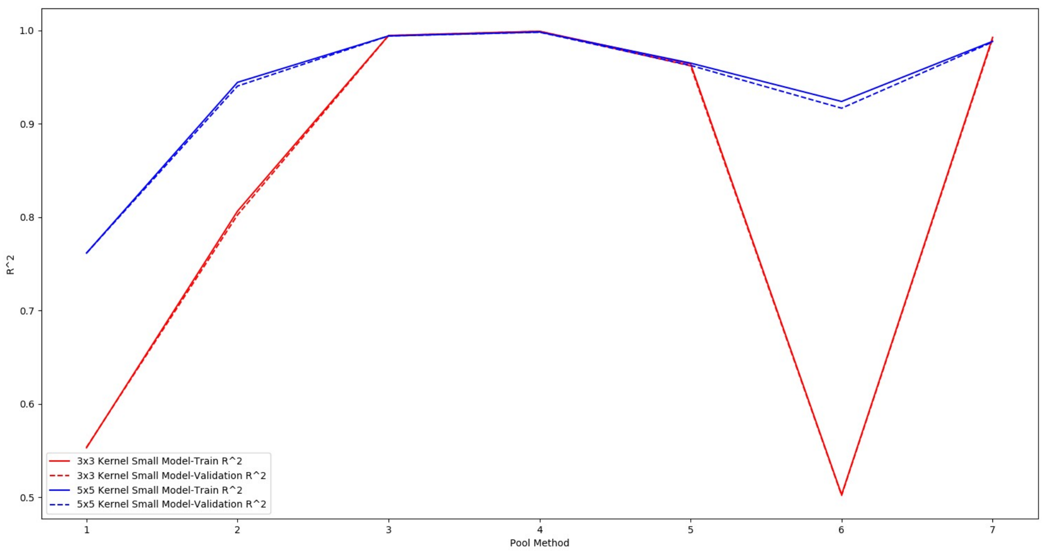
The above seven pools will be tested again on small and big model with base 16 and batch size 128. Moreover, note that that these pools are applied consecutively from the biggest pool to the smallest, beginning at the encoder's middle layer, if possible, because with these configurations we yielded better results. The results are the following:



From the results we observe that neither model overfits much. However it is seem that in methods (1) and (6) where in fact the encoder returns a column vector, the big model performs better, probably because it does dimensionality reduction to more coordinates(in big into 1024 elements while in small model 64 elements). Still, it is seem that Big Model performs better than the Small one. In terms of overfit neither model seem to overfit much with only one exception. Finally, it is clear that the third and fourth pooling methods are the best, but because fourth method does only one 2x2 pooling while third method does two 2x2 pools, we think that someone should prefer the latter pooling method because the encoder does a dimensionality reduction to less dimensions than the fourth one, thus it captures better the input features.

Kernel Size

Initially, we were reluctant if it was worth to check another kernel sizes like 5x5 because we thought that either way it would produce worse results than having a kernel size of 3x3. However, we tested the 5x5 kernel size in the Small Model that we tested on pooling methods because it produced on average worse results than the Big Model and the results are the following:



It is clear that on models that are using the 7x7 pooling layer, greater kernel size yield better results without too much overfitting. Note that, that we tried having a 7x7 kernel on convolutional layer and this yielded on first pooling method even better results. Moreover, looking more closely on the results it is seem that on the other pooling methods the model performs less better, but not much. Thus, it is pretty reasonable for someone to use 5x5(or 7x7) kernels instead of 3x3 layers.

Conclusion

Taking the above experiments, into consideration we think that the best model is the small one with base 64 that uses two 2x2 max pooling layers with 3x3 kernel size and 128 batch size in order to run on reasonable times.

Classifier Evaluation

Number of neurons

Using too few neurons in the hidden layers will result in something called underfitting. Underfitting occurs when there are too few neurons in the hidden layers to adequately detect the signals in a complicated data set.

Using too many neurons in the hidden layers can result in several problems. First, too many neurons in the hidden layers may result in overfitting. Overfitting occurs when the neural network has so much information processing capacity that the limited amount of information contained in the training set is not enough to train all of the neurons in the hidden layers. A second problem can occur even when the training data is sufficient. An inordinately large number of neurons in the hidden layers can increase the time it takes to train the network. The amount of training time can increase to the point that it is impossible to adequately train the neural network. Obviously, some compromise must be reached between too many and too few neurons in the hidden layers.

- Rules of thumbs:
    - The number of hidden neurons should be between the size of the input layer and the size of the output layer.
    - The number of hidden neurons should be 2/3 the size of the input layer, plus the size of the output layer.
    - The number of hidden neurons should be less than twice the size of the input layer.

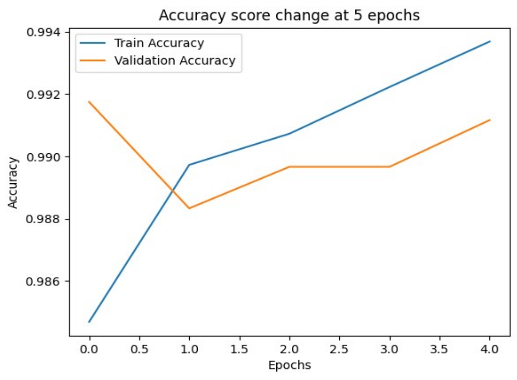
Following first rule, since input is 28x28 and output is 10(merged model) choosing neurons between 32 and 512 should make our model perform very well.
Indeed,from the tests we obtain that training with neurons = 64 or 128 or 256 we get the best results. Whereas using neurons = 1024 we can obtain a slight overfit.

Batch size

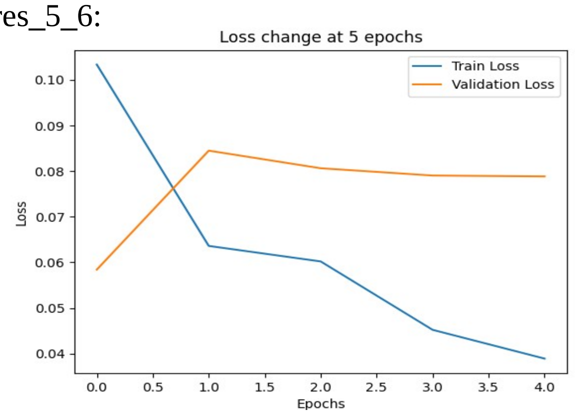
- Using a small batch size causes the model to start from high loss and end up in low loss. Training with a small batch size might require a small learning rate to maintain stability because of the high variance in the estimate of the gradient. The total runtime can be very high as a result of the need to make more steps, both because of the reduced learning rate and because it takes more steps to observe the entire training set. That being said and observed while we trained our model,in a small batch size we have to use a dropout layer because it is easier for the model to overfit. With the metricts that we use batch size between 32 and 256 performs very well.

- We obtain that the larger the batch size is:
    - The slower the training loss decreases.
    - The higher the minimum validation loss.
    - The less time it takes to train per epoch.
    - The more epochs it takes to converge to the minimum validation loss.

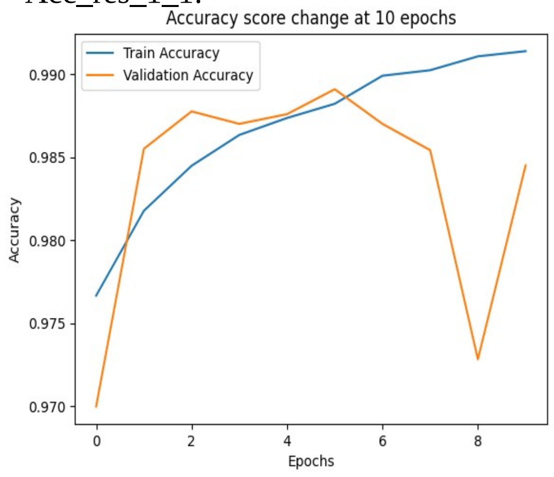
Acc_res_5_6:



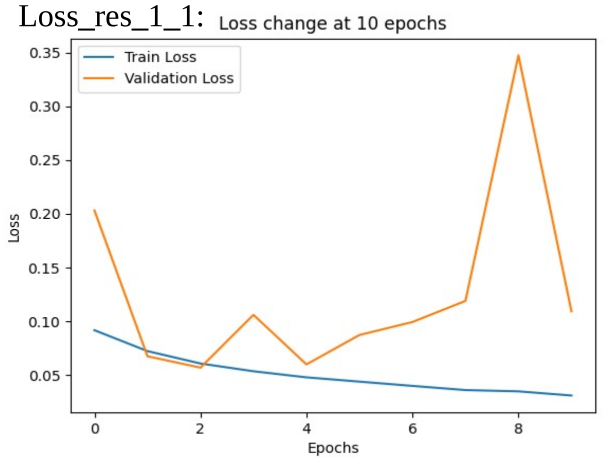
Loss_res_5_6:



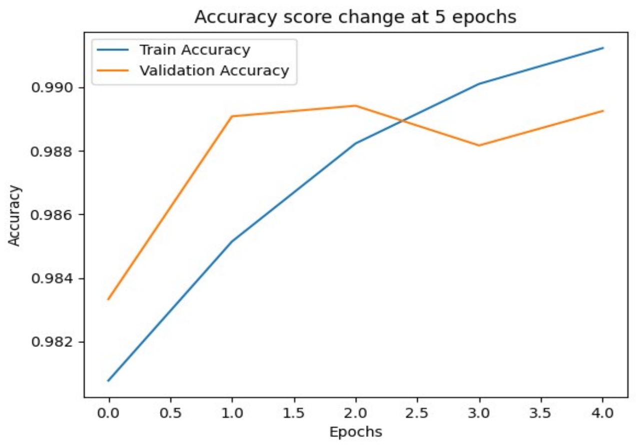
Acc_res_1_1:



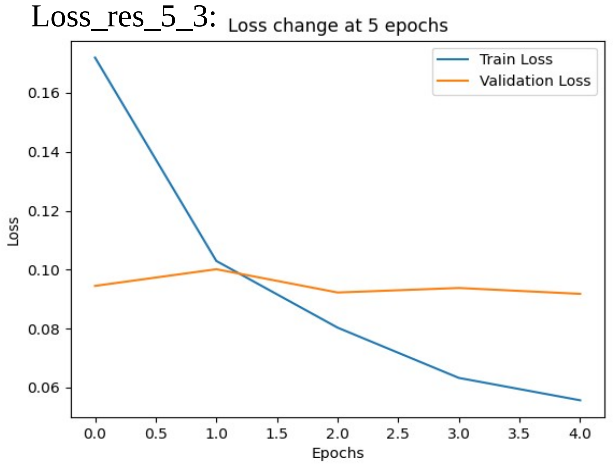
Loss_res_1_1:



Acc_res_5_3:



Loss_res_5_3:



Dropout

- Through testing we observe that a dropout layer of 0.1 or 0.2 after FC layer is enough to avoid overfit assuming we use neurons and batch size in the range described above.
- Also we can use the dropout layer after flatten layer.We observe that a little higher dropout(0.3) than the one after FC layer, performs very well,but because dropout is generally used on FC layers,we mostly used on FC layers too.

- Also,with neurons at 128 and batch size at 128(res_1_13) we observed that without dropout no overfit happens and classifier converges at 5 epochs as well as merged_model. The same applies with neurons = 64 and batch size = 32, dropout = 0(res_2_1).

Convergence

(see figures above)

- We observed that classifier model converges to 0.99 at around 5 to 10 epochs
  If we use a small batch and the more neurons we use the faster that classifier converges(res_5_2,res_5_3).Assuming that we use the values above then at most after 15
  epochs the classifier converges at 0.985-0.99.(see res_1_17 or res_2_4).
- Merge model converges to 0.99 at most after 5 epochs.

Note: See Classification/results directory for more results.