Problem Set 2

CMSC 828L

Eleftheria Briakou

October 9, 2018

1 Image classification approach

For both MNIST and Fashion MNIST datasets, I chose to experiment with simple cnn configurations, as shown in Figure 11. Following, I describe the basic structural units and hyperparameters of the system as well as the intuition behind their usage:

- **Feature extraction:** Convolutional layers produces output feature maps of the input image; the ReLu activation function is used at their outputs to introduce nonlinearities to the model. Given that the number of parameters that the model has to learn is large, I also used a pooling layer to downsample the dimensionality of the feature maps produced by the convolutional layer using the maximum operator, and finally I experimented with the dropout regularization to avoid overfitting.
- **Classification:** The second part of the network (fully connected with one hidden layer) uses the aforementioned features to classify the input image into one of the 10 classes (the softmax activation functions converts the outputs into probabilities).
- **Learning:** As our problem is a multi-class classification task I chose to minimize the categorical cross entropy, using the Adadelta()[1] optimizer from keras.
- **Hyperparameters:**
  - $(i \times i)$: the convolution filter
  - n: the number of filters
  - $k \in [0, 1)$: the dropout parameter
  - $(j \times j)$: the pooling filter
  - M: number of convolutional layers
  - H: number of nodes of fully connected layer[2]

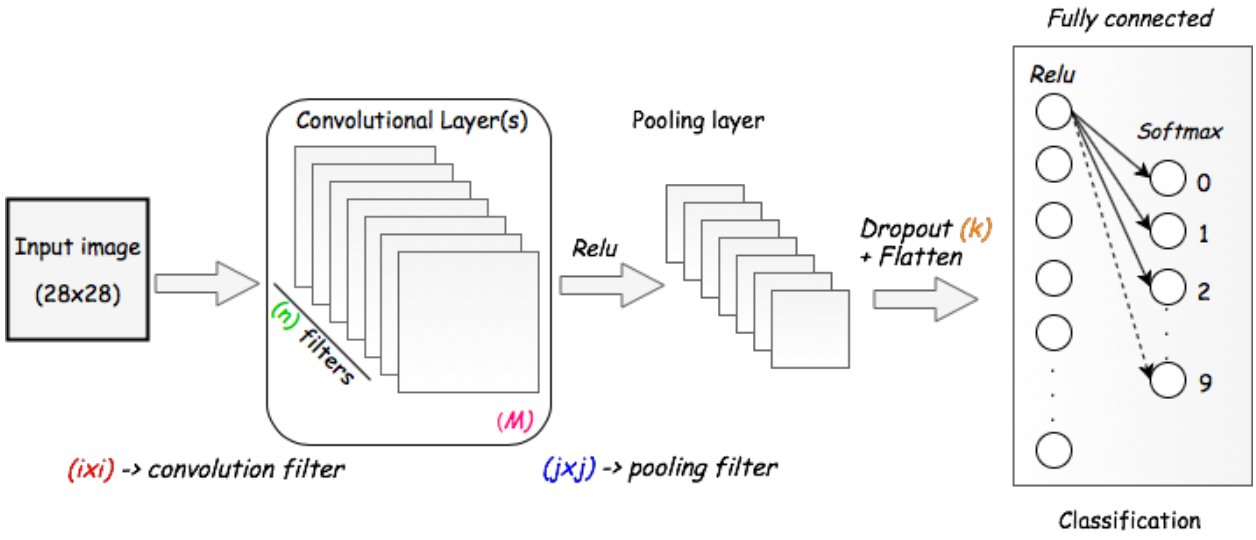


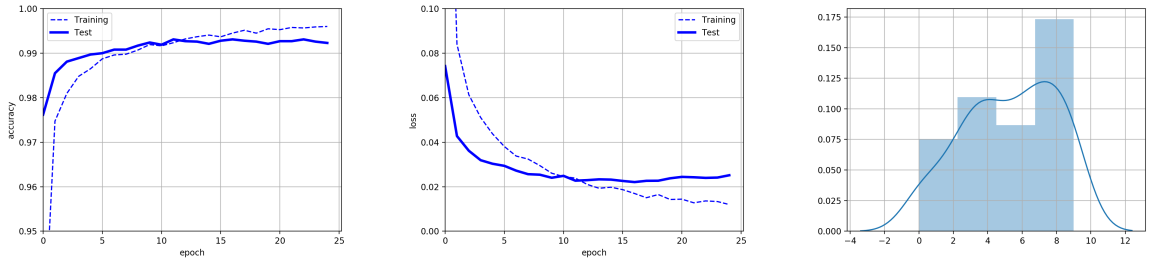
Figure 1: Abstract depiction of the basic structural elements and tunable hyperparameters of the neural network configuration used for image classification.

All in all, I tuned the hyperparameters of the above configuration using a grid search method for the two image datasets. In the following sections I present the results that I got for the best configurations. Plots of the thorough search could be found in the Appendices sections.

[1] Experimentation with other optimizers showed the adadelta performs better.

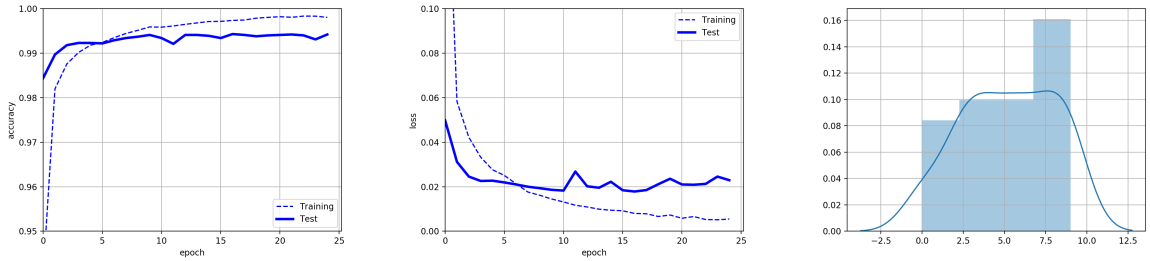
[2] The submitted code does not support a grid search for this parameter; for the following experiments its value is $H = 128$.

2 MNIST dataset



(a) Accuracy (b) Loss function (c) Distribution of misclassified

Figure 2: Results of best configuration, for $M = 1$ ($i = 7$, $n = 32$, $j = 2$, $k = 0.3$). Test accuracy: **99.23**%



(a) Accuracy (b) Loss function (c) Distribution of misclassified

Figure 3: Results of best configuration, for $M = 2$ ($i = 7$, $n = 32$, $j = 2$, $k = 0.3$). Test accuracy: **99.42**%

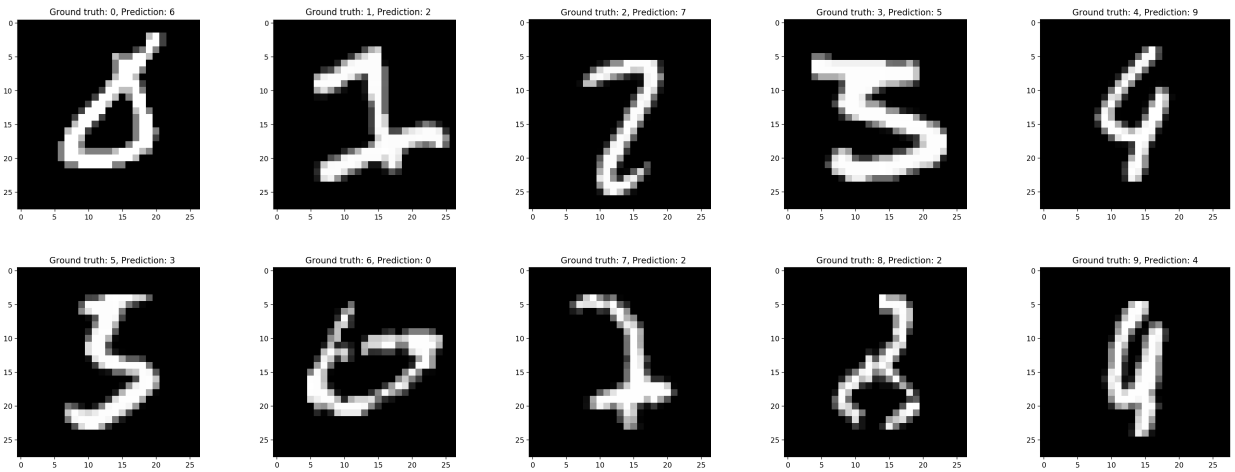


Figure 4: Misclassified examples from each cluster. Generally I noticed that most of the examples my neural network failed to correctly classify were difficult even for a human to label correctly. So, this failure is attributed to the intrinsic difficulty of the hand written recognition task, as there is a variety of different writing styles that could 'confuse' both the automatic task and humans.

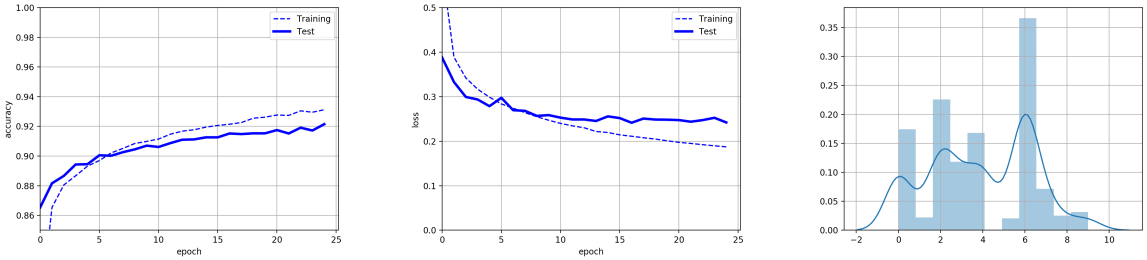
Generally, the neural network configuration that I chose seems to work sufficiently well even when a single convolutional layer is used. The testing accuracy and loss do not deviate a lot from the training results, which indicates that with the right choice of hyperparameters I avoid overfitting. Moreover — except for the above analysis— I printed out the weights and biases of the final layer after training is completed to ensure that my neural network learns reasonable parameters. This information is presented in Figure 14.[3]

[3]This is also produced by the code; I will only include this for this dataset in my report.

```
2018-10-07 22:18:04,322 : INFO : Print outs of weights (final layer): [[ 0.06306177  0.04785202  0.18371262 ... -0.04483008  0.0947363
  -0.28137362]
 [-0.09190955 -0.36204827 -0.24910194 ...  0.15235129 -0.1656088
   0.25050434]
 [ 0.15298937 -0.33838138  0.12720741 ... -0.5445941  -0.34527314
   0.10130877]
 ...
 [ 0.11407878 -0.23308662 -0.37369508 ...  0.2937545  -0.36080208
  -0.38828963]
 [ 0.1793865   0.23781414  0.27612907 ... -0.19750199 -0.20444243
  -0.22474931]
 [-0.35866195  0.15809442  0.21962786 ... -0.2512921   0.21070586
  -0.415122  ]]
2018-10-07 22:18:04,322 : INFO : Print outs for biases (final layer): [-0.03768442  0.08282702 -0.12108482  0.04221249  0.01632063 -0.03158664
 -0.11115552 -0.07987185 -0.00967062  0.01520945]
```

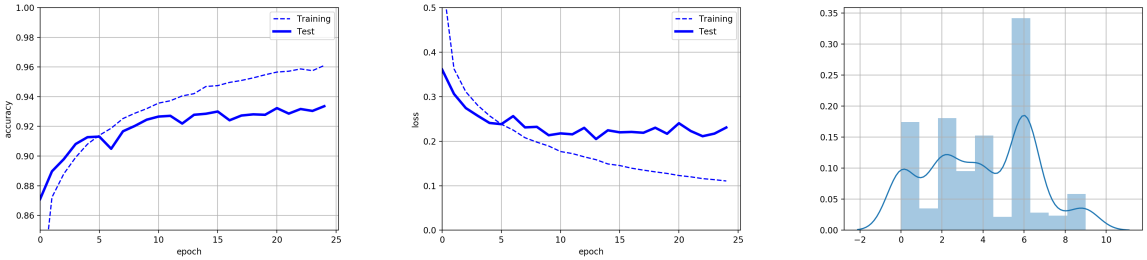
Figure 5: Representative print outs of weights and biases.

3 Fashion MNIST



(a) Accuracy (b) Loss function (c) Distriuubtion of misclassified

Figure 6: Results of best configuration, for $M = 1$ ($i = 3$, $n = 64$, $j = 2$, $k = 0.3$). Test accuracy: **92.15**%



(a) Accuracy (b) Loss function (c) Distriuubtion of misclassified

Figure 7: Results of best configuration for $M = 2$ ($i = 3$, $n = 64$, $j = 2$, $k = 0.3$). Test accuracy: **93.36**%

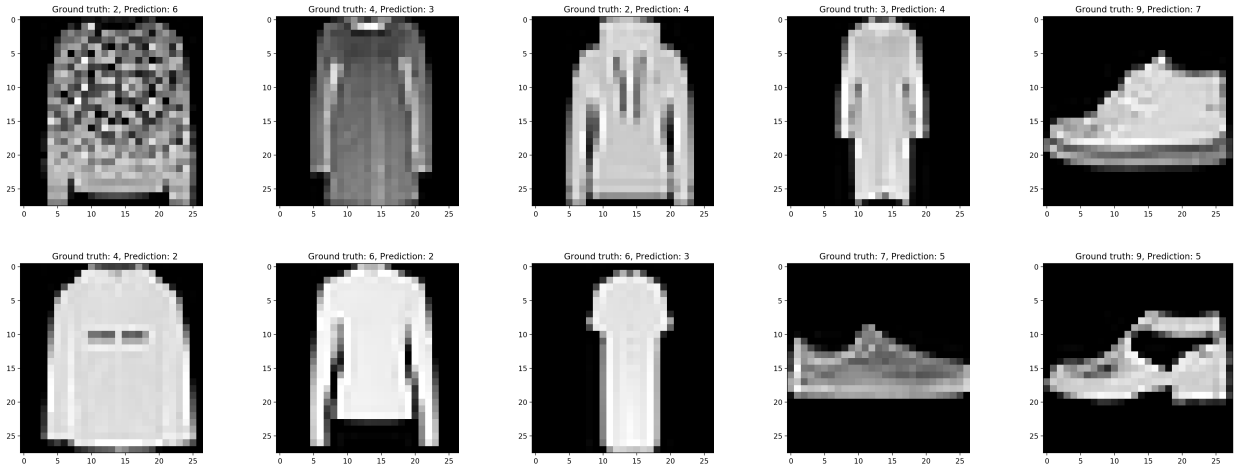


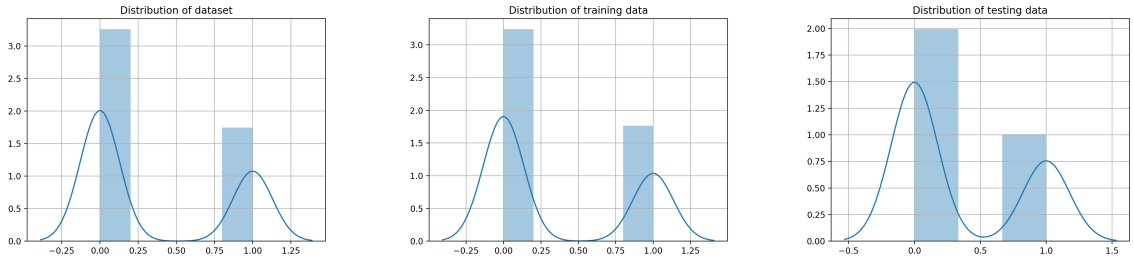
Figure 8: Misclassified examples from each cluster. Again, I notice that most of the misclassified examples were also hard to classify for humans. Encouragingly the wrong predictions are correlated with the ground truth values (e.g., instead of predicting that an image is a sandal it predicts a sneaker, which are both shoes).

Generally, the Fashion MNIST dataset is more challenging than MNIST. However, the same architecture produces satisfactory results for both datasets which implies that there is a correlation between them. It also worth mentioning though that different hyperparameters lead to the best results for each of them.

4 Breast Cancer

4.1 Data preparation

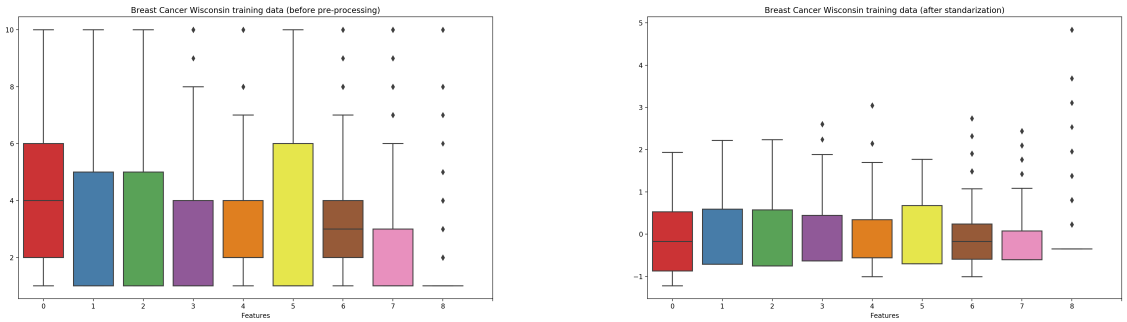
For this binary classification task I had to pre-process the given data, before start training the model. To gain an inside into the data, I plotted the distribution of the dataset over the two candidate classes, where 0 stands for *benign* and 1 for *malignant*. Figure 9 (a) depicts the results. Generally, I noticed that the dataset is unbalanced among the two classes; this imbalance should also be reflected to both the training and test sets (as it could be interpreted as a prior knowledge for these two classes), as seen in Figure 9 (b) and (c). For the purposes of this homework I didn't create a validation set, so the test set is going to be used for avoiding overfitting in the hyper-parameter tuning section.



(a) Dataset (b) Training (85%) (c) Test (15%)

Figure 9: Distribution of breast cancer dataset and training/test subsets over the two candidate classes.

Except for splitting the data, I also performed data standarization, so that the features are normally distributed (mean 0, variance 1). In Figure 10 (a) we can notice the high variance in the distributions of the features, and how this is addressed after the standarization in Figure 10 (b). The reason for this rescaling is that all features should lie in the same range, so that they could contribute equally to the output result; also, gradient descent works better when this pre-processing step is applied.



(a) (b)

Figure 10: Boxplots depicting the distribution of features before and after applying standarization.

4.2 Neural network

For this task a chose to experiment with a fully connected neural network as shown in Figure 11. Following, I describe the basic structural units and hyperparameters of the system as well as the intuition behind their usage:

- **Architecture:** Fully connected neural network with one or more hidden layers that uses the ReLu activation function at the outputs of the hidden nodes to introduce nonlinearities to the system. Finally, the softmax activation function is used at the output layer and converts the outputs into probabilities over the two candidate classes.

- **Learning:** As our problem is a binary classification task I chose the binary cross entropy as the loss function, with Adam() stochastic optimizer from keras, since it consists the recommended technique[4]. As for my evaluation metric I used accuracy. However, I should mention that ideally precision-recall would have been better metrics for this task, since there is also a class imbalance in the dataset. Although, these metrics are currently removed from keras.
- **Hyperparameters:**
  - N: number of nodes of hidden layer
  - b: batch size
  - L: number of hidden layers

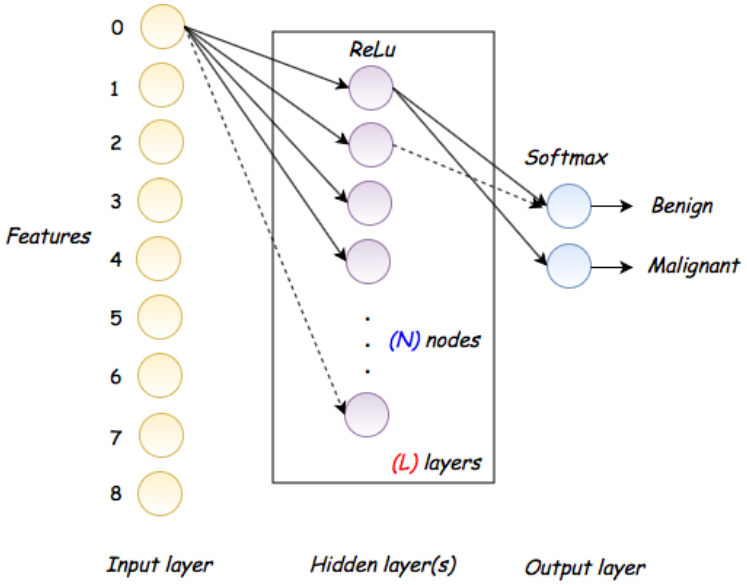
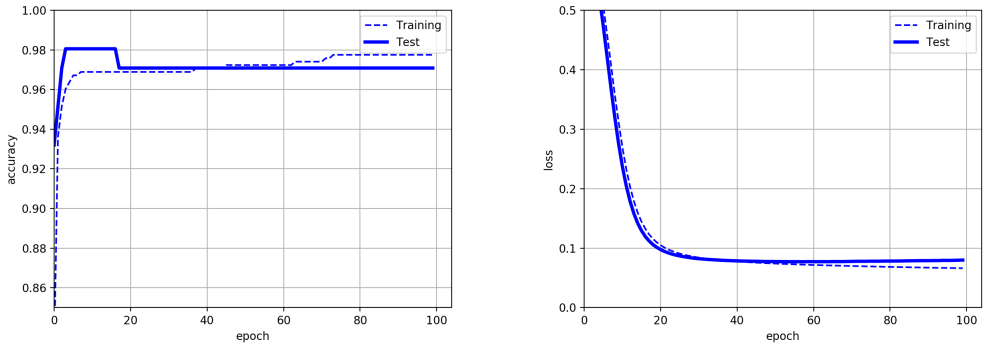


Figure 11: Abstract configuration of fully connected neural network used for binary classification on Breast Cancer Wisconsin Data set.



(a) Accuracy (b) Loss function

Figure 12: Results of best nn configuration ($N = 30$, $b = 100$, $L = 1$). Test accuracy: **97.09**%

Given that the number of features (9) and the complexity of the problem (binary) is small, a simple hidden layer was found to be enough to produce satisfactory results in terms of accuracy. The above configuration leads to the best results (Figure 12), after a grid search over the hyperparameters. (For a thorough study of the grid search results see the Appendix.)

Furthermore, the testing accuracy and loss do not deviate a lot from the training results, which indicates that with the right choice of hyperparameters I avoid overfitting. Moreover — except for the above analysis— I printed out the weights and biases of the final layer after training is completed to ensure that my neural network learns reasonable parameters. This information is presented in Figure 14.[5] Finally, Figure 13 shows the 3 incorrect classified examples for my best configuration.

[4]Experimentation with other optimizers from keras has also shown that Adam() gives the better results.
[5]This is also produced by the code; I will only include this for this dataset in my report.

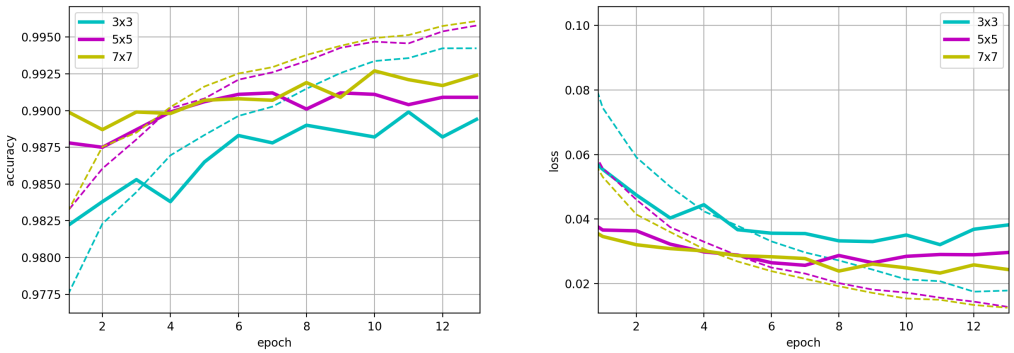
```
2018-10-08 23:04:38,152 : INFO : Misclassified data: [0, 0, 0]
2018-10-08 23:04:38,153 : INFO : 0 Incorrect classified example: [ 0.18458885 -0.06008473  0.24451762  0.08740399  0.35485595  0.39428312
  0.2321961   1.4107983  -0.347963  ]
2018-10-08 23:04:38,153 : INFO : 1 Incorrect classified example: [ 0.5338599   1.8907038   1.2368706   0.80523247  0.80841357  1.217444
  0.2321961  -0.2692762  -0.347963  ]
2018-10-08 23:04:38,153 : INFO : 2 Incorrect classified example: [ 0.18458885 -0.7103476  -0.41705108  2.5998037   0.35485595  0.39428312
 -0.5925986  -0.6052911  -0.347963  ]
```

Figure 13: Misclassified examples from each cluster. Generally I noticed that all of the examples that my neural network failed to correctly classify belonged to the 0 class. This could be attributed to the fact that the number of examples in this class is generally larger than that of the second one.

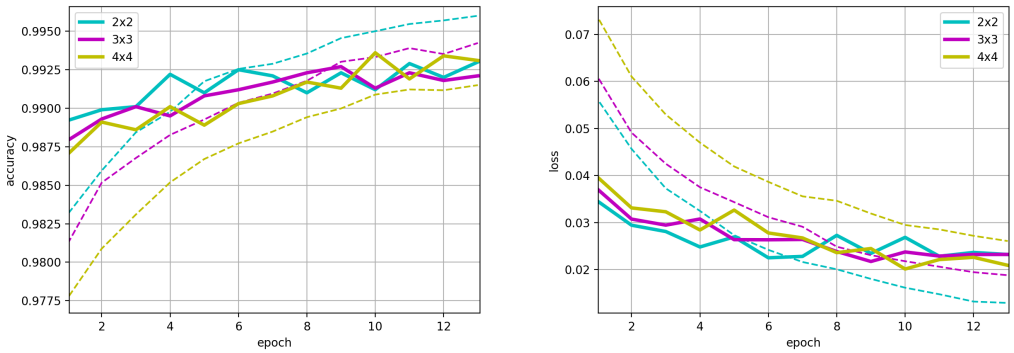
```
2018-10-08 23:04:32,297 : INFO : Print outs of weights (final layer): [[ 0.22649065 -0.24646872]
 [-0.22310796  0.2735642 ]
 [ 0.46856448 -0.38092867]
 [-0.24736886  0.26357687]
 [-0.3152401   0.24870183]
 [-0.05654091  0.16089416]
 [-0.14352556  0.25335032]
 [-0.0477487   0.11166039]
 [-0.06786313  0.03039065]
 [-0.17856105  0.27429208]
 [-0.14710557  0.2744668 ]
 [-0.10934357  0.17384703]
 [ 0.23314434 -0.23450577]
 [-0.12833889  0.11561456]
 [-0.19381028  0.19405493]
 [ 0.25775167 -0.287764  ]
 [-0.19173948  0.19169323]
 [-0.15132004  0.14744368]
 [ 0.39299765 -0.4501297 ]
 [-0.16029368  0.06522088]
 [-0.23649089  0.20346025]
 [-0.01050247  0.07769499]
 [ 0.38538063 -0.28087   ]
 [-0.11510061  0.13167056]
 [ 0.3532574  -0.32478228]
 [ 0.5421851  -0.53829914]
 [ 0.13425201 -0.24196614]
 [-0.25232372  0.25414976]
 [ 0.49531975 -0.48615605]
 [ 0.35480517 -0.2944202 ]]
2018-10-08 23:04:32,298 : INFO : Print outs for biases (final layer): [ 0.03748452 -0.03748439]
```

Figure 14: Representative print outs of weights and biases.

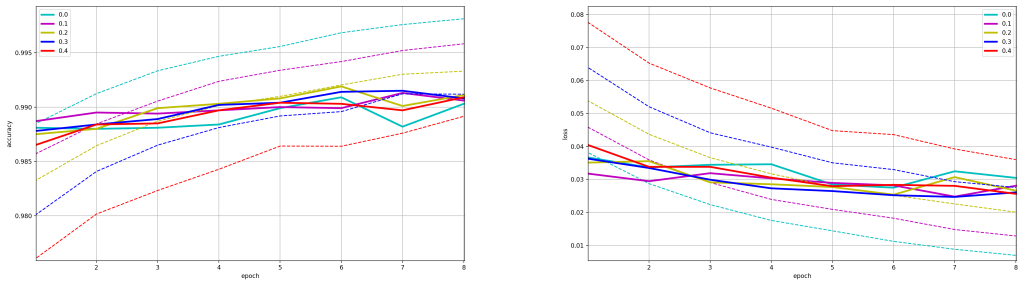
4.3 Appendix: Hyperparameter tuning



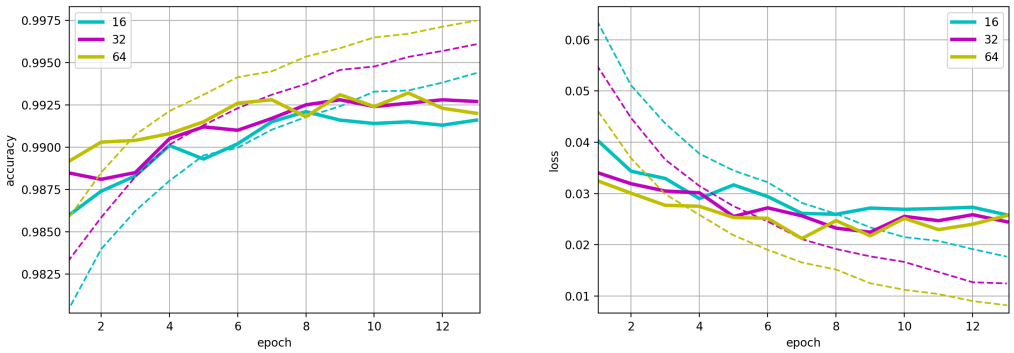
(a) Different filter sizes.



(b) Different pooling sizes.

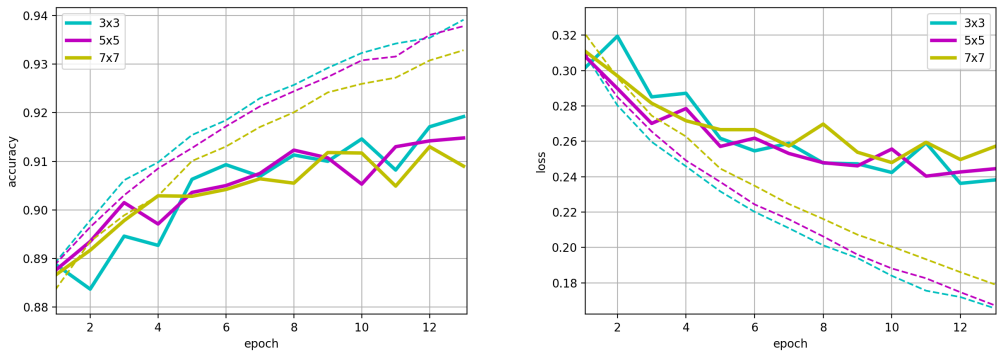


(c) Different dropout configurations.

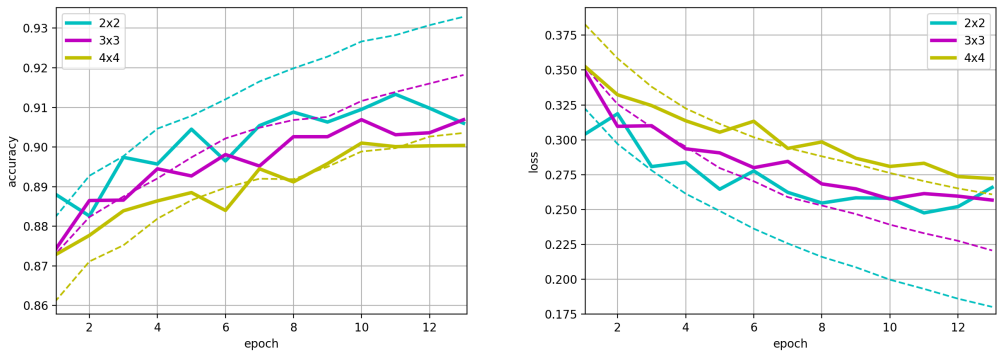


(d) Different dropout configurations.

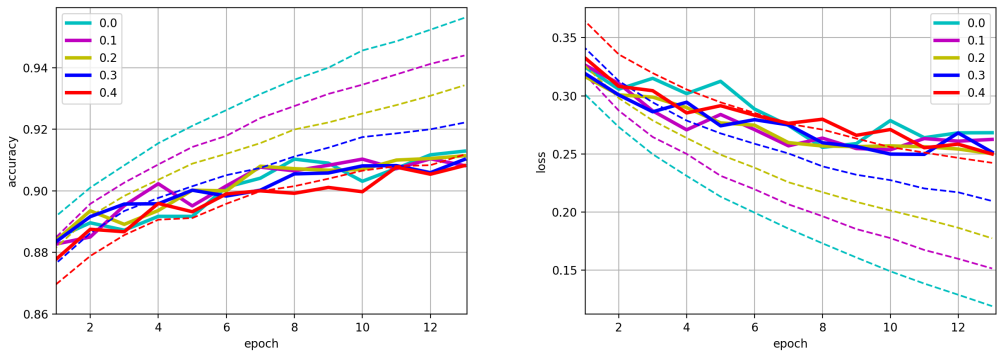
Figure 15: Tuning hyperparameters of cnn layer for MNIST classification. Dashed lines correspond to accurancies and losses of training sets; continuous lines correspond to test sets.



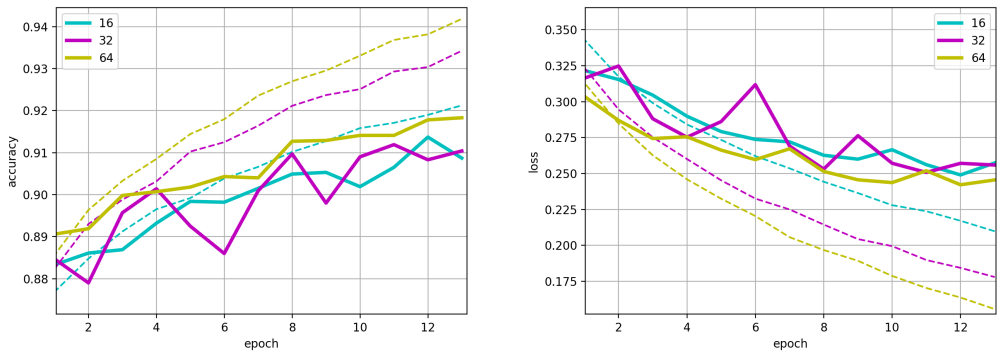
(a) Different filter sizes.



(b) Different pooling sizes.

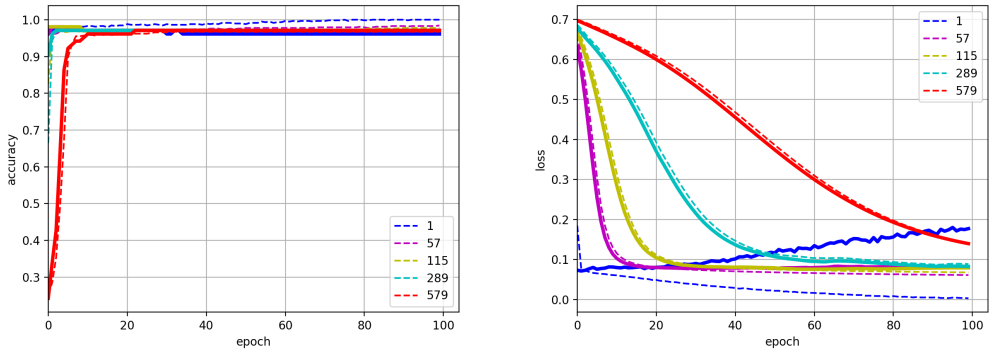


(c) Different dropout configurations.

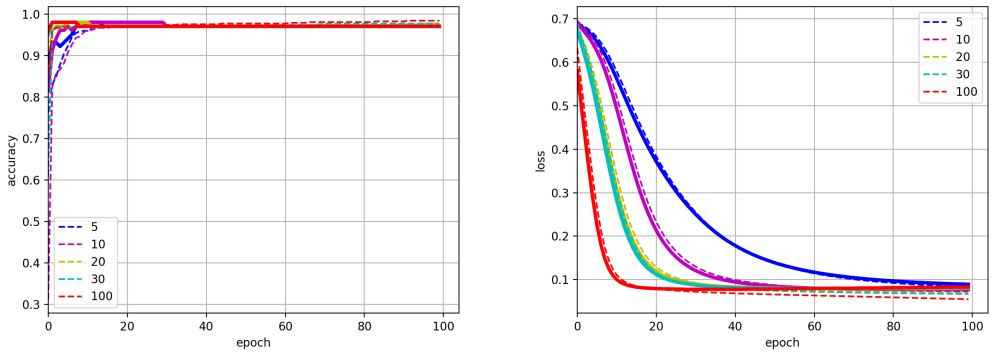


(d) Different dropout configurations.

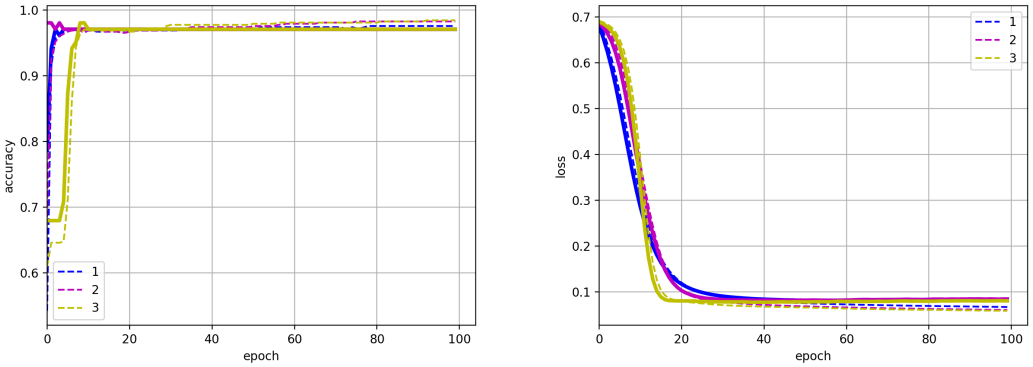
Figure 16: Tuning hyperparameters of cnn layer for FMNIST classification. Dashed lines correspond to accurancies and losses of training sets; continuous lines correspond to test sets.



(a) Different batch sizes (b).



(b) Different node sizes for hidden layer (N).



(c) Different number of hidden layers (L).

Figure 17: Tuning hyperparameters of ann layer for breast cancer classification. Dashed lines correspond to accuracies and losses of training sets; continuous lines correspond to test sets.