

Machine Learning Project

Task 1

Duy Nguyen Dinh
dinh@rhrk.uni-kl.de

Minh Duc Duong
duong@rhrk.uni-kl.de

—
—

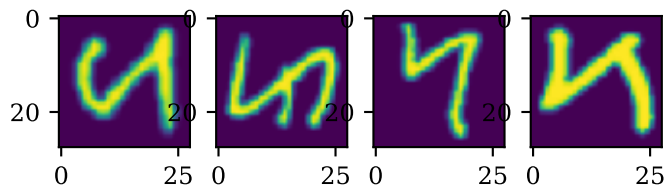
November 29, 2021

1 K Nearest Neighbor Classification

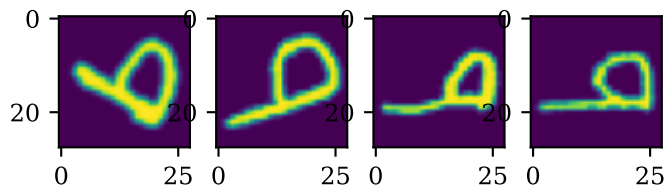
1.1 Classes of the data set

First we look into the data set, which will be used to fit the model as well as evaluate the accuracy of the model. The data consists of a total of 15000 images, each one was assigned with a specific label. Here is what each class in the data set looks like:

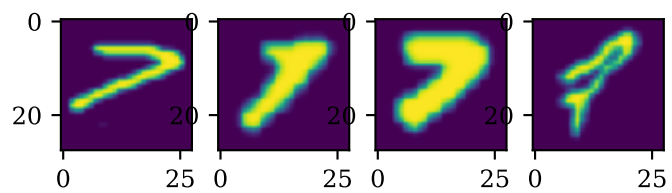
Class 0



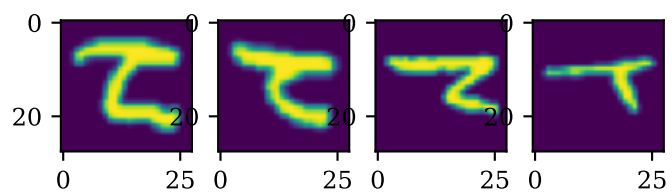
Class 1



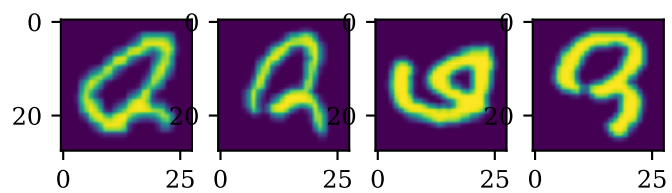
Class 2



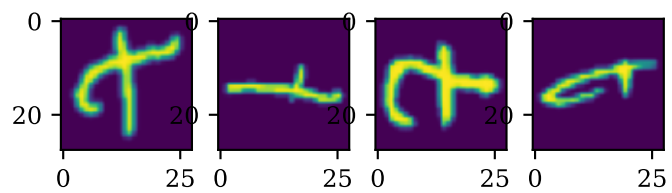
Class 3



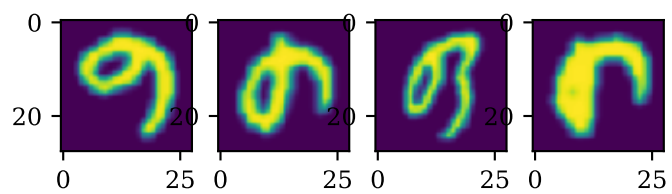
Class 4



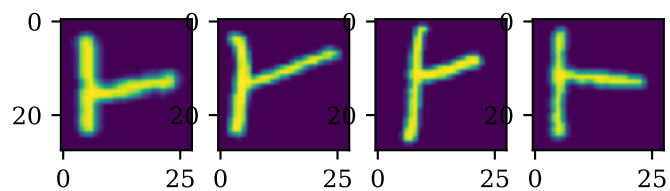
Class 5



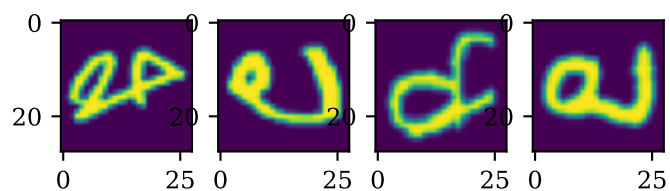
Class 6



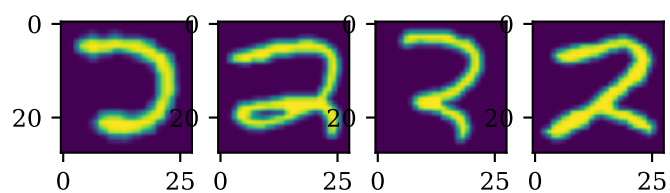
Class 7



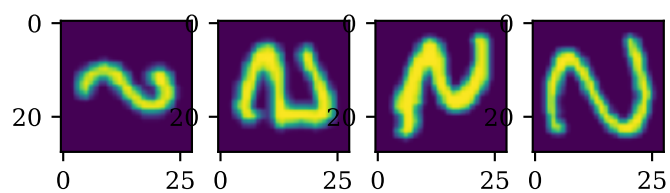
Class 8



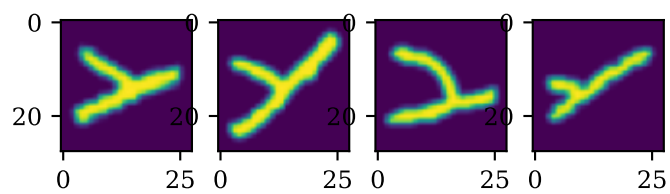
Class 9

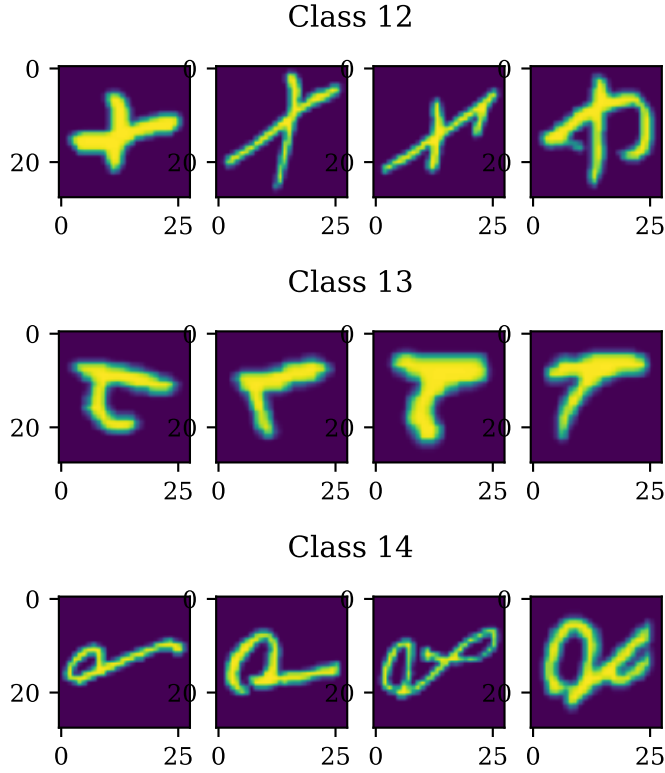


Class 10



Class 11





There are 15 different classes in the data, which was counted from 0 to 14. Each of these classes may have a unique pattern: For example, Images in class 7 look like a letter 'T', which was rotated 90 degree to the left. Images in class 1 look like a letter 'q', which was rotated 90 degree to the right. It is important to note that through normal observation, it can be very challenging to recognize the pattern or the common properties of all images in the same class. Therefore it is necessary to create a classification, which can be used to determine the class of object that can not be done with simple human observation.

1.2 Implementation of KNN

A class was implemented to define every KNN-classifier. A classifier has many different properties:

- `k`: The amount of images that we need to identify the label of calculating image
- `dist_function`: The method to calculate the distance between image

- **filter**: Filter For the calculating image, which is required later into the task
- **return_neighbor**: Check if we want to save the k-nearest neighbors for each image, which is useful later to identify neighbors of misclassified images

Listing 1: KNN class

```
class KNN:
    def __init__(self, k=5, dist_function=euclidean_distance
                , filter=None, return_neighbor=False):
        self.k = k
        self.dist_function = dist_function
        self.return_neighbor = return_neighbor
        self.filter = filter

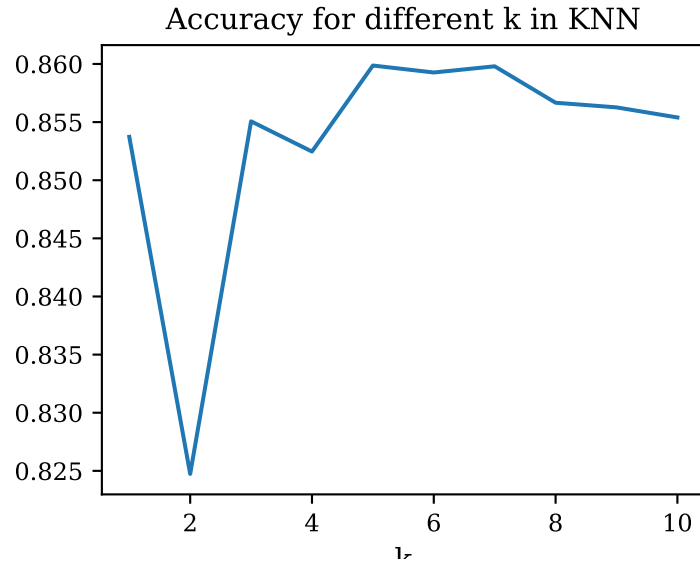
    def fit(self, X, y):
        if self.filter is not None:
            X = np.array([convolve(i, self.filter) for i in X])
        self.train_x = X
        self.train_y = y

    def predict(self, X):
        if self.filter is not None:
            X = np.array([convolve(i, self.filter) for i in X])
        return [knn(x, self.train_x, self.k, self.train_y
                    , self.dist_function, self.return_neighbor) for x in X]
```

The fit method will get the data as "train data", so we will calculate the distance between the images, which we need to predict the label, with these "train-images". The method predict will call the function knn, which calculate the label for each image in the set. The algorithm of this function was demonstrated in the task sheet and the implementation can be found in knn.py

1.3 m-fold cross validation on different k

We implemented m-fold cross validation and the Euclidean distance functions as described in the task sheet. As the sheet required, we run 5 fold cross validation on the entire set of 15000 images with k vary from 1 to 10.



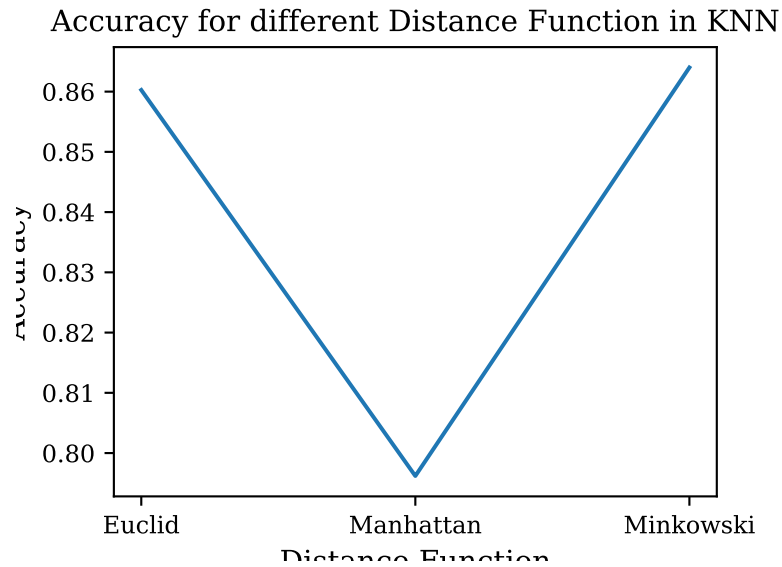
The peak of accuracy is achieved with $k=5$. Therefore all of the classifier will have $k=5$ from now on.

1.4 Is the best current k a good choice?

While it is a good indicator to choose the k , which has the best accuracy at the moment, it may not be the best choice to test on new fresh images. As more images will be used to fit the classifier, new images come with many unseen properties or the distribution of classes can be changed. If we use the approach KNN, it should be a good practice to test out on different values of k , especially since k is not the only factor that decides the accuracy of the model (for example the amount, quality and distribution of data also matters).

1.5 KNN with different distance metric

The Euclidean distance function is a specific case of Minkowski distance function with the scalar of 2. This time we test the accuracy of Euclidean metric compared to Manhattan metric (Minkowski with scalar of 1) and Minkowski metric with scalar of 3.



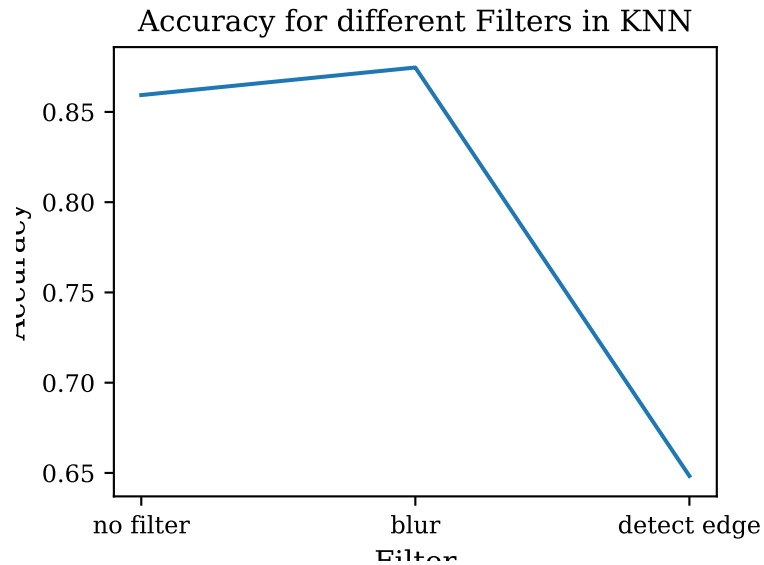
The result shown, the accuracy of Euclidean metric and 3-Minkowski metric are similar and Manhattan metric has a 6% lower in accuracy.

1.6 Feature Engineering

The convolutional operation is implemented in `knn.py` with the instruction presented in the task sheet.

1.7 Experiment with filters

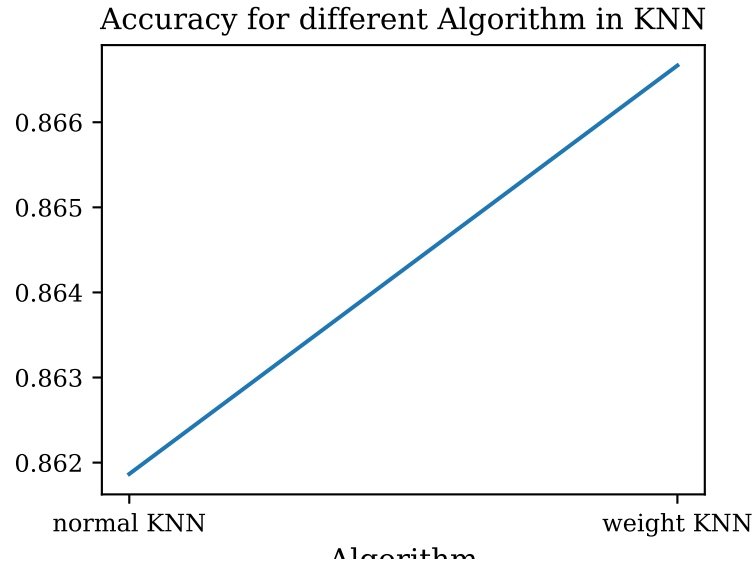
With the implemented convolutional operation, we test the classifier in 3 different cases (without filter, blur filter and edge filter) using 5-fold cross validation.



Blur filter improve our KNN-classifier a little while we have a significantly drop in accuracy with edge detection approach.

1.8 Weight KNN

Now we assigned weight into each neighbor, which is inversely proportional to the distance. This means the closer the neighbor to the test image, the more impact it will make to the result of the prediction. We modify our knn function with a new added parameter `inverse_modifier` to calculate the weight of each neighbor ($\text{inverse_modifier} / \text{distance}$)

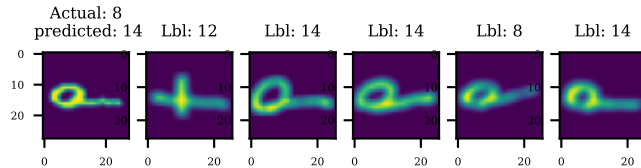


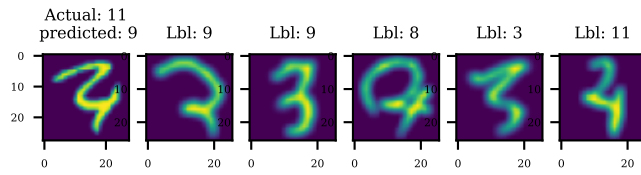
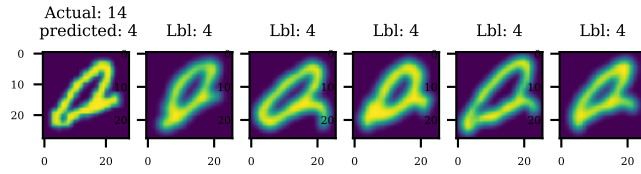
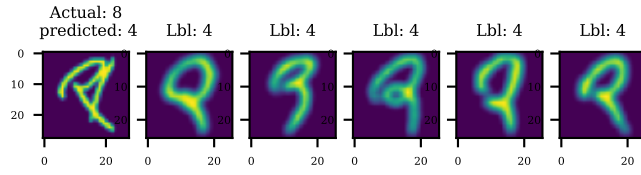
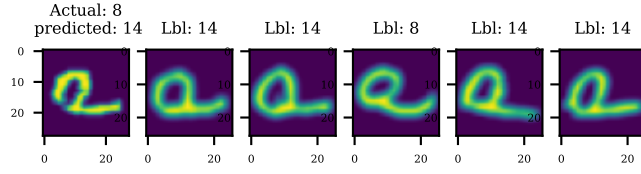
We find very small increment in accuracy with Weight KNN.

1.9 Misclassified samples

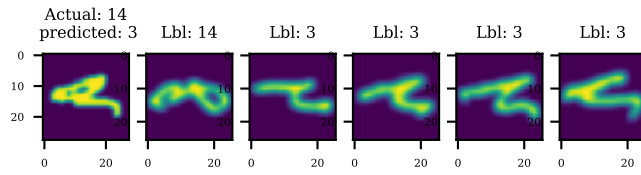
As the task sheet required, for at least three different distance measures and features obtained from two different filters, we plot the nearest neighbors to 5 random misclassified samples. To fulfill the requirement, we plot 3 different cases.

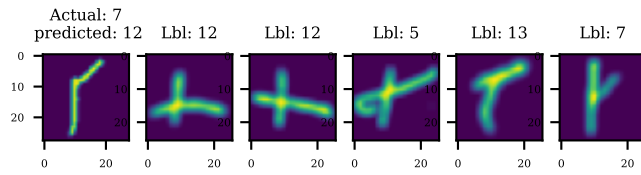
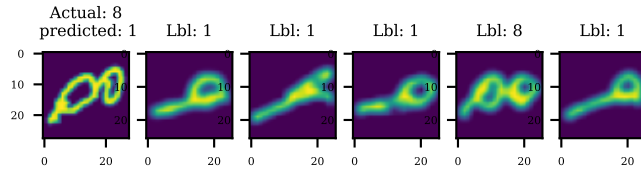
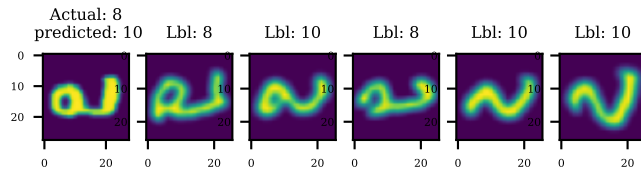
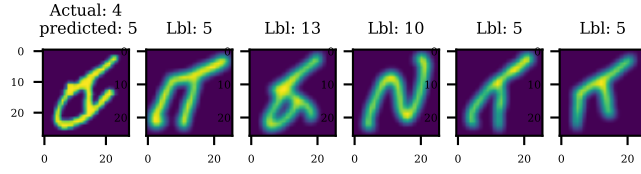
The first case is with a classifier with Euclidean distance metric and blur filter:



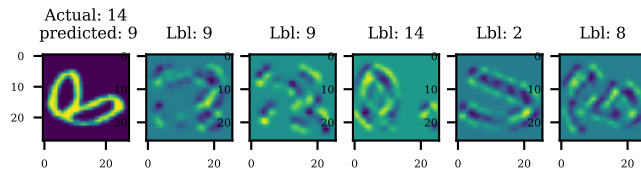


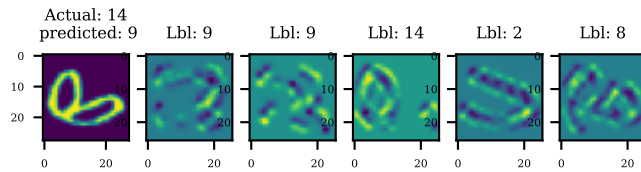
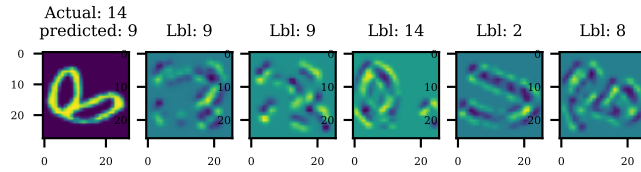
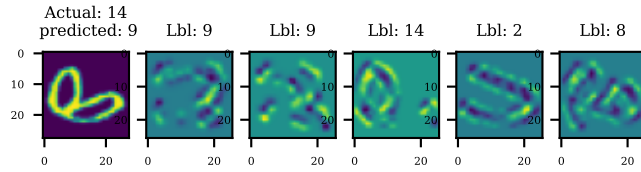
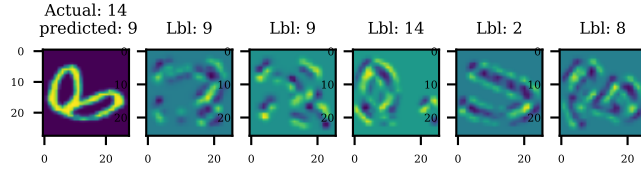
The second case is with a classifier with Manhattan distance metric and blur filter:





The third case is with a classifier with Manhattan distance metric and edge filter:





Many misclassified images has a very close pattern compare to their neighbors but the result of the predictions are still incorrect. Because we have many images of different classes that look identical to each other, many was predicted with the wrong label.

1.10 Conclusion

We implemented the algorithm for K-nearest-neighbors, which calculates the distance between the current image and other fitted images in the classifier. The classifier uses one of three different metrics, based on the scalar of

Minkowski distance function. The accuracy of each model is determined by m-Fold-Cross-Validation. We also implemented convolution operation with filter. Weight-KNN does not count the number of each labels from k-nearest neighbors but the closer the neighbor to the calculating image, the more impact it will make to the prediction. Lastly we shown the misclassified samples from different models.

Most of the classifiers use $k=5$ as this value yield the best accuracy from range of 1 to 10.

2 Deep Neural Network

2.1 Neural Network Term

Before implementing, we need to present some definitions related to neural network.

- **Activation function:** Activation function defines the output of each neuron in the network, given (set of) input. Some of the functions are Sigmoid, ReLU
- **Loss function:** Loss function is the function that calculate the prediction error of the model, such as Mean Square Error, Cross Entropy...
- **Parameter** is specifications from the model, which can be estimated from the data (e.g. weight, bias). **Hyper-parameter** is specifications that there are no clear way to optimize and need to manually changed to find the best values. (e.g. k in KNN classifier, learning rate,...)
- **Optimizer:** Algorithm/Method used to change the hyper-parameters of the network for examples: learning rate, weight... (Adam, Gradient Descent...)
- **Epoch:** Training all the given data in one cycle
- **Under-fitting:** Learning a too simple classifier that is not complex enough to describe the training data well
- **Over-fitting:** Learning a too complex classifier that fits the training data “too well” (does not “generalize” to new data)
- **Training set:** Training set is a set a data, which is used to train a model. Normally given as a pair of input and the actual output.

- Test set: Test set is a set of data, which is used to test how good (accurate) the current model is.
- Validation set: Validation set is a set of data which is used to tune the hyper-parameters of the model.

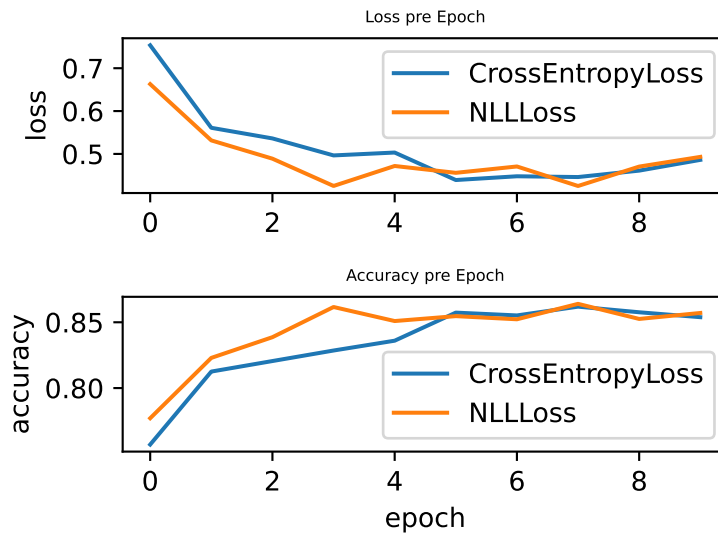
2.2 Implementing Deep Neural Network

As for the given task we implemented a linear neural network (layer: 784, 256, 64, 15) and convolution neural network (layer: 1, 6, 16, 15).

- learning_rate = 0.01
- batch_size = 128
- num_epoch = 10
- Loss function: Cross Entropy
- Optimizer: Adam

2.3 Changing Specification

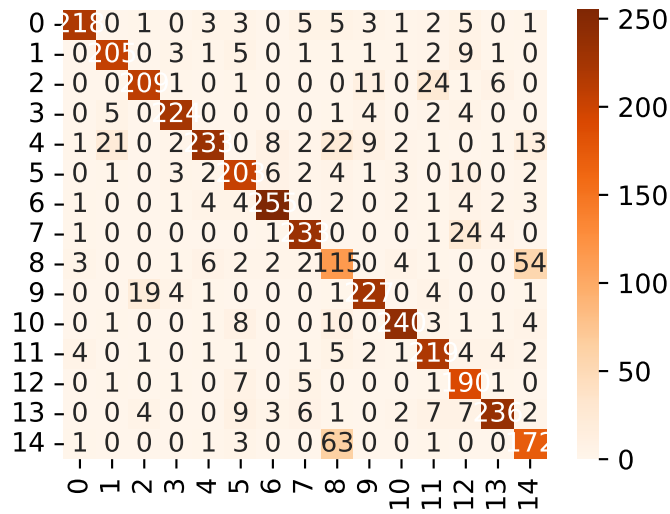
As we found our best number of layers and neurons for each layer. We compare our current CNN with another but with different loss function.



After 8 epochs, the accuracy of each model peaks and close to each other.

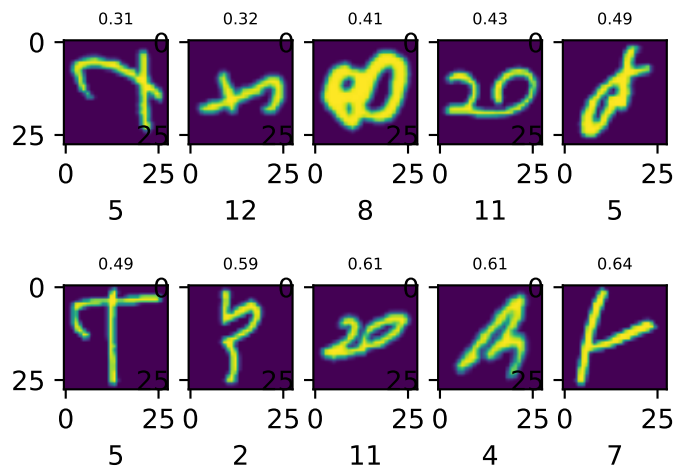
2.4 Confusion Matrix and Confidence

Confusion Matrix for our current CNN

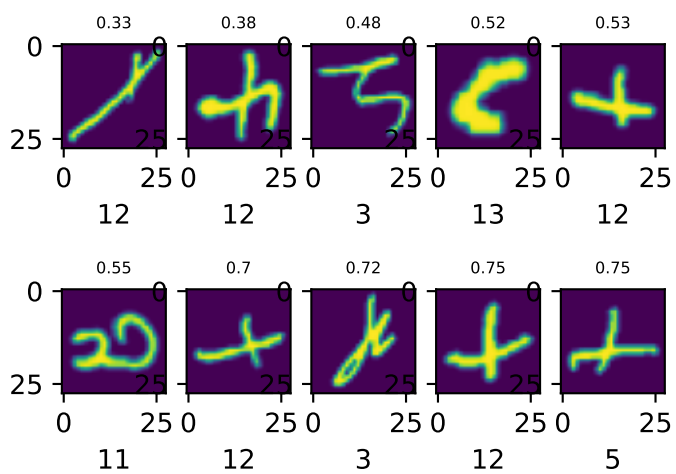


10 most unconfident each class:

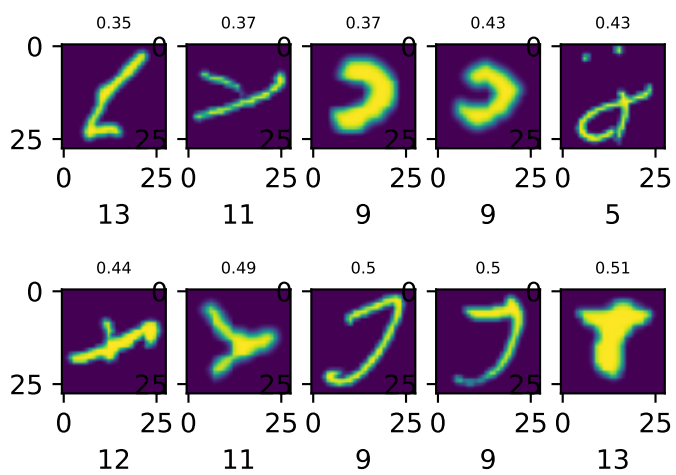
Incorrect, most unconfident images, Class 0



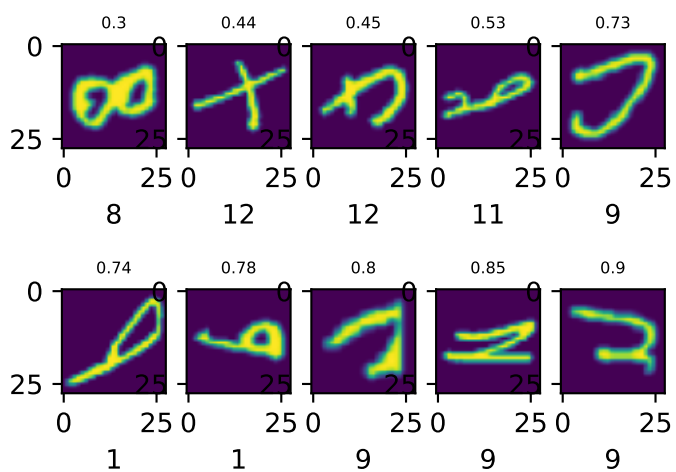
Incorrect, most unconfident images, Class 1



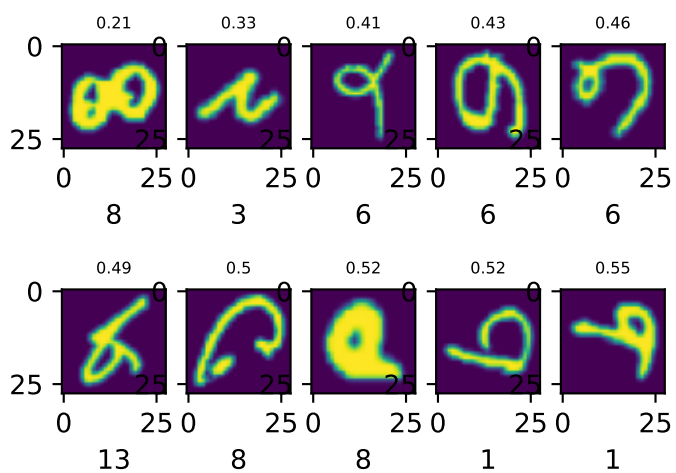
Incorrect, most unconfident images, Class 2



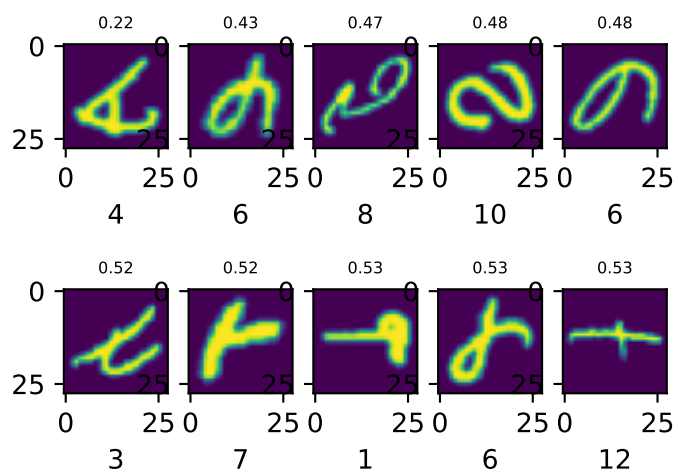
Incorrect, most unconfident images, Class 3



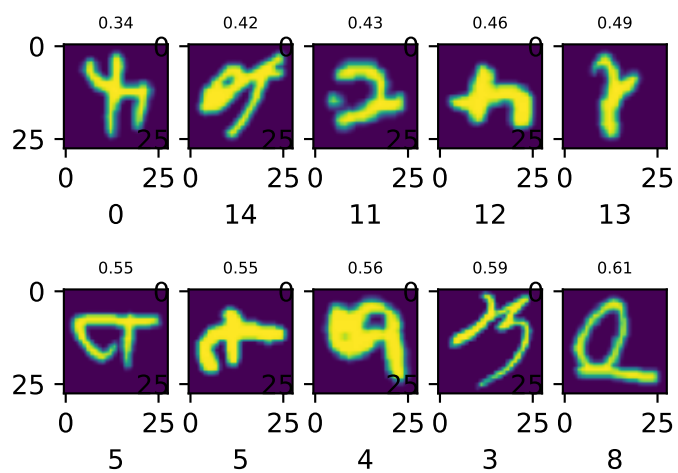
Incorrect, most unconfident images, Class 4



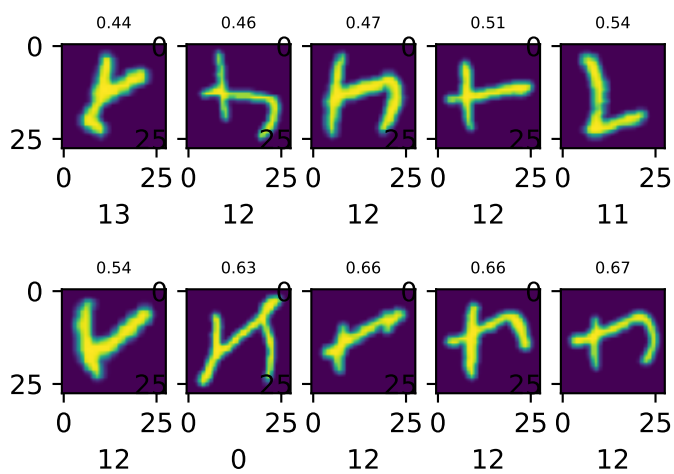
Incorrect, most unconfident images, Class 5



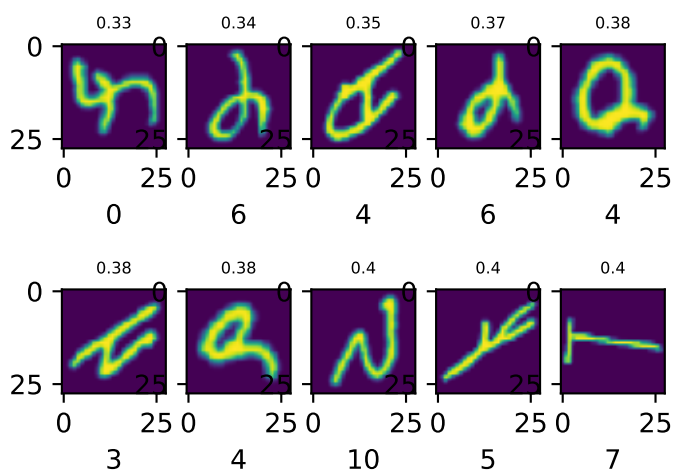
Incorrect, most unconfident images, Class 6



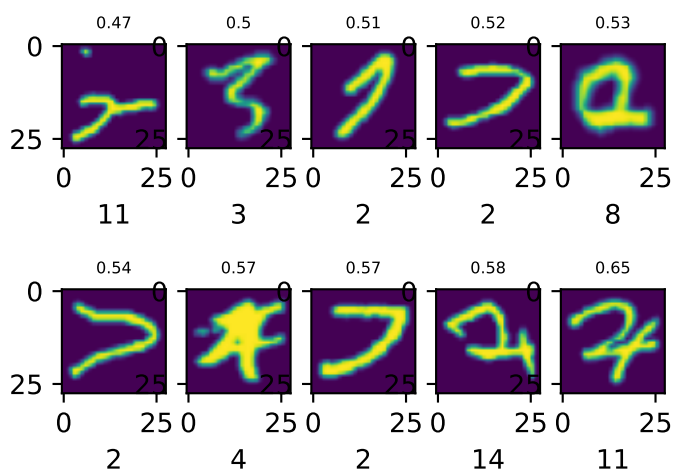
Incorrect, most unconfident images, Class 7



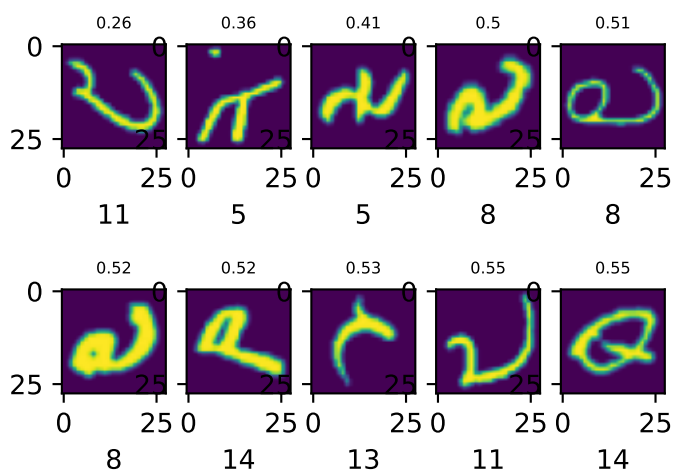
Incorrect, most unconfident images, Class 8



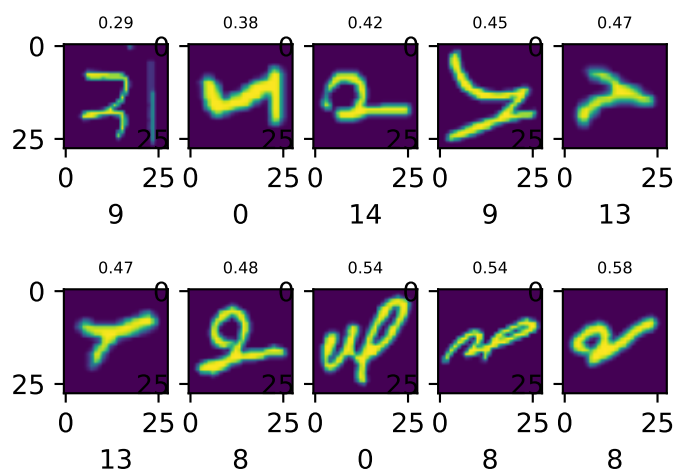
Incorrect, most unconfident images, Class 9



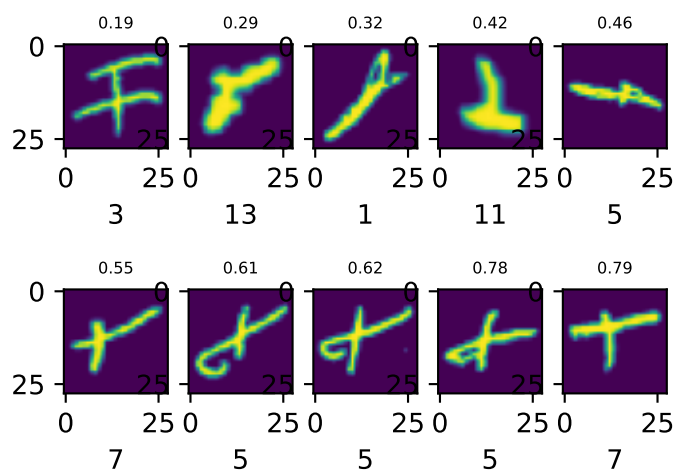
Incorrect, most unconfident images, Class 10



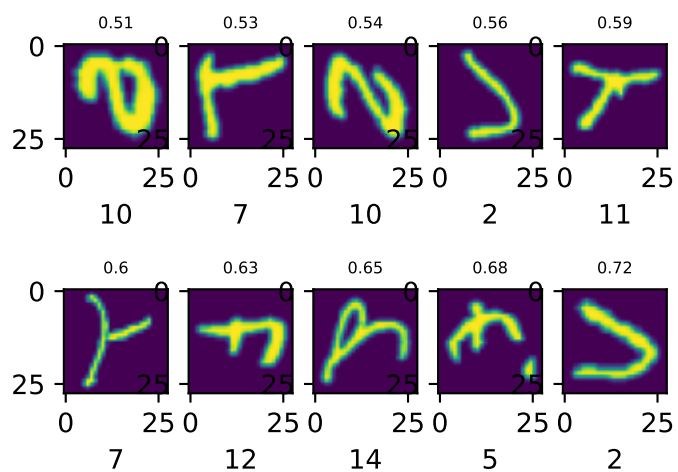
Incorrect, most unconfident images, Class 11



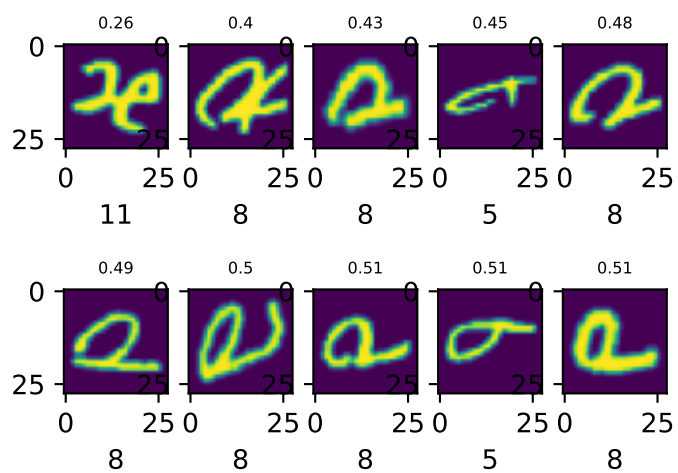
Incorrect, most unconfident images, Class 12



Incorrect, most unconfident images, Class 13

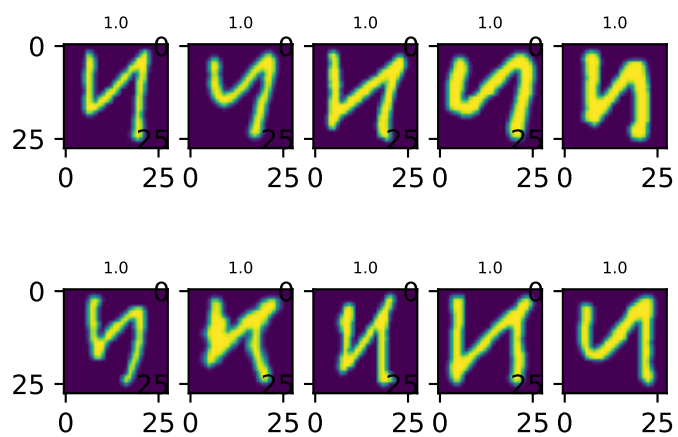


Incorrect, most unconfident images, Class 14

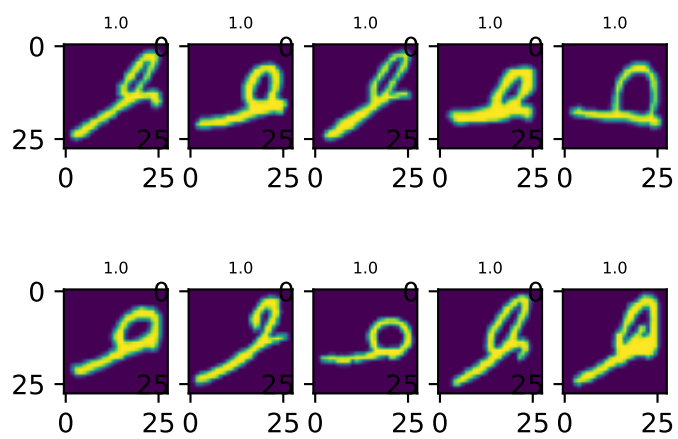


10 most confident each class:

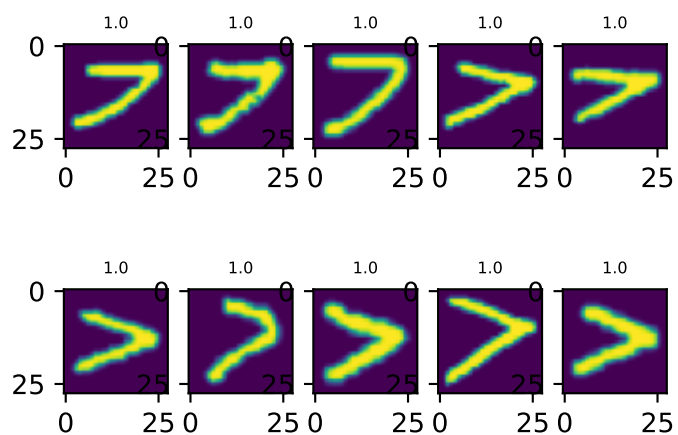
Correct, most confident images, Class 0



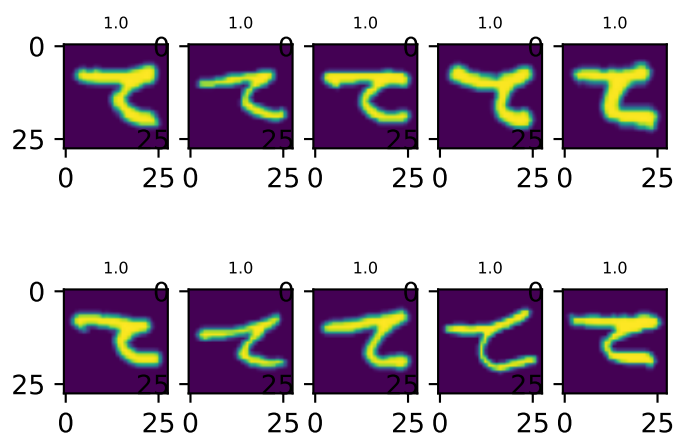
Correct, most confident images, Class 1



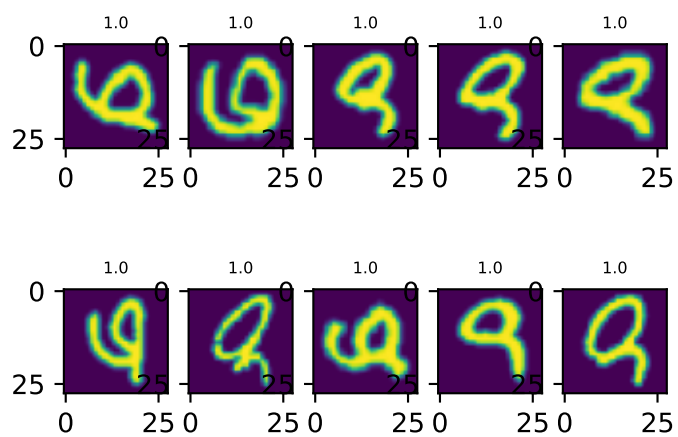
Correct, most confident images, Class 2



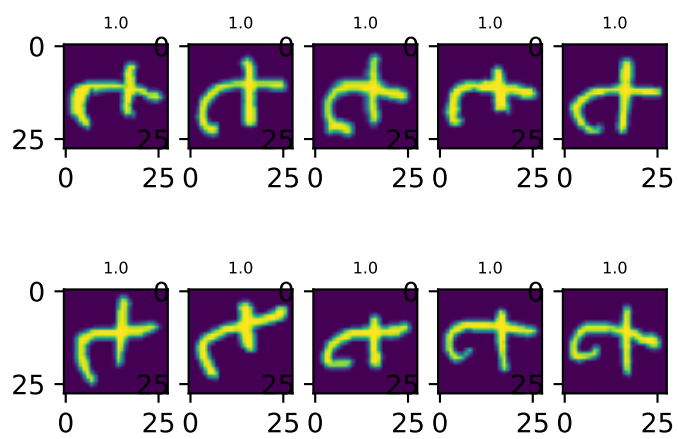
Correct, most confident images, Class 3



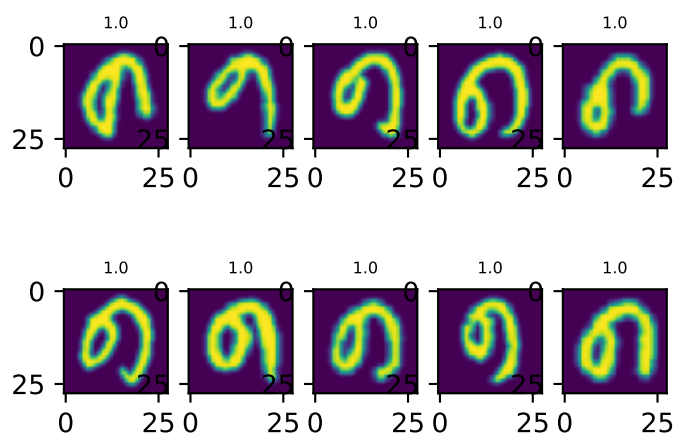
Correct, most confident images, Class 4



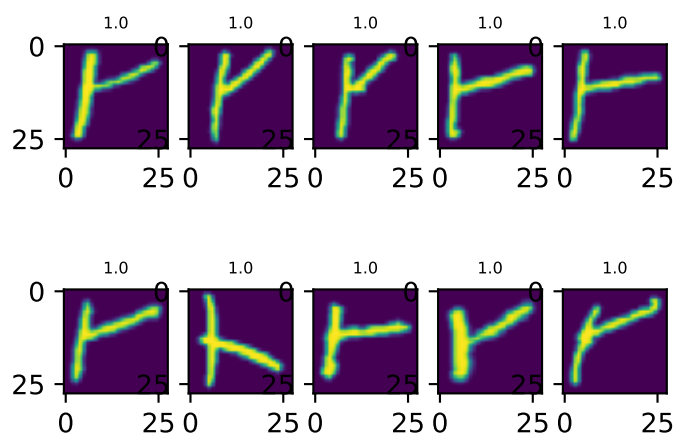
Correct, most confident images, Class 5



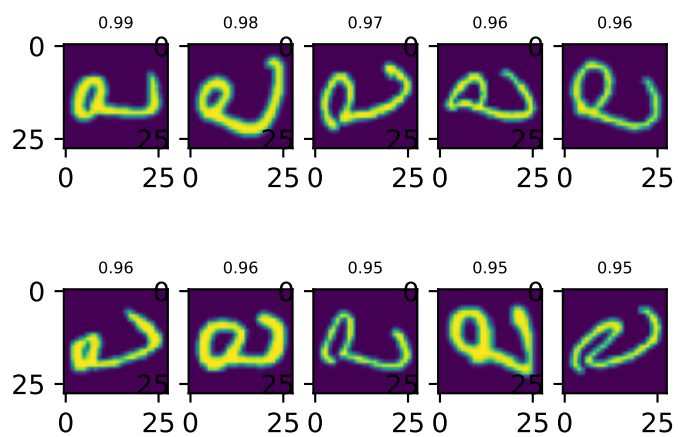
Correct, most confident images, Class 6



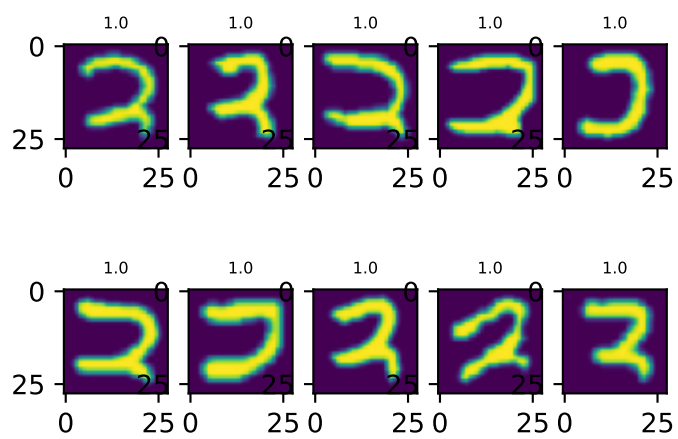
Correct, most confident images, Class 7



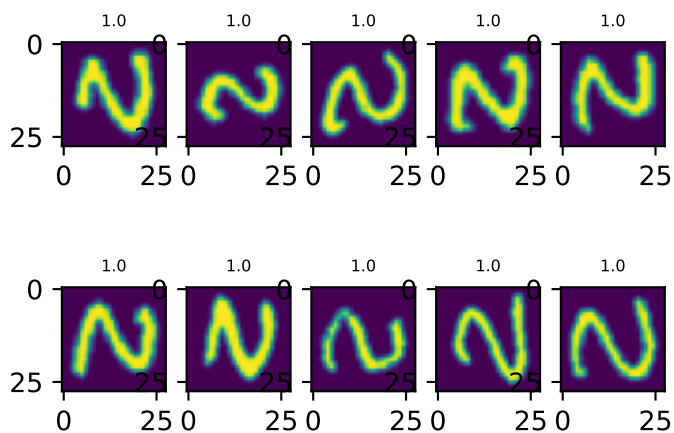
Correct, most confident images, Class 8



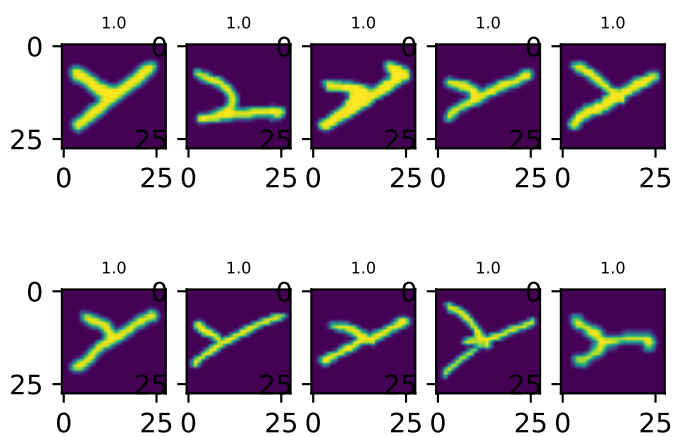
Correct, most confident images, Class 9



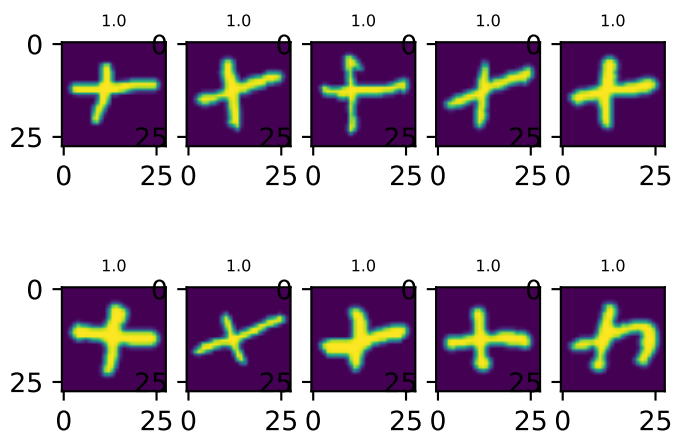
Correct, most confident images, Class 10



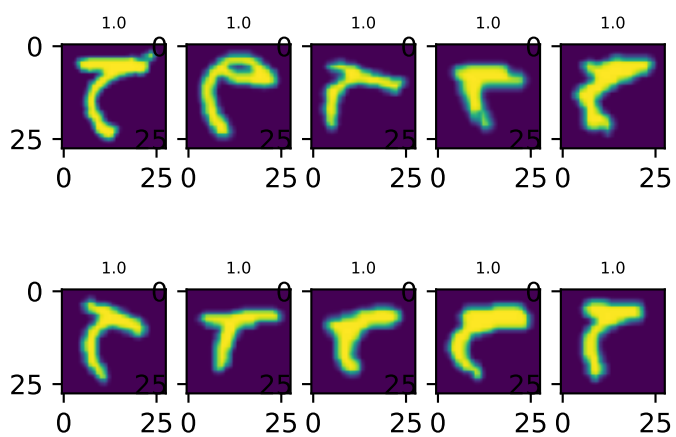
Correct, most confident images, Class 11



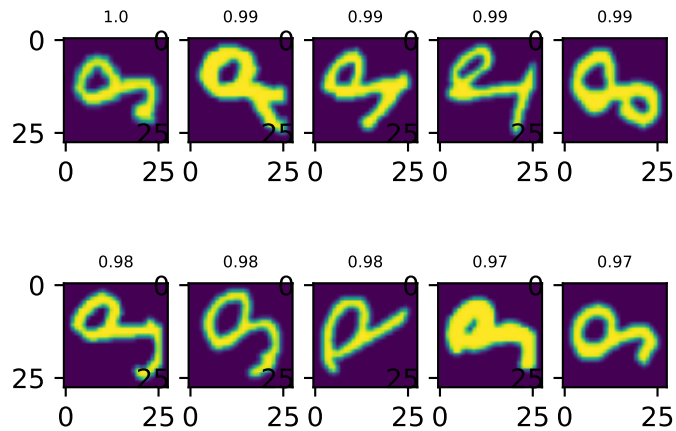
Correct, most confident images, Class 12



Correct, most confident images, Class 13



Correct, most confident images, Class 14



As expected, for unconfident images in each class, they does not look identical or share a common pattern with each other. The opposite applies for confident images.

2.5 _____

2.6 Batch Normalization

Batch Normalization is implemented as shown in task sheet.

