Machine Learning Project
Task 1 Report

Albert Garaev, Ksenia Novikova, Mukhammadsodik Khabibulloev

27.11.2021

1 K Nearest Neighbour Classification

1.1 Tasks

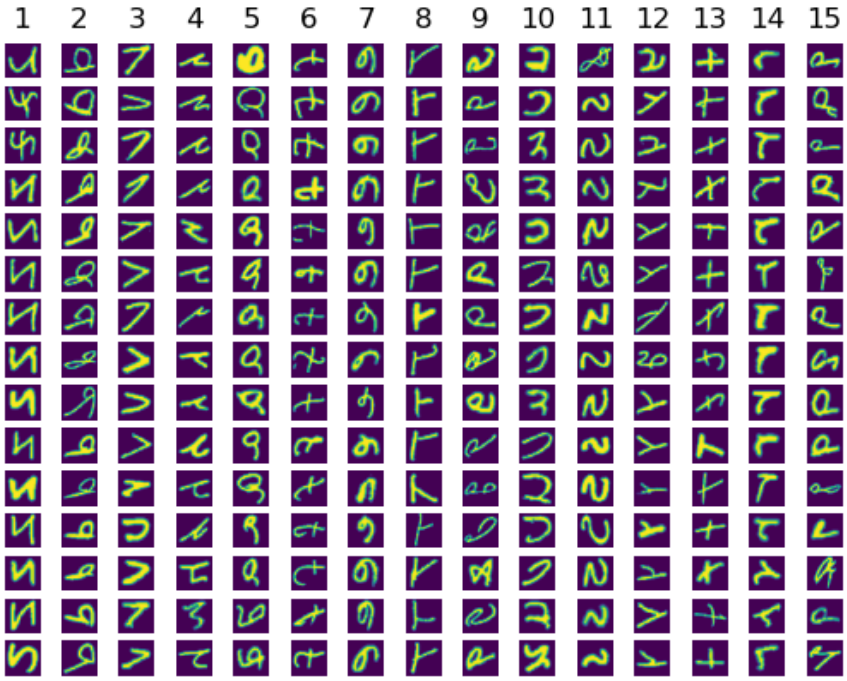


Figure 1: Examples of all classes

The Figure1 shows the plot of 15 random images from each of 15 classes. From plotted data, we can assume that the dataset contains 15 letters of the alphabet in their handwritten form.

For this dataset we implemented KNN algorithm with 5-fold cross validation and different distance measures. The source code of cross validation function is shown below.

```
def cross_validation(X, Y, Xtest, Ytest, m=5):
        trainX = X
        trainY = Y
        kn = KNN()
        neighb = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
        trainX_fold = np.array_split(trainX, m)
        trainY_fold = np.array_split(trainY, m)
        k_to_acc = {}
        for k in neighb:
                k_to_acc[k] = list()
        for k in neighb:
                print('k=', k)
        for i in range(m):
                X_val = trainX_fold[i]
                Y_val = trainY_fold[i]
                X_tr = np.vstack((trainX_fold[0:i] + trainX_fold[i + 1:]))
                Y_tr = np.vstack((trainY_fold[0:i] + trainY_fold[i + 1:])).ravel()
                knnf = KNN()
                knnf.fit(X_tr, Y_tr)
                pred = knnf.predict(X_val, k=k)
                acc = np.mean(pred == Y_val)
                k_to_acc[k].append(acc)
        scores = []
        best_k = 0
        best_acc = 0
        for k in neighb:
                acc = np.mean(k_to_acc[k])
                print(acc)
                scores.append(acc)
                if acc > best_acc:
                        best_acc = acc
                        best_k = k
        clf = KNN()
        clf.fit(trainX, trainY)
        y_test_pred = clf.predict(Xtest, k=best_k)
        acc = np.mean(y_test_pred == Ytest)
        print('Acc:{}, best_k:{}'.format(acc, best_k))
```

Except the Euclidean distance, we also used Manhattan distance metric to measure distance between vectors and to evaluate our algorithm. According to this Manhattan distance, the distance between two points is equal to the sum of the absolute values of the differences of their coordinates.

This code shows functions for distance metrics.

```
        def euclidean_distance(a, b):
        return np.sqrt(np.sum(np.square(a - b), axis=1))

        def manhattan_distance(a, b):
        return np.abs(np.array(a) - np.array(b)).sum(axis=1)

        def haming_distance(a, b):
        return sum(el1 != el2 for el1, el2 in zip(a, b))
```

Using euclidean distance, we got best accuracy with k=4. The plot of dependence of accuracy on k is shown in Figure 2.

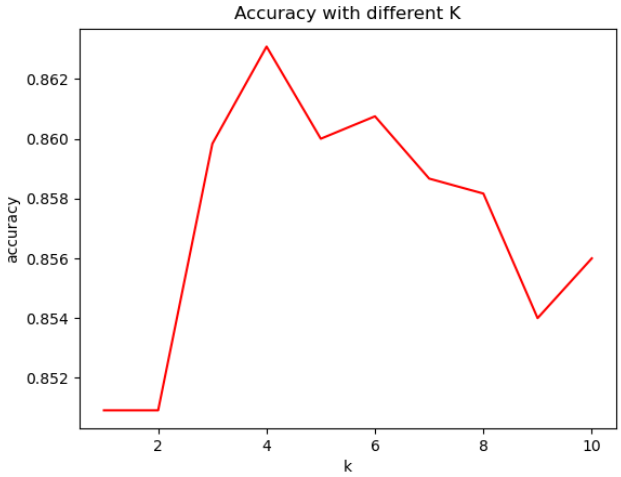


Figure 2: Accuracy with different k (Euclidean distance)

Using manhattan distance, we got best accuracy with k=6. The plot of dependence of accuracy on k is shown in Figure 3.

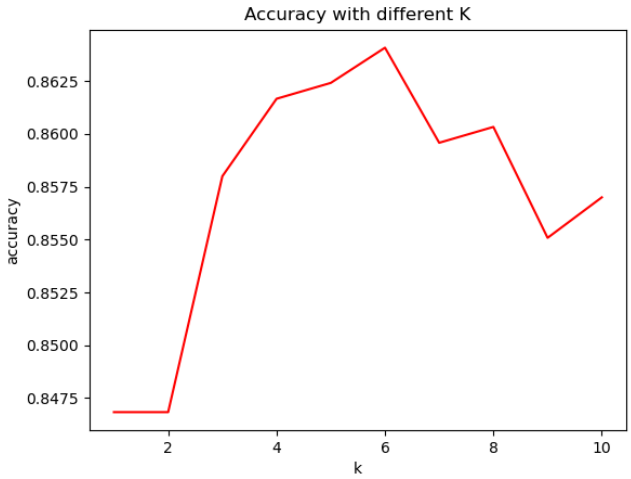


Figure 3: Accuracy with different k (Manhattan distance)

At the next step we implemented convolutional operation and applied filters to images. The first filter blures images and the second is edge detection filter. The code of convolutional function is shown bellow.

```
def convolve2D(image, kernel, padding=0, strides=1):
    kernel = np.flipud(np.fliplr(kernel))
    xKernShape = kernel.shape[0]
    yKernShape = kernel.shape[1]
    xImgShape = image.shape[0]
    yImgShape = image.shape[1]

    xOutput = int(((xImgShape - xKernShape + 2 * padding) / strides) - 1)
    yOutput = int(((yImgShape - yKernShape + 2 * padding) / strides) - 1)
    output = np.zeros((xOutput, yOutput))

    if padding != 0:
        imagePadded = np.zeros((image.shape[0] + padding*2, image.shape[1]
 + padding*2))
        imagePadded[int(padding):int(-1 * padding), int(padding):int(-1 * padding)] = image
        print(imagePadded)
    else:
        imagePadded = image

    for y in range(image.shape[1]):
        # Exit Convolution
        if y > image.shape[1] - yKernShape:
            break
        if y % strides == 0:
            for x in range(image.shape[0]):
                if x > image.shape[0] - xKernShape:
                    break
                try:
                    if x % strides == 0:
                        output[x, y] =
 (kernel * imagePadded[x: x + xKernShape, y: y + yKernShape]).sum()
                except:
                    break
    return output
```

Figure 4 shows images of dataset after bluring.

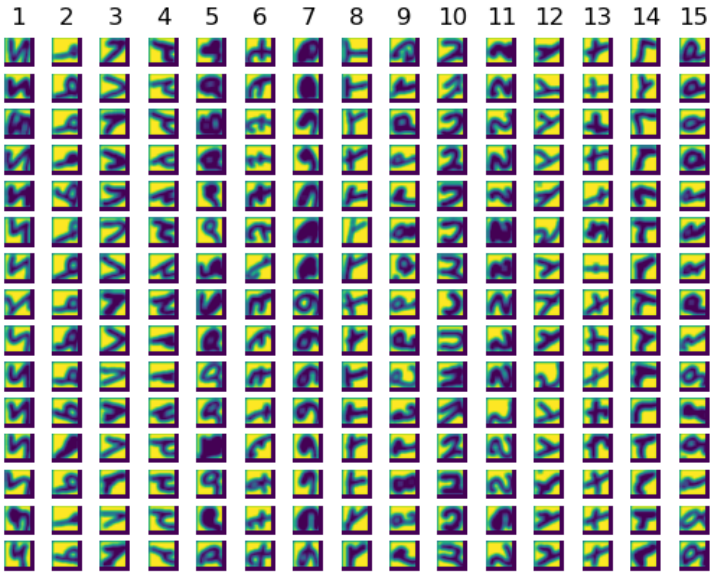


Figure 4: Images with blur filter

Figure 5 shows images after edge detection filter.

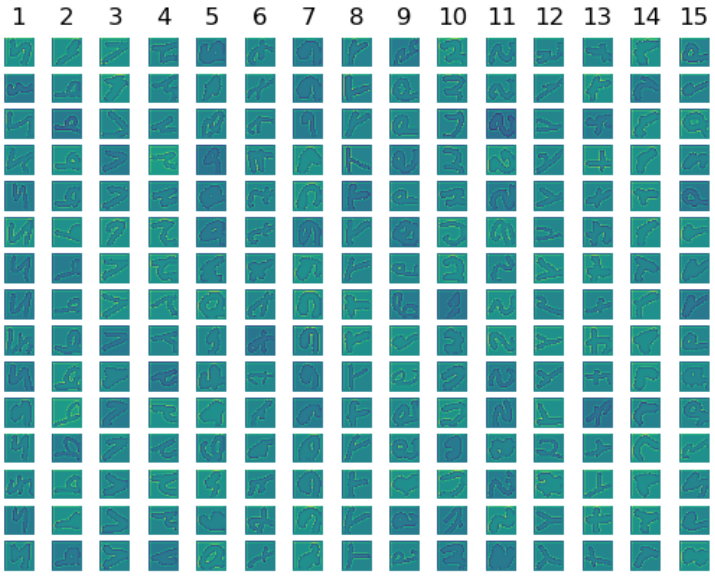


Figure 5: Images with edge detection filter

We applied knn algorithm with 5-fold cross validation on filtered images. Figures 6 and 7 shows results of accuracy measuring. On images with edge detection filter we got accuracy less than 50. The plots of dependence of accuracy on k are shown at Figure 6 and Figure 7.

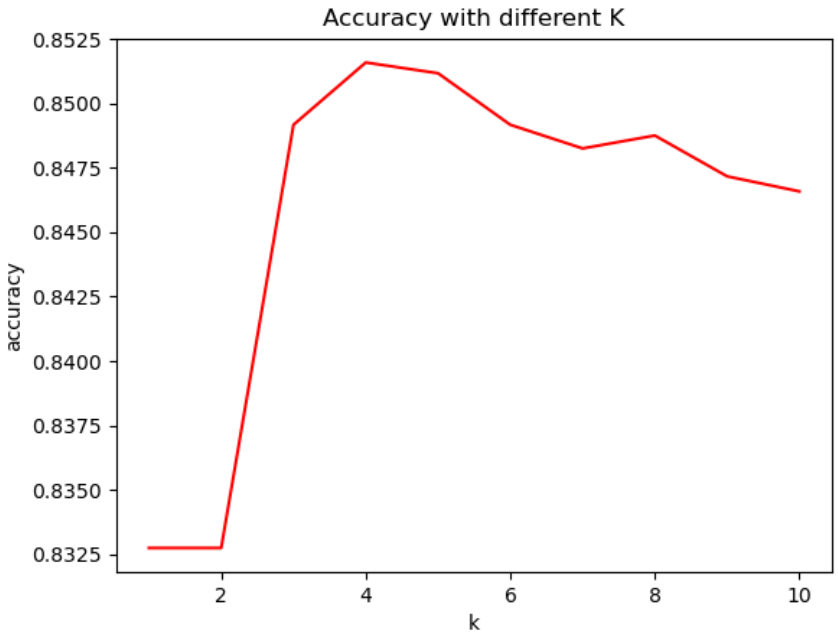


Figure 6: Accuracy with different k (Blured images)

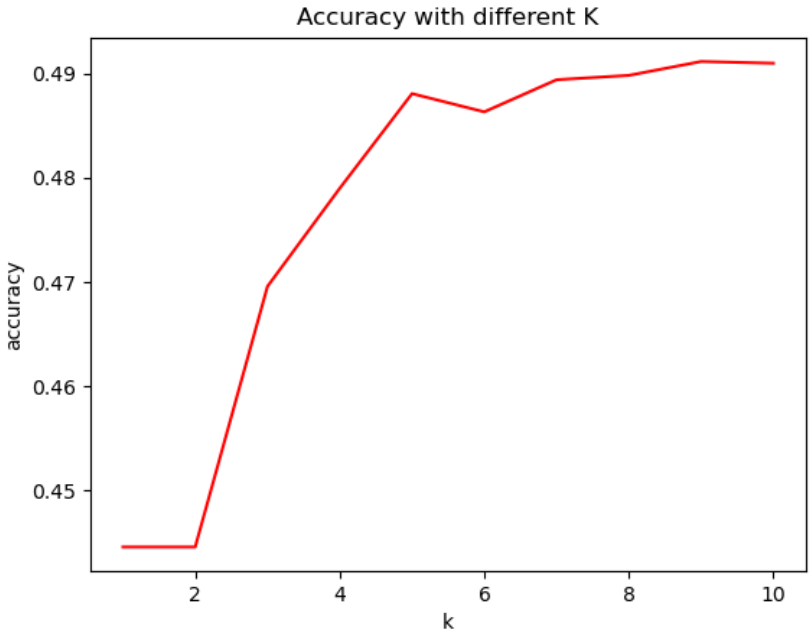


Figure 7: Accuracy with different k (Images with edge detection)

Next, we used extended version of kNN algorithm with weighted kNN. Using euclidean distance metrics with weighted kNN we got the best accuracy with k=6. The plot of dependence of accuracy on k is shown in Figure 8.

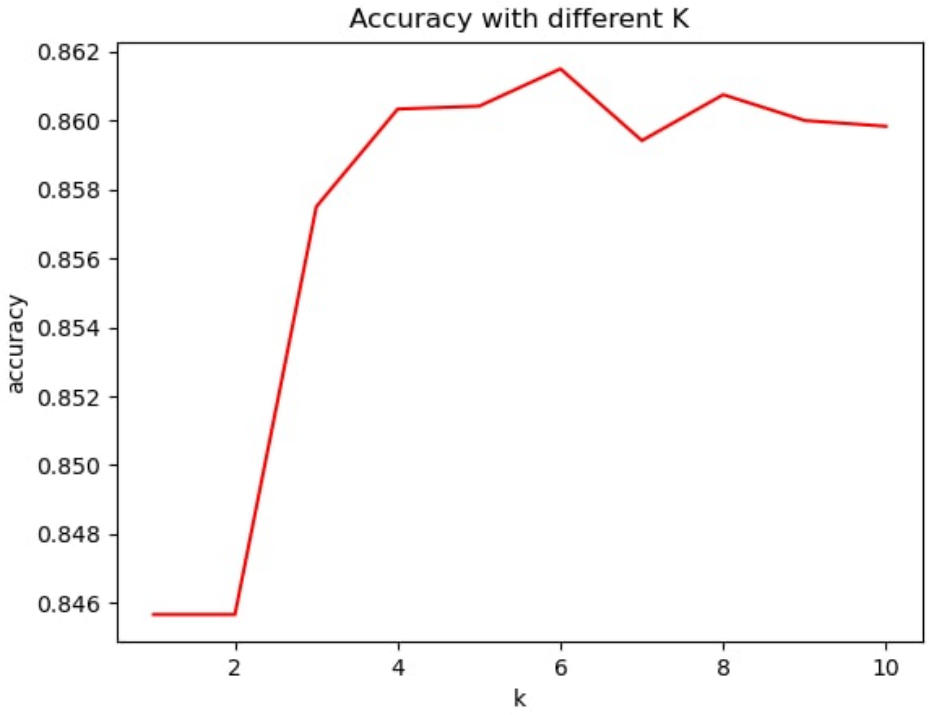


Figure 8: Accuracy on weighted kNN

2 Deep Neural Networks

2.1 Tasks

Activation function. In artificial neural networks, the neuron activation function determines the output signal, which is determined by the input signal or a set of input signals. For instance, using activation function Rectified linear unit (ReLU), if the input value is less than 0, then this function equates to zero, otherwise the value remains the same.

Loss function. The loss function is used to calculate the error between real and received responses. Our global goal is to minimize this error. Thus, the loss function brings neural network training closer to this goal. An example of a loss function is cross-entropy, which describes the distance between the actual outcome (probability) and the expected outcome (probability), that is, the smaller the cross-entropy value, the closer the two probability distributions.

Hyperparameters is a manually configurable parameter used to regulate network learning. For instance, a model hyperparameter is size of a neural network and its topology.

Optimizers determine the optimal set of model parameters, such as weight, so that the model produces the best results when solving a specific problem. Examples of optimizers: SGD, Adam, Adagrad, Adamax.

Epoch. One epoch has occurred, which means the entire dataset has passed through the neural network in the forward and backward directions only once. With an increase in the number of epochs, the weights of the neural network change more and more times. One epoch leads to underfitting, and an excess of epochs leads to overfitting.

Underfitting might occur when using insufficiently complex models. The training algorithm does not provide a sufficiently small value of the average error on the training sample.

Overfitting. The model explains well only examples from the training set, adapting to training examples, instead of learning to classify examples that were not involved in training (losing the ability to generalize).

Training set is a sequence of data that a neural network uses. It contains examples with true values like tags, classes, metrics. Unlabeled sets are also used to train neural networks.

Test set is used for estimation of the generalization error of the model which we choose. The test set should be used only at the end of the data analysis.

Validation set is used for estimation of prediction error, in this regard we can choose optimal model. The division into sets can be carried out approximately in the following proportions: training set 60%, test set 20%, validation set 20%. Sometimes there is no validation set, in this case we can divide data like training set 70%, test set 30%.

Training network on the Strange Symbols dataset.

Number of epochs and the training loss are shown in Figure 9
Number of epochs and the accuracy are shown in Figure 10

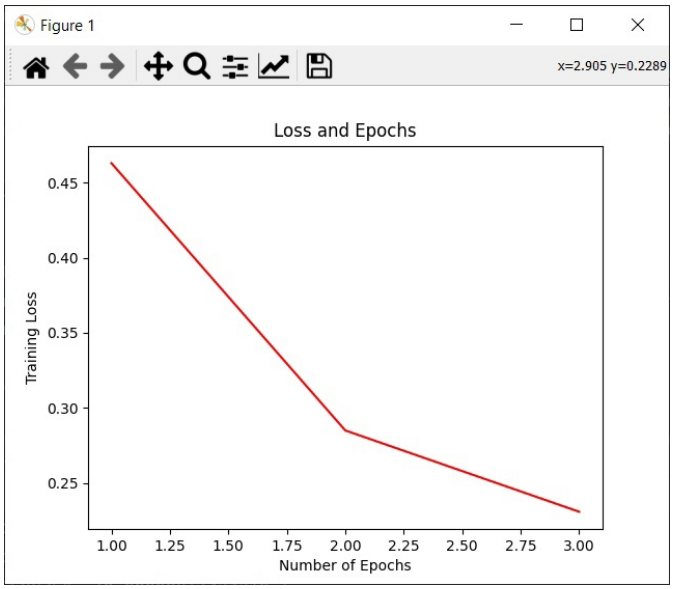


Figure 9: Number of epochs and the training loss

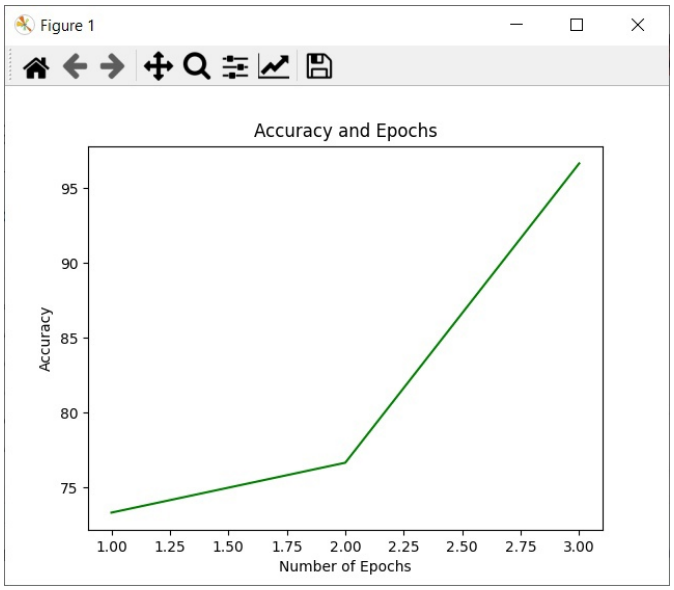


Figure 10: Number of epochs and the accuracy

Description of our best architecture of network.

```
class Net(nn.Module):
def __init__(self):
super(Net, self).__init__()

# Declaration of  the layers for feature extraction
self.features = nn.Sequential(nn.Conv2d(
in_channels=1, #Number of channels in the input image
out_channels=10, #Number of channels produced by the convolution
kernel_size=3, #Size of the convolving kernel
stride=1,
padding=1),
nn.ReLU(inplace=True),
nn.Conv2d(
in_channels=10,
out_channels=30,
kernel_size=3,
stride=1,
padding=1),
nn.MaxPool2d(2, 2),
nn.ReLU(inplace=True),
nn.BatchNorm2d(30),
nn.Conv2d(
in_channels=30,
out_channels=90,
kernel_size=3,
stride=1,
padding=1),
nn.ReLU(inplace=True),
nn.BatchNorm2d(90)) #90 number of channels from input

# Layers for classification
# In feature extraction layers, 1 max merge layer, which halves the
# height and width of the image, so we get a size of 14 x 14 (28/2)
# with the last output out_channels 90.
# We transfer them to a sequential layer.
# Here we used hidden layers of 512 and 256 neurons.
self.classifier = nn.Sequential(
nn.Linear(14 * 14 * 90, 512),
nn.ReLU(inplace=True),
nn.Dropout(0.5), # input randomly zeroes with probability p=0.5
nn.Linear(512, 256),
nn.ReLU(inplace=True),
nn.Linear(256, 15)) #Output layer 15 (number of classes)

```

The confusion matrix is shown in Figure 11. The output will be a series of 15 (number of classes) lists. The diagonal values from top to bottom from left to right is the number of correctly predicted values for each class.

```
Confusion matrix
[[ 96   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0  91   0   0   0   0   0   0   2   1   0   0   0   0   0]
 [  0   0  94   0   0   0   0   0   0   1   0   2   0   0   1]
 [  0   0   0  82   0   0   0   0   0   1   0   0   1   0   0]
 [  0   0   0   0  97   0   1   0   2   1   0   1   0   1   3]
 [  0   0   0   0   0  97   0   0   0   1   0   0   0   0   0]
 [  0   0   0   0   0   0  94   0   0   0   0   0   0   1   1]
 [  0   0   0   0   0   0   0  91   0   0   0   0   5   1   1]
 [  0   0   0   0   0   0   1   0  59   0   2   3   0   0  10]
 [  0   0   3   1   1   0   0   0   0  87   0   2   1   0   0]
 [  0   0   0   0   0   0   0   0   1   0  94   0   0   0   0]
 [  0   0   2   0   1   0   0   0   1   0   0 129   0   0   0]
 [  0   0   0   0   0   2   0   4   0   0   0   0  99   0   0]
 [  0   0   2   0   0   0   0   1   0   0   0   0   1  98   0]
 [  0   0   0   0   4   0   0   0  15   1   1   1   0   0 103]]
```

Figure 11: Confusion matrix

As we can see at the confusion matrix the worst result (59) of recognition show the class 8. We can see how images of this class looks like in Figure 12.

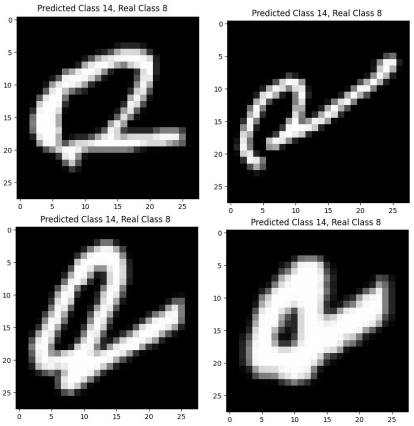


Figure 12: Predictions for class 8

We can see that model predict for class 8 that it is class 14. We can see how images of class 14 looks in Figure 13.

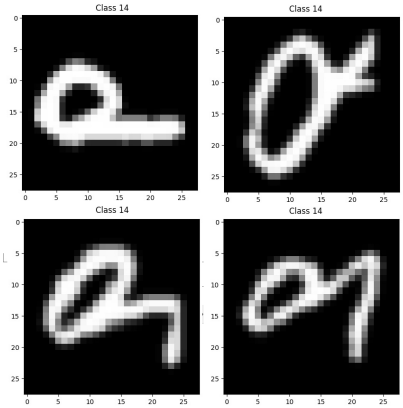
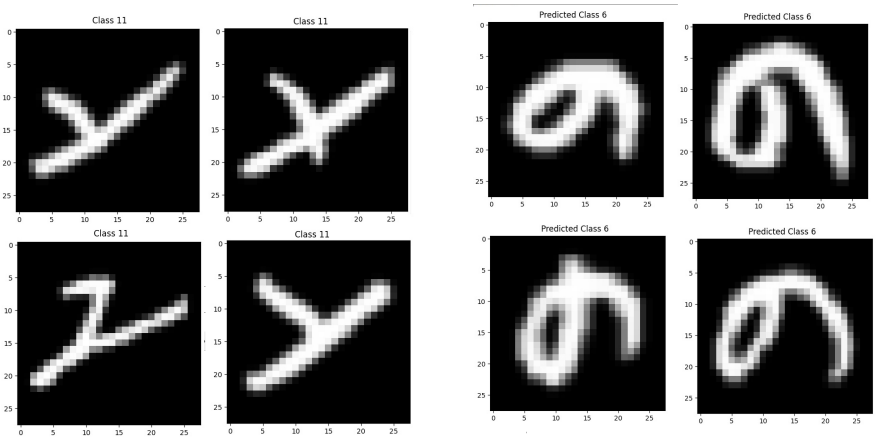


Figure 13: Images of class 14

We can see that the images of classes 8 and 14 looks almost the same, this fact affects the accuracy of predictions.
Now we see the well predicted classes, for instance, class 11 and class 6 in Figure 14.



(a) (b)

Figure 14: Images of class 11 and class 6

We can see that the images of this classes are different from each other and have unique features, this fact helps for better prediction.