Problem 4.1

A operation performed by a convolutional layer is technically a cross-correlation and not a convolution, as the name would suggest. For symmetric filters these two operations are equivalent. Nevertheless in machine learning it is often called convolution and therefore I will call it a convolution in the following sections.

Convolution Input Output Relation

First, a 1D convolutional layer with nonlinear activation $\sigma(\cdot)$ two filters $\mathbf{w}^{(0)}$ and $\mathbf{w}^{(1)}$ with one channel input $\mathbf{x}$ is considered. The two output channels are given by

$$z_k^{(0)} = \sigma\left(\sum_{m=0}^{M-1} w_m^{(0)} x_{k+m} + b^{(0)}\right) \tag{1}$$

$$z_k^{(1)} = \sigma\left(\sum_{m=0}^{M-1} w_m^{(1)} x_{k+m} + b^{(1)}\right). \tag{2}$$

The nonlinear function is applied to the convolution of the input with the corresponding filter and an added bias term. The input shape in TF is (b, n, c_i) where n is the sequence length, b the batch size and c_i the number of input channels. The weights have the TF shape $(M, c_\mathrm{i}, c_\mathrm{o})$ with the filter length M and the number of output channels c_o. The convolution can be thought of as sliding the filter along the sequence and summing up all the element wise multiplications. The strides argument controls by how many positions the filter is moved to the right at each step. The output sequence length gets smaller when strides is set to a higher value.

Secondly, a 1D convolution layer with nonlinear activation $\sigma(\cdot)$ one filter $\mathbf{w}$ and two channel input $\mathbf{x}^{(0)}$, $\mathbf{x}^{(1)}$ is considered. Now only one output channel is produced

$$z_k^{(0)} = \sigma\left(\sum_{j=0}^{c_\mathrm{i}-1}\sum_{m=0}^{M-1} w_m^{(j)} x_{k+m}^{(j)} + b\right). \tag{3}$$

The shapes mentioned above apply here too, $c_\mathrm{i} = 2$ and $c_\mathrm{o} = 1$ in this case.

Lastly, a 2D convolution layer with nonlinear activation $\sigma(\cdot)$ one filter $\mathbf{W}$ and one channel input $\mathbf{X}$ is considered.

$$Z_{k,j} = \sigma\left(\sum_{m_1=0}^{M_\mathrm{h}-1}\sum_{m_2=0}^{M_\mathrm{w}-1} W_{m_1,m_2} X_{k+m_1,j+m_2} + b\right) \tag{4}$$

The input as well as the filter are now two dimensional. The summation runs over height M_h and width M_w of the filter. The TF input shape is $(b, h_\mathrm{i}, w_\mathrm{i}, c_\mathrm{i})$ corresponding to batch size, height, width and number of channels on the input. The TF weight shape is $(M_\mathrm{h}, M_\mathrm{w}, c_\mathrm{i}, c_\mathrm{o})$. Strides can now be set independently for height and width, the interpretation is again the same but the filter is slid across both dimensions of the input.

Backpropagation

Considering the MSE loss function

$$J(\mathbf{y}, \mathbf{w}, \mathbf{x}) = \frac{1}{K} \sum_{k=1}^{K} (y_k - z_k)^2 \tag{5}$$

and a 1D convolution with linear activation, where the second sum was obtained by a change of summation variable

$$z_k = \sum_{m=0}^{M-1} w_m x_{k+m} + b = \sum_{l=k}^{k+M-1} x_l w_{l-k} + b \tag{6}$$

the partial derivates of the loss and the convolution output are given by

$$\frac{\partial J}{\partial z_k} = \frac{2}{K} (z_k - y_k) \tag{7}$$

$$\frac{\partial z_k}{\partial w_m} = x_{k+m} \tag{8}$$

$$\frac{\partial z_k}{\partial x_l} = w_{l-k} \tag{9}$$

The gradient of the loss is now affected by multiple elements and is therefore given by a summation

$$\frac{\partial J}{\partial w_m} = \sum_{k=1}^{K} \frac{\partial J}{\partial z_k} \frac{\partial z_k}{\partial w_m} = \frac{1}{K} \sum_{k=1}^{K} 2 (z_k - y_k) x_{k+m} \tag{10}$$

$$\frac{\partial J}{\partial x_l} = \sum_{k=1}^{K} \frac{\partial J}{\partial z_k} \frac{\partial z_k}{\partial x_l} = \frac{1}{K} \sum_{k=1}^{K} 2 (z_k - y_k) w_{l-k} \tag{11}$$

Mathematically the gradient in (10) is a cross-correlation, the gradient in (11) is now a true convolution.

Image Filtering

The two filters

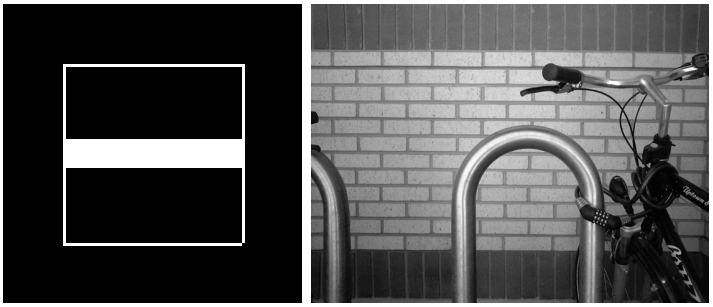
$$\mathbf{W}_{(0)} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad \mathbf{W}_{(1)} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{12}$$

were applied to the original images in Figure 1.

The outputs are shown in Figure 2. The first filter acts like a differentiation in horizontal direction, the second one like a differentation in vertical direction. For the edges image the first filter produced an output in response to the left and right edges, the second filter in response to the upper and lower edges and the white bar in the

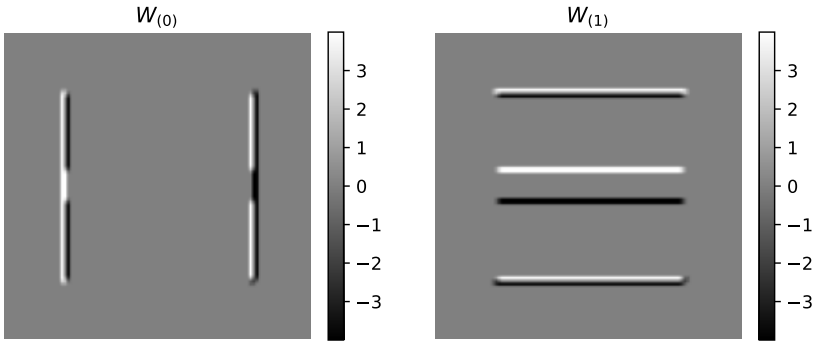
center. Looking at the filtered left edge, it can be seen that the output is a white line followed by a black line (going from left to right). This is due to the white line on the black background in the original image. When differentiated it produces a positive value followed by a negative value.

In the filtered bike images it can be seen that the first filter highlights the vertical edges of the tiles in the background. The second filter highlights the horizontal edges.

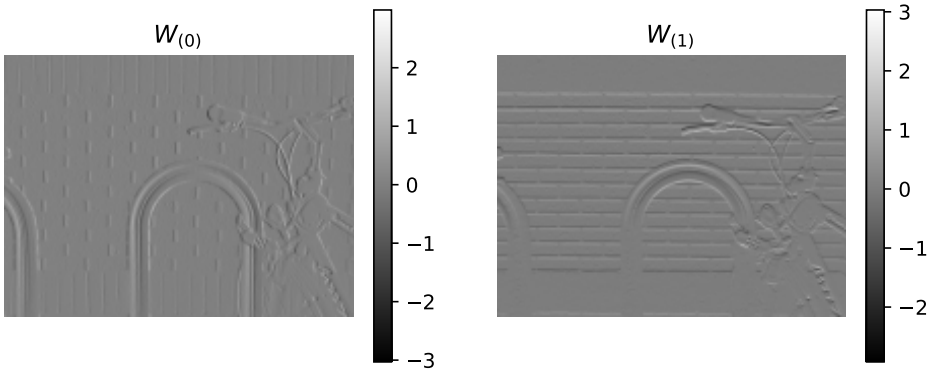


(a) Edges (b) Bike

Figure 1: Original images



(a) Edges



(b) Bike

Figure 2: Filtered images

Problem 4.2

Dataset

A random sample of all of the classes are shown in Figure 3 with the class label annotated on the top. For training the neural network the class labels where converted to a one hot encoding.

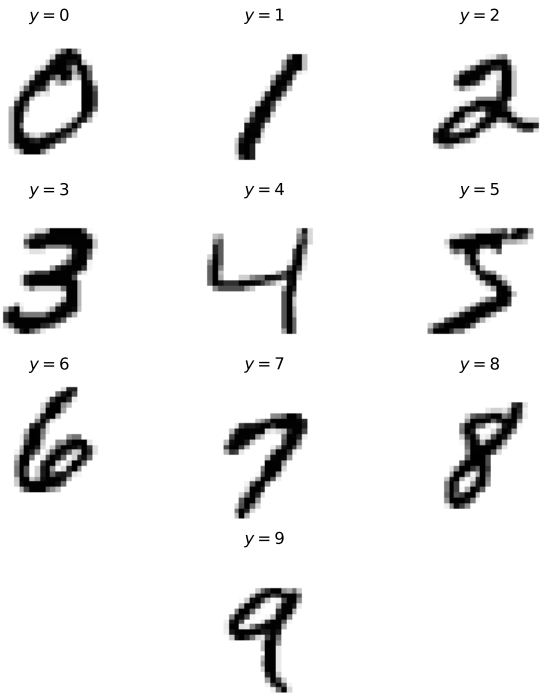


Figure 3: Samples from the 10 different classes

SVC

The SVC with RBF kernel trained on the whole training set achieved a final accuracy on test set of 97.92 %.

Neural network classifier

The neural network architecture is shown in Figure 4, the model summary is shown below.

For the Conv1 layer a kernel size of 4×4, 16 filters and a stride of $(2, 2)$ was used. For the Conv2 layer a kernel size of 3×3, 16 filters and a stride of $(2, 2)$ was used. For the Conv3 layer a kernel size of 3×3, 32 filters and a stride of $(1, 1)$ was used.

All layers, except the Output layer use a `ReLu` activation. The dropout layers use a drop rate of 0.5.

For the Output layer a linear activation was used. The predicted label is then decided by selecting the highest of the 10 output values. Another approach would be to use a softmax activation on the output. This has the advantage, that the outputs can then be interpreted as probabilities.

Using the same notation is in Problem 4.1 the number of parameters can be computed by accounting for the weights of one kernel $M_\mathrm{h}M_\mathrm{w}$ both for each input and each output channel $M_\mathrm{h}M_\mathrm{w}c_\mathrm{i}c_\mathrm{o}$ and the c_o bias terms. In total

$$M_\mathrm{h}M_\mathrm{w}c_\mathrm{i}c_\mathrm{o} + c_\mathrm{o}. \tag{13}$$

For example for Conv3 $M_\mathrm{h} = M_\mathrm{w} = 3$, $c_\mathrm{i} = 16$, $c_\mathrm{o} = 32$ adding up to $3\times3\times16\times32+32 = 4640$ parameters.

For the Dense1 layer it is $512 \times 256 + 512$ due to the 512 input and 256 output size.

For padding set to valid the output height is given by

$$h_\mathrm{o} = \mathrm{floor}\left(\frac{h_\mathrm{i} - M_\mathrm{h}}{\mathrm{stride}} + 1\right). \tag{14}$$

The stride is in the denominator as it sets the number of positions the filter is moved each step. The filter height is subtracted from the input height in the numerator because for valid padding no values are added to the input. As all my kernels have the same size in both directions and also the stride was set to the same values the width and height are the same at the output of each layer.

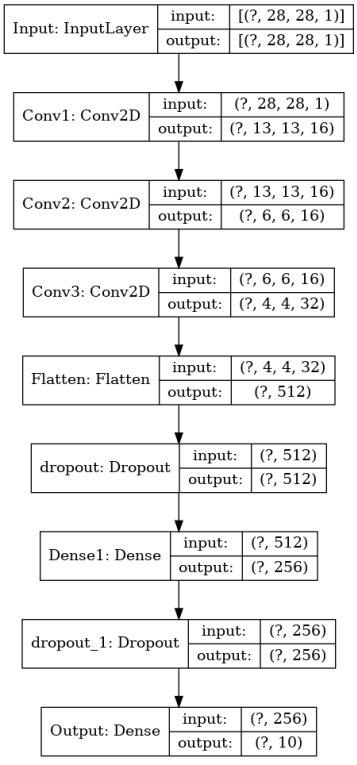


Figure 4: Neural network architecture

Layer (type)	Output Shape	Param #
Conv1 (Conv2D)	(None, 13, 13, 16)	272
Conv2 (Conv2D)	(None, 6, 6, 16)	2320
Conv3 (Conv2D)	(None, 4, 4, 32)	4640
Flatten (Flatten)	(None, 512)	0
dropout (Dropout)	(None, 512)	0
Dense1 (Dense)	(None, 256)	131328
dropout_1 (Dropout)	(None, 256)	0
Output (Dense)	(None, 10)	2570

Total params: 141,130
Trainable params: 141,130
Non-trainable params: 0

The batch size was set to 64, the number of epochs to 20 and the Adam optimizer has been used. `CategoricalCrossEntropy` has been used as a loss function. When the final layer is a linear layer the option `from_logits` has to be set to true. If a softmax layer were to be used this option has to be set to false.

The learning curve is shown in Figure 5. The performance is better on the test set because dropout is not active when evaluating the performance on the test set. The final accuracy on the test set is 99.18 %.

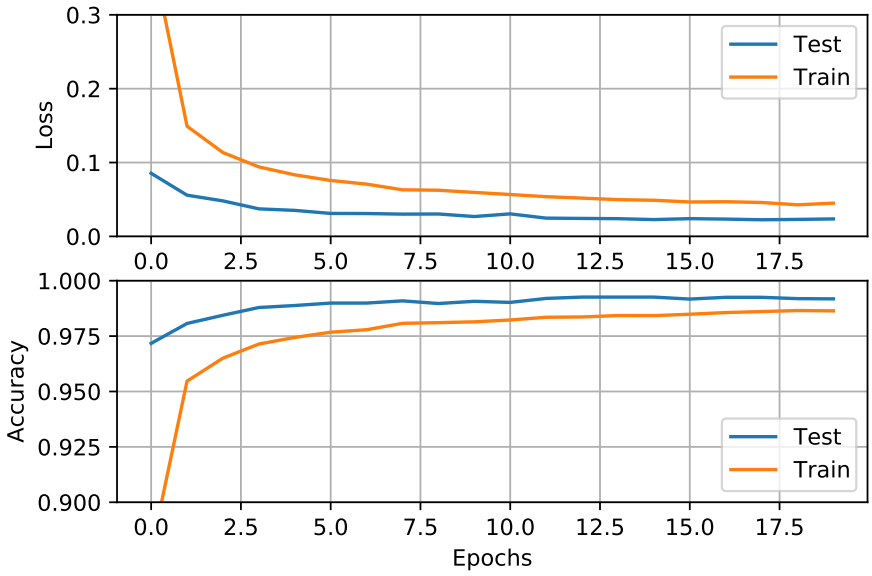


Figure 5: Learning curves

Some samples that have not been classified correctly are shown in Figure 6 with the predicted and true labels. The multiclass confusion matrix is shown in Table 1. For example 9 of the samples that are actually 5 get classified as a 3, this is the most common error.

Samples with label 5 are the ones that get misclassified most often with 13 out of 879 (1.48 %). On the samples with label 1 the highest accuracy is achieved with only 3 out of 1132 misclassified (0.27 %).

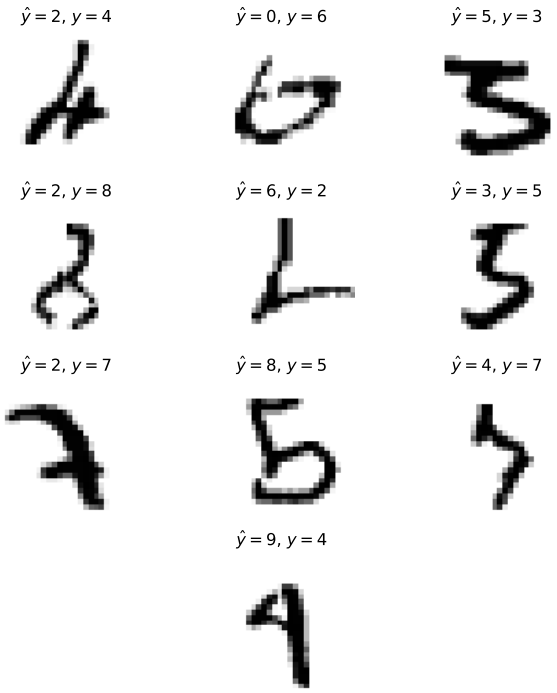


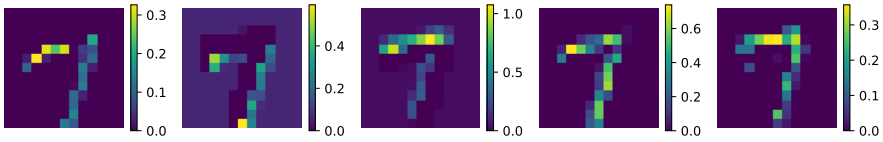
Figure 6: Incorrectly classified samples

y \ $\hat{y}$	0	1	2	3	4	5	6	7	8	9
0	977	0	1	0	0	0	1	1	0	0
1	0	1132	1	0	0	0	1	1	0	0
2	0	0	1027	0	1	0	3	1	0	0
3	0	0	1	1003	0	3	0	2	1	0
4	0	0	1	0	975	0	2	0	0	4
5	0	0	0	9	0	879	1	1	1	1
6	3	2	0	0	1	1	950	0	1	0
7	0	2	6	0	1	0	0	1018	0	1
8	3	1	3	1	0	0	0	1	962	3
9	0	2	0	2	5	0	0	5	0	995

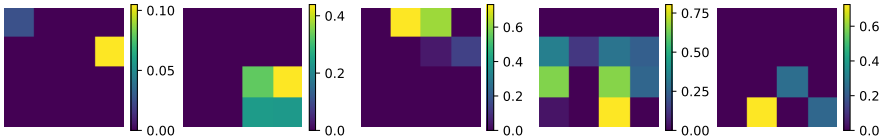
Table 1: Confusion Matrix

An input sample and the 5 filtered outputs after the first and last convolutional layer are shown in Figure 7. The first layer acts as a feature extractor as can be seen in the plots. For example the middle filter extracts the horizontal line of the input. After the third convolutional layer the size is only 4×4 and the original image is not recognizable anymore.

(a) Input



(b) Output after Conv1 Layer



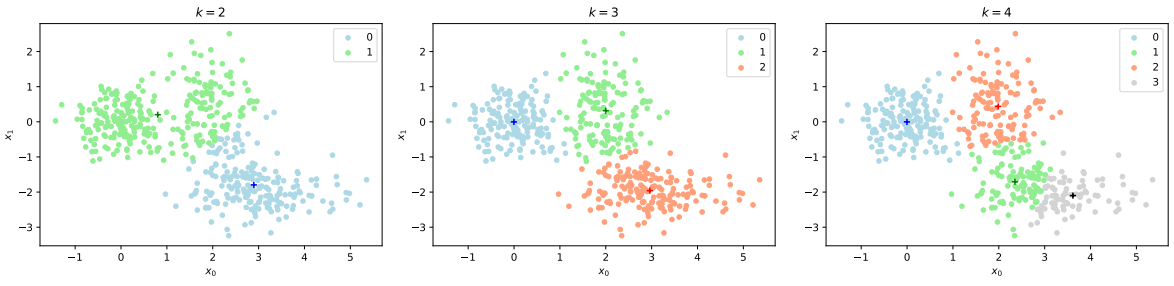
(c) Output after Conv3 Layer

Figure 7: Heatmap of the output after the first and last convolutional layer

Problem 4.3

kMeans

Clustering results for different values of k with the standard kMeans algorithm are shown in Figure 8. In Figure 9 a different result for $k = 4$ is shown, indicating that kMeans does not always converge to the same solution. The clustering result depends on the initial centroids. If the initialization is poor kMeans can get stuck in a local minimum leading to a different result.



(a) $k = 2$ (b) $k = 3$ (c) $k = 4$

Figure 8: Clustering results for different k

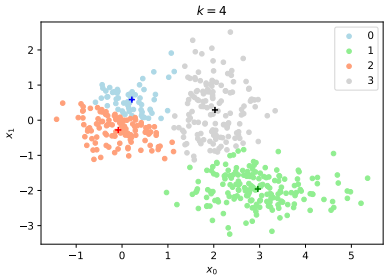


Figure 9: Different clustering results for $k = 4$

Samples from the region $x_0 > 4$ have been selected to obtain the visualization of the training process in Figure 10. This poor choice of initialization leads to a high number of iterations needed until the algorithm converges. The cluster centroids move little by little during iteration.

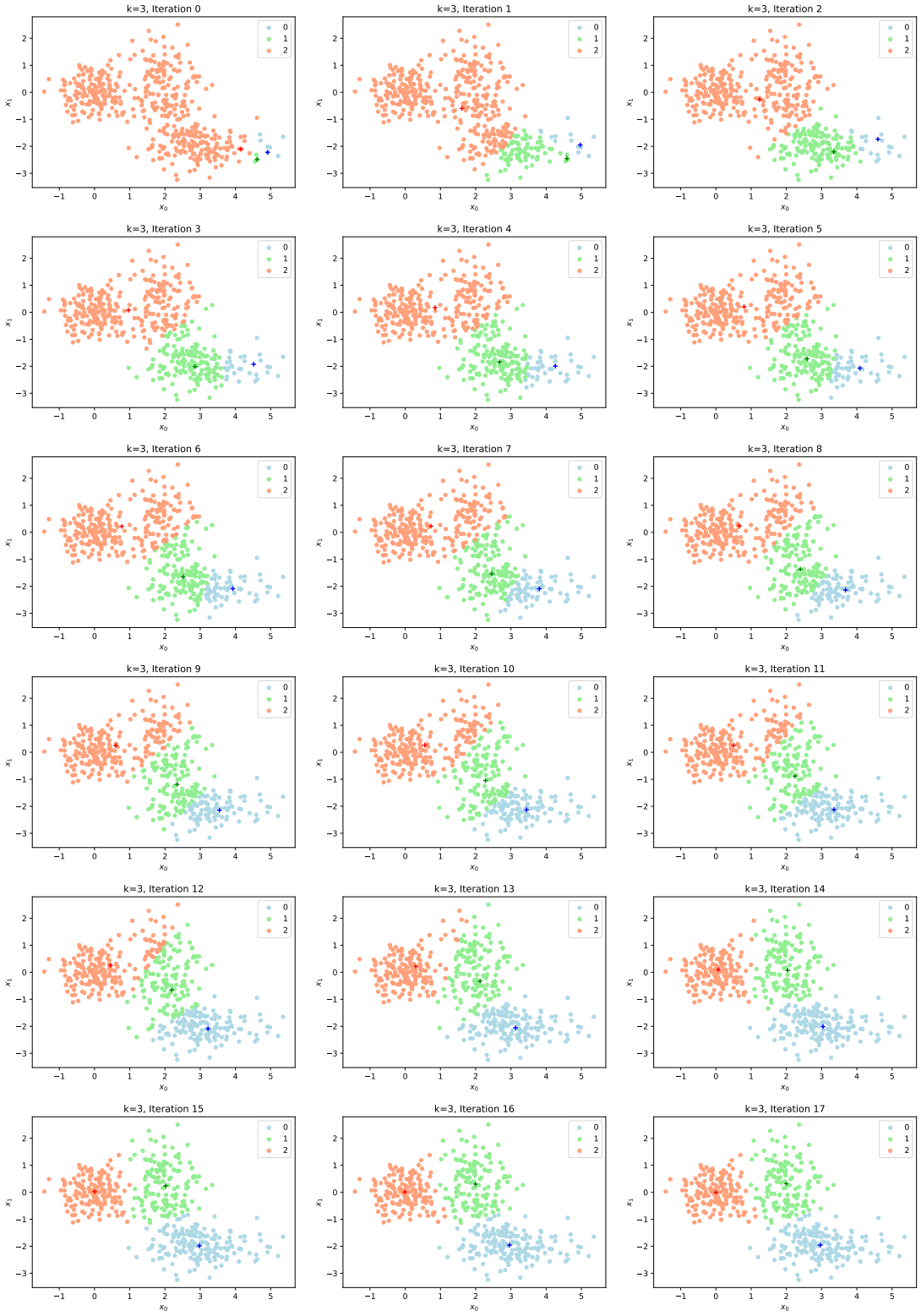


Figure 10: kMeans Iteration

kMeans++

To overcome the issues of the standard kMeans algorithm the kMeans++ algorithm has been implemented. It uses a different initialization scheme where samples close to an already selected centroid are less likely to be chosen. This should prevent such poor initializations as shown above.

The visualization is shown in Figure 11 it can be seen that the initial selection is already a good estimate. It takes just five iterations for the algorithm to converge in this case.

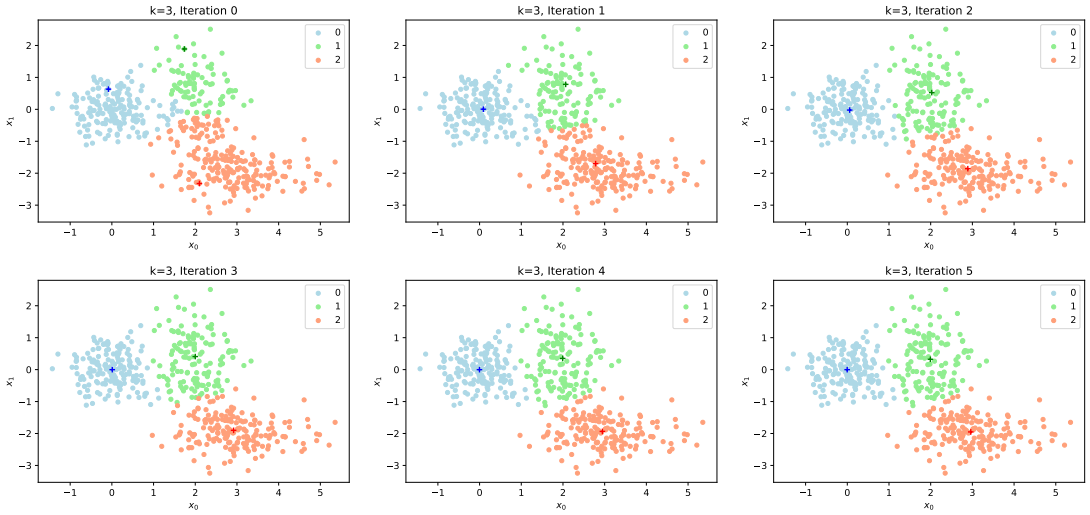


Figure 11: kMeans++ Iteration

MNIST Clustering

The kMeans++ method has been tested on the MNIST dataset. A histogram of the assigned cluster labels and the true labels is shown in Figure 12. The true labels are almost evenly distributed whereas with the clustering e.g. six is assigned less frequent.

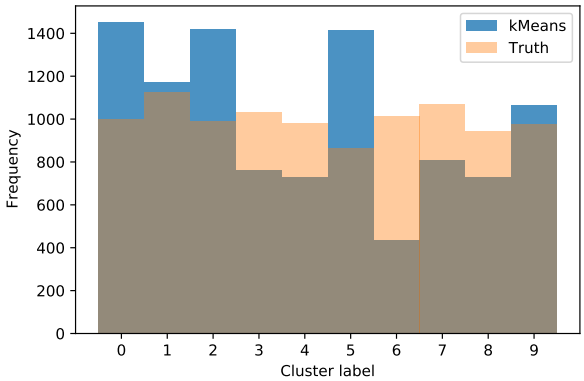


Figure 12: Histogram of predicted and true cluster labels in the dataset

The $k = 10$ centroids are shown in Figure 13 and the assigned labels $\hat{y}$ on top. The labels have been assigned by finding the most frequent true label in the corresponding cluster. It can be seen that some centroids have the same $\hat{y}$ e.g. 1 and 6 meaning that kMeans wasn't able to cluster the samples correctly.

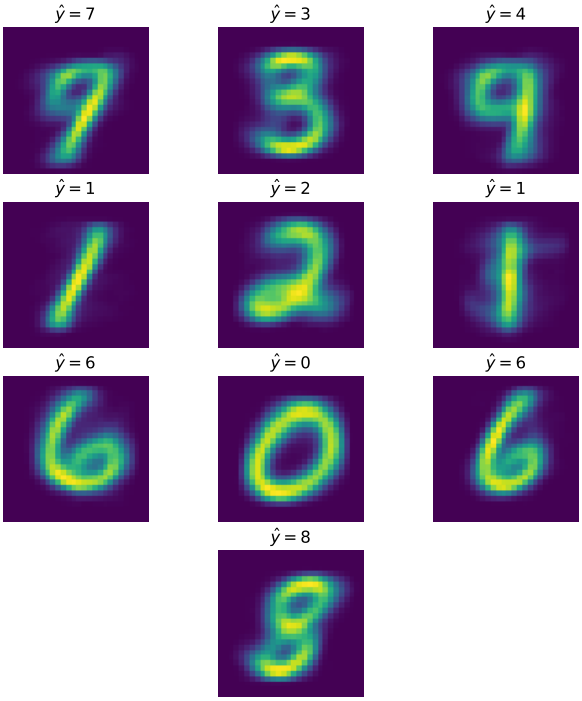


Figure 13: Centroids