DAT405 Assignment 7 – Group 111

Jihad Almahal - (5 hrs)
Oscar Karbin - (5 hrs)

March 20, 2023

1 Preprocessing. In the notebook, the data is downloaded from an external server and im- ported into the notebook environment using the mnist.load_data() function call.

1.1 Explain the data pre-processing highlighted in the notebook

The data pre-processing highlighted in the notebook is as follows:

The pixel values of the images are converted from integers to floating point numbers using the astype() method of NumPy arrays, which sets the data type of the array to the specified type (in this case, 'float32'). This is done for both the training and test sets.

The pixel values are normalized to the range [0, 1] by dividing each pixel value by the maximum value of a pixel (255). This is also done for both the training and test sets.

The labels for the training and test sets are converted from class vectors to binary class matrices using the to_categorical() function of the Keras utils module. This converts the integer class labels to binary vectors of length equal to the number of classes, where the value of the corresponding class index is set to 1 and all other values are set to 0.

These pre-processing steps are important to ensure that the input data is in a suitable format for training a neural network. Converting the pixel values to floating point numbers and normalizing them to the range [0, 1] helps to ensure that the input values are on a similar scale, which can improve training performance. Converting the class labels to binary class matrices is necessary for multi-class classification tasks like the MNIST dataset, where the goal is to predict one of several possible classes.

2 Network model, training, and changing hyper-parameters

2.1 How many layers does the network in the notebook have? How many neurons does each layer have? What activation functions and why are these appropriate for this ap- plication? What is the total number of parameters for the network? Why do the input and output layers have the dimensions they have?

The network in the notebook has four layers:

The first layer is a Flatten layer, which takes the input image and converts it from a 2D array of pixels to a 1D array. This layer has no neurons, but it reshapes the input data into a format that can be processed by the following layers.

The second and third layers are both Dense layers, which are fully connected layers of neurons. Both layers have 64 neurons and use the ReLU (rectified linear unit) activation function. ReLU is a commonly used activation function in neural networks, as it is computationally efficient and has been shown to be effective for many types of data. It returns 0 for negative inputs and the input value for positive inputs. This type of non-linearity can help the network learn more complex and nonlinear relationships between the input and output.

The final layer is also a Dense layer with 10 neurons, which is equal to the number of classes in the dataset. It uses the softmax activation function, which is appropriate for multi-class classification tasks like this one. Softmax outputs a probability distribution over the classes, with each value representing the probability that the input belongs to that class.

The total number of parameters in the network can be calculated as follows: We start by considering the number of connections between the input layer and the first hidden layer, which is 784 multiplied by 64. Next, we add the 64 bias terms in the first hidden layer. We repeat this process for the connections between the first hidden layer and the second hidden layer, which is 64 multiplied by 64. Finally, we consider the connections between the second hidden layer and the output layer, which is 64 multiplied by 10, plus 10 bias terms. Therefore, the total number of parameters in the network is 55,050(784*64 +64 + 64*64 +64 +10*64 + 10 = 55050).

The input layer has the dimensions it has because the input images are 28x28 pixel grayscale images. The output layer has 10 neurons because there are 10 possible classes in the MNIST dataset.

2.2 What loss function is used to train the network? What is the functional form (a math- ematical expression) of the loss function? and how should we interpret it? Why is it appropriate for the problem at hand?

The categorical cross-entropy loss function is used to train the network. The mathematical expression for this loss function is:

$$Loss = -\sum_{i=1}^{C} y_i \log(\hat{y}_i)$$

where y_i is the target from the dataset, $\hat{y}$ is the predicted output of the model, and C is the number of classes.

This loss function is appropriate for the problem at hand because it is commonly used for multi-class classification problems. It penalizes the model when the predicted probability distribution of the classes is different from the true distribution. In other words, it encourages the model to predict the correct class with high confidence, and penalizes the model when it predicts the wrong class or is uncertain about its prediction.

2.3 Train the network for 10 epochs and plot the training and validation accuracy for each epoch.

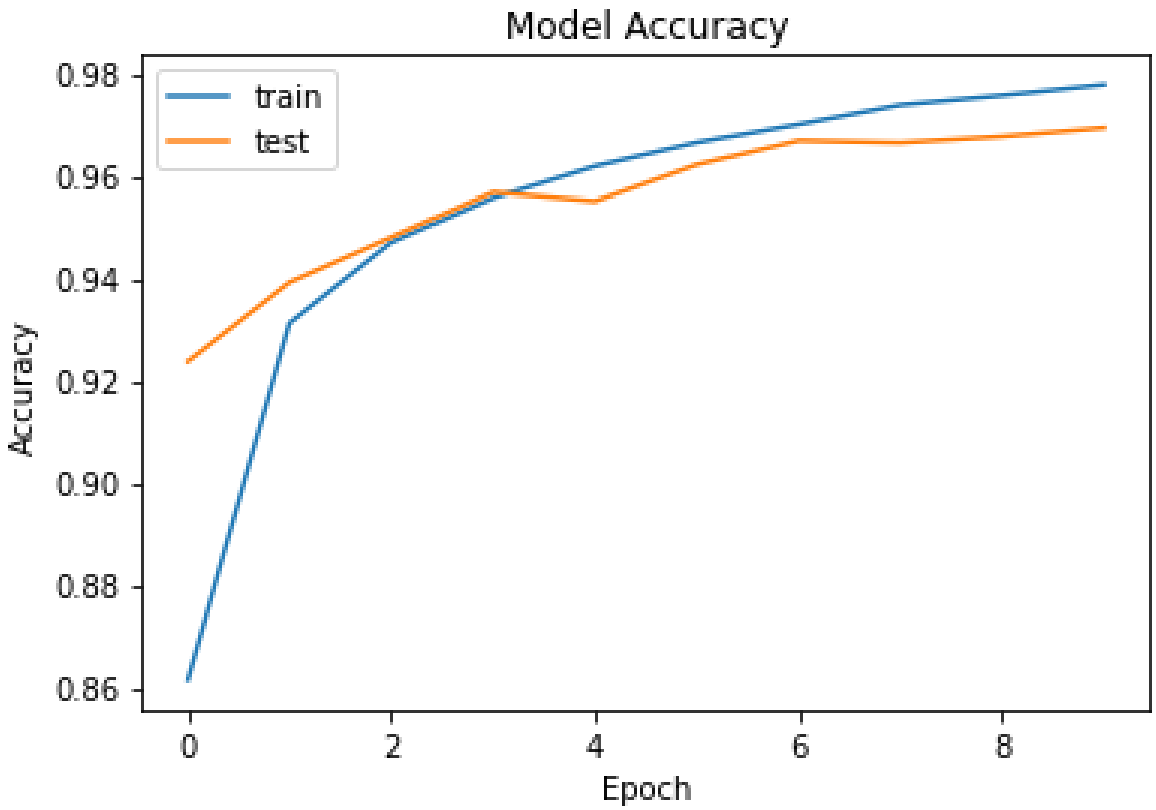


Figure 1: training and validation accuracy for each epoch.

We can see that the training accuracy increases with each epoch while the validation accuracy initially increases but starts to plateau after the fourth epoch. This indicates that the model may be overfitting to the training data after a certain number of epochs.

The final test accuracy of the model is 0.9694 and the test loss is 0.0985.

2.4 Update Model to implement a new three-layer model

With the new updated model with 40 epochs and layers that have 500 and 300 units we get a test loss: 0.0660 and a test accuracy: 0.9817.

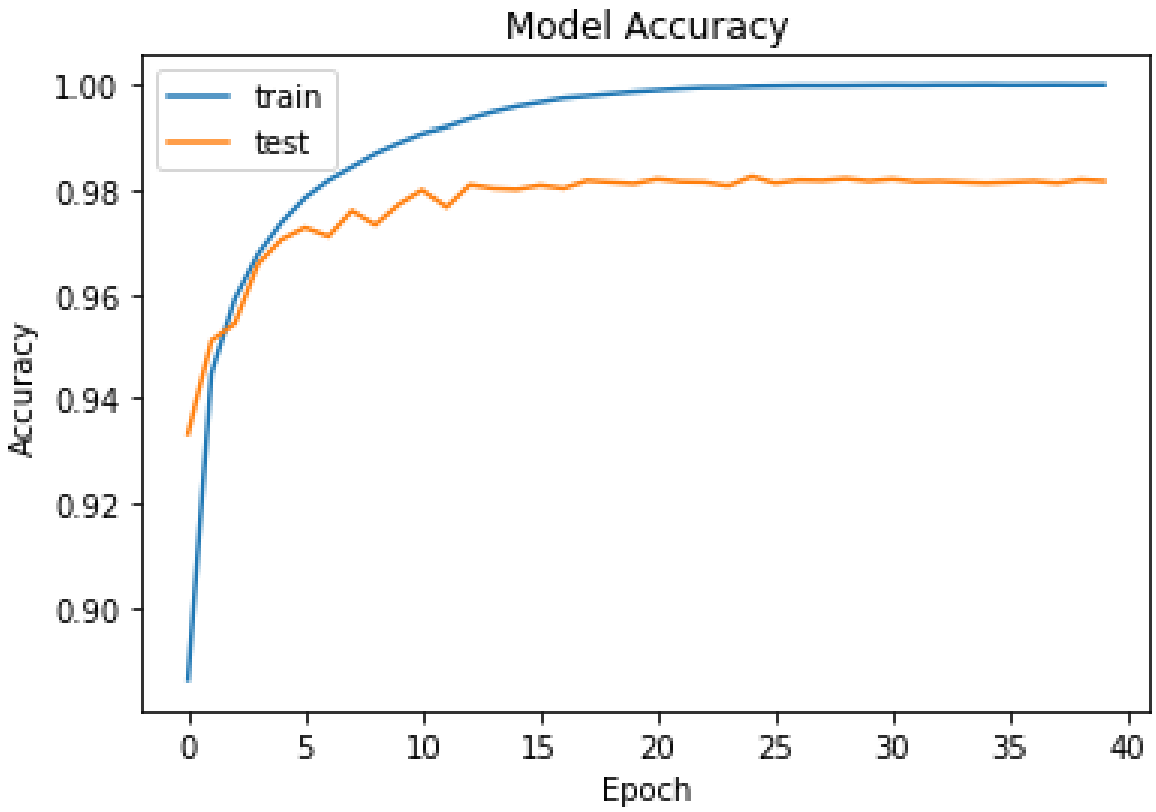
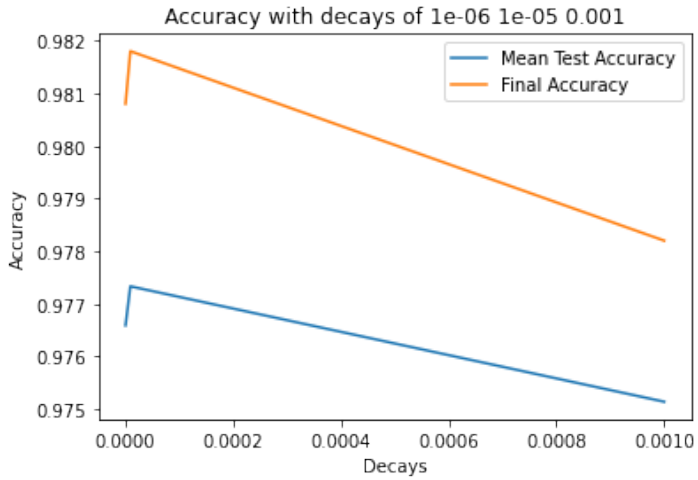
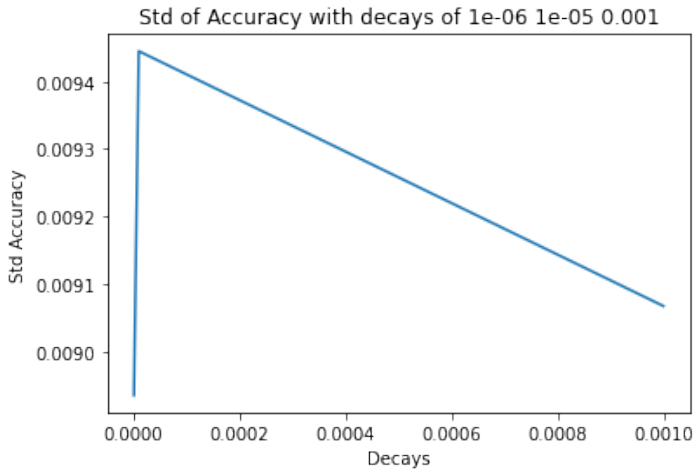


Figure 2: The new model with 40 epochs.

We trained the new model with 3 different decays: 0.000001, 0.00001, 0.001 and calculated the mean test accuracy and the standard deviation test accuracy.

Decays	**Test Accuracy**	**Mean Test Accuracy**	**STD Test Accuracy**
0.000001	0.9807	0.9765	0.00893
0.00001	0.9818	0.9773	0.00944
0.001	0.9782	0.9751	0.00906





Your results we obtained are very close to Hinton's reported results. However, it's important to note that there are many factors that can influence the results of a machine learning model, including the specific implementation details and hyperparameters used.

Some factors that may influence the results include the specific versions of the libraries and frameworks used, the hardware and software environment in which the model was trained, the specific preprocessing steps used for the data, the specific regularization techniques and hyperparameters used, and the specific architecture and hyperparameters of the neural network model.

Additionally, it's worth noting that Hinton may have had access to additional information about the MNIST database that is not publicly available, which could have influenced his results. However, without more information about Hinton's specific implementation and training process, it's impossible to say for certain what factors may have contributed to any differences in results.

3 Convolutional layers

3.1 Design a model that makes use of at least one convolutional layer

The model we used has two convolutional layers, each followed by a max pooling layer to downsample the output. It also has a dropout layer to prevent overfitting and a dense layer with 128 neurons. The output layer has 10 neurons for the 10 possible classes. With this model we were able to reach over 99% accuracy

```
Epoch 1/10
469/469 [==============================] - 53s 111ms/step - loss: 0.3011 - accuracy: 0.9072 - val_loss: 0.0616 - val_accuracy: 0.9795
Epoch 2/10
469/469 [==============================] - 51s 109ms/step - loss: 0.0953 - accuracy: 0.9717 - val_loss: 0.0428 - val_accuracy: 0.9854
Epoch 3/10
469/469 [==============================] - 51s 110ms/step - loss: 0.0696 - accuracy: 0.9793 - val_loss: 0.0371 - val_accuracy: 0.9874
Epoch 4/10
469/469 [==============================] - 52s 112ms/step - loss: 0.0548 - accuracy: 0.9837 - val_loss: 0.0298 - val_accuracy: 0.9894
Epoch 5/10
469/469 [==============================] - 51s 110ms/step - loss: 0.0472 - accuracy: 0.9865 - val_loss: 0.0256 - val_accuracy: 0.9915
Epoch 6/10
469/469 [==============================] - 53s 113ms/step - loss: 0.0387 - accuracy: 0.9885 - val_loss: 0.0280 - val_accuracy: 0.9907
Epoch 7/10
469/469 [==============================] - 51s 108ms/step - loss: 0.0362 - accuracy: 0.9892 - val_loss: 0.0249 - val_accuracy: 0.9914
Epoch 8/10
469/469 [==============================] - 51s 108ms/step - loss: 0.0306 - accuracy: 0.9900 - val_loss: 0.0214 - val_accuracy: 0.9931
Epoch 9/10
469/469 [==============================] - 51s 110ms/step - loss: 0.0276 - accuracy: 0.9912 - val_loss: 0.0220 - val_accuracy: 0.9927
Epoch 10/10
469/469 [==============================] - 51s 109ms/step - loss: 0.0251 - accuracy: 0.9919 - val_loss: 0.0245 - val_accuracy: 0.9923
Test loss: 0.024467185139656067, Test accuracy 0.9922999739646912
```

Figure 3: Result from running the model.

3.2 Discuss the differences and potential benefits of using convolutional layers over fully connected ones for the application?

Convolutional layers are specifically designed for image processing tasks such as object detection, recognition, and segmentation. They take advantage of the spatial relationships between pixels in an image by applying filters or kernels that scan the image and extract features. While fully connected layers connect every neuron in one layer to every neuron in the next layer, without considering the structure of the input.

Convolutional layers allow the model to learn local features and patterns in the image, and are better at handling data with high spatial correlation. This means that they can detect patterns regardless of where they appear in the image, and can handle variations in the position, size, and orientation of the object being classified. Additionally convolutional layers typically have far fewer parameters than fully connected layers, which makes them less prone to overfitting and allows the model to generalize better to new data. Overall, the use of convolutional layers has revolutionized image processing tasks and has led to significant improvements in accuracy and performance for a variety of computer vision applications.