University of Bath

Department of Computer Science

MSc – Artificial Intelligence

Foundations and Frontiers of Machine Learning

Graded Assignment 2 – Deep Learning

Participating Students:

Name	Student No	Weight
Anas Rezk	219421780	50%
Mohammed Bin Ali Alhaj	219543484	50%

[Google Drive Link]

Task-1 Data Visualization

MNIST dataset contains 60,000 training images and 10,000 testing images of handwritten digits, see figure-1.1, 10 balanced classes have around 6,000 training samples each and another 1000 testing samples for all classes.

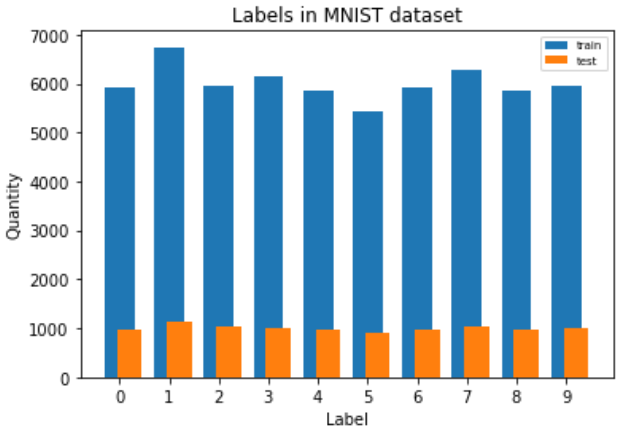


Figure 1.1

We resize the 28x28 sample images to a flat 784-ffeatures array. However, as visualization beyond 3D is not feasible, PCA offers a solution to transform a set of correlated features in high dimensional space (i.e., 784) into a series of uncorrelated features in low dimensional space. The 2-PCA components explain 0.1/ 0.071 of original train dataset variations and 0.1/0.075 on test dataset. We demonstrate that 154 components preserve 95% of data variance, see figure-1.2

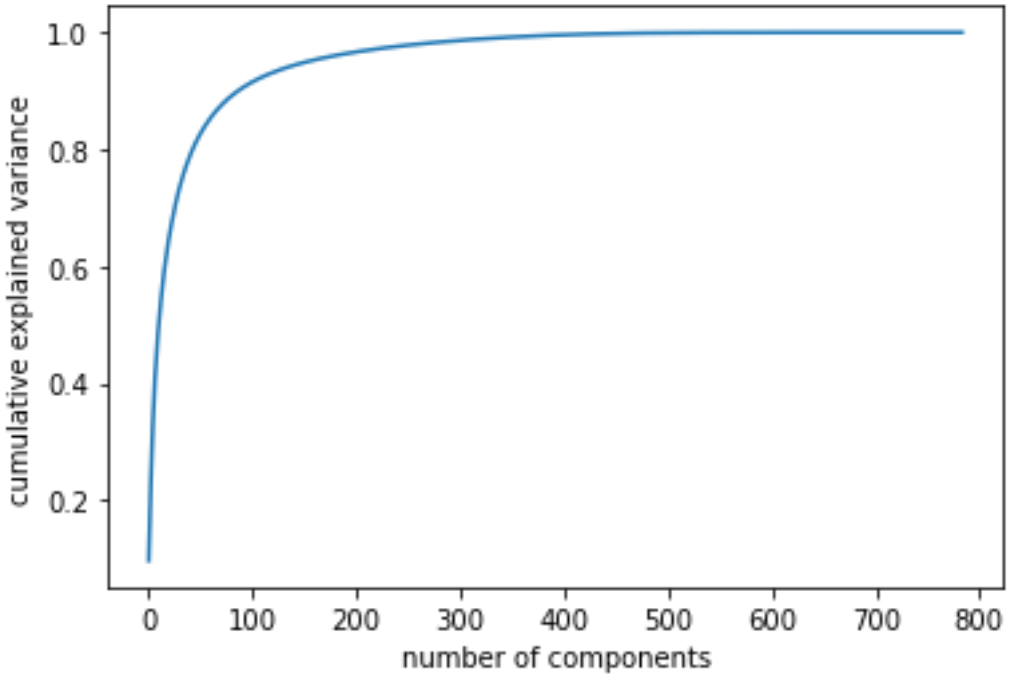


Figure 1.2

Figure-1.3 shows the spread of 60,000 digits in 2D space. Scatter points overlap around the graph center but less jumbled in border areas. We can recognize that (0,1), (0,7), and (1,6) pairs are linearly separable. These observations are then re-confirmed in figure-1.6 that plots the first 400 samples of test dataset (less-dense space). Figure-1.7 suggests that (0,4), (3,6), (6,7), and (6,9) pairs are also linearly separable.

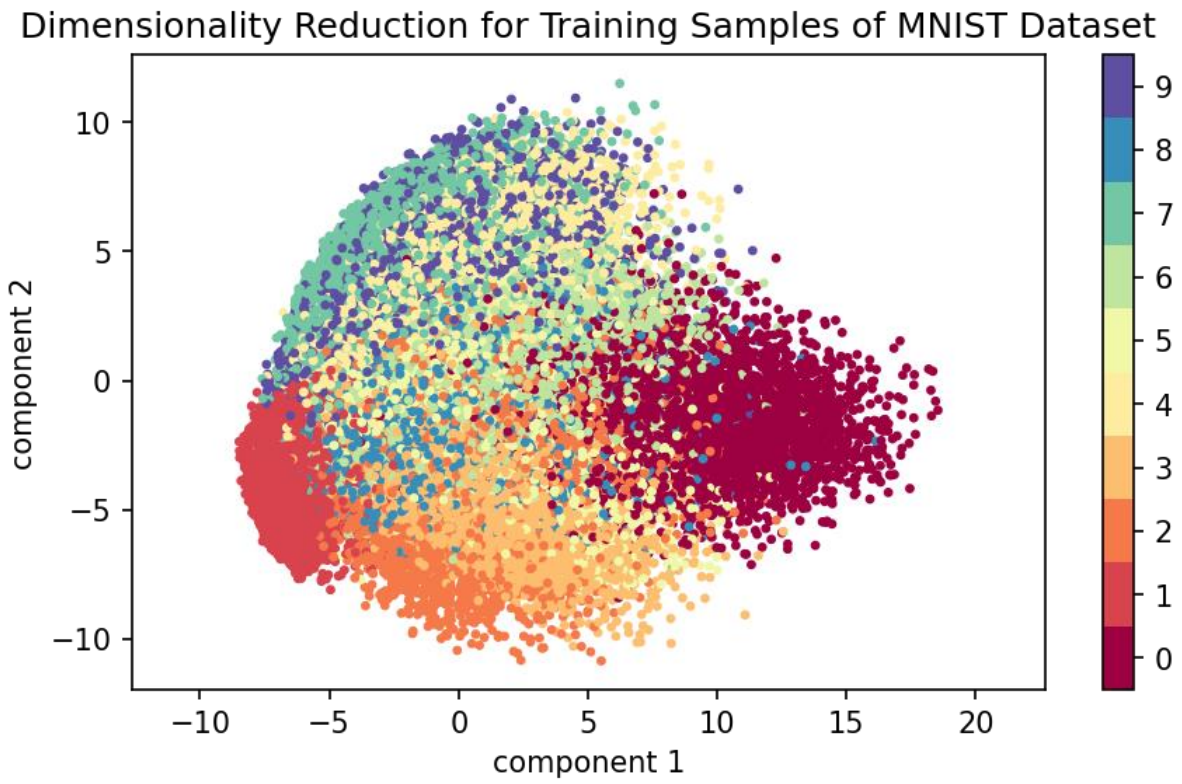


Figure 1.3

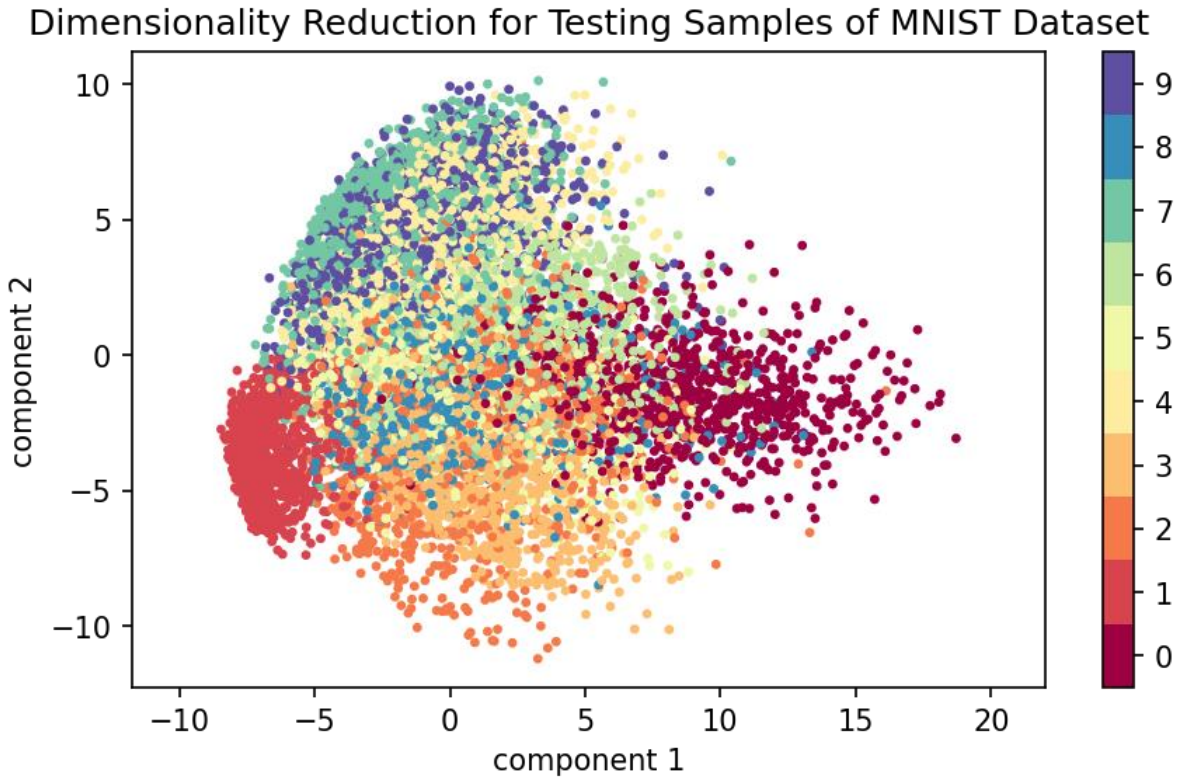


Figure 1.4: Additional Dimensionality Reduction plot for Test data

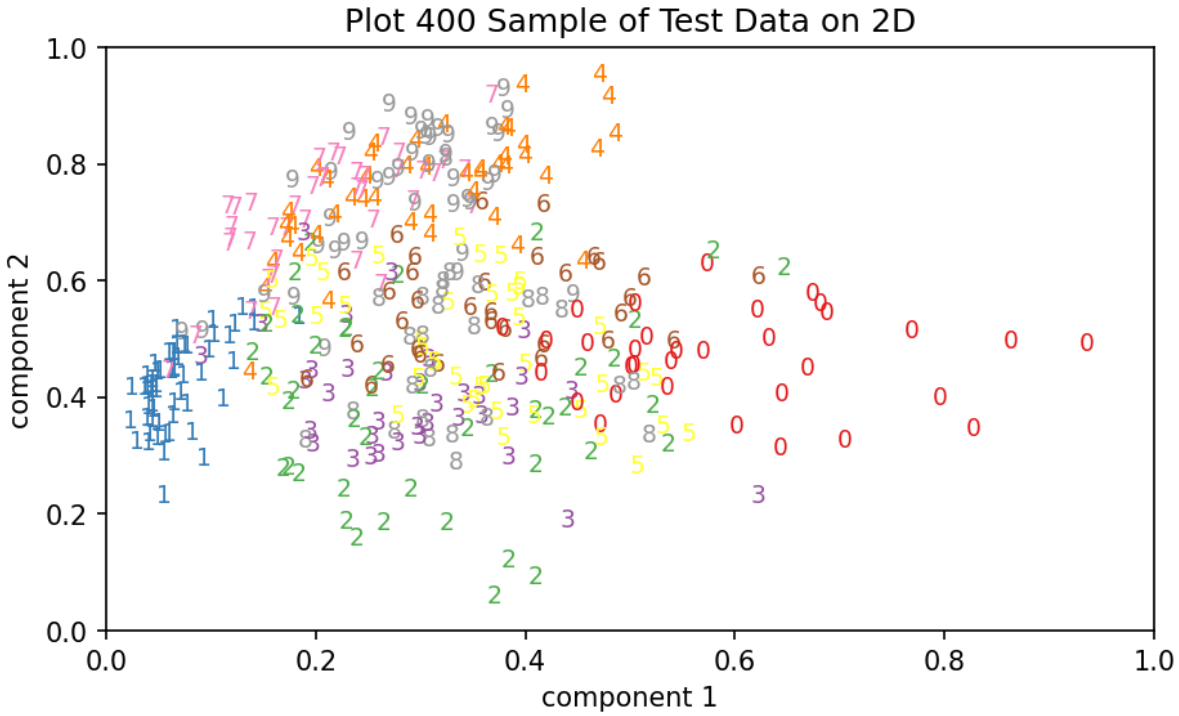


Figure 1.5

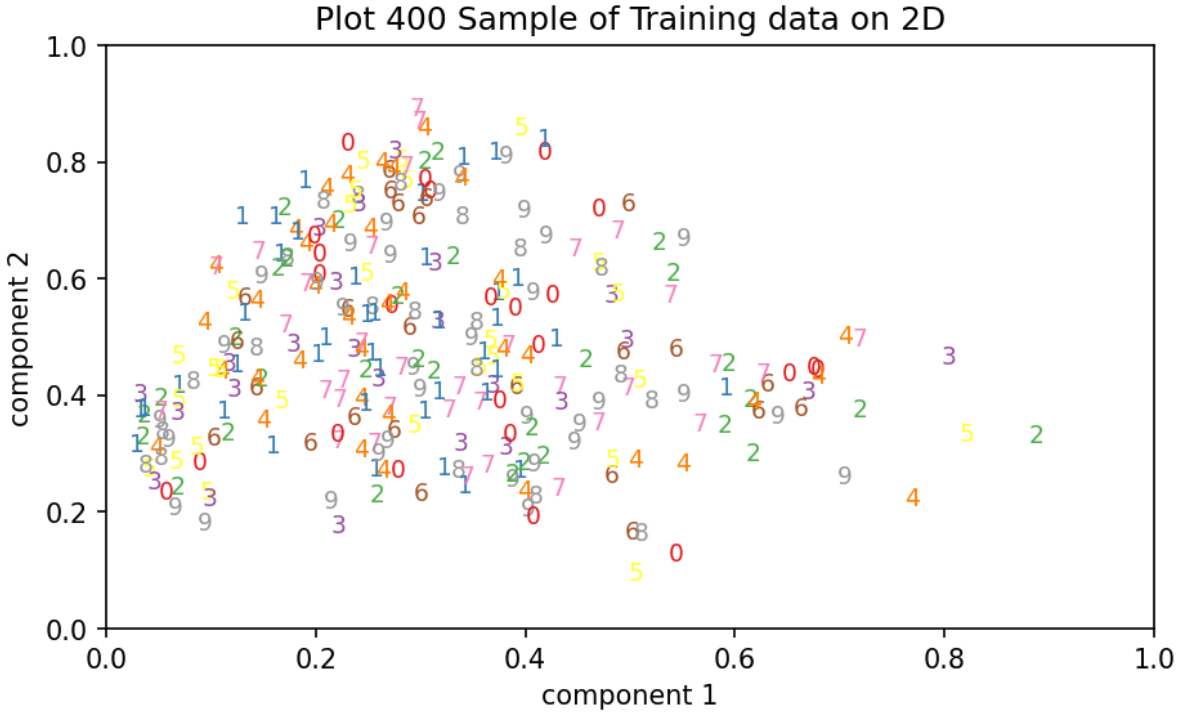


Figure 1.6: Additional plot for 400 samples from train data

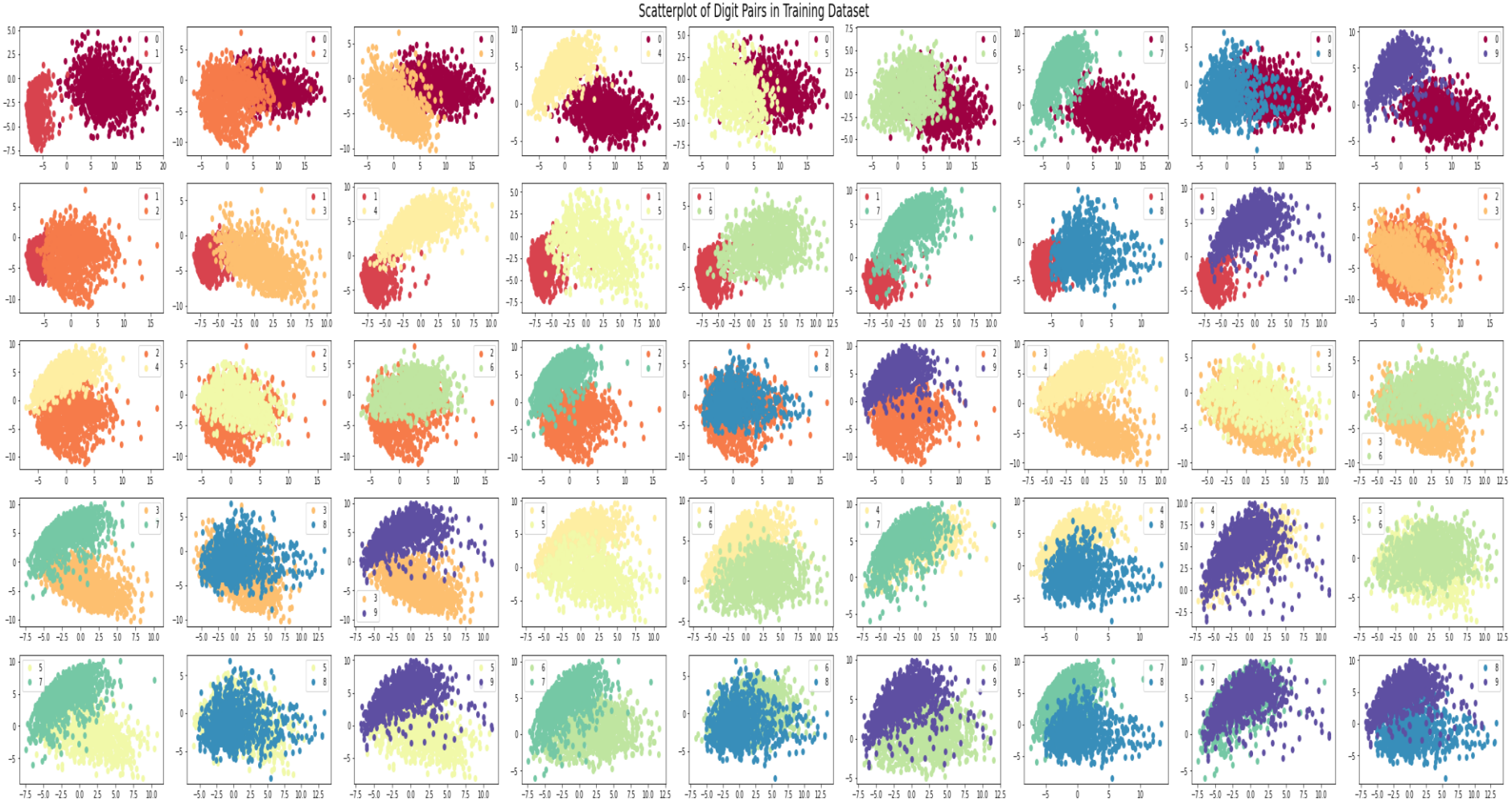


Figure 1.7

Task-2 Perceptron

We construct a single-layer perceptron that could receive an input $x \in R^{n \times m}$ and return a binary labels prediction, i.e. [-1,1]. After going through the feedforward function (predict), We setup a step function (signum), see figure-2.1 as an activation function. This will scale the input values to range between -1 to 1 which's useful for binary classification schemes.

$$g(x) = wT\,x + b$$

$$f(x) = sign(wT\,x + b)$$

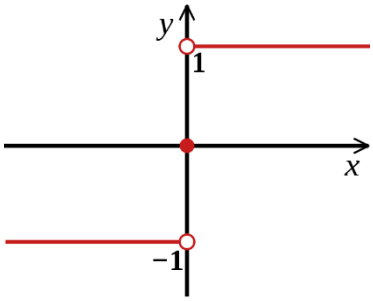


Figure 2.1

We use class dissimilarity loss function:

$$J_{soft}(\theta) = \frac{1}{N}\sum_{n=1}^{N} loss(y_n,\ f_n(x)\,) = \frac{1}{N}\sum_{n=1}^{N} f_n(x) - y_n$$

We apply gradient descent to update weights and bias only when $y_i \neq f_i(x)$

$$W_{new} = W_{old} + learning\ rate * \sum_{j=1}^{m} (\,y_j - f_j(x)\,)\,x_j$$

$$b_{new} = b_{old} + learning\ rate * \sum_{j=1}^{m} (\,y_n - f_n(x)\,)$$

When training the perceptron to distinguish between 0 and 1 samples, model converges to a local minimum in less than 70 iterations using different multiple learning-rates, see figure-2.2.

```
When learning rate is 0.004, single perceptron learning progresses as following:
epoch 1, loss = 0.01231741018555073
epoch 2, loss = 0.0022108172127911566
epoch 3, loss = 0.0019739439399921043
epoch 4, loss = 0.0015001973943939992
epoch 5, loss = 0.0011843663639952626
epoch 6, loss = 0.0009474930911962101
epoch 7, loss = 0.0008685353335965259
epoch 8, loss = 0.00047374654559810504
epoch 9, loss = 0.0009474930911962101
epoch 10, loss = 0.0006316620607974733
epoch 11, loss = 0.00047374654559810504
epoch 12, loss = 0.00047374654559810504
epoch 13, loss = 0.0006316620607974733
epoch 14, loss = 0.0003158310303987367
epoch 15, loss = 0.0003158310303987367
epoch 16, loss = 0.0
```

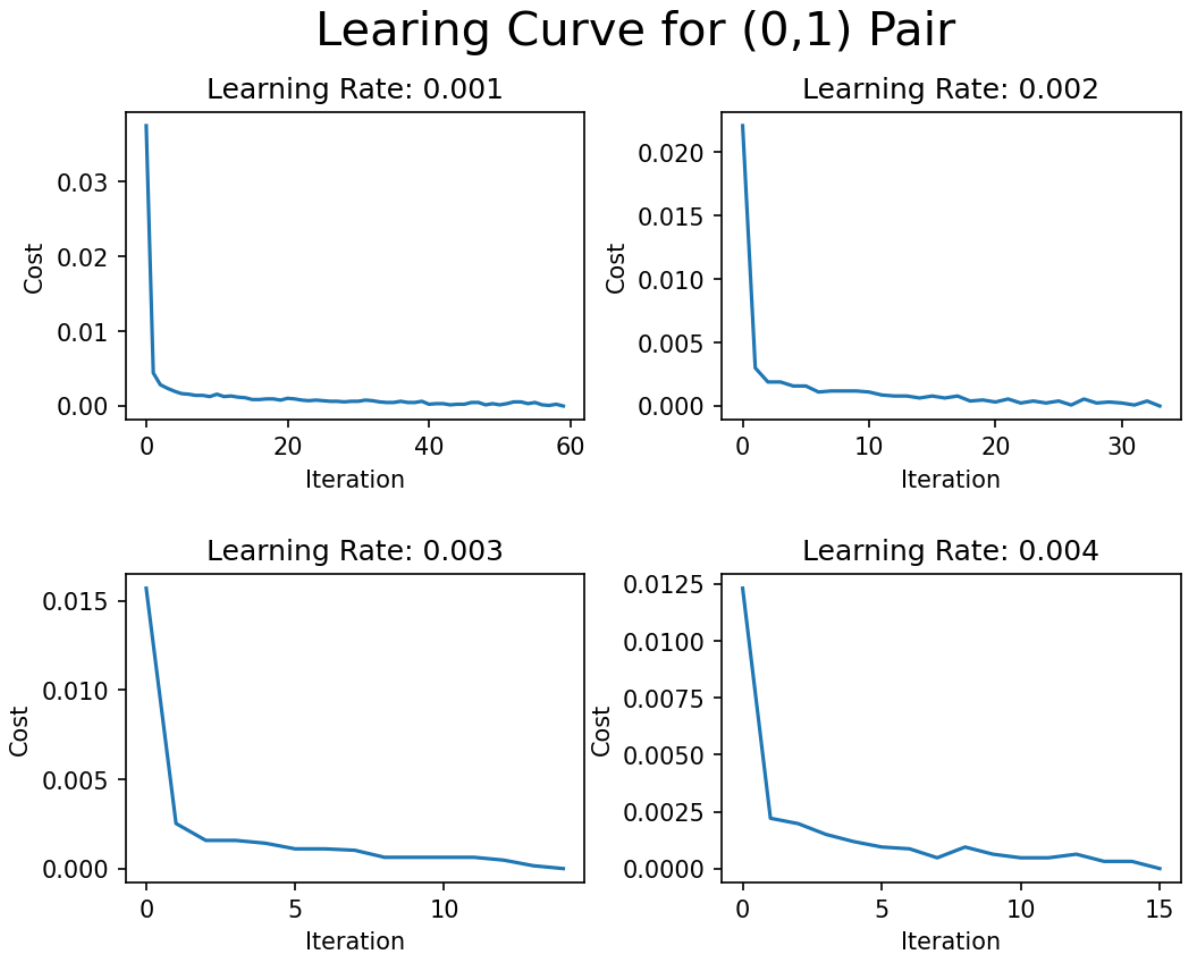


Figure 2.2

Figure-2.3-a represents the Perceptron's weights which shows a fuzzy shape of a Zero. The pixels that draw the shape of 0 have a dark blue hue, which means that these weights are negative and of high absolute value. Because of this, when those pixels are activated by receiving a zero, the perceptron result will be negative, since those weights are negative. In contrast, the 1 is expected to pass through the center of the image, hence the important weights with high positive values are in the center of the image, so that the output of the perceptron is to be positive when it receives an image of a 1. Outside the center in the 4 corners, we note that the weights values fluctuate in both end of 0 indicating that these pixels will not contribute much to activation of the perceptron on either side. Evaluating the perceptron's accuracy with the test dataset confirms that the machine has learned to differentiate the classes as the image formed by the weights show.

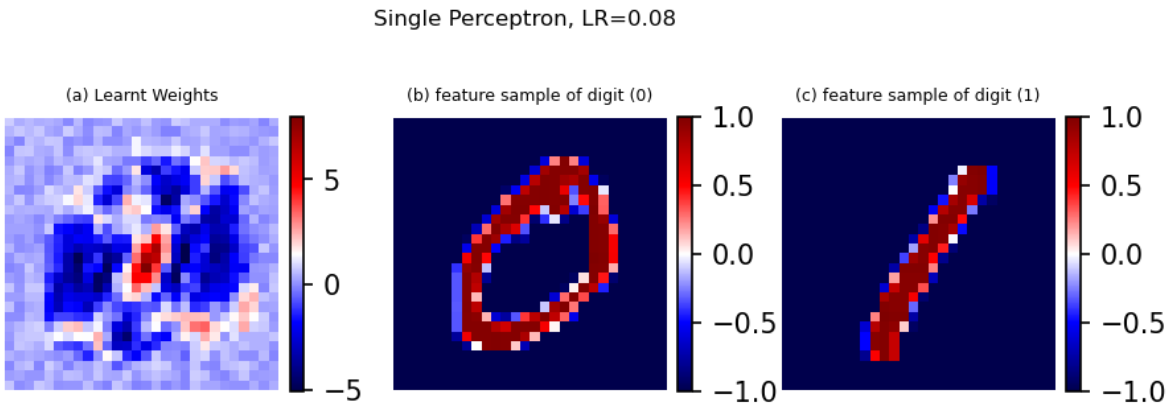


Figure 2.3

As we note in task 1, pair 0 and 1 is linearly separable so the perceptron does a good job in classifying them. We note that if training samples are linearly separable, then single layer perceptron's learning converges, and we can reach almost zero-training loss. On the other hand, when training samples are not linearly separable, learning curve oscillates and model cannot reach the predefined loss threshold (0.001), see figure-2.4.

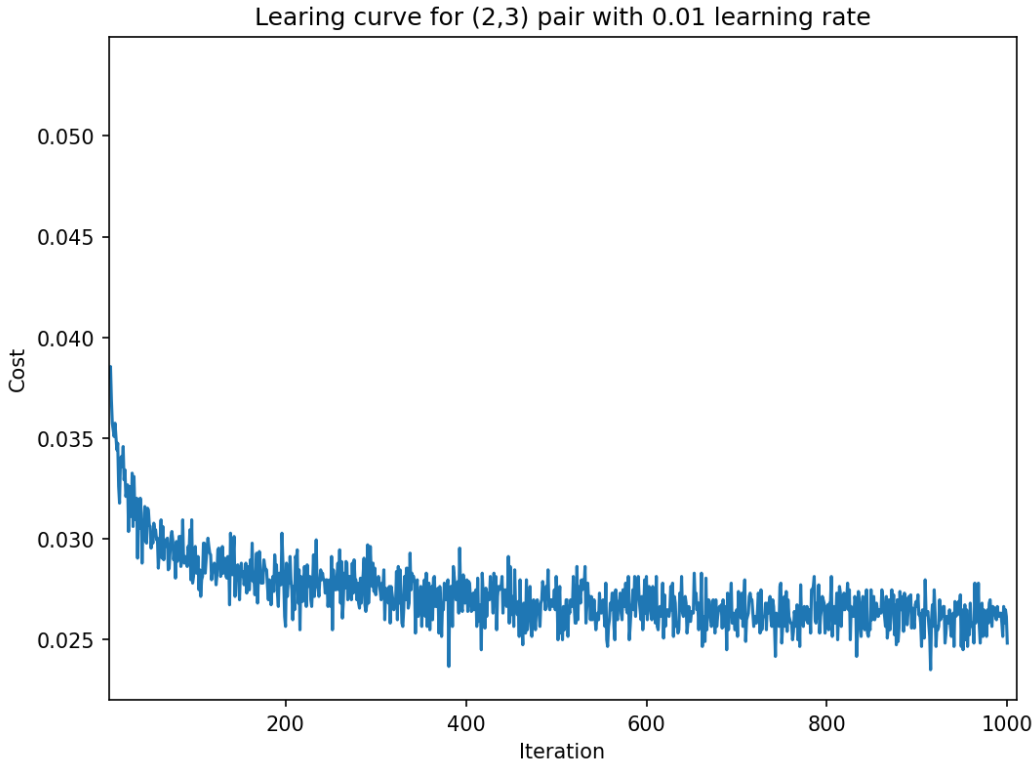


Figure 2.4

Using our model, we could find out the list of linearly separable pairs.

```
Following is a list of linearly seperable pairs:
(0, 1)  pair took 2.36 sec after going for 24 iteration
(0, 4)  pair took 4.62 sec after going for 52 iteration
(0, 7)  pair took 2.51 sec after going for 36 iteration
(1, 6)  pair took 2.37 sec after going for 30 iteration
(3, 6)  pair took 6.12 sec after going for 82 iteration
(6, 7)  pair took 2.34 sec after going for 23 iteration
(6, 9)  pair took 1.40 sec after going for 19 iteration
```

This fact is supported well by the results of training the single perceptron on various pairs, see Table Table-2.1. Some pairs prove to be linearly separable, e.g. (0,1), (0,7) where we note the relatively short training time and low iterations number. These two pairs achieve the highest accuracies among all pairs (>99%). On the other hand, perceptron couldn't converge when they were trained on pairs that are linearly inseparable. Training stopped only when iteration number threshold was reached, e.g. (2,3), (2,4), (3,8), (4,9), (5,6) and (1,2). Results of predictions on these pairs are attached in appendix.

Table 2.1

Pair	0, 1	2, 3	2, 4	3, 8	4, 9	5, 6	0, 7	1, 2
Accuracy %	99.91	94.73	97.42	95.46	96.53	94.05	99.4	98.57
Training Iteration	19	1001	1001	1001	1001	1001	57	1001
Training Time (sec)	1.98	80.61	67.78	75.68	71.2	71.55	2.95	99.41

Task-3 (Multi-Layer Perceptron)

Considerations for Task 3 and 4:

- **Network Depth:** refers to the number of layers with trainable parameters excluding the 10 neurons output layer.
- **Network Width:** refers to the largest number of Neurons/channels among all network layers.
- Models training for task 3 and 4 was done on both Google premium GPU and on local CPU (Intel(R) Core(TM) i7-10700 CPU @ 2.90GHz and 32 GB RAM).
- We have taken 10% of the training data for validation during the training to ensure that test data is not leaked to the model during training.
- For task 3.2 and 4.2, since the aim is not to seek the best performing model but to compare and discuss different architectures, we refrained from fine-tuning the models using techniques such as early-stop, regularization, dropout.. etc, especially when encountering overfitting. However, we have pointed that out in our observations.
-

3.1:

We have created an MLP model with the given architectures, trained it for 10 epochs using Cross Entropy loss with Adam optimizer and learning-rate of 0.001.

Table-3.1 shows results on both training and testing datasets.

```
#build MLP baseline model based on the configuration given in the assignment

MLP_model = Sequential()
MLP_model.add(Dense(1000, input_dim=784, activation = 'relu'))
MLP_model.add(Dense(1000,activation='relu'))
MLP_model.add(Dense(10,activation='softmax'))
MLP_model.summary()
```

```
Model: "sequential_18"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_48 (Dense)            (None, 1000)              785000

 dense_49 (Dense)            (None, 1000)              1001000

 dense_50 (Dense)            (None, 10)                10010

=================================================================
Total params: 1,796,010
Trainable params: 1,796,010
Non-trainable params: 0
```

Model	# Of Parameters	Depth	Accuracy (Training)	Accuracy (Testing)
MLP Baseline	1,796,010	2	98.87 %	97.69 %

Table 3.1: MLP Baseline Model Accuracy on Train and Test Data

The confusion matrix shows good classification for all labels. Notably, 36 instances of '4' have been misclassified as 9 which are approaching to human error on differentiating these hand-written digits.

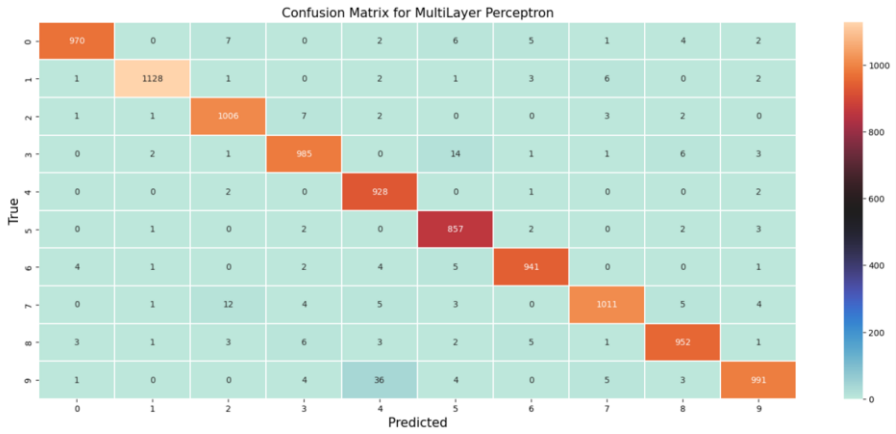


Figure 3.2 MLP Baseline Model Confusion Matrix

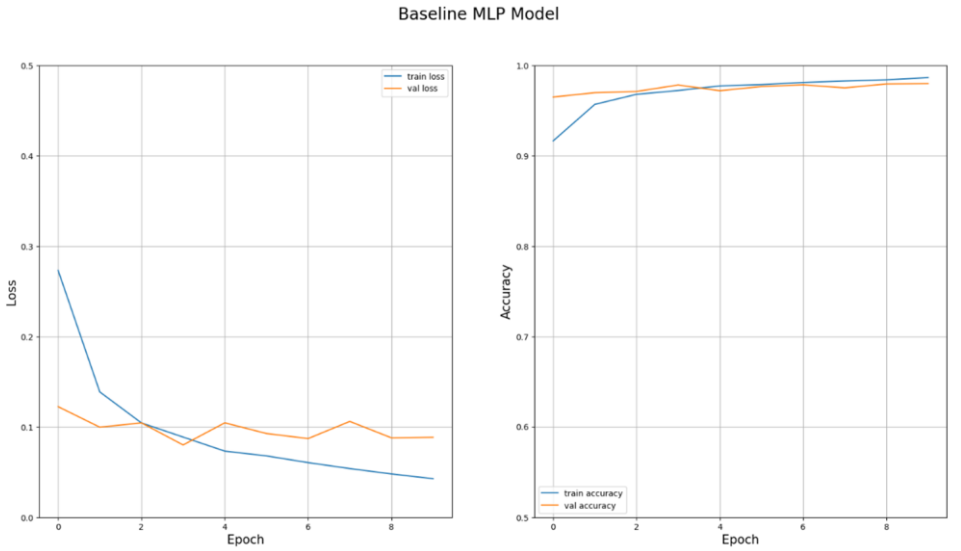


Figure 3.3 Learning Curves - MLP Baseline Model

Observations:

- Unlike the training curves, validation curves start with better results from the beginning and that's because training scores are calculated during the epoch while validation scores are calculated at the end of the epoch. We are using low batch size (50) which means that each epoch performs backpropagation 1080 times (# of batches) before one epoch is completed.
- Validation curves remain almost flat throughout the training indicating signs of minimal overfitting visible especially after epoch 4 that can be tackled using early stopping, but that is not a major concern, the model still performs well on the left-out test-data.

3.2

For this task and task **4.2**, our strategy was to use grid-search to optimize model validation accuracy with various number of layers [2, 3, 4, 5]. To achieve this, we used "keras tuner" that allows to search various number of models architectures (5 trials for each number of hidden layers), then we picked the best and worst performing models, trained them on 10 epochs using both Google premium GPU and local CPU using batch size of 50 and learning rate of 0.001, and report the accuracy on the test dataset. Below table summarizes our findings:

Model	# Of Parameters	# Of Hidden Layers (Depth)	Max Width	Width Configurations	Accuracy (Testing)	Time - CPU (seconds)	Time - GPU (seconds)
MLP Baseline Model	1,796,010	2	1000	Dense_1 (1000) Dense_2 (1000)	97.69	116.2	35.3
Additional Model 1	1,324,362	2	992	Dense_1 (992) Dense_2 (544)	97.26	87.09	35.55
Additional Model 2	1,861,578	3	1088	Dense_1 (768) Dense_2 (1088) Dense_3 (384)	97.15	125.84	33.51
Additional Model 3	4,253,258	4	1632	Dense_1 (512) Dense_2 (1280) Dense_3 (1632) Dense_4 (672)	97.07	303.95	35.29
Additional Model 4	3,405,418	5	1984	Dense_1 (224) Dense_2 (1984) Dense_3 (768) Dense_4 (1088) Dense_5 (384)	97.04	243.04	37.96
Additional Model 5	848,170	2	960	Dense_1 (480) Dense_2 (960)	97.27	64.24	31.39
Additional Model 6	3,165,482	3	1632	Dense_1 (512) Dense_2 (1280) Dense_3 (1632)	96.94	225.66	33.64
Additional Model 7	3,085,354	4	1184	Dense_1 (480) Dense_2 (960) Dense_3 (1184) Dense_4 (928)	96.93	212.1	34.76
Additional Model 8	4,458,314	5	1184	Dense_1 (960) Dense_2 (1184) Dense_3 (928) Dense_4 (992) Dense_5 (544)	96.43	295.14	36.63

Table 3.2 MLP Models Accuracy vs Depth vs Complexities vs Width vs Train Time

The table shows comparison of 9 different MLP models including the baseline where we can observe the following:

- The models' accuracies are high (+96%) and close to each other including baseline model, since the MNIST Handwritten Dataset is considerably easy for classification, it's the "Hello World" of Computer Vision (Gupta, 2020), it also helps that MNIST images are centered, scaled, and normalized before the model sees them.
- The highest performing models are of different complexities, number of layers and max width, hence there's no evidence of a direct relationship between model performance and a single architectural choice, rather a mixture of all.
- The training time for all models on Google-Premium-GPU is very similar for all models because Google-GPU distributes the tasks over multiple compute units in parallel, therefore it can't be a valid comparison factor.
- Considering training time on CPU, there's direct relationship between models' complexities and time taken for training and that makes intuitive sense.

Task 4 - CNN:

4.1

We have created a CNN classifier with the given configurations, trained it for 10 epochs using Cross Entropy loss with Adam optimizer and learning-rate of 0.001.

Below are the accuracies on both training and testing datasets.

```
# Baseline Model based on the given parameters from the assignment.

CNN_model = Sequential()

CNN_model.add(Conv2D(32, (4, 4), activation='relu', input_shape=(28, 28, 1)))
CNN_model.add(Conv2D(64, (4, 4), activation='relu', strides=2))
CNN_model.add(Conv2D(128, (4, 4), activation='relu', strides=2))
CNN_model.add(Flatten())
CNN_model.add(Dense(10, activation='softmax'))
CNN_model.summary()
plot_model(CNN_model, show_shapes=True) # visualize the CNN artitecture
```

```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 25, 25, 32)        544

conv2d_1 (Conv2D)            (None, 11, 11, 64)        32832

conv2d_2 (Conv2D)            (None, 4, 4, 128)         131200

flatten (Flatten)            (None, 2048)              0

dense_39 (Dense)             (None, 10)                20490

=================================================================
Total params: 185,066
Trainable params: 185,066
Non-trainable params: 0
```

Model	# Of Parameters	# Of Layers	Accuracy (Training)	Accuracy (Testing)
CNN Baseline	185,066	3	99.69 %	98.98 %

Table 4.1: CNN Baseline Model Accuracy on Train and Test Data

The confusion matrix below demonstrates almost perfect classification across all labels. False labelling hasn't exceeded 8 on any pair.

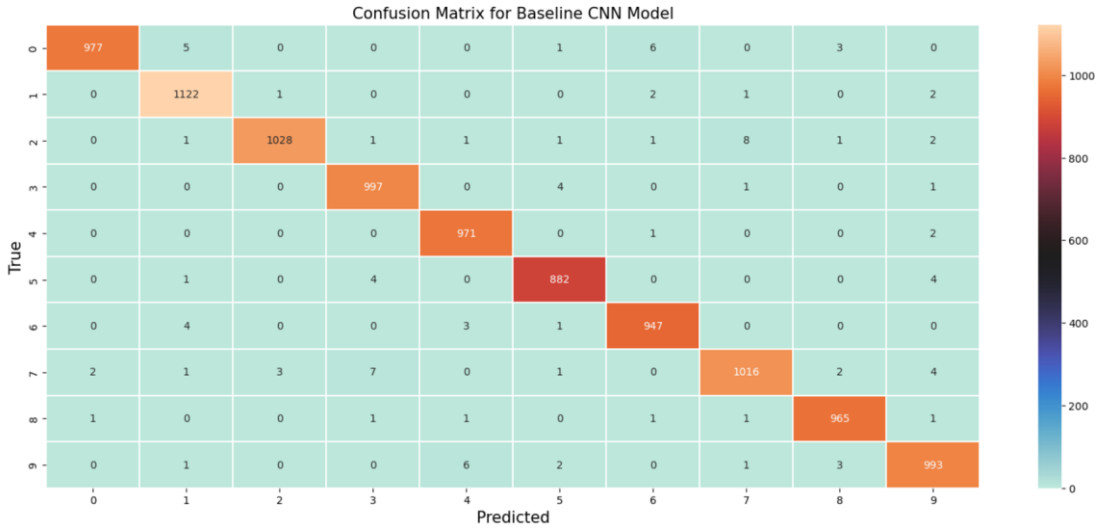


Figure 4.1 CNN Baseline Model Confusion Matrix

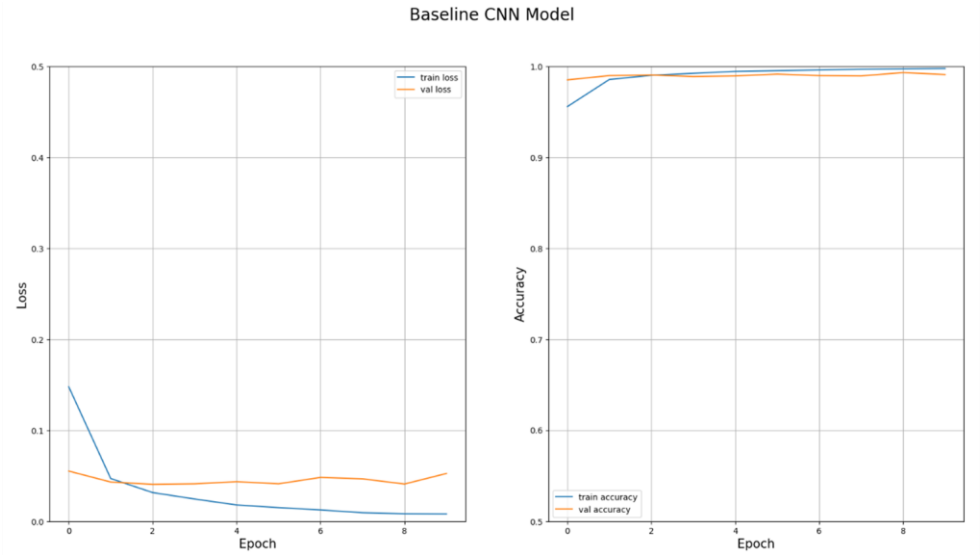


Figure 4.2 Learning Curves - CNN Baseline Model trained on 10 epochs.

Observations:

- Like task 3.1, validation curves start with better results from the beginning because training scores are calculated during the epoch while validation scores are calculated at the end of the epoch.
- As learning curve for baseline model trained on 10 epochs shows clear symptom of overfit after epoch 4, we retrained the model on 4 epochs only. Although the new model has less overfit, we noticed that test accuracy for the later model is lower than the former with 98.93%. Going forward to task 4.2, we choose to train models on more epochs to get a better accuracy acknowledging the fact that resulting models suffer from overfit.

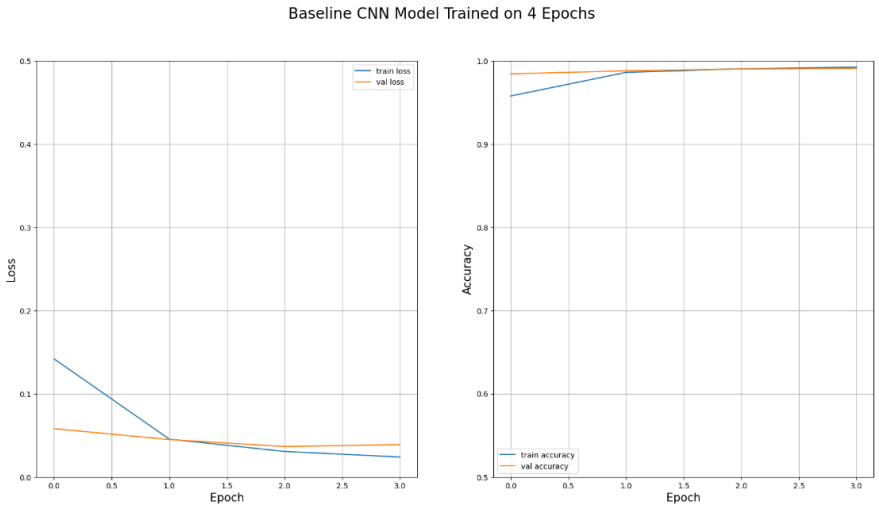


Figure 4.3 Learning Curves - CNN Baseline Model trained on 4 epochs

4.2:

Like task 3.2, we have created an additional 8 CNN models with different depths, widths and complexities using keras tuner and picked the highest and lowest performing models out of 5 configurations per each number of conv layers [2, 3, 4, 5], trained them for 10 epochs on Google GPU and local CPU using batch size of 50 and learning rate of 0.001. We have maintained the type of CNN structure, just added extra Conv layers with no MaxPooling to investigate the effect of different Conv depths while using different number of filters and kernels.

Below table summarizes our findings:

Model	# Of Parameters	# Of Conv Layers (Depth)	Max Width	Width Configurations	Accuracy (Testing)	Time - CPU (seconds)	Time - GPU (seconds)
CNN Baseline Model	185,066	3	128	Conv_1_filter (32, 4x4) Conv_2_filter (64, 4x4) Conv_3_filter (128, 4x4)	98.98	240	44.45
Additional Model 1	804,778	2	192	Conv_1_filter (192, 3x3) Conv_2_filter (96, 4x4)	98.55	2,533	55.36
Additional Model 2	579,178	3	288	Conv_1_filter (64, 3x3) Conv_2_filter (288, 2x2) Conv_3_filter (64, 3x3)	98.67	2,280	48.46
Additional Model 3	2,474,218	4	416	Conv_1_filter (128, 3x3) Conv_2_filter (416, 2x2) Conv_3_filter (128, 3x3) Conv_4_filter (320, 3x3)	99	6,498	80.82
Additional Model 4	2,414,954	5	416	Conv_1_filter (128, 3x3) Conv_2_filter (416, 2x2) Conv_3_filter (128, 3x3) Conv_4_filter (320, 3x3) Conv_5_filter (256, 2x2)	98.76	7,848	90.32
Additional Model 5	2,814,698	2	416	Conv_1_filter (128, 3x3) Conv_2_filter (416, 2x2)	97.87	1,908	51.22
Additional Model 6	903,114	3	320	Conv_1_filter (320, 3x3) Conv_2_filter (256, 2x2) Conv_3_filter (64, 4x4)	98.47	5,074	62.24
Additional Model 7	1,676,874	4	288	Conv_1_filter (64, 3x3) Conv_2_filter (288, 2x2) Conv_3_filter (64, 3x3) Conv_4_filter (288, 3x3)	99	2,950	60.26
Additional Model 8	5,255,786	5	512	Conv_1_filter (224, 2x2) Conv_2_filter (480, 3x3) Conv_3_filter (288, 2x2) Conv_4_filter (256, 2x2) Conv_5_filter (512, 3x3)	98.6	14,521	135.55

Table 4.4 CNN Models Accuracy vs Depth vs Complexities vs Width vs Train Time

The table shows comparison of 9 different CNN models including the baseline where we can observe the following:

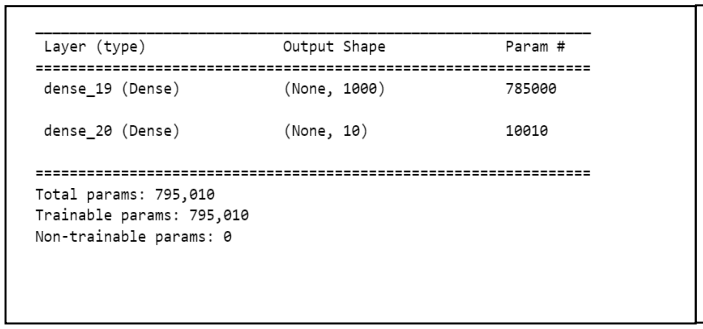
- Like MLP the models' accuracy performances are close to each, but CNN outperforms MLP.
- Models' accuracies are high (98%) with different complexities, number of layers and width, hence there's no direct relationship between model performance and a single parameter, rather a mixture of all including the filters configurations.
- Model 3 and 7 outperformed the baseline model, but at a very expensive and unnecessary cost (Time and computation power!!).
- There's evidence of a direct relationship between models' complexities and time taken for training and that makes intuitive sense, it's never a good idea to train large models on CPU, GPU is worth the investment.

4.3 MLP vs CNN:

From table 3.2 and 4.2 we can notice that:

- CNN outperforms MLP in all configurations, and that's because CNN maintains spatial relation between pixels, and if the images were of high resolution, CNN superiority would've been crystal clear.
- It's obvious that the time taken to train CNN models is way more than that for MLP because task 4.2's purpose was to maximize validation accuracy (by experimenting with different architectures). Hence, we did not capitalize on CNN convolution power to down-sample images to reduce trainable parameters by use of e.g. Pooling layers and strides>1, therefore number of parameters was high. Furthermore, CNN models converged in less epochs than MLP (see learning curves in Notebook), but we went all the way to 10 epochs for the sake of comparison.

We trained 2 more models with single hidden/conv layer - to show the superiority of CNN over MLP - for 2 epochs.

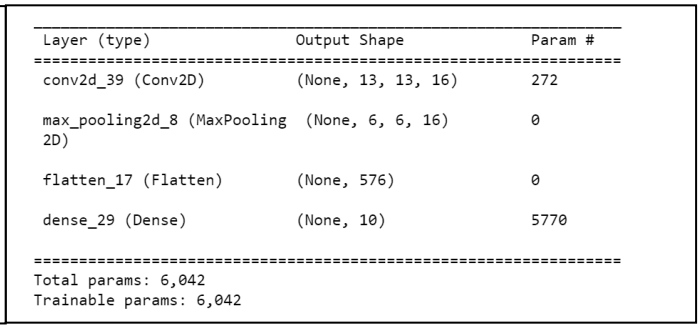


```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 dense_19 (Dense)             (None, 1000)              785000

 dense_20 (Dense)             (None, 10)                10010

=================================================================
Total params: 795,010
Trainable params: 795,010
Non-trainable params: 0
```

MLP Model Architecture



```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d_39 (Conv2D)           (None, 13, 13, 16)        272

 max_pooling2d_8 (MaxPooling  (None, 6, 6, 16)          0
 2D)

 flatten_17 (Flatten)         (None, 576)               0

 dense_29 (Dense)             (None, 10)                5770

=================================================================
Total params: 6,042
Trainable params: 6,042
```

CNN Model Architecture

Model	# Of Parameters	# Of Conv Layers (Depth)	Max Width	Width Configurations	Accuracy (Testing)	Time - CPU (seconds)
MLP Bonus Model	795,010	1	1000	Dense_1 (1000)	**96.26**	**10.13**
CNN Bonus Model	6,042	1	16	Conv_1_filter (16, 4x4)	**97.35**	**5.2**

Table 4.3: CNN vs MLP 1-Hidden/Conv Layer configuration.

From table-4.3, we see how CNN crushed MLP with half-time training and better accuracy with much fewer parameters. Overall, we can say that MLP can perform well in simple images classification, but CNN is much better especially when images become complex, it just must be properly designed.

Task 5

5.1 CNN Filters Visualization:

All visualizations are done for the baseline model in task 4.1.

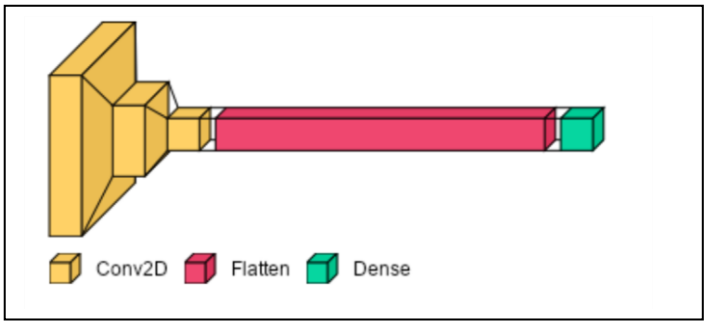


Figure 5.1 CNN Baseline model Schematic Visualization

There are 2 ways to visualize CNN filters:

1. **Observe and Visualize Filters Learnt Weights.**

   First Conv-layer has 32 filters with 1 channel per each, second Conv-layer has 64 filters with 32 channels per each, third Conv layer has 128 filters with 64 channels per each. We visualize all filters in first Conv layer, but only first channel of each filter for the other two layer, since it won't be useful to list down all of them as little non-observable squares summing to $(64 * 32 + 128 * 64) = 10{,}240$ filters. However, we have shown all of them in the notebook.

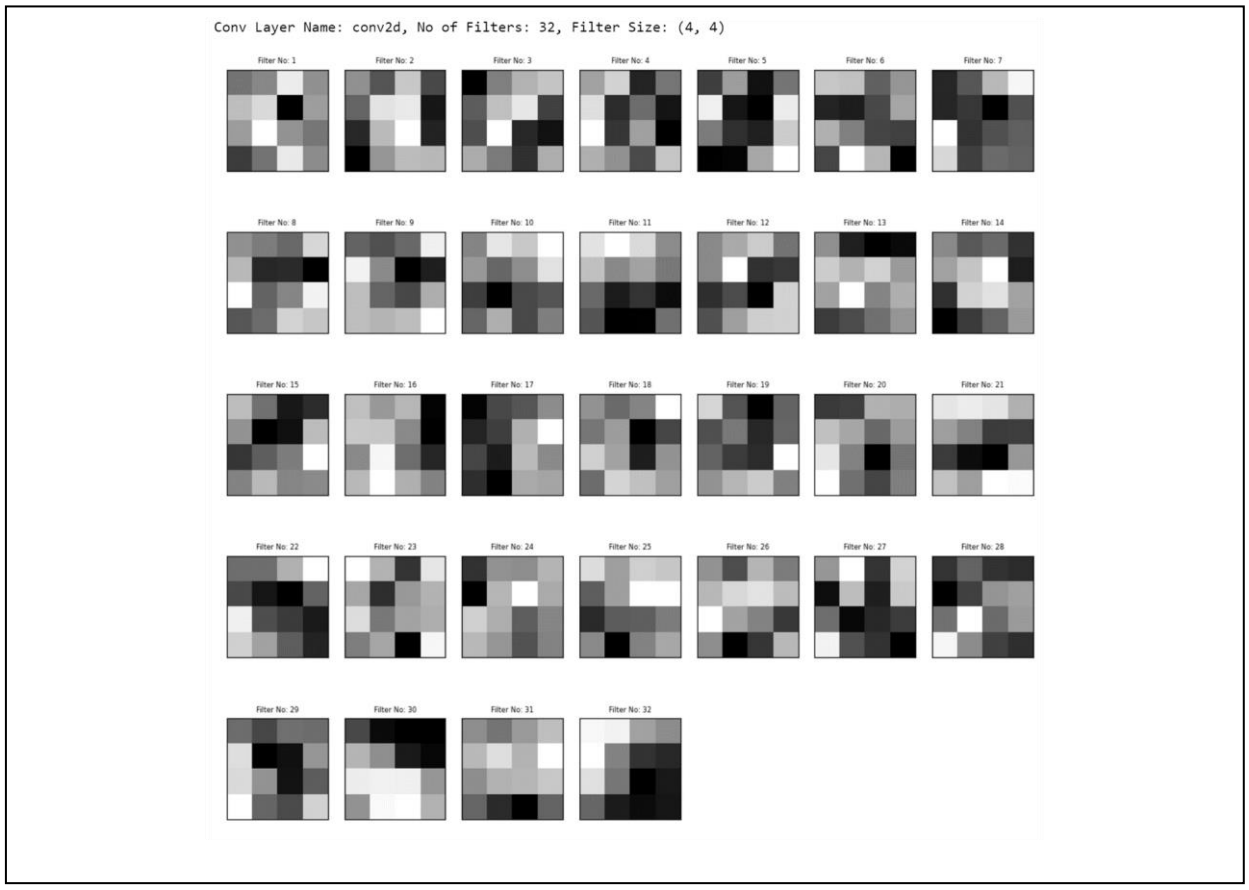


Figure 5.2 Filters Visualization using learnt weights.

Figure 5.2 Filters Visualization using learnt weights.

2. **Using Gradient Ascent in Input Space.**

   Here we search for input that maximizes filters output by applying gradient ascent to the value of the input image to maximize the response of a specific filter, starting from random noisy input image. The resulting input image will be one that the chosen filter is maximally responsive to.

   The process is to build a loss function that maximizes the value of a given filter in each convolution layer, then use stochastic gradient ascent to adjust the values of the input image to maximize this activation value. (Chollet, 2021)

We will discuss the observation of filters visualization together with feature maps visualization.

5.2 Activations / Feature Maps Visualization

The output of a convolution layer is a 3D volume. The height and width correspond to the dimensions of the feature-map, and each depth (channel) is a distinct feature-map encoding independent features, number of feature-maps from each conv layer correspond to the number of filters in that layer.

Observations:

- Weights visualization in fig 5.2 shows dark squares indicating small weights and light squares indicate large weights, some filters appear to detect gradient of bright-darks or dark-bright such as filter 32 in first layer.
- Bright areas in feature maps are the “activated” regions, meaning the filter detected the pattern it was looking for.
- First Conv layer activations maintain almost the full shape and most of the information in the image. In CNN architectures the first layers usually act as edge detectors. (Dertat, 2017)
- As we go deeper into the network, the activations become more complex and abstract. It starts encoding high-level features such as (edges, curves, angels, vertical/horizontal/diagonal lines. etc.) and this becomes clear when looking at the filter’s visualization by maximal activation in fig 5.3 & 5.4 second layers.
- The deeper we go, we see that many filters don’t get activated, meaning that the pattern encoded by the filter wasn’t found in the input image and our model is reaching its learning capacity.
- The activations of deeper layers carry less information about input image as a whole and more information about the class itself such as the line shape in the selected "9" image or the curves and angels in the "2".

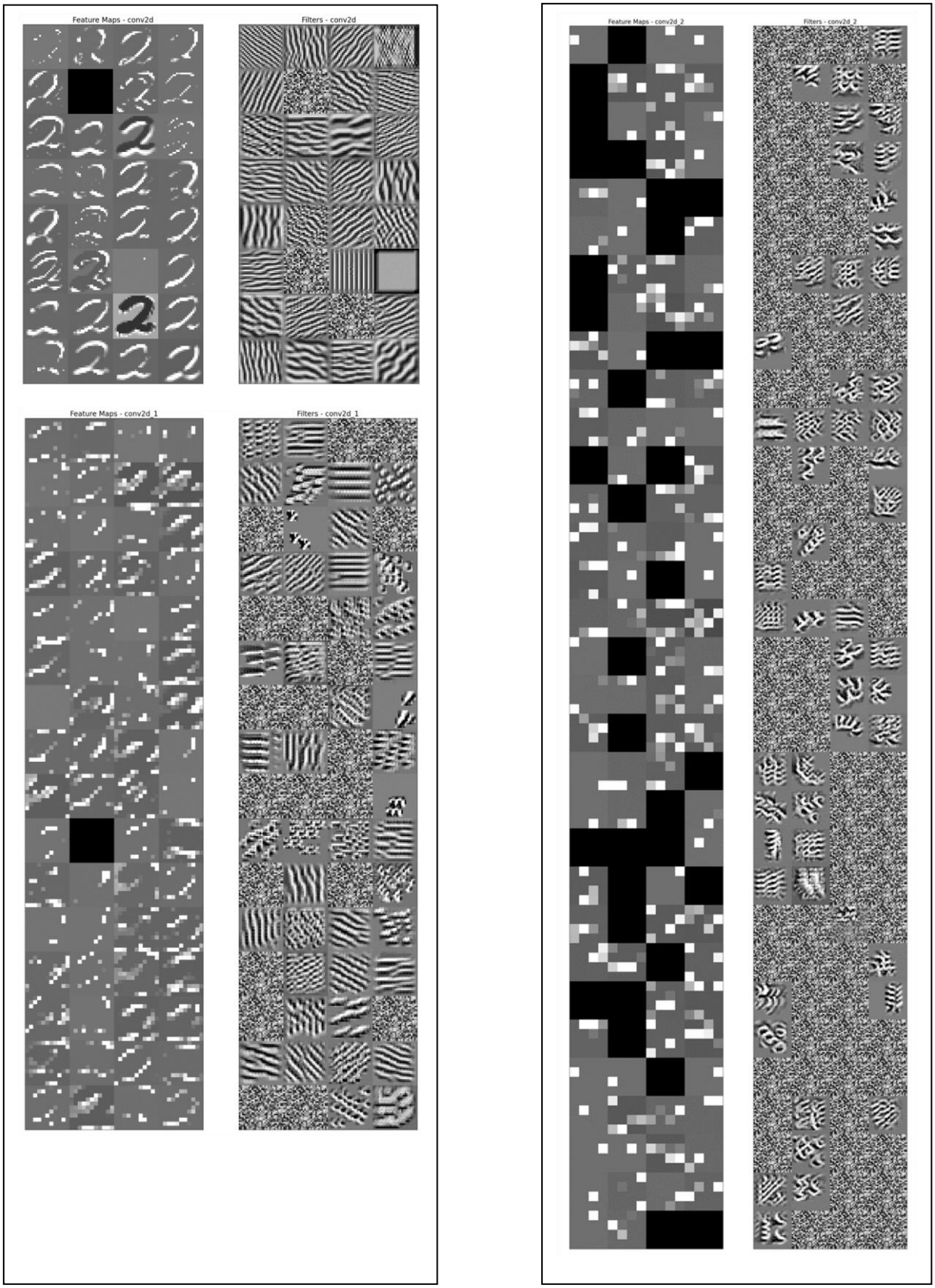


Figure 5.3 Feature Maps vs Filters visualized by maximal activation in input space for "2".

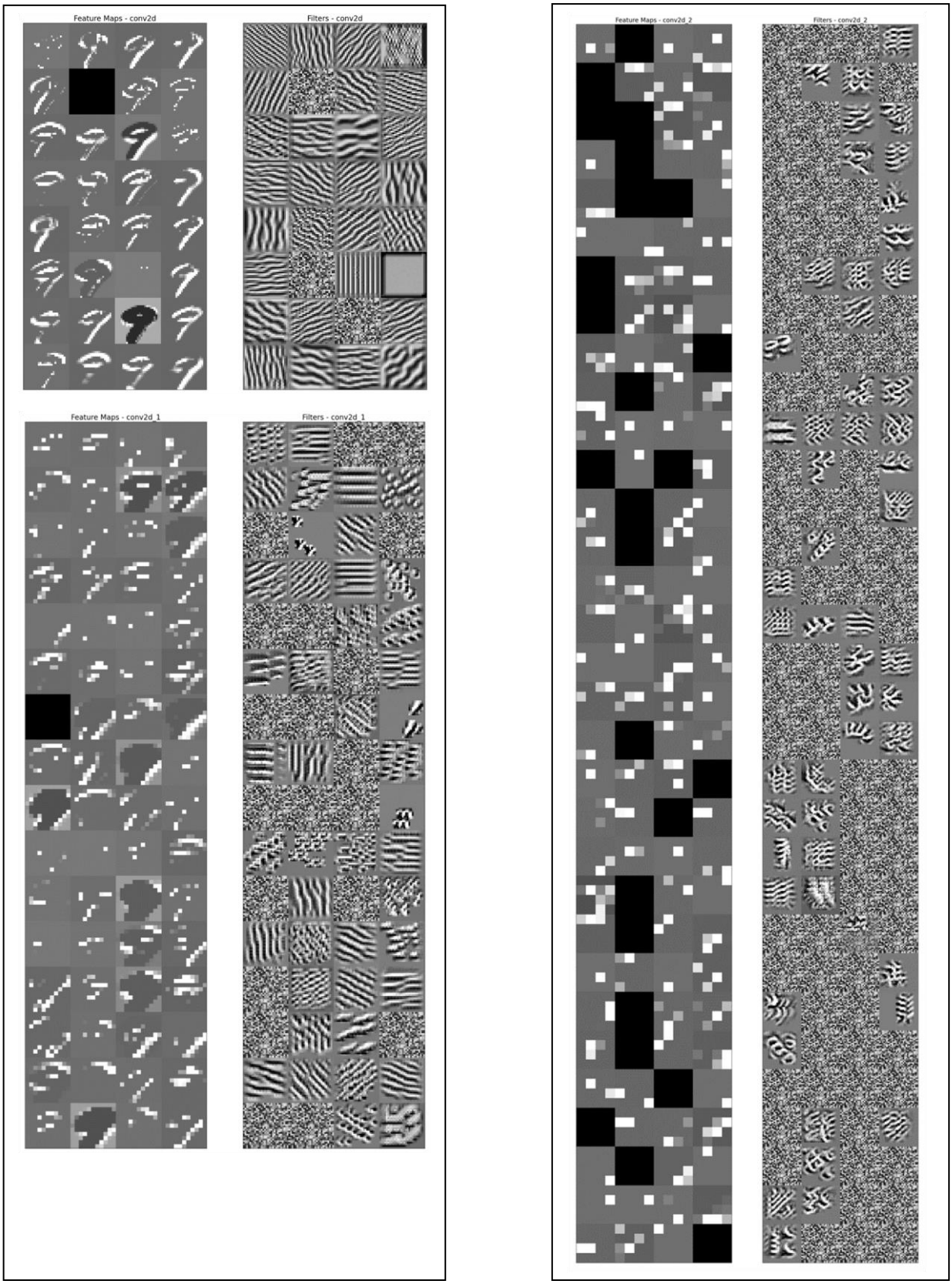


Figure 5.4Feature Maps vs Filters visualized by maximal activation in input space for "9".

Task 5.3 - Deep Dream

Deep Dream is similar to filter visualization using gradient ascent, but instead of starting from a random noise image, we start with the image we want to create deep dream for, and try to maximize the network output by updating the input image. In this way, we're enhancing the features that the network recognizes in the image. Different layers yield different results, lower ones tend to produce geometric patterns and textures, while higher ones produce more abstract shapes that resemble what the network has seen during its training process, resulting in a dream-like image.

Steps:

- Forward target image through our trained CNN.
- Select the layer of choice, (we applied to all the 3 layers).
- Calculate the activations of each layer.
- Calculate the gradient of the image with respect to the activations of each layer.
- Update the image to increase these activations by applying gradient ascent., hence we enhance the patterns seen by the network resulting in a dream-like hallucinated image.
- Repeat this process until the image shows all what each layer wanted to learn.

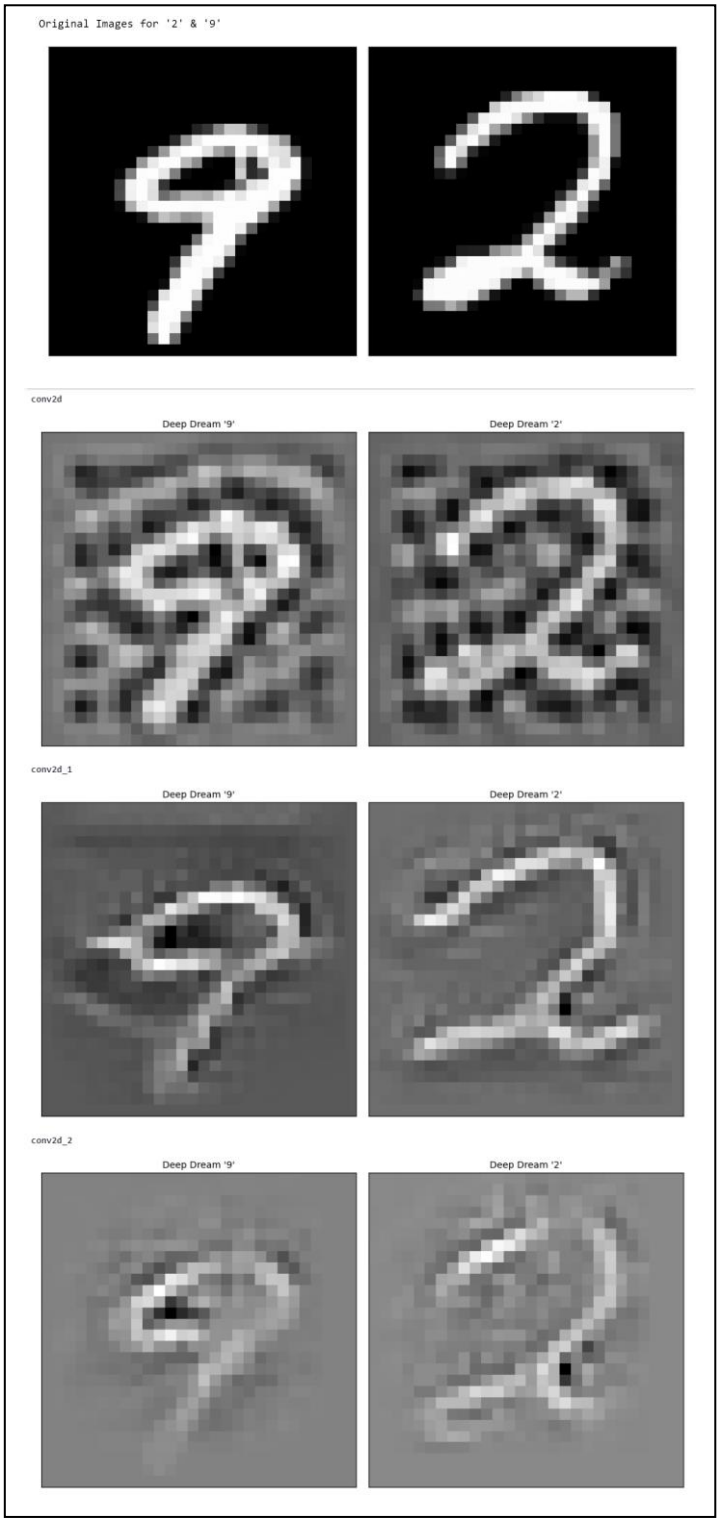


Figure 5.5 Deep Dream images from all CNN layers for "2" and "9" iterations=500, lr=0.2

Observations:

As the input images for '2' and '9' are of grey scale and low resolution, the process won't be able to activate many patterns other than the black and white scale with some basic patterns such as grey/black/white shades or edges and curves that correspond to different neurons interests.

As the first conv layer looks for main edges and mostly maintains overall shape of the image, we can see that deep dream image it produced maintains the overall shape and adds the activated textures and main edges all over it, but going to the deeper layers, we can see the high level features such as small grey squares that surrounds each shape in the original images were amplified and visible all over the resulted image to becoming the dominant pattern in the deepest layer.

Note: We have included more deep dream images with different iterations and learning rates with various colored plots in the Jupyter Notebook.

Task-6 MTL

Task 6.1

Two CNN classifiers with identical architecture (besides output layer dimensions) were trained separately on 60,000 single-channel image samples to perform classification tasks: task-1 concerns labelling the image into 10 commodities and task-2 handles the classification into 3 groups. Both classifiers have good learning curves with minimal signs of overfitting. After training for 5 epochs, see figure-6.1, training and validation accuracies exceed 90% whereas their accuracies on test dataset were 88.40% and 92.80% respectively.

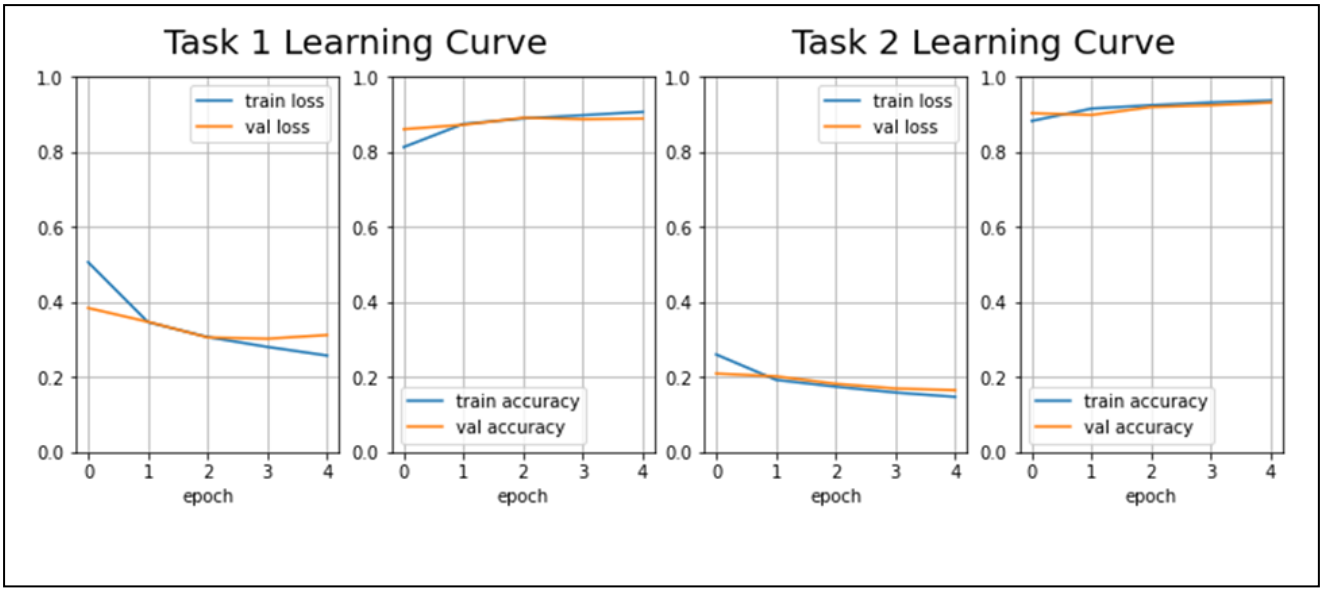


Figure 6.1

Figure 6.2 shows task-1 classifier has misclassified many instances as shirts, T-shirts, and coats. Pullovers are mistakenly labelled as either coats or shirts. Similarly, shirts are misclassified and tagged as t-shirts, coat, and pullover. The second classifier almost masters telling when images belong to shoes category, see figure-6.3, but performs worse in unisex and gendered categories.

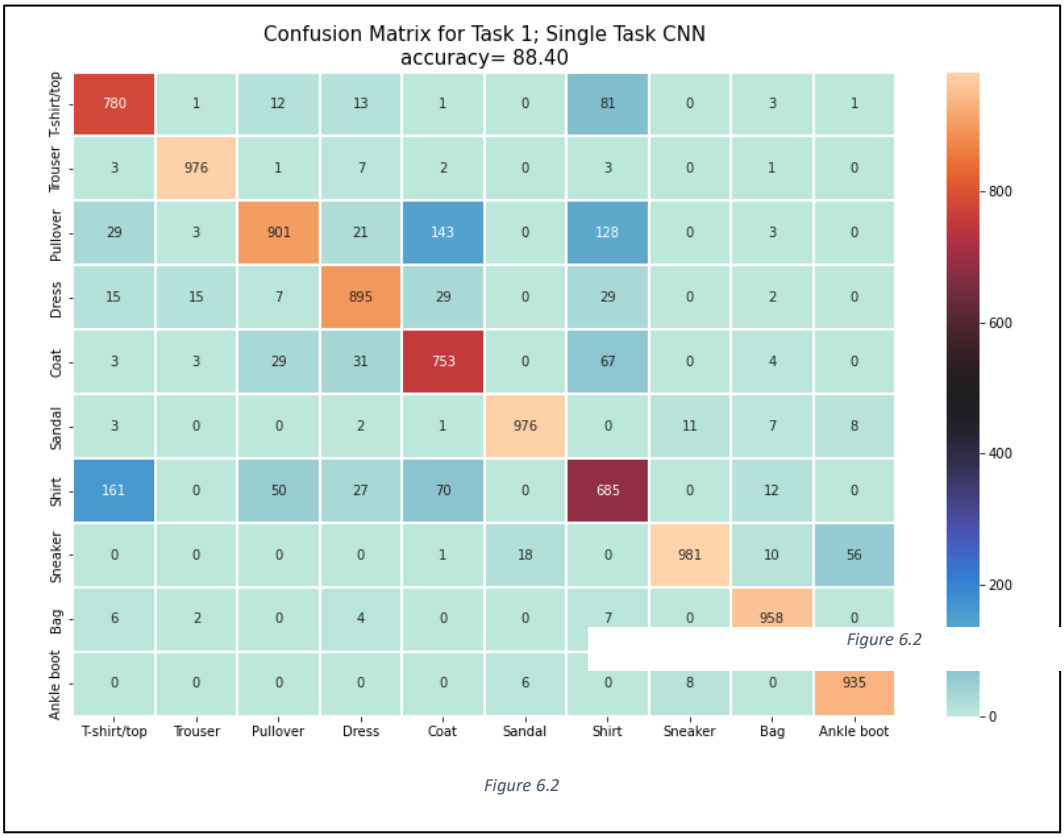


Figure 6.2

Figure 6.2

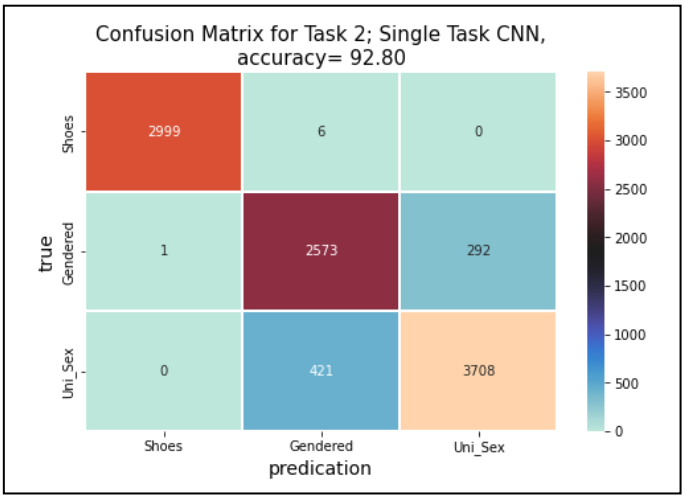


Figure 6.3

Task 6.2

We constructed an MTL model that has two heads: 3-output categorization and 10-output classification and devise a loss function that balances the priority of learning between the two heads. We tested the model on unseen test data and recorded the accuracy in table-6.1 for different values of λ.

When λ=0, task 2 dominates the training processes as we notice that the model isn't concerned with learning task-1, so we note a low accuracy and high training and validation loss, see figure-6.4. Accuracy on test data is very low, i.e., 10.2%. On the other hand, model performs much better on task 2 and demonstrates high accuracy and low loss. Model suffers overfitting to task-2, this is indeed noted in the overall training and validation loss curves.

It's totally the opposite when λ=1, task-1 dominates the training processes, task-1 accuracy is high compared to task-2's, task-1 loss in training and validation is much lower than task-2. Overfitting is still characteristic of this model, see figure-6.8, accuracy on task-2 is 28.56% compared to 89.9% for task-1.

λ =	0	0.3	0.5	0.7	1
Classification accuracy (Task1)	10.2	90.27	90.14	90.32	89.9
Categorization accuracy (Task2)	92.47	93.77	93.37	94	28.56

table 6.1

When λ = 0.7, we note that the model learns well both tasks, task-1&2 accuracies are high and loss in training is decreasing, see figure-6.7, loss in validation starts decreasing then picks up again after 4th epoch for both task 1 and 2, model has minimal overfitting to perform task 1 and 2, accuracy on test data is higher than 90% on both tasks.

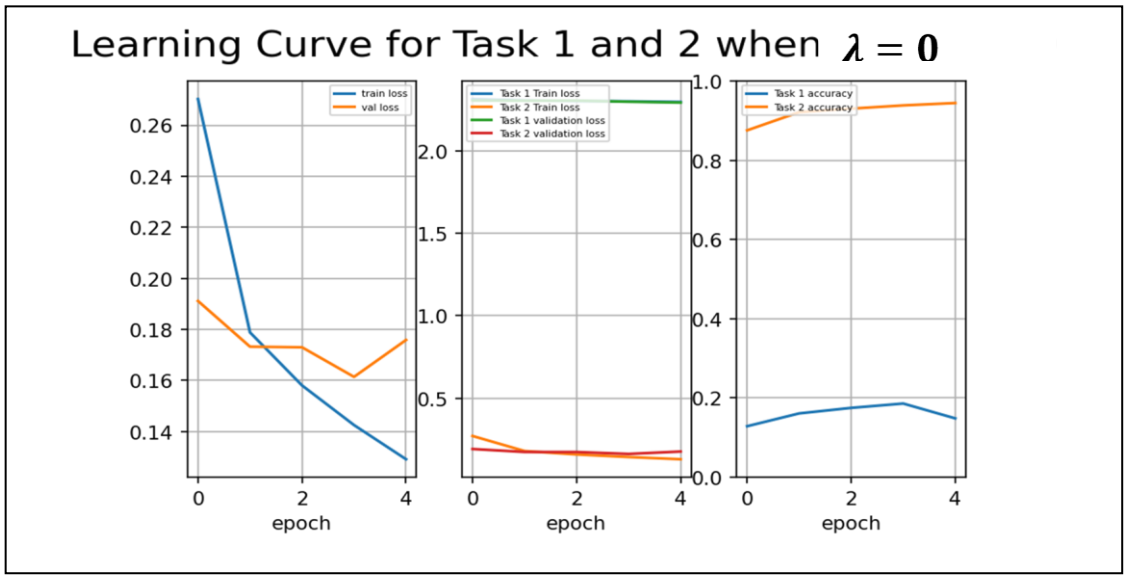


Figure 6.4

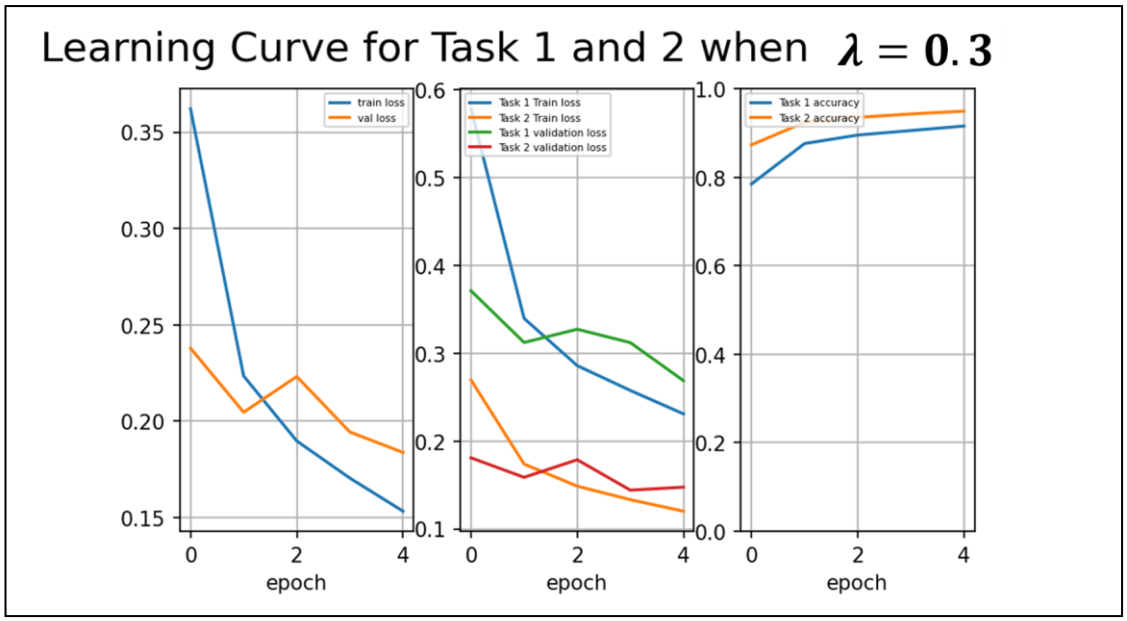


Figure 6.5

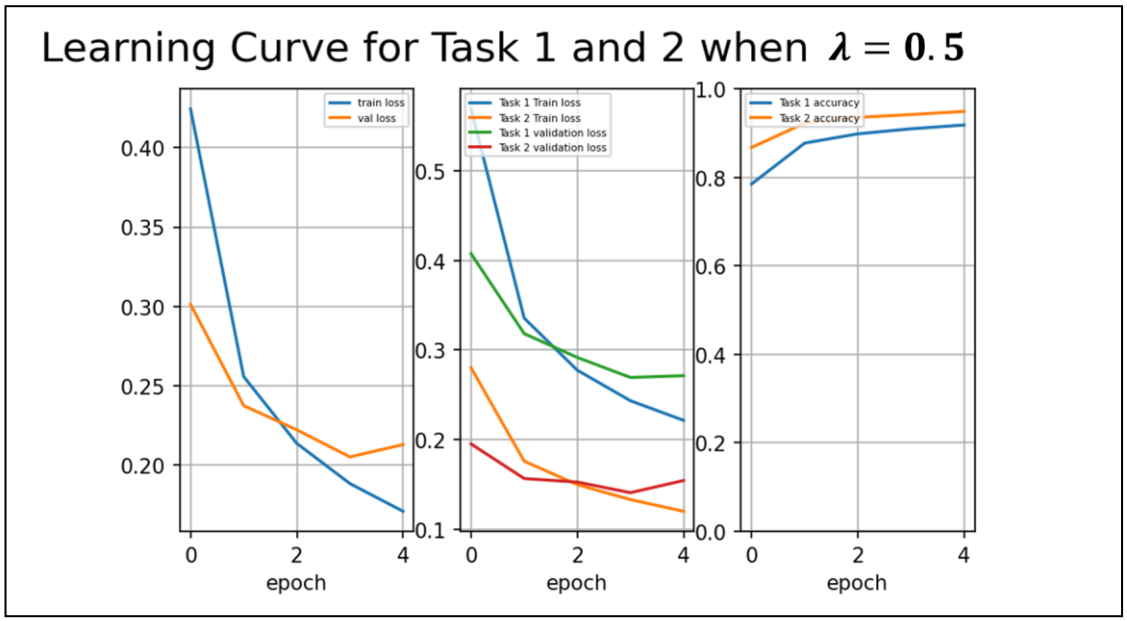


Figure 6.6

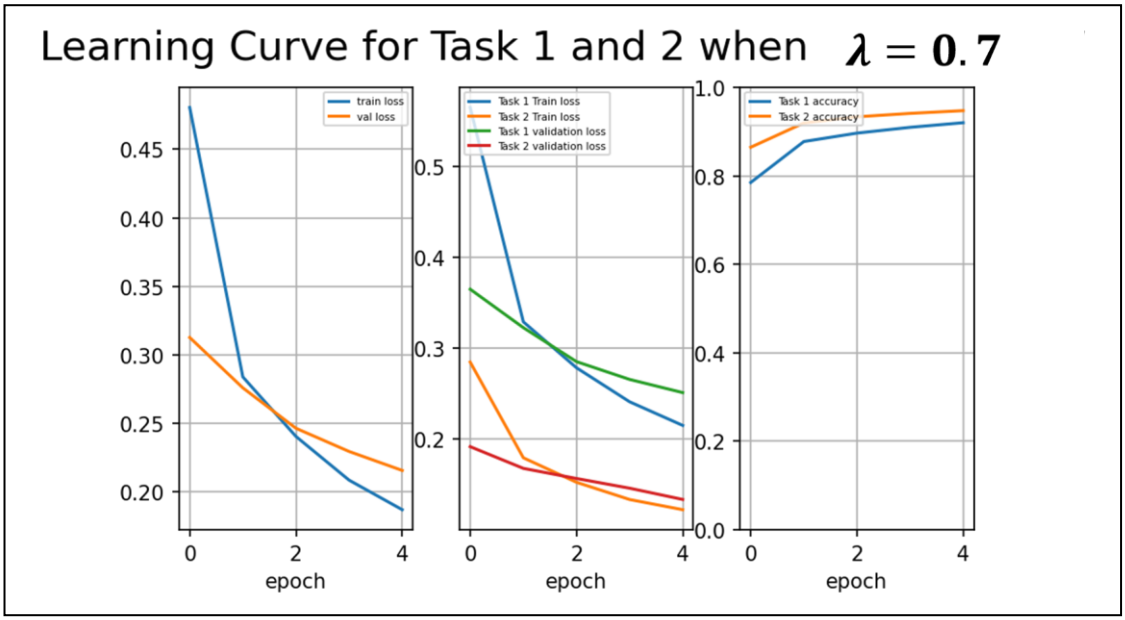


Figure 6.7

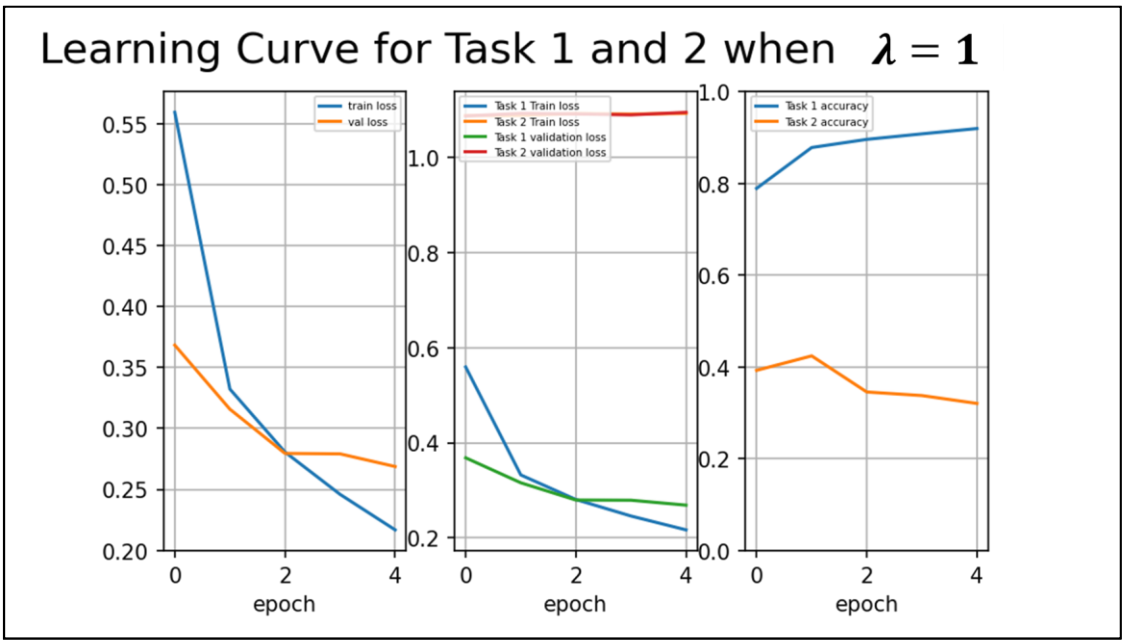


Figure 6.8

In theory, hard parameter sharing reduces the risk of overfitting, (Ruder, 2017). Our MTL model starts to show signs of overfitting at epoch 4, see figure-6.7, in comparison to task-1-STL where overfitting starts at epoch 3, see figure-6.1. This means that the model is trained for one more epoch without suffering overfitting. MTL-CNN outperforms single-task CNN in both tasks with higher accuracies, see table-6.2. It is evident that MTL improves the performance of each task by optimizing multiple tasks jointly and utilizing the correlation between related tasks. We note that MTL-CNN performs better in classifying images of dress, coat, and shirt (task-1) and their corresponding categories: gendered and unisex images (task-2). This proves a feature in MTL (attention focusing), MTL can help the model focus its attention on those features that matter as other tasks provide additional weight for relevant features. Also, given that different tasks have different noise patterns, a model that learns two tasks simultaneously can learn generalize better. Learning just task-1 bears the risk of overfitting to task-1,

while learning 1 and 2 jointly enables the model to obtain a better representation through averaging the noise patterns.

	Baseline (STL)	Δ	MTL
Classification accuracy (Task1)	88.4	**1.92**	90.32
Categorization accuracy (Task2)	92.8	**+1,20**	94

Table 6.2

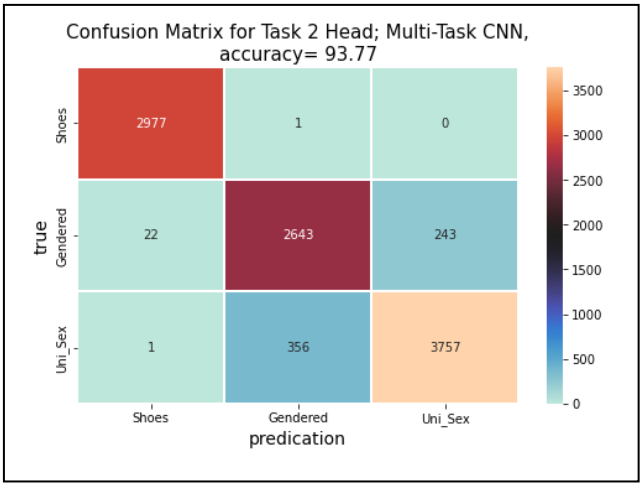


Figure 6.9

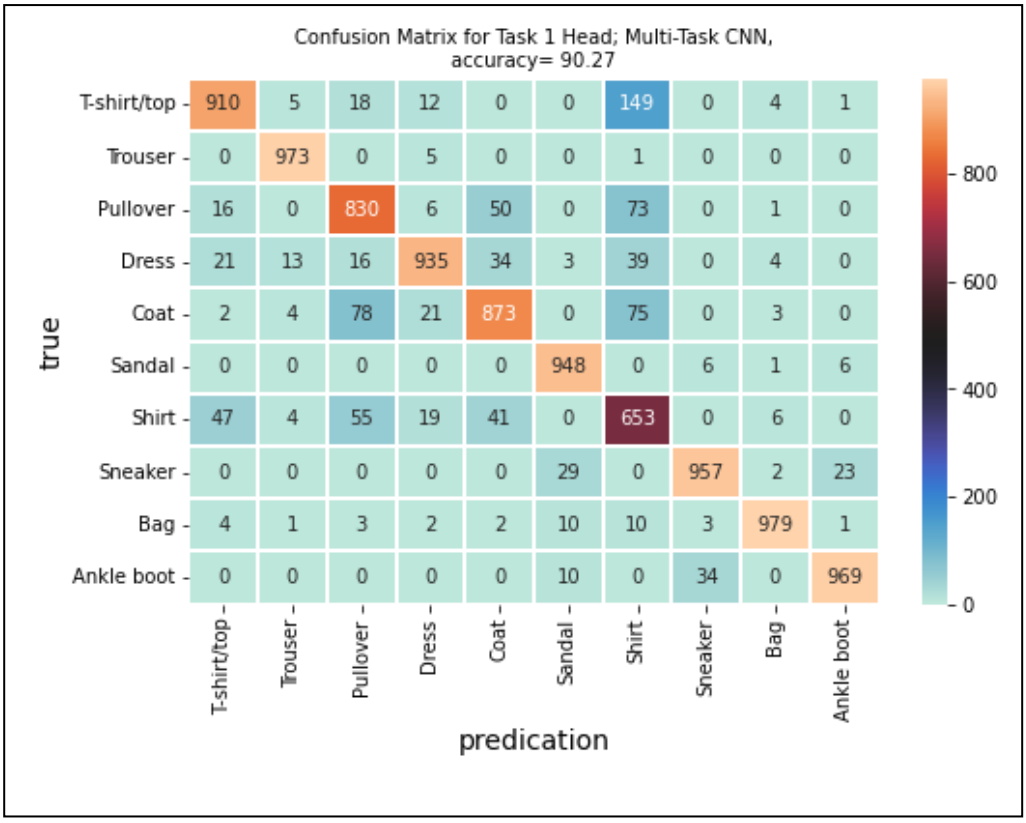


Figure 6.10

The following are essential factors to consider when using MTL:

- the synergy between different tasks to be solved by the model as they need to have some inherent correlation, (Kundu, 2023). In our example, 10-item categorization is a subcategory of the 3-category classification (i.e., related tasks).
- MTL Optimization: different tasks are required to be properly balanced to enable the model to converge to a state that is beneficial for all tasks. It is desirable to learn a balanced global task weight to avoid some tasks dominating the training process.
- Shared Network Architecture Design: it is important to develop a network of multi-task learning that exploits the shared information among the multiple tasks while maximally preserving the specific information of specific tasks.
- Task Scheduling: most MTL models decide on which tasks to train on in an epoch in a very simple way, either training on all tasks at each step or randomly sampling a subset of tasks to train on, (Kundu, 2023).

MTL offers multiple advantages like:

- Reduced inference time as multiple tasks are solved at the same time.
- Decrease the overall computational cost as several tasks are trained together instead of training each of them individually.
- Improved data efficiency
- Faster model convergence
- Reduced model overfitting due to shared representations.

However, MTL has disadvantage as task gradients may interfere, and multiple summed losses may make the optimization landscape more challenging which in turn would reduce the quality of predication. This is referred to as a negative transfer that can be reduced by using soft parameter sharing (Zhang et al., 2023). In fact, multi-task performance can suffer so much that smaller independent networks are often superior. This may be because the tasks must be learned at different rates or because one task may dominate the learning leading to poor performance on other tasks. Nevertheless, when task objectives do not interfere much with each other, performance on both tasks can be maintained or even improved when jointly trained. Intuitively, this loss or gain of quality seems to depend on the relationship between the jointly trained tasks, (Kundu, 2023).

References:

Chollet, F. (2021) *Deep learning with python*. S.l.: O'REILLY MEDIA.

Dertat, A. (2017) *Applied deep learning - part 4: Convolutional Neural Networks*, *Medium*. Towards Data Science. Available at: https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2 (Accessed: February 23, 2023).

Dinesh (2019) *CNN vs MLP for image classification*, *Medium*. Analytics Vidhya. Available at: https://medium.com/analytics-vidhya/cnn-convolutional-neural-network-8d0a292b4498#:~:text=The%20weights%20are%20smaller%20and,connected%20rather%20than%20fully%20connected. (Accessed: February 24, 2023).

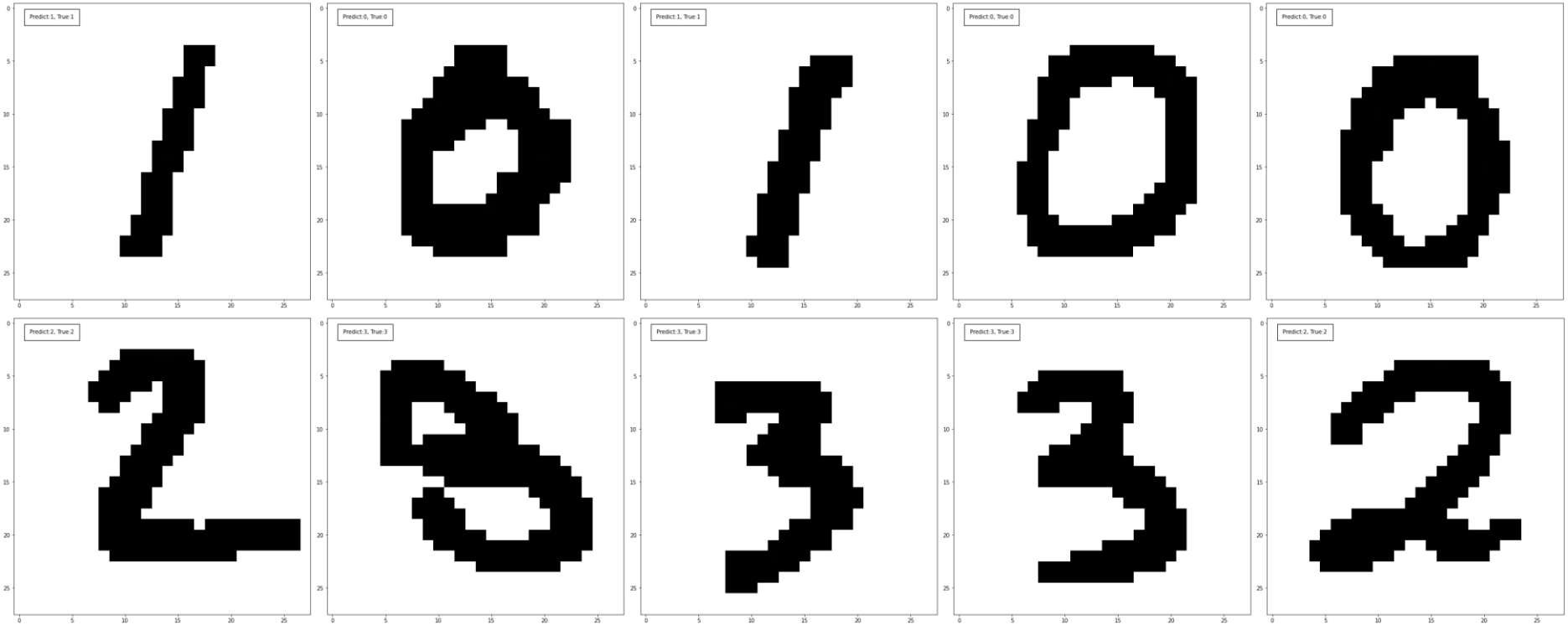
Gupta, J. (2020) *Going beyond 99% - mnist handwritten digits recognition*, *Medium*. Towards Data Science. Available at: https://towardsdatascience.com/going-beyond-99-mnist-handwritten-digits-recognition-cfff96337392#:~:text=The%20MNIST%20Handwritten%20Digits%20dataset,correctly%20classifying%20the%20handwritten%20digits. (Accessed: February 23, 2023).

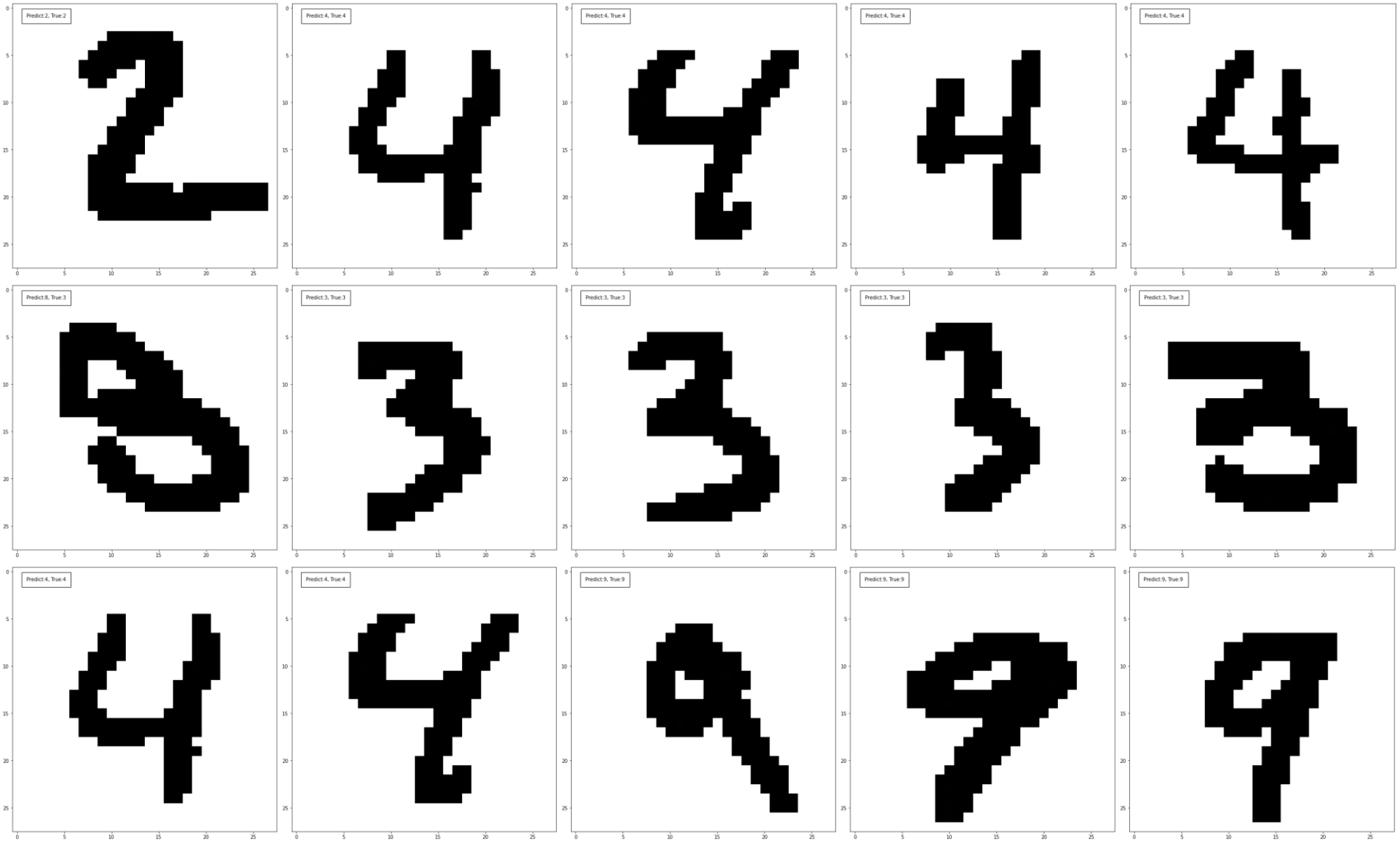
Kundu, R., 2023. *Multi-Task Learning in ML: Optimization & Use Cases [Overview]* [Online]. Available from: https://www.v7labs.com/blog/multi-task-learning-guide, https://www.v7labs.com/blog/multi-task-learning-guide [Accessed 18 February 2023].

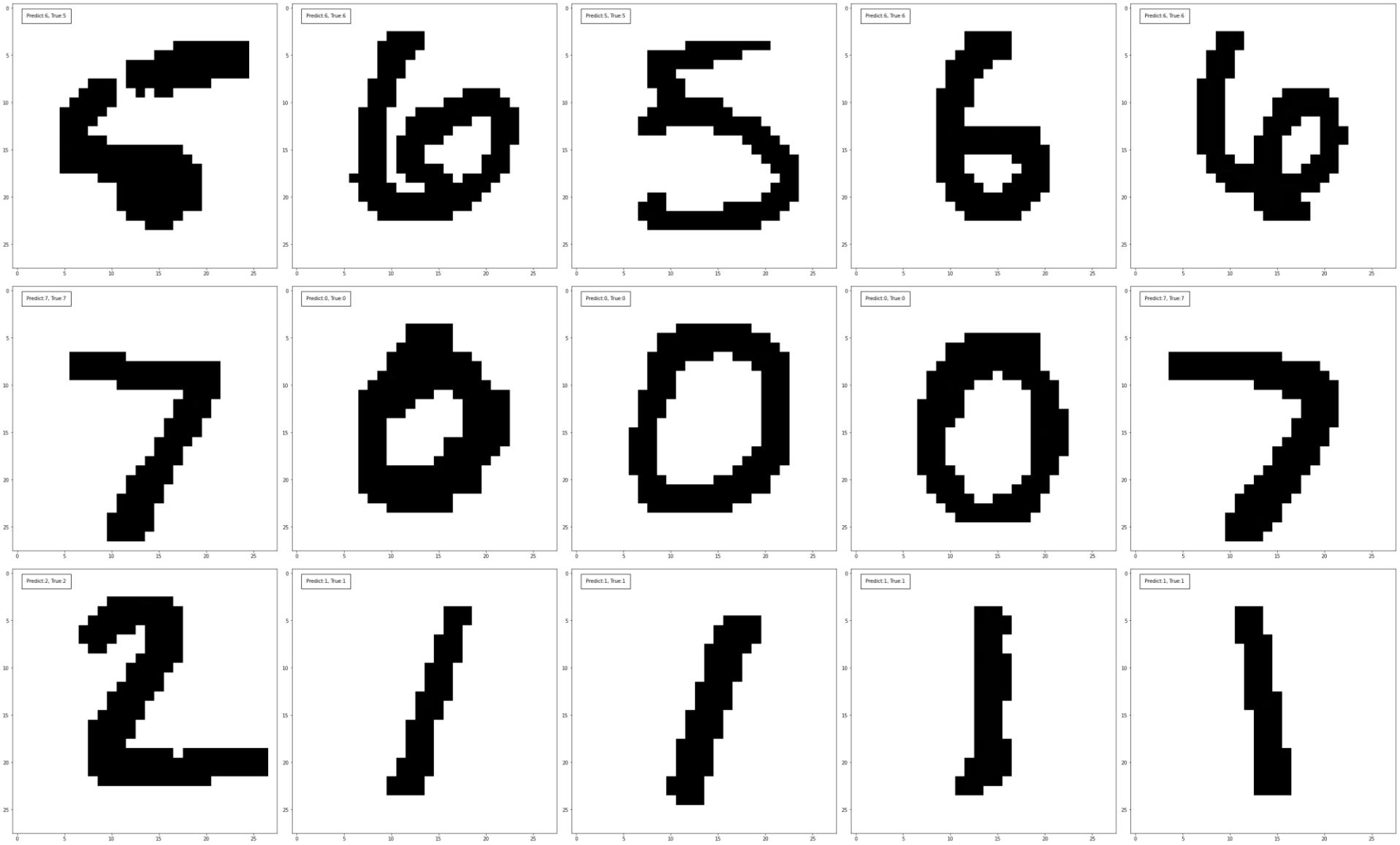
Ruder, S., 2017. *An Overview of Multi-Task Learning in Deep Neural Networks* [Online]. [Online]. Available from: http://arxiv.org/abs/1706.05098 [Accessed 18 February 2023].

Zhang, W., Deng, L., Zhang, L. and Wu, D., 2023. A Survey on Negative Transfer. *IEEE/CAA Journal of Automatica Sinica* [Online], 10(2), pp.305–329. Available from: https://doi.org/10.1109/JAS.2022.106004.

APPENDIX


Predict:1, True:1
Predict:0, True:0
Predict:1, True:1
Predict:0, True:0
Predict:0, True:0
Predict:2, True:2
Predict:3, True:3
Predict:3, True:3
Predict:3, True:3
Predict:2, True:2


Predict:2, True:2
Predict:4, True:4
Predict:4, True:4
Predict:4, True:4
Predict:4, True:4
Predict:8, True:3
Predict:3, True:3
Predict:3, True:3
Predict:3, True:3
Predict:3, True:3
Predict:4, True:4
Predict:4, True:4
Predict:9, True:9
Predict:9, True:9
Predict:9, True:9


Predict:6, True:5
Predict:6, True:6
Predict:5, True:5
Predict:6, True:6
Predict:6, True:6
Predict:7, True:7
Predict:0, True:0
Predict:0, True:0
Predict:0, True:0
Predict:7, True:7
Predict:2, True:2
Predict:1, True:1
Predict:1, True:1
Predict:1, True:1
Predict:1, True:1