Task 2

1. Part I

1.1. Step I - Preprocessing Images :

- **Channel change :** The image is firstly Converted from BGR to Grayscale to reduce the number of channels from 3 to 1. As the images are inherently Black and White, 3 channel data would be redundant and unnecessarily increase the dimensions and hence feature input into the model.
- **Cropping the Image :** Input images of the dataset have a dimension of 900 x 1200 so if we directly resize them to 28 x 28, the quality of image would have decreased and there were high chances of feature loss, in order to avoid this, roi (the part of image which contains pixels about the class to be predicted) is identified and cropped. To do so, coordinates of first and last pixel both horizontally and vertically which are black(in this case pixels containing the digits/alphabets) are found using opencv and then the image array is sliced with a gap of 20 on each side
- **Padding and Resizing :** After resizing the image a final padding of 15 width has been provided in order to aid the kernel during processing of the image, padding is added to the frame of the image to allow for more space for the kernel to cover the image.
- **Color Reversal :** The pixels containing the alphanumeric data/ classes have been provided white color and rest have been provided black,contrary to what was present in the original data.
- After final resizing, rescaling by a factor of 1/255 in order to get the final input range between 0 to 1 and reshaping an image array to four dimensions (1,28,28,1) it is ready as input into a CNN.

Before

After

1.2. Architecture of the Model

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 32)        320
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 14, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
dropout_2 (Dropout)          (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 7, 7, 64)          36928
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 4, 4, 64)          0
_________________________________________________________________
dropout_3 (Dropout)          (None, 4, 4, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               262400
_________________________________________________________________
batch_normalization_1 (Batch (None, 256)               1024
_________________________________________________________________
dropout_4 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 62)                15934
=================================================================
Total params: 335,102
Trainable params: 334,590
Non-trainable params: 512
_________________________________________________________________
```

- 3 Convolutional layers with ReLU activation function, padding as ‘same’, kernel size as 3 and filters as 32, 64, 64 have been used. Models with kernel size 5 and various combinations of filter sizes including 24,32,48,64,128 were tried to find the one that would fit the given data and bring best results.
- MaxPooling2D layers have been used to down sample the data volume with ‘same’ padding

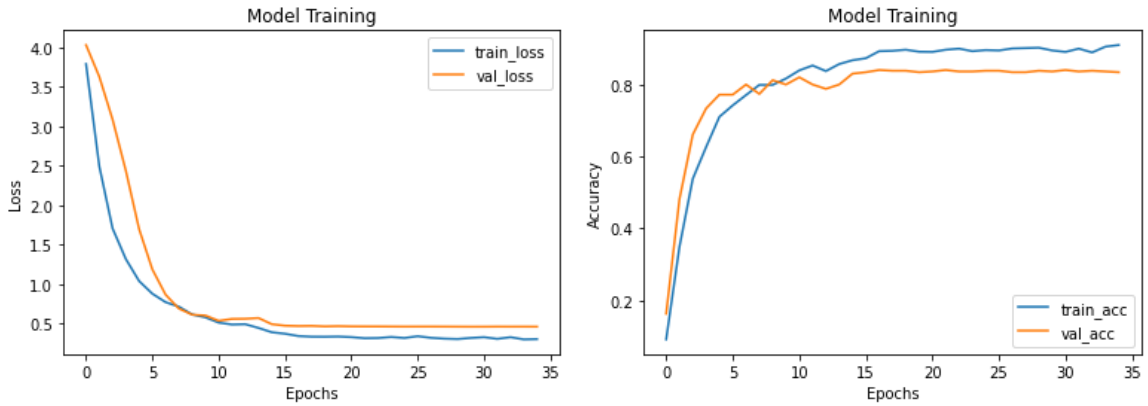
- Dropout of 0.15, 0.2 and 0.2 have been used in Dropout layer 1,2,3 respectively in order to prevent overfitting. These values were tuned on a trial and evaluate method.
- After flattening the feature maps a dense layer of 256 neurons, a batch normalization layer to stabilise the neural network, 4rth dropout layer of 0.35 and then finally a classification layer with softmax activation function with 62 neurons are connected to complete the architecture of the base model.

1.3. Training

- **Optimizer Used :** Adam, Learning rate : 1e-3, Loss : categorical_crossentropy
- **Callbacks :**
    a) Early Stopping : Monitors validation loss with patience of 5. Employed to avoid overfitting and end the training when appropriate and also restores the weights corresponding to the last best epoch.
    b) Reduce Learning Rate on Plateau : To avoid stagnant learning and before stopping the training, model benefits by reducing the learning rate by a factor of 0.1 with a patience of 3 epochs and it monitors validation accuracy in this case.
    c) Model Checkpoint : In order to save model checkpoints.
- **Batch size :** 32
- **Maximum Epochs :** 50

S.No.	Metric	BaseModel
1	Epochs Used	34
2	Training Accuracy	0.9025
3	Validation Accuracy	0.8347
4	Training Loss	0.3152
5	Validation Loss	0.4586
6	Number of Classes	62

Table 1. Results for Model 1



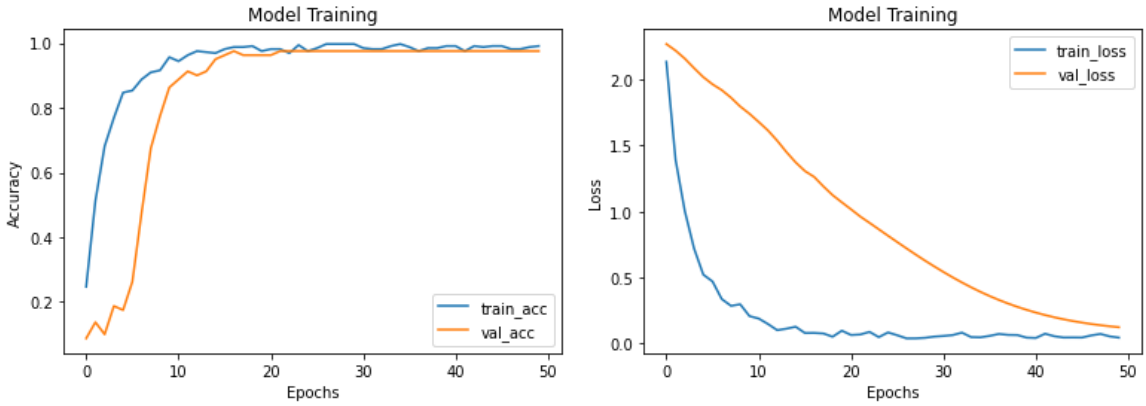
2. Part II

2.1. **Data 1 :** Mnist1 - Extracted 10 classes (0-9 digits) from the data in part I

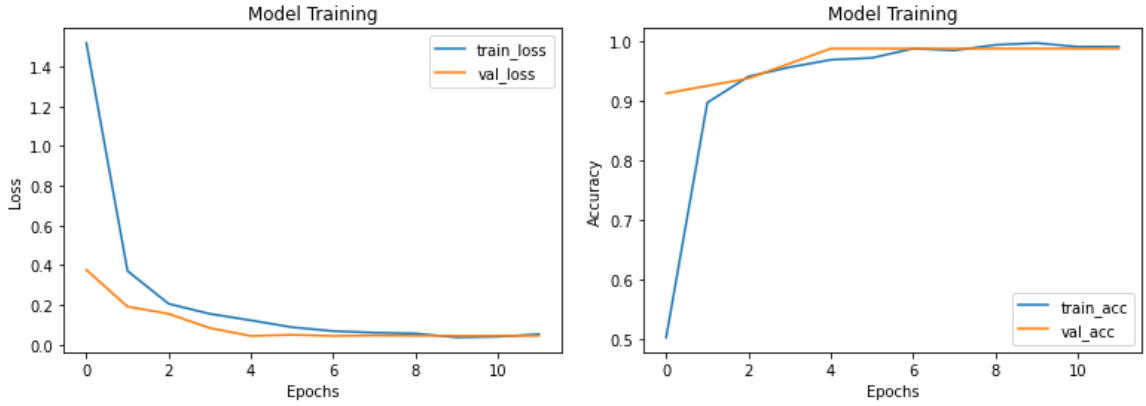
- After preprocessing the data as in Section 1.1, data is once trained on a randomly initialised neural network and then by initialising the weights using a pretrained model (using model in part I as the basemodel). Training parameters are set as in Section 1.3
- **Observations :**

S.No.	Model	Random_MNIST1	Pretrained_MNIST1
1	Epochs Used	50	12
2	Training Accuracy	0.9969	0.9973
3	Validation Accuracy	0.975	0.9875
4	Training Loss	0.0367	0.0358
5	Validation Loss	0.1236	0.044
6	Number of Classes	10	10
7	Val Loss in Epoch 1	2.26	0.37

Table 2. Comparison for Part II data 1



Randomly Initialised



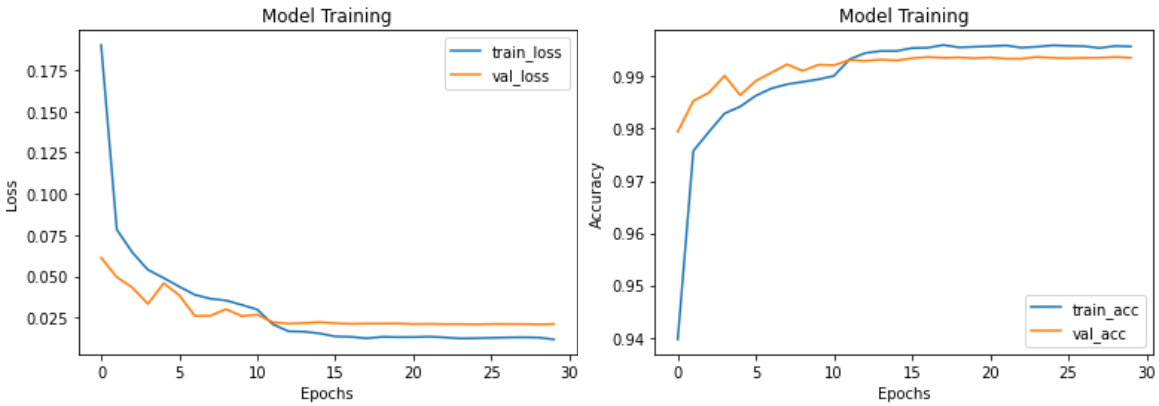
Using a pretrained model

2.2. **Data 2 :** Mnist2 - Standard MNIST data (http://yann.lecun.com/exdb/mnist/)

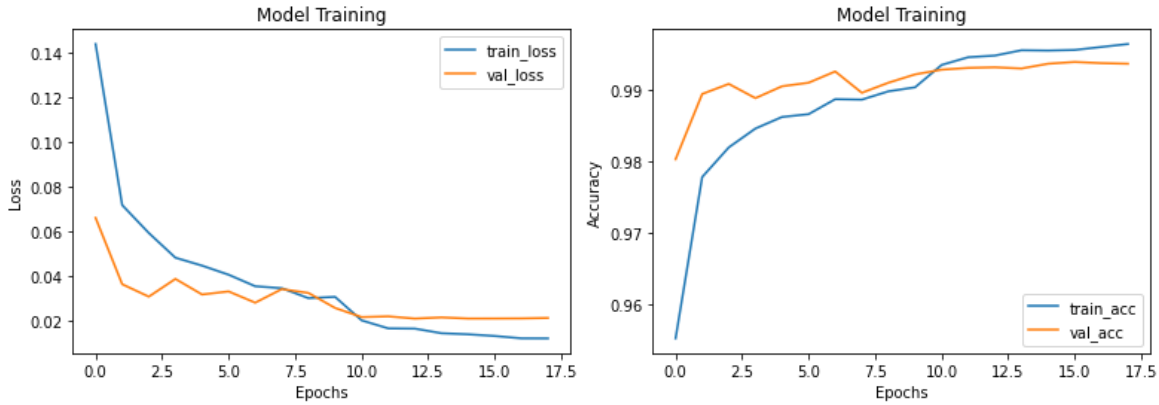
- After loading the data into array, and rescaling it is trained once on a randomly initialised neural network and then by initialising the weights using a pretrained model (using model in part I as the basemodel) and the training parameters are kept same as Section 1.3
- Observations :

S.No.	Model	Random_MNIST2	Pretrained_MNIST2
1	Epochs Used	30	18
2	Training Accuracy	0.9958	0.9961
3	Validation Accuracy	0.9936	0.9938
4	Training Loss	0.0129	0.0124
5	Validation Loss	0.0209	0.021
6	Number of Classes	10	10
7	Test Accuracy	0.9953	0.9949

Table 3. Comparison for Part II data 2



Randomly Initialised



Using a pretrained model

2.3. Inference :

- As we can see, initialising the weights of the model increases its efficiency and helps reach better results in a lesser time period. This happened because pretraining **helped to set an initial bar of weights which identified with the data well** and hence **makes it quicker and easier for the model to understand and learn the data**.
- Apart from the fact that pretrained model helps the model to learn data faster and better, it is also noteworthy that we observe similar accuracies and losses for Data 1 and Data2 whereas Data 1 has only 320 training samples whereas Data 2 has 48000.We may also infer that pretraining is helpful when we have less data as pretraining helps learn data faster and initialising the weights **helped us to overcome the shortcoming of having less data**.

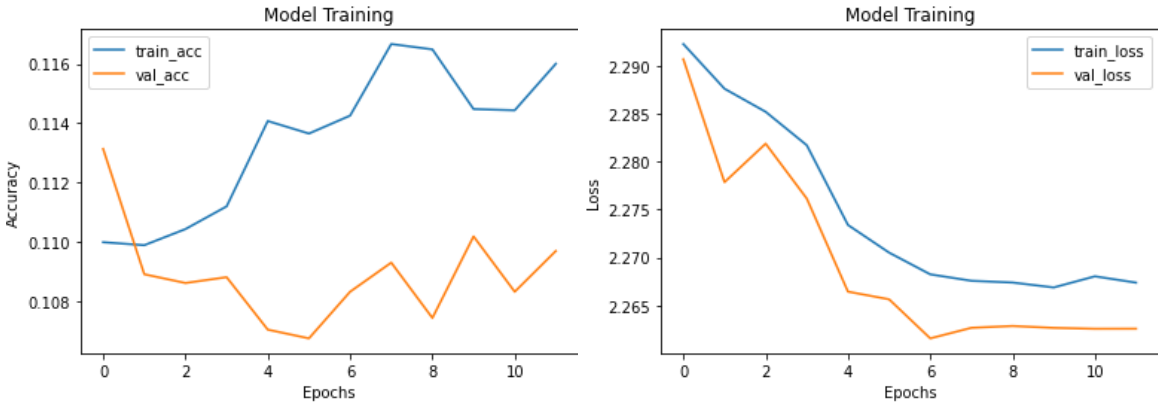
3. Part III - (Data 3 : Data provided in Part 3)

3.1. After loading the data into array, and rescaling it is trained once on a randomly initialised neural network and then by initialising the weights using a pretrained model (using model in part I as the basemodel) and the training parameters are kept same as Section 1.3

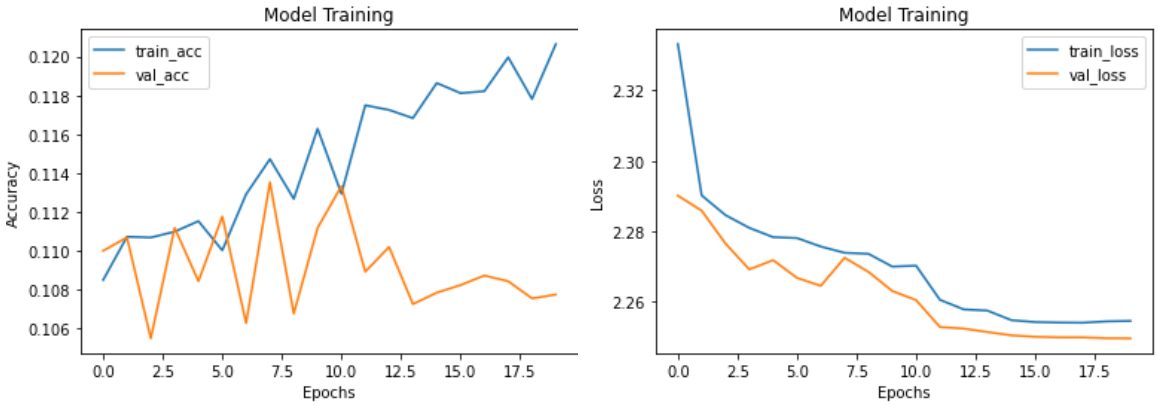
3.2. Observations :

S.No.	Model	Random_MNIST3	Pretrained_MNIST3
1	Epochs Used	20	20
2	Training Accuracy	0.116	0.1217
3	Validation Accuracy	0.1097	0.1077
4	Training Loss	2.2674	2.2542
5	Validation Loss	2.2626	2.2496
6	Number of Classes	10	10
7	Test Accuracy	1.00E-04	5.00E-04

Table 4. Comparison for Data 3



Randomly Initialised



Using a pretrained model

3.3. **Analysis :** When manually looked into the data it was found that the data is jumbled up in the sense that it labelled wrong. In the directories of each label respectively multiple instances of images corresponding to other labels are present. This is the reason that we get poor results on the data. It reflects positively about the model and the architecture that it fails to work and learn on incorrect data.

Result Summary

Result Summary								
S.No.	Model	BaseModel	Random_ MNIST1	Pretrained _MNIST1	Random_ MNIST2	Pretrained _MNIST2	Random_ MNIST3	Pretrained_ MNIST3
1	Epochs Used	34	50	12	30	18	20	20
2	Training Accuracy	0.9025	0.9969	0.9973	0.9958	0.9961	0.116	0.1217
3	Validation Accuracy	0.8347	0.975	0.9875	0.9936	0.9938	0.1097	0.1077
4	Training Loss	0.3152	0.0367	0.0358	0.0129	0.0124	2.2674	2.2542
5	Validation Loss	0.4586	0.1236	0.044	0.0209	0.021	2.2626	2.2496
6	Number of Classes	62	10	10	10	10	10	10
7	Test Accuracy	-	-	-	0.9953	0.9949	1.00E-04	5.00E-04