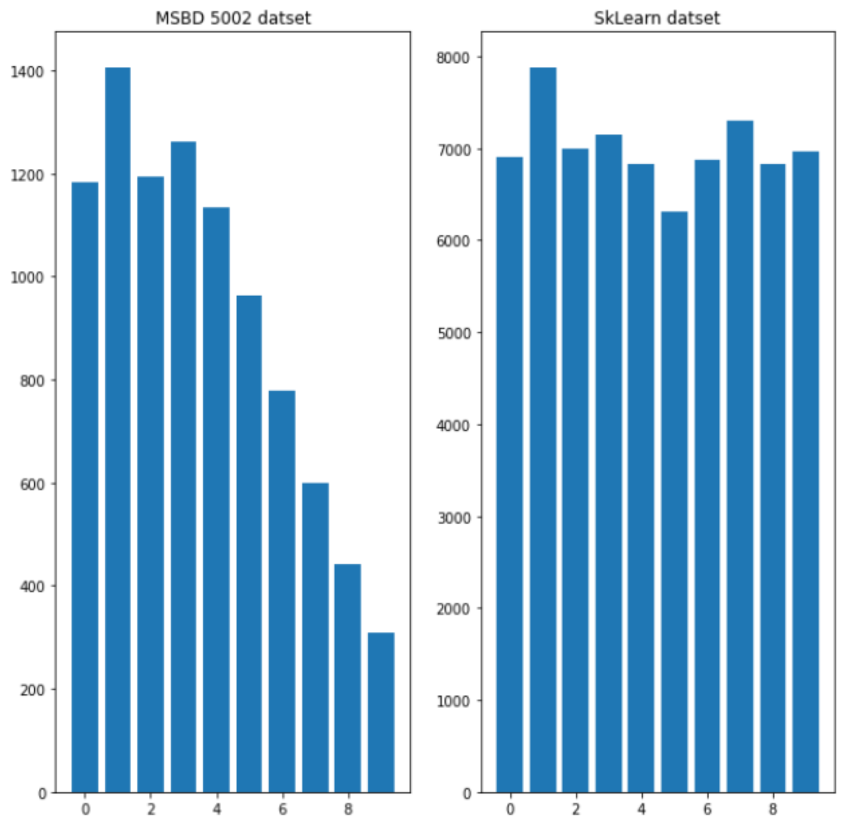
Q3 Report

Task 1:

1) The dataset is not balanced. We see a high imbalance, where there are a large number of labels of low value labels and starting from label 5, it starts to progressively decrease. To show a comparison, I have used the Sklearn MNIST dataset, where the dataset is much more balanced. Figure 1 shows a bar graph of the distribution of the datasets.

   Figure1



To overcome this imbalance, I have carried out random oversampling of the dataset as recommended in Lecture 8, Slide 98.

2) Figure 2 shows a visualization of the dataset for MSBD 5002, while Figure 3 shows a visualization for the MNIST dataset. It is clear that Data Augmentation[1] has been carried out for the dataset of MSBD 5002, while the images in MNIST dataset are normal. Data augmentation is a technique to artificially increase the variety and size of a dataset by generating many realistic variants of a training instance. In this dataset it seems that the images have been rotated as per Figure 2, and also the brightness and lighting of images. This is a good technique as the model is forced to be tolerant and more generalized to inputs without overfitting the dataset. Data Augmentation is a technique that is also used in AlexNet which won the 2012 ImageNet ILSVRC challenge.
If you used Tensorflow to do the data augmentation, the following examples of code may have been used:

```
augmentationlayer = tf.keras.models.Sequential()
#random flip
augmentationlayer.add(keras.layers.experimental.preprocessing.RandomFlip("horizon
tal_and_vertical"))
```

The above piece of code will randomly flip your image to augment it.

Figure 2

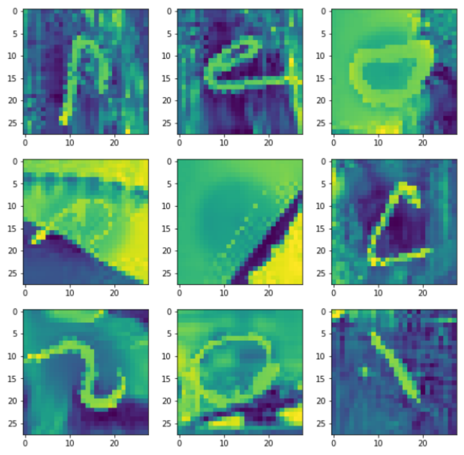
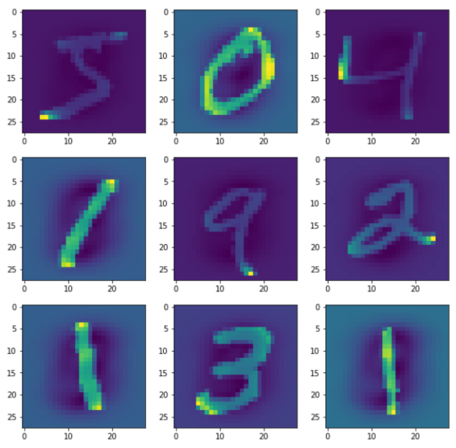


Figure 3



Task 2

For this Task I have experimented with 2 different models.

The first model is a basic neural network 4 layer neural network, with 3 hidden layers and 1 output layer. The input shape to the model is a flattened array of shape (784,1).

Model Architecture

The model consists of the following:

1) First hidden layer has 400 units with a kernel initializer of “He_normal” (this will be explained in subsequent sections)
2) This layer is followed by a Batch Normalization layer
3) Next I have used an activation layer which uses the Exponential Linear Unit as the Activation function instead of the typical Rectified Linear Unit.
4) 2 more hidden layers with identical properties by hidden units of 300 and 400 are used respectively
5) Lastly, I have an output layer of 10 units to predict 10 labels, with a softmax activation function
6) For the optimization function, I have used the Adam optimizer (will be explained in subsequent sections)

Kernel Initializers

When a neural network simply uses a mean of zero and standard deviation of 1 to initialize weights in its hidden units, the neural network is very susceptible for gradients become either smaller and smaller as it moves to lower levels during back propagation, or it may become larger. This results in the algorithm either dying out becomes there wont be weight updates as the gradient is too small or the algorithm diverges and the weight updates are too big due to large gradients.

It was proposed by Glorot and Bengio[2], that to avoid this we need to have the outputs of each layer and the gradients that are being back propagated to have the same variance as the input. Hence kernel initializers were proposed. Technically, to ensure equal variance we need the inputs to be the same as that of the number of neurons of a given layer. As a compromise it was selected to use the average of the input and output.

Glorot Normal initialization is as follows:

Mean 0 and variance of $\frac{1}{fan_{avg}}$ where fan_{avg} is equal to $(fan_{in} + fan_{out})/2$

Now the He Normalization I have used was proposed by K. He et al. in 2015 and uses a variance of $\frac{2}{fan_{in}}$. This initialize is well suited when the Activation function is using Linear unit variances such as Exponential Linear Unit and Rectified Linear uni.

Activation Functions

I have used the Exponential Linear Unit instead of ReLU as ReLU tends to die, or be dormant when the output of the linear function of a hidden unit is negative. However the Exponential linear unit tends to saturate at -1 at very negative linear function outputs, or gives a positive value when the linear function output is greater than 0. With the Exponential Linear Unit, it was noticed by Djork-Arné Clevert et al. that the training time is reduced too.

Batch Normalization layers

Just like we normalize values before inputting into a model, it would be good if we can normalize between layers of a neural network too. Batch normalization allows to add a zero-centering operation and normalizing the output from the previous hidden layer before it is passed into the activation layer. During training a running mean and running standard deviation is maintained by the model, which is then applied on the validation and test data for normalizing.

Adam Optimizer

Instead of using a normal Stochastic Gradient Descent Optimizer, I have used the Adam Optimizer. Stochastic Gradient Descent tends to be too slow. Adam Optimizer is a combination of the Momentum Optimizer and the RMSProp optimizer. With respect to the Momentum Optimization step, instead of applying the gradient directly to the weight update, it first substracts the local gradient from a momentum vector. For the RMSProp step, it squares the local gradient and obtains a scaling vector.

This is proceeded by dividing the momentum vector by the scaling vector before applying it to the weight update. This ensures that training occurs faster, and the global minima is reached faster. Also it will scale the momentum vector to ensure that learning does not slow down too fast, and will converge to a global optimum.

This mode achieves an accuracy of 70.1%

The second model consists of Convolutional Layers. Model architecture is as follows:

1) 3 convolutional layers with 20,64 and 128 filters respectively are used.
2) The final convolutional layer is connected to a pooling layer which takes the maximum of the 2x2 default rectangular receptive region
3) The result is flattened to a single array and fed into a dropout layer.
4) The result from the drop out layer is fed into a normal Dense hidden layer of 128 units.
5) Lastly there is one more batch normalization layer and dropout layer followed by a dense layer of 10 units for prediction.

Convolutional layers

In a convolutional layer, each layer is a stack of feature maps. Each neuron in a feature map is connected to the previous layer, limited to its receptive region defined by the kernel_size parameter. The receptive region is connected to all the rows and columns covered by that region and it extends across the entire stack of feature maps of the previous convolutional layer. Also the input in a convolutional layer is of shape (height,width,channel). This allows for each feature map's neuron to learn multiple complex features of the input.

Dropout layers

Dropout layers are a away of preventing overfitting of the model. It was first proposed by Hinton et al. in 2012. At every training step, we specify a drop out rate where a certain number of neurons out of each layer, including the input layer will not be used in training. It may be active in the next step however.

The simple idea behind this is to force neurons to be independent of itself and not have high reliance of out neurons in the network. This results in a more generalized model that is robust.

On the test set it provides an accuracy of 77%

To be used for testing on another test dataset, I have included the models in hdf5 format as model_one.h5 and model_cnn.h5.

The Q3_predicted_results.csv results are using model_cnn.h5.

[1] Hands On Machine Learning with Scikit Learn, Keras and Tensorflow, Aurelien Geron