CSE5ML **ASSIGNMENT 1**
NAME:HAMIM SHAIKH
STUDENTID:19901125
This is my individual work and I haven't copied this work from anyone

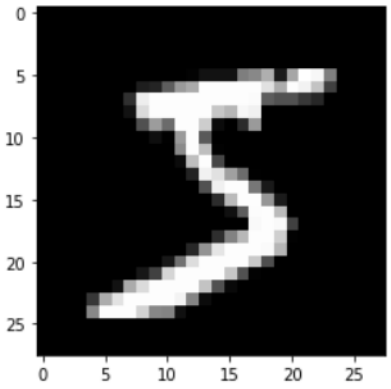
1 Objective

To build NNs for classifying handwritten digits in the MNIST database, train it on the training set and test it on the test set.Aanalyze 3 model its corresponding layers and architecture

2 Exploring dataset

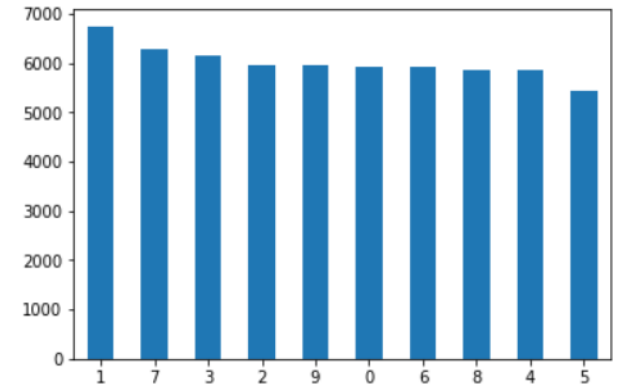
The data-set consist of images of numbers from 0 to 9 written by hand. The data-set consist of 60,000 data points of training data and 10,000 data points of testing data. Each image has size of 28×28 pixels.
Image of a number(5) in training data-set

```
plt.imshow(x_train[0],cmap=plt.cm.gray)
plt.show()
```



Before we create a CNN model.We need to explore the data.Its important to have information about the class labels.As we don't want any class imbalance to happen while modelling.

Class labels in training data-set



Based on the bar-chart shown above it is pretty clear that the majority the data points represents the class variable 1.Class variable 5 represents the least amount of data points. Further looking at the distribution of percentages of data-points within classes with image shown below it is clear that there is not much class imbalance taking place

Percentage of Class labels

	Percentage of Classes
1	11.236667
7	10.441667
3	10.218333
2	9.930000
9	9.915000
0	9.871667
6	9.863333
8	9.751667
4	9.736667
5	9.035000

3 Prepossessing

1. Reshaping the training and testing data-set data to a tensor of shape $(numsamples, imageheight, imagewidth, numc$ from $(60000, 28, 28)$ to $(60000, 28, 28, 1)$ and $(10000, 28, 28)$ to $(10000, 28, 28, 1)$.We value 1 in the reshape function indicates the color channel with $depth = 1$

2. We re-scale the image data-set from (0-255) to (0.0 to 1.0).The re-scaling of input tries to keep the the range of weights in a small range to avoid any weights occupying a large value so that our CNN model performs better. We do this by simply dividing all the set of values with 255.

3. A one hot encoding allows the representation of categorical data to be more expressive. Many machine learning algorithms cannot work with categorical data directly. The categories must be converted into numbers.

4 Analyzing Model

A Convolutional neural network is mainly used in images/object detection.The above image shows complete description of how a CNN was constructed.In order to build a "deep" neural network, we can stack several layers.We will have 3 variations of the model.Within the these variations there will be similar set of layers that will be added successively.Sequential library is imported as every neural network as this library is responsible for creating this kind of neural network.

Different layers which will be mentioned in our Convolutional neural network

1. First,layer is an input layer which will be present in every variation of our model with input shape of 28×28.However,choosing the kernel_constrain and activation function may later result in change of overall accuracy in later stages.This will also have convolutional layer which can also be added as another layer as done in all 3 models.

2. In our model we will be having a dropout layer with a specific drop out rate which deactivate random set of neurons to avoid any case of overfitting.

3. In our model we will be having 2 dimensional maxpooling layer which will be responsible for capturing objects/Images/features more easily as it will focus more on pixels with higer intensity values

4. A dense layer is just an another layer with each input node connected with output node.It takes a positive integer, dimensionality of the output space.

5. Flatten layer simply flattens multidimensional array into a simple 1 dimensional array.

6. Finally the output layer.In our case we will be having 10 output neurons as we have 10 class variables.

4.1 Model 1

Model 1

```
#Model 1
#Model with only two layers
model = Sequential()
model.add(Conv2D(32, (5, 5), activation='sigmoid', input_shape=(28, 28, 1)))
model.add(MaxPooling2D((2, 2)))
model.summary()
batch_size = 100
epochs = 5
model.add(Flatten())
model.add(Dense(10, activation='softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])

model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=0)

print("Evaluating on testing data")
test_loss, test_acc = model.evaluate(x_test, y_test)

print('Test loss:', test_loss)
print('Test accuracy:', test_acc)
```

Model 1 architecture

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 24, 24, 32)        832
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 12, 12, 32)        0
=================================================================
Total params: 832
Trainable params: 832
Non-trainable params: 0
_________________________________________________________________
Evaluating on testing data
10000/10000 [==============================] - 1s 82us/sample - loss: 0.4003 - acc: 0.8843
Test loss: 0.40027640377283097
Test accuracy: 0.8843
```

1. The learning rate given here has default value
2. The first layer has has 32 filters.The number of parameters of the CONV2D dense layers corresponds to the weight matrix W of 5×5 and a b bias for each of the filters is 832 parameters.
3. Second layer,2 dimensional maxpool layer with a window size of 2×2 doesn't require parameter since its a mathematical operation

4.2 Model 2

Model 2 Code

```
# Build the model
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(28, 28, 1), padding='same', activation='sigmoid', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Conv2D(32, (3, 3), activation='sigmoid', padding='same', kernel_constraint=maxnorm(3)))
model.add(MaxPooling2D())
model.add(Flatten())
model.add(Dense(512, activation='sigmoid', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
# Compile model
epochs = 5
lrate = 0.002
decay = lrate/epochs
sgd = SGD(lr=lrate, momentum=0.7, decay=decay, nesterov=False)
model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])
model.summary()
# Fit the model
model.fit(x_train, y_train, epochs=epochs, batch_size=60, verbose=1)
# Final evaluation of the model
scores = model.evaluate(x_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
Model: "sequential_3"
```

Model 2 architecture

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_5 (Conv2D)            (None, 28, 28, 32)        320
_________________________________________________________________
dropout_5 (Dropout)          (None, 28, 28, 32)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 28, 28, 32)        9248
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
flatten_3 (Flatten)          (None, 6272)              0
_________________________________________________________________
dense_4 (Dense)              (None, 512)               3211776
_________________________________________________________________
dropout_6 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_5 (Dense)              (None, 10)                5130
=================================================================
Total params: 3,226,474
Trainable params: 3,226,474
Non-trainable params: 0
```

1. The learning rate provided here is 0.002
2. The first layer has has 32 filters.The number of parameters of the CONV2D dense layers corresponds to the weight matrix W of 3×3 and a b bias for each of the filters is $(32 \times (9 + 1))$ which gives us 320 parameters.
3. Second layer,dropout doesn't require parameter since its a mathematical operation
4. The size of resulting second convolution layer 28×28.The value 9248 corresponds to the fact that there are 32 filters with 289 parameters
5. Similarly, at dense layer 4 there are 512 filters with 6,273 parameters.
6. Finally, the last dense layer has 10 outputs with $512 * 10 + 1$ parameters

4.3 Model 3

Model 3

```
#Best possible model
model = Sequential()
# add Convolutional layers
model.add(Conv2D(filters=32, kernel_size=(3,3), activation='relu', padding='same',
                 input_shape=(28, 28, 1)))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(filters=64, kernel_size=(3,3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(filters=64, kernel_size=(3,3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Flatten())
# Densely connected layers
model.add(Dense(128, activation='relu'))
# output layer
model.add(Dense(10, activation='softmax'))
# compile with adam optimizer & categorical_crossentropy loss function
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1
          )
print("Evaluating on testing data")
test_loss, test_acc = model.evaluate(x_test, y_test)

print('Test loss:', test_loss)
print('Test accuracy:', test_acc)
```

Model 3 architecture

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 28, 28, 32)        320
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 14, 14, 64)        18496
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 7, 7, 64)          36928
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 3, 3, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 576)               0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               73856
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290
=================================================================
Total params: 130,890
Trainable params: 130,890
Non-trainable params: 0
```

1. The learning rate provided here is default value of 0.001
2. The first layer has has 32 filters.The number of parameters of the CONV2D dense layers corresponds to the weight matrix W of 3×3 and a b bias for each of the filters is $(32 \times (9 + 1))$ which gives us 320 parameters.
3. There are total of 9 layers including input and output layer
4. The value of all the parameter is calculated in similar as shown in model 1.
5. This model is much more deeper(more hidden layers) than the model 1.Hence it may help to fit the data well.

5 Evaluation of the 3 models

5.1 Model 1

Model 1 has mainly uses sigmoid as an activation function.The depth of the model is just two layers with an accuracy of 88.4% on test data. The optimizer used here is Stochastic gradient descent with a batch size of 100. Overall accuracy may increase in models 2 by increasing the depth neural network.Also its possible that the global minima is not obtained due vanishing gradient problem because of sigmoid function,hence another possible way to increase the accuracy is by changing the activation function.
Model 1 accuracy

```
_________________________________________________________________
Evaluating on testing data
10000/10000 [==============================] - 1s 82us/sample - loss: 0.4003 - acc: 0.8843
Test loss: 0.40027640377283097
Test accuracy: 0.8843
```

5.2 Model 2

Based on previous experience the depth of the model has been increased however we din't get the expected results indicating that we must change the activation function one way or another to have any improvement whatsoever since accuracy here is as mere as $just11\%$
This is a typical case of underfitting in which model has fails to acquire the feaures from the training data.
Model 2 accuracy

```
Epoch 1/5
60000/60000 [==============================] - 114s 2ms/step - loss: 2.4027 - accuracy: 0.0996
Epoch 2/5
60000/60000 [==============================] - 114s 2ms/step - loss: 2.3245 - accuracy: 0.1025
Epoch 3/5
60000/60000 [==============================] - 114s 2ms/step - loss: 2.3128 - accuracy: 0.1032
Epoch 4/5
60000/60000 [==============================] - 114s 2ms/step - loss: 2.3090 - accuracy: 0.1059
Epoch 5/5
60000/60000 [==============================] - 118s 2ms/step - loss: 2.3075 - accuracy: 0.1052
Evaluating on testing data
10000/10000 [==============================] - 4s 353us/step
Test loss: 2.3008178928375242
Test accuracy: 0.11349999904632568
```

5.3 Model 3

Finally, a suitable model was achieved after trying out number of iterations on model hyperparameters.The relu activation function and corresponding adding of more hidden layers helped to get the best accuracy.The combination of adam optimizer which in our case helped to be best way reducing loss at every step of training process and the relu function which helped us to acieve the global optima.
The overall accuracy attained her was 99% on the test data.Further the evaluation on training data shows similar results. **Model 3 accuracy**

```
Train on 60000 samples
Epoch 1/5
60000/60000 [==============================] - 35s 588us/sample - loss: 0.2085 - acc: 0.9365
Epoch 2/5
60000/60000 [==============================] - 40s 672us/sample - loss: 0.0521 - acc: 0.9835
Epoch 3/5
60000/60000 [==============================] - 38s 638us/sample - loss: 0.0353 - acc: 0.9892
Epoch 4/5
60000/60000 [==============================] - 37s 625us/sample - loss: 0.0280 - acc: 0.9911
Epoch 5/5
60000/60000 [==============================] - 39s 645us/sample - loss: 0.0222 - acc: 0.9931
Evaluating on testing data
10000/10000 [==============================] - 2s 190us/sample - loss: 0.0303 - acc: 0.9902
Test loss: 0.030344770431693178
Test accuracy: 0.9902
```

6 Conclusion

After detailed analysis and performing number of iteration by changing different hyperparameters Model 3 outperformed Model 1 and Model 2.Further using the relu activation function , adam optimizer and increasing the number of layers helped to achieve an accuracy of 99%