AMERICAN INTERNATIONAL UNIVERSITY BANGADESH

Shifat, Sirajum Munira

22-92314-1

MScCS (A)

Spring 2021-2022

Date: 13 March 2022

MID ASSESSMENT

COMPUTER VISION & PATTERN RECOGNITION

Course Teacher
DEBAJYOTI KARMAKER

Title: CNN Model for CIFAR 10 Dataset

The main goal of this CNN model for CIFAR 10 dataset is to achieve possible higher accuracy. The whole training process is explained below.
At first, I have imported necessary libraries and then downloaded the CIFAR10 dataset using "keras". Then using loop printed the images of that dataset. There are 10 different classes of images in this dataset.

```
[ ] import tensorflow as tf
    from tensorflow import keras
    import numpy as np
    import matplotlib.pyplot as plt
```

```
[ ] (train_images, train_labels), (test_images, test_labels) = keras.datasets.cifar10.load_data()
    print(train_images.shape)
    print(train_labels.shape)
    print(test_images.shape)
    print(test_labels.shape)
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 2s 0us/step
170508288/170498071 [==============================] - 2s 0us/step
(50000, 32, 32, 3)
(50000, 1)
(10000, 32, 32, 3)
(10000, 1)
```

```
[ ] categories = ["airplane", "automobile", "bird", "cat", "deer", "dog", "frog", "horse", "ship", "truck"]
```

```
[ ] plt.figure(figsize = (20,20), dpi =120 )
    for i in range(100):
      plt.subplot(10, 10, i+1)
      plt.imshow(train_images[i])
      plt.xticks([])
      plt.yticks([])
      plt.xlabel(categories[train_labels[i][0]])
    plt.show()
```

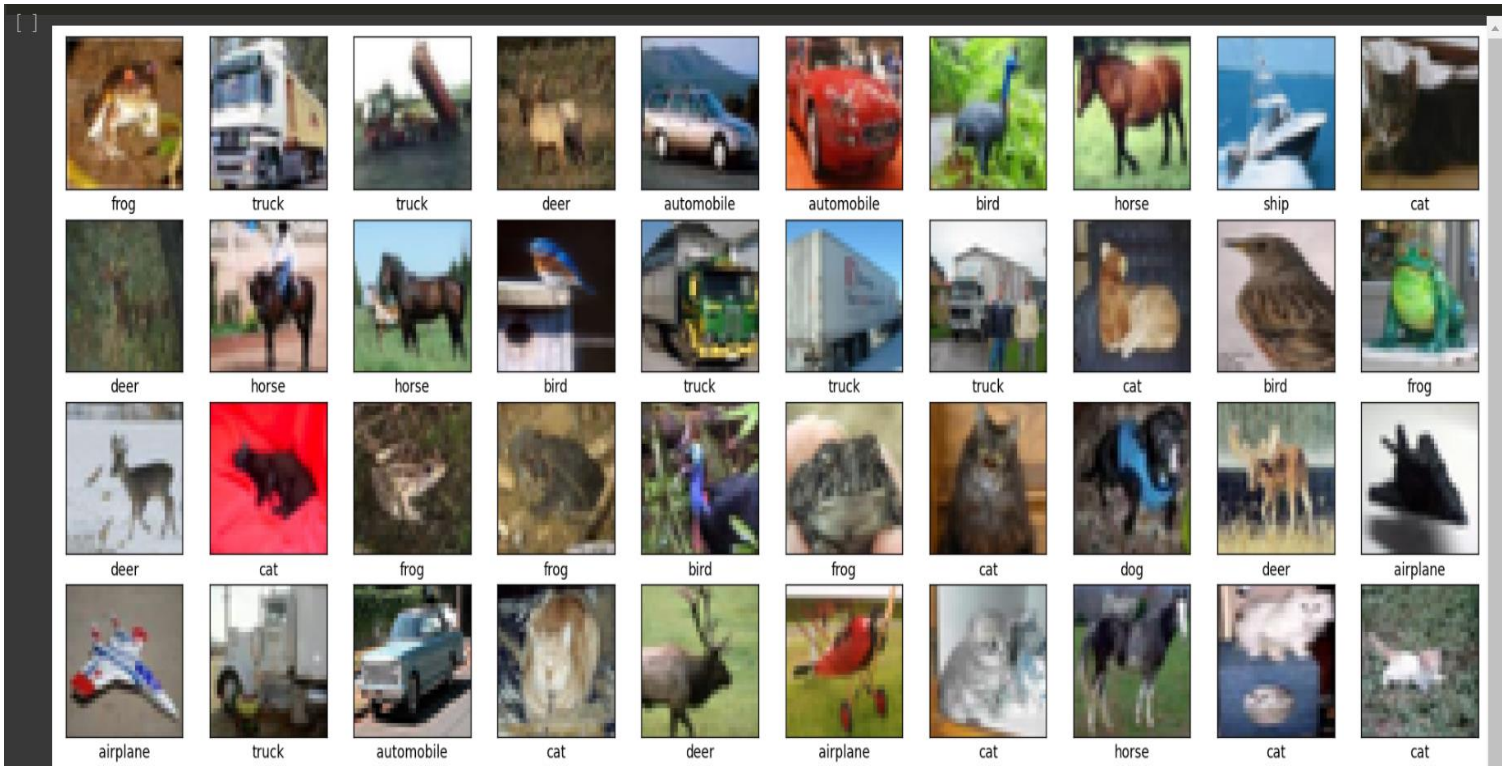


Fig: Images of the dataset

As the images are RGB, so the range of array for each image is 0 to 255. For training purpose, I normalized the images' array in range between 0 to 1.

```
train_images_norm = train_images.astype('float32') / 255
test_images_norm = test_images.astype('float32') /255
```

I have tried to build three model for the dataset. And then compare those model to check which model has given the accurate output.

First Model:

First Model

```python
model = keras.Sequential([
    keras.Input(shape = (32, 32, 3)),
    keras.layers.Conv2D(filters = 32, kernel_size = (3, 3), padding='same', activation = "relu"),
    keras.layers.Conv2D(filters = 32, kernel_size = (3, 3), padding='same',activation = "relu"),
    keras.layers.MaxPooling2D(pool_size=(2, 2)),
    keras.layers.Conv2D(filters = 64, kernel_size = (3, 3), padding='same', activation = "relu"),
    keras.layers.Conv2D(filters = 64, kernel_size = (3, 3), padding='same', activation = "relu"),
    keras.layers.MaxPooling2D(pool_size=(2, 2)),
    keras.layers.Conv2D(filters = 128, kernel_size = (3, 3), padding='same', activation = "relu"),
    keras.layers.Conv2D(filters = 128, kernel_size = (3, 3), padding='same', activation = "relu"),
    keras.layers.MaxPooling2D(pool_size=(2, 2)),
    keras.layers.Flatten(),
    keras.layers.Dense(units = 128, activation = "relu"),
    keras.layers.Dense(units = 10, activation = "softmax")
])

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 32)        896

 conv2d_1 (Conv2D)           (None, 32, 32, 32)        9248

 max_pooling2d (MaxPooling2D  (None, 16, 16, 32)       0
 )

 conv2d_2 (Conv2D)           (None, 16, 16, 64)        18496

 conv2d_3 (Conv2D)           (None, 16, 16, 64)        36928

 max_pooling2d_1 (MaxPooling  (None, 8, 8, 64)         0
 2D)

 conv2d_4 (Conv2D)           (None, 8, 8, 128)         73856

 conv2d_5 (Conv2D)           (None, 8, 8, 128)         147584

 max_pooling2d_2 (MaxPooling  (None, 4, 4, 128)        0
 2D)

 flatten (Flatten)           (None, 2048)              0

 dense (Dense)               (None, 128)               262272

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 550,570
Trainable params: 550,570
Non-trainable params: 0
_________________________________________________________________
```

For the first model, I have used 6 Convo2d layers with filers of 32, 64 and 128, the kernel size was 3x3. I kept the padding same and used Relu activation function. I also used 2 Maxpooling layers, 1 Flatten and 2 dense layers with the units of 128 and 10.

After building the model, I used three different optimizers with 0.001 learning rate and three different losses each time to compile the model using 200 epochs, 128 batch size and 30% validation split.

Optimizer: Adam

Loss: spare categorical crossentropy

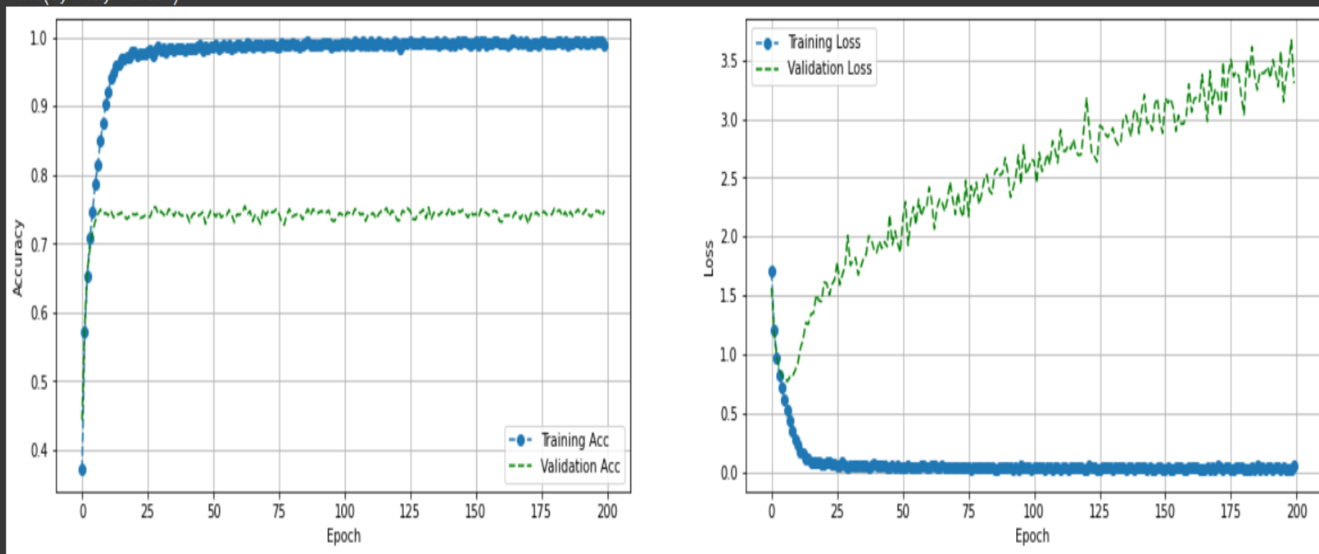
```
model.compile(
    optimizer = keras.optimizers.Adam(learning_rate=0.001),
    loss = keras.losses.sparse_categorical_crossentropy,
    metrics = ['accuracy']
)
```

```
h1 = model.fit(x = train_images_norm, y = train_labels, epochs = 200, batch_size = 128, validation_split = 0.3)
```

```
274/274 [==============================] - 10s 35ms/step - loss: 0.0285 - accuracy: 0.9924 - val_loss: 3.3994 - val_accuracy: 0.7464
Epoch 188/200
274/274 [==============================] - 10s 35ms/step - loss: 0.0349 - accuracy: 0.9917 - val_loss: 3.3863 - val_accuracy: 0.7398
Epoch 189/200
274/274 [==============================] - 11s 40ms/step - loss: 0.0268 - accuracy: 0.9931 - val_loss: 3.3986 - val_accuracy: 0.7454
Epoch 190/200
274/274 [==============================] - 10s 35ms/step - loss: 0.0303 - accuracy: 0.9929 - val_loss: 3.4445 - val_accuracy: 0.7495
Epoch 191/200
274/274 [==============================] - 10s 35ms/step - loss: 0.0322 - accuracy: 0.9915 - val_loss: 3.3462 - val_accuracy: 0.7426
Epoch 192/200
274/274 [==============================] - 10s 35ms/step - loss: 0.0246 - accuracy: 0.9940 - val_loss: 3.5019 - val_accuracy: 0.7436
Epoch 193/200
274/274 [==============================] - 10s 36ms/step - loss: 0.0312 - accuracy: 0.9925 - val_loss: 3.3980 - val_accuracy: 0.7499
Epoch 194/200
274/274 [==============================] - 10s 35ms/step - loss: 0.0313 - accuracy: 0.9923 - val_loss: 3.2760 - val_accuracy: 0.7419
Epoch 195/200
274/274 [==============================] - 10s 35ms/step - loss: 0.0203 - accuracy: 0.9944 - val_loss: 3.5793 - val_accuracy: 0.7391
Epoch 196/200
274/274 [==============================] - 10s 35ms/step - loss: 0.0394 - accuracy: 0.9906 - val_loss: 3.1457 - val_accuracy: 0.7459
Epoch 197/200
274/274 [==============================] - 10s 35ms/step - loss: 0.0247 - accuracy: 0.9935 - val_loss: 3.3616 - val_accuracy: 0.7475
Epoch 198/200
274/274 [==============================] - 10s 35ms/step - loss: 0.0219 - accuracy: 0.9939 - val_loss: 3.4663 - val_accuracy: 0.7465
Epoch 199/200
274/274 [==============================] - 11s 40ms/step - loss: 0.0245 - accuracy: 0.9937 - val_loss: 3.6697 - val_accuracy: 0.7401
Epoch 200/200
274/274 [==============================] - 10s 36ms/step - loss: 0.0512 - accuracy: 0.9881 - val_loss: 3.3029 - val_accuracy: 0.7478
```

Here I achieved around 98.81% training accuracy with approximately 5% loss, but the validation difference between training accuracy and validation accuracy was high from which it can be

```
Text(0, 0.5, 'Loss')
```



decided that this model is not accurate. Tough I have tried other two optimizer and loss function too observe the training.

Optimizer: RMSprop
Loss: categorical hinge

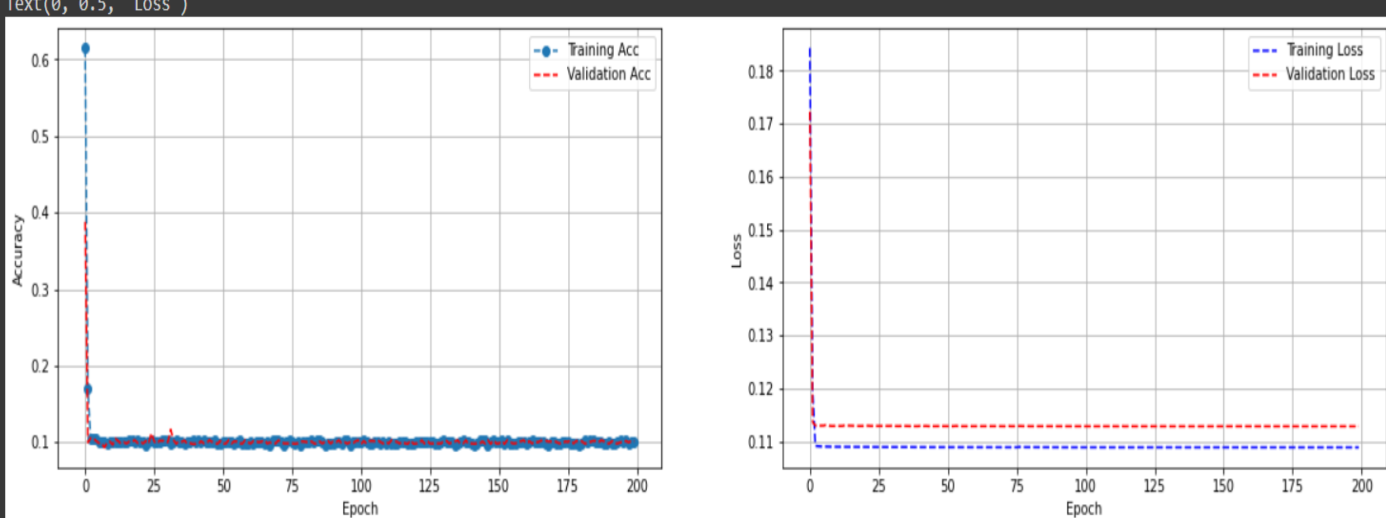
```
[7] model.compile(
        optimizer = keras.optimizers.RMSprop(learning_rate=0.001),
        loss = keras.losses.categorical_hinge,
        metrics = ['accuracy']
    )

[ ] h2 = model.fit(x = train_images_norm, y = train_labels, epochs = 200, batch_size = 128, validation_split = 0.3)
```

```
274/274 [============================] - 10s 38ms/step - loss: 0.1088 - accuracy: 0.1013 - val_loss: 0.1128 - val_accuracy: 0.1017
Epoch 187/200
274/274 [============================] - 10s 38ms/step - loss: 0.1088 - accuracy: 0.1001 - val_loss: 0.1128 - val_accuracy: 0.0999
Epoch 188/200
274/274 [============================] - 10s 38ms/step - loss: 0.1088 - accuracy: 0.1002 - val_loss: 0.1128 - val_accuracy: 0.1025
Epoch 189/200
274/274 [============================] - 10s 38ms/step - loss: 0.1088 - accuracy: 0.0976 - val_loss: 0.1128 - val_accuracy: 0.1014
Epoch 190/200
274/274 [============================] - 10s 38ms/step - loss: 0.1088 - accuracy: 0.0977 - val_loss: 0.1128 - val_accuracy: 0.0984
Epoch 191/200
274/274 [============================] - 10s 38ms/step - loss: 0.1088 - accuracy: 0.1011 - val_loss: 0.1128 - val_accuracy: 0.0983
Epoch 192/200
274/274 [============================] - 10s 38ms/step - loss: 0.1088 - accuracy: 0.0993 - val_loss: 0.1128 - val_accuracy: 0.1017
Epoch 193/200
274/274 [============================] - 10s 38ms/step - loss: 0.1088 - accuracy: 0.0991 - val_loss: 0.1128 - val_accuracy: 0.0983
Epoch 194/200
274/274 [============================] - 10s 38ms/step - loss: 0.1088 - accuracy: 0.1013 - val_loss: 0.1128 - val_accuracy: 0.0984
Epoch 195/200
274/274 [============================] - 10s 38ms/step - loss: 0.1088 - accuracy: 0.1023 - val_loss: 0.1128 - val_accuracy: 0.0999
Epoch 196/200
274/274 [============================] - 10s 38ms/step - loss: 0.1088 - accuracy: 0.1000 - val_loss: 0.1128 - val_accuracy: 0.0999
Epoch 197/200
274/274 [============================] - 10s 38ms/step - loss: 0.1088 - accuracy: 0.1019 - val_loss: 0.1128 - val_accuracy: 0.1017
Epoch 198/200
274/274 [============================] - 10s 38ms/step - loss: 0.1088 - accuracy: 0.0985 - val_loss: 0.1128 - val_accuracy: 0.0999
Epoch 199/200
274/274 [============================] - 10s 38ms/step - loss: 0.1088 - accuracy: 0.1006 - val_loss: 0.1128 - val_accuracy: 0.1017
Epoch 200/200
274/274 [============================] - 10s 38ms/step - loss: 0.1088 - accuracy: 0.0997 - val_loss: 0.1128 - val_accuracy: 0.1014
```

In this second history the training accuracy was 0.0997 and validation accuracy was 0.1014 where the difference was not so huge but the accuracy was very poor. And also, the losses were pretty much close.

```
Text(0, 0.5, 'Loss')
```



Optimizer: SGD

Loss: mean squared logarithmic error

```
[16] model.compile(
        optimizer = keras.optimizers.SGD(learning_rate=0.001),
        loss = keras.losses.mean_squared_logarithmic_error ,
        metrics = ['accuracy']
    )
```

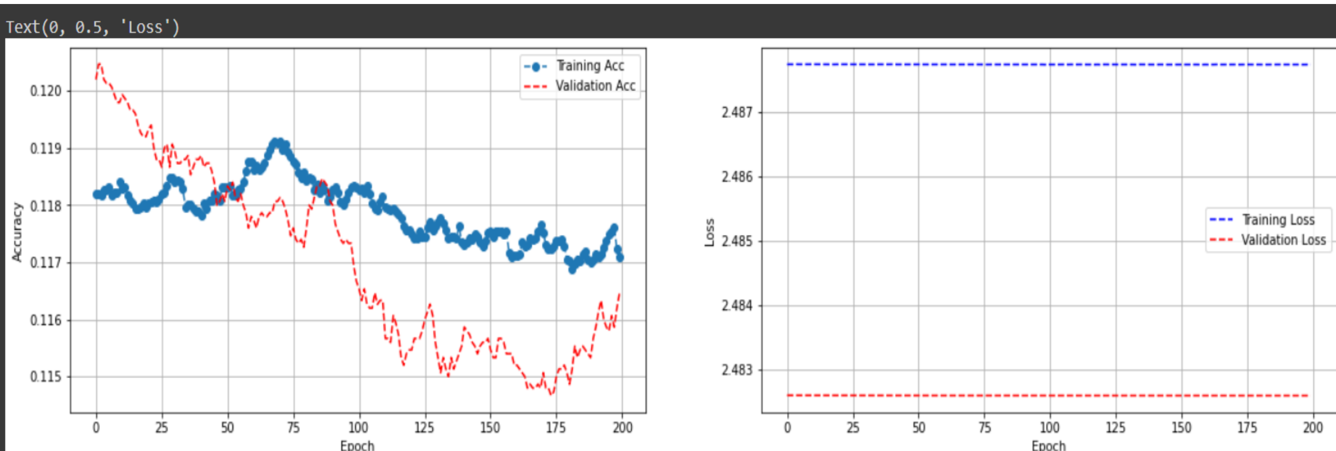
```
[17] h3 = model.fit(x = train_images_norm, y = train_labels, epochs = 200, batch_size = 128, validation_split = 0.3)
```

```
Epoch 196/200
274/274 [==============================] - 12s 44ms/step - loss: 2.4877 - accuracy: 0.1175 - val_loss: 2.4826 - val_accuracy: 0.1158
Epoch 197/200
274/274 [==============================] - 12s 44ms/step - loss: 2.4877 - accuracy: 0.1175 - val_loss: 2.4826 - val_accuracy: 0.1161
Epoch 198/200
274/274 [==============================] - 12s 45ms/step - loss: 2.4877 - accuracy: 0.1176 - val_loss: 2.4826 - val_accuracy: 0.1159
Epoch 199/200
274/274 [==============================] - 12s 45ms/step - loss: 2.4877 - accuracy: 0.1172 - val_loss: 2.4826 - val_accuracy: 0.1162
Epoch 200/200
274/274 [==============================] - 12s 44ms/step - loss: 2.4877 - accuracy: 0.1171 - val_loss: 2.4826 - val_accuracy: 0.1165
```

```
[18] model.evaluate(test_images_norm,test_labels)

313/313 [==============================] - 3s 8ms/step - loss: 2.4862 - accuracy: 0.1123
[2.4861888885498047, 0.11230000108480453]
```

```
Text(0, 0.5, 'Loss')
```



In third history training accuracy and validation accuracy were pretty much close which were accordingly 0.1171 and 0.1165. The losses also close but huge.

Second Model:

Second Model

```python
model = keras.Sequential([
    keras.Input(shape = (32, 32, 3)),
    keras.layers.Conv2D(filters = 32, kernel_size = (3, 3), padding='same', activation = "relu"),
    keras.layers.Dropout(0.5),
    keras.layers.Conv2D(filters = 32, kernel_size = (3, 3), padding='same',activation = "relu"),
    keras.layers.MaxPooling2D(pool_size=(2, 2)),
    keras.layers.Conv2D(filters = 64, kernel_size = (3, 3), padding='same', activation = "relu"),
    keras.layers.Dropout(0.5),
    keras.layers.Conv2D(filters = 64, kernel_size = (3, 3), padding='same', activation = "relu"),
    keras.layers.MaxPooling2D(pool_size=(2, 2)),
    keras.layers.Flatten(),
    keras.layers.Dense(units = 128, activation = "relu"),
    keras.layers.Dense(units = 10, activation = "softmax")
])

model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_8 (Conv2D)           (None, 32, 32, 32)        896

 dropout_4 (Dropout)         (None, 32, 32, 32)        0

 conv2d_9 (Conv2D)           (None, 32, 32, 32)        9248

 max_pooling2d_4 (MaxPooling  (None, 16, 16, 32)       0
 2D)

 conv2d_10 (Conv2D)          (None, 16, 16, 64)        18496

 dropout_5 (Dropout)         (None, 16, 16, 64)        0

 conv2d_11 (Conv2D)          (None, 16, 16, 64)        36928

 max_pooling2d_5 (MaxPooling  (None, 8, 8, 64)         0
 2D)

 flatten_2 (Flatten)         (None, 4096)              0

 dense_4 (Dense)             (None, 128)               524416

 dense_5 (Dense)             (None, 10)                1290

=================================================================
Total params: 591,274
Trainable params: 591,274
Non-trainable params: 0
_________________________________________________________________
```

For the second model, I have used 4 Convo2d layers with filers of 32, 64 and 128, the kernel size was 3x3. I kept the padding same and used Relu activation function. I also used 2 Maxpooling layers, 1 Flatten and 2 dense layers with the units of 128 and 10. Here I also used 50% dropout 2 times to avoid the overfitting issue.

After building the model, I used three different optimizers with 0.001 learning rate and three different losses each time to compile the model using 50 epochs, 20 batch size and 30% validation split.

Optimizer: Adam

Loss: spare categorical crossentropy

```
[13] model.compile(
         optimizer = keras.optimizers.Adam(learning_rate=0.001),
         loss = keras.losses.sparse_categorical_crossentropy,
         metrics = ['accuracy']
     )

[14] h1 = model.fit(x = train_images_norm, y = train_labels, epochs = 50, batch_size = 20, validation_split = 0.3)
```

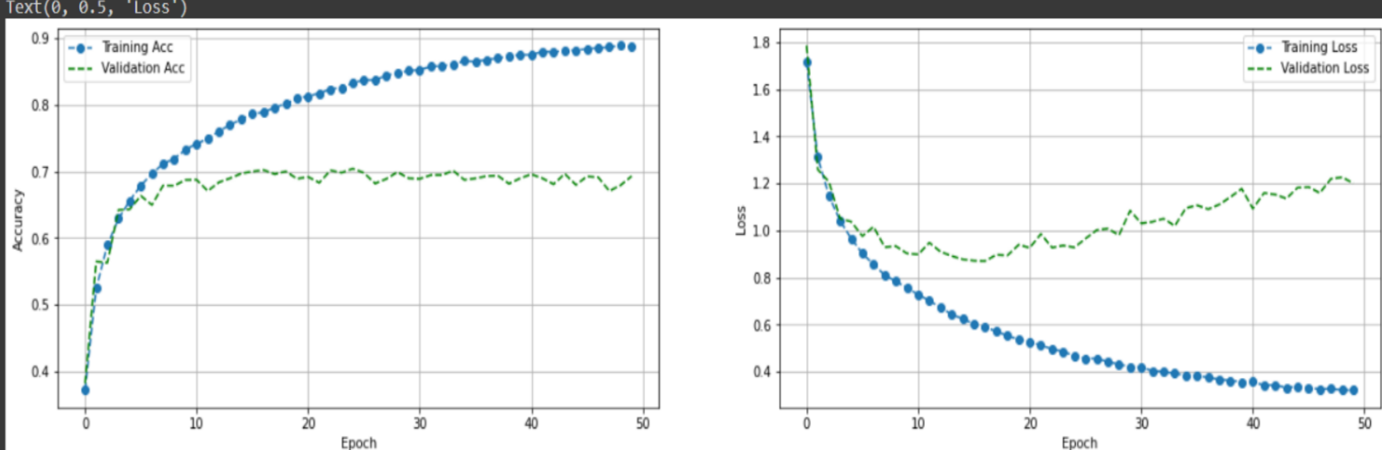
```
1750/1750 [==============================] - 29s 16ms/step - loss: 0.3357 - accuracy: 0.8823 - val_loss: 1.1812 - val_accuracy: 0.6799
Epoch 46/50
1750/1750 [==============================] - 29s 16ms/step - loss: 0.3313 - accuracy: 0.8842 - val_loss: 1.1844 - val_accuracy: 0.6927
Epoch 47/50
1750/1750 [==============================] - 29s 16ms/step - loss: 0.3250 - accuracy: 0.8858 - val_loss: 1.1591 - val_accuracy: 0.6913
Epoch 48/50
1750/1750 [==============================] - 27s 15ms/step - loss: 0.3274 - accuracy: 0.8871 - val_loss: 1.2193 - val_accuracy: 0.6707
Epoch 49/50
1750/1750 [==============================] - 29s 16ms/step - loss: 0.3208 - accuracy: 0.8894 - val_loss: 1.2259 - val_accuracy: 0.6794
Epoch 50/50
1750/1750 [==============================] - 26s 15ms/step - loss: 0.3204 - accuracy: 0.8873 - val_loss: 1.2003 - val_accuracy: 0.6931

[15] test_accuracy = model.evaluate(test_images_norm,test_labels)

313/313 [==============================] - 2s 7ms/step - loss: 1.2315 - accuracy: 0.6840
```

Here I achieved training accuracy around 88.73% and validation accuracy around 69.31%. Also, I observed the test accuracy which was around 68.40% that was close to validation accuracy. Both

```
Text(0, 0.5, 'Loss')
```



training and validation losses had huge difference. But validation loss was almost close to test loss.

Optimizer: RMSprop
Loss: categorical hinge

```
[17] model.compile(
         optimizer = keras.optimizers.RMSprop(learning_rate=0.001),
         loss = keras.losses.categorical_hinge,
         metrics = ['accuracy']
     )
```

```
[18] h2 = model.fit(x = train_images_norm, y = train_labels, epochs = 50, batch_size = 20, validation_split = 0.3)
```

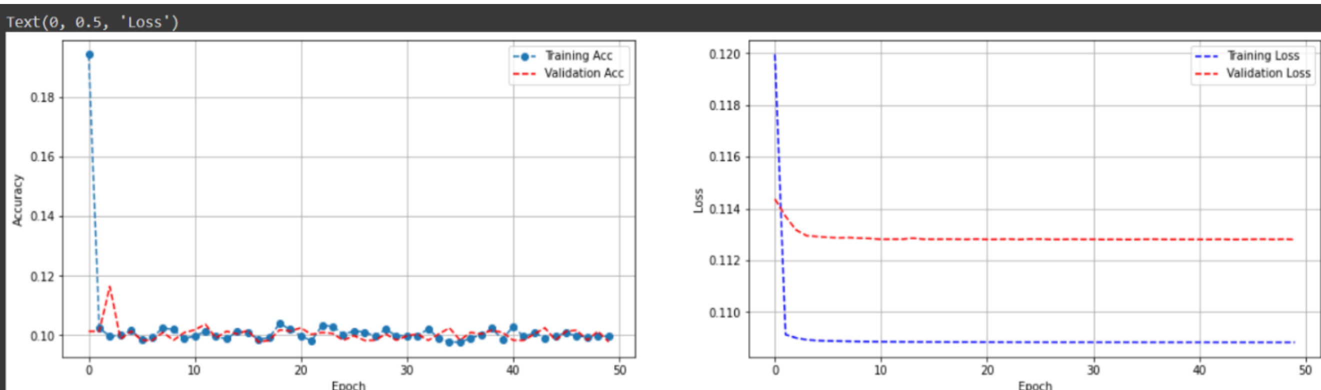
```
1750/1750 [==============================] - 32s 19ms/step - loss: 0.1088 - accuracy: 0.1030 - val_loss: 0.1128 - val_accuracy: 0.0984
Epoch 42/50
1750/1750 [==============================] - 32s 18ms/step - loss: 0.1088 - accuracy: 0.0996 - val_loss: 0.1128 - val_accuracy: 0.0983
Epoch 43/50
1750/1750 [==============================] - 32s 18ms/step - loss: 0.1088 - accuracy: 0.1008 - val_loss: 0.1128 - val_accuracy: 0.1005
Epoch 44/50
1750/1750 [==============================] - 32s 18ms/step - loss: 0.1088 - accuracy: 0.0992 - val_loss: 0.1128 - val_accuracy: 0.1025
Epoch 45/50
1750/1750 [==============================] - 34s 20ms/step - loss: 0.1088 - accuracy: 0.0999 - val_loss: 0.1128 - val_accuracy: 0.0984
Epoch 46/50
1750/1750 [==============================] - 35s 20ms/step - loss: 0.1088 - accuracy: 0.1008 - val_loss: 0.1128 - val_accuracy: 0.1014
Epoch 47/50
1750/1750 [==============================] - 35s 20ms/step - loss: 0.1088 - accuracy: 0.0998 - val_loss: 0.1128 - val_accuracy: 0.1017
Epoch 48/50
1750/1750 [==============================] - 32s 18ms/step - loss: 0.1088 - accuracy: 0.0995 - val_loss: 0.1128 - val_accuracy: 0.0984
Epoch 49/50
1750/1750 [==============================] - 35s 20ms/step - loss: 0.1088 - accuracy: 0.0997 - val_loss: 0.1128 - val_accuracy: 0.1014
Epoch 50/50
1750/1750 [==============================] - 32s 18ms/step - loss: 0.1088 - accuracy: 0.0997 - val_loss: 0.1128 - val_accuracy: 0.0979
```

```
[19] test_accuracy = model.evaluate(test_images_norm,test_labels)

313/313 [==============================] - 2s 7ms/step - loss: 0.1100 - accuracy: 0.1000
```

Here I achieved training accuracy around 0.0997 and validation accuracy around 0.0979 which were pretty much close. Also, I observed the test accuracy which was around 0.1000 that was

```
Text(0, 0.5, 'Loss')
```



greater than validation accuracy and training accuracy. Both training and validation losses had close difference. Both losses were almost close to test loss.

Optimizer: SGD

Loss: mean squared logarithmic error

```
[21] model.compile(
         optimizer = keras.optimizers.SGD(learning_rate=0.001),
         loss = keras.losses.mean_squared_logarithmic_error ,
         metrics = ['accuracy']
     )
```

```
[22] h3 = model.fit(x = train_images_norm, y = train_labels, epochs = 50, batch_size = 20, validation_split = 0.3)
```

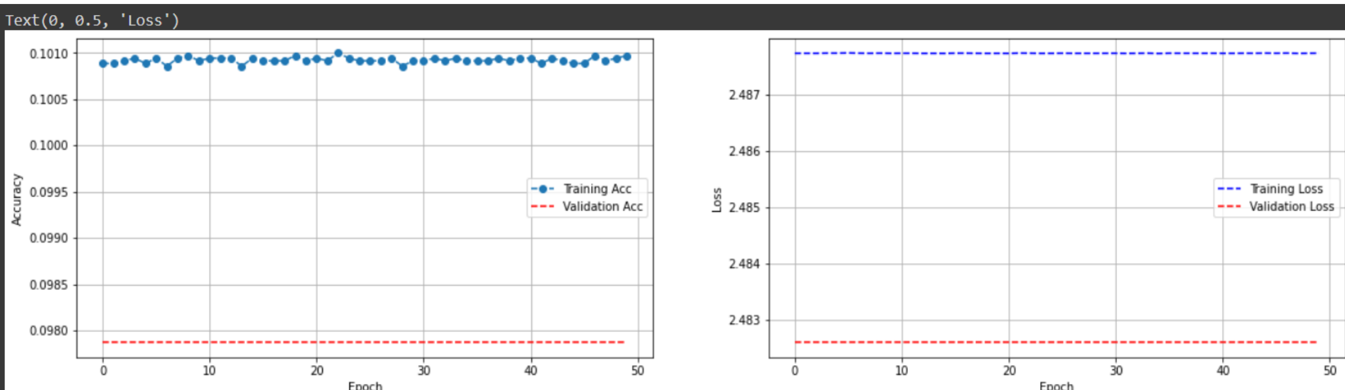
```
Epoch 45/50
1750/1750 [==============================] - 29s 16ms/step - loss: 2.4877 - accuracy: 0.1009 - val_loss: 2.4826 - val_accuracy: 0.0979
Epoch 46/50
1750/1750 [==============================] - 29s 16ms/step - loss: 2.4877 - accuracy: 0.1009 - val_loss: 2.4826 - val_accuracy: 0.0979
Epoch 47/50
1750/1750 [==============================] - 27s 15ms/step - loss: 2.4877 - accuracy: 0.1010 - val_loss: 2.4826 - val_accuracy: 0.0979
Epoch 48/50
1750/1750 [==============================] - 29s 17ms/step - loss: 2.4877 - accuracy: 0.1009 - val_loss: 2.4826 - val_accuracy: 0.0979
Epoch 49/50
1750/1750 [==============================] - 27s 15ms/step - loss: 2.4877 - accuracy: 0.1009 - val_loss: 2.4826 - val_accuracy: 0.0979
Epoch 50/50
1750/1750 [==============================] - 27s 15ms/step - loss: 2.4877 - accuracy: 0.1010 - val_loss: 2.4826 - val_accuracy: 0.0979
```

```
[23] model.evaluate(test_images_norm,test_labels)

313/313 [==============================] - 2s 7ms/step - loss: 2.4862 - accuracy: 0.1000
[2.486187219619751, 0.10000000149011612]
```

In this history the training accuracy was 0.1010 and validation accuracy was 0.0979. Both training and validation losses were according to 2.4877 and 2.4826 which was almost same. Test accuracy

```
Text(0, 0.5, 'Loss')
```



was 0.1000 and loss 2.4862. Here the validation accuracy was flat as the changes of validation accuracy were not frequent.

Third Model:

```
model = keras.Sequential([
    keras.Input(shape = (32, 32, 3)),
    keras.layers.Conv2D(filters = 32, kernel_size = (3, 3), padding='same', activation = "relu"),
    keras.layers.Dropout(0.5),
    keras.layers.Conv2D(filters = 32, kernel_size = (3, 3), padding='same',activation = "relu"),
    keras.layers.MaxPooling2D(pool_size=(2, 2)),
    keras.layers.BatchNormalization(),
    keras.layers.Dropout(0.8),
    keras.layers.Conv2D(filters = 64, kernel_size = (3, 3), padding='same', activation = "relu"),
    keras.layers.MaxPooling2D(pool_size=(2, 2)),
    keras.layers.Conv2D(filters = 128, kernel_size = (3, 3), padding='same', activation = "relu"),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters = 128, kernel_size = (3, 3), padding='same', activation = "relu"),
    keras.layers.Dropout(0.5),
    keras.layers.MaxPooling2D(pool_size=(2, 2)),
    keras.layers.Flatten(),
    keras.layers.Dense(units = 128, activation = "relu"),
    keras.layers.Dense(units = 10, activation = "softmax")
])

model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_11 (Conv2D)          (None, 32, 32, 32)        896

 dropout_6 (Dropout)         (None, 32, 32, 32)        0

 conv2d_12 (Conv2D)          (None, 32, 32, 32)        9248

 max_pooling2d_6 (MaxPooling  (None, 16, 16, 32)       0
 2D)

 batch_normalization_1 (Batc  (None, 16, 16, 32)       128
 hNormalization)

 dropout_7 (Dropout)         (None, 16, 16, 32)        0

 conv2d_13 (Conv2D)          (None, 16, 16, 64)        18496

 max_pooling2d_7 (MaxPooling  (None, 8, 8, 64)         0
 2D)

 conv2d_14 (Conv2D)          (None, 8, 8, 128)         73856

 batch_normalization_2 (Batc  (None, 8, 8, 128)        512
 hNormalization)

 conv2d_15 (Conv2D)          (None, 8, 8, 128)         147584

 dropout_8 (Dropout)         (None, 8, 8, 128)         0

 max_pooling2d_8 (MaxPooling  (None, 4, 4, 128)        0
 2D)
```

```
 flatten_2 (Flatten)         (None, 2048)              0

 dense_4 (Dense)             (None, 128)               262272

 dense_5 (Dense)             (None, 10)                1290

=================================================================
Total params: 514,282
Trainable params: 513,962
Non-trainable params: 320
_________________________________________________________________
```

For the third model, I have used 5 Convo2d layers with filers of 32, 64 and 128, the kernel size was 3x3. I kept the padding same and used Relu activation function. I also used 3 Maxpooling layers, 1 Flatten and 2 dense layers with the units of 128 and 10. Here I also used two 50% dropout and one 80% dropout as well as used BatchNormalization two times to avoid the overfitting issue.

After building the model, I used three different optimizers with 0.001 learning rate and three different losses each time to compile the model using 70 epochs, 28 batch size and 20% validation split.

Optimizer: Adam

Loss: spare categorical crossentropy

```
[11] model.compile(
        optimizer = keras.optimizers.Adam(learning_rate=0.001),
        loss = keras.losses.sparse_categorical_crossentropy,
        metrics = ['accuracy']
     )

[12] h1 = model.fit(x = train_images_norm, y = train_labels, epochs = 70, batch_size = 28, validation_split = 0.2)
```

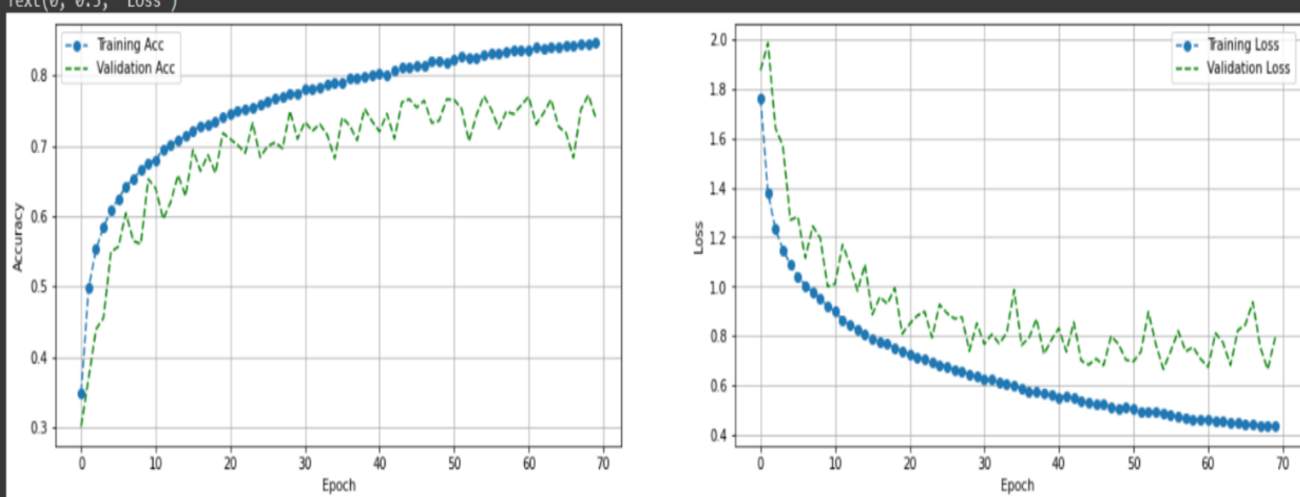
```
Epoch 65/70
1429/1429 [==============================] - 23s 16ms/step - loss: 0.4496 - accuracy: 0.8411 - val_loss: 0.8236 - val_accuracy: 0.7273
Epoch 66/70
1429/1429 [==============================] - 24s 17ms/step - loss: 0.4472 - accuracy: 0.8420 - val_loss: 0.8468 - val_accuracy: 0.7182
Epoch 67/70
1429/1429 [==============================] - 25s 18ms/step - loss: 0.4414 - accuracy: 0.8435 - val_loss: 0.9376 - val_accuracy: 0.6829
Epoch 68/70
1429/1429 [==============================] - 25s 18ms/step - loss: 0.4412 - accuracy: 0.8437 - val_loss: 0.7551 - val_accuracy: 0.7513
Epoch 69/70
1429/1429 [==============================] - 23s 16ms/step - loss: 0.4395 - accuracy: 0.8446 - val_loss: 0.6652 - val_accuracy: 0.7723
Epoch 70/70
1429/1429 [==============================] - 24s 17ms/step - loss: 0.4352 - accuracy: 0.8462 - val_loss: 0.7948 - val_accuracy: 0.7392
```

```
[13] test_accuracy = model.evaluate(test_images_norm,test_labels)
```

```
313/313 [==============================] - 2s 7ms/step - loss: 0.8156 - accuracy: 0.7327
```

In this first history of third model the training accuracy was around 84.62% and validation accuracy was around 73.92%. The training loss and validation loss were accordingly 0.8462 and

```
Text(0, 0.5, 'Loss')
```



0.7948. Also, the test accuracy was almost 81.56% and loss 0.8156. Tough losses were much higher but this model has consistency between training and validation.

Optimizer: RMSprop
Loss: categorical hinge

```
[15] model.compile(
         optimizer = keras.optimizers.RMSprop(learning_rate=0.001),
         loss = keras.losses.categorical_hinge,
         metrics = ['accuracy']
     )
```

```
[16] h2 = model.fit(x = train_images_norm, y = train_labels, epochs = 70, batch_size = 28, validation_split = 0.2)
```

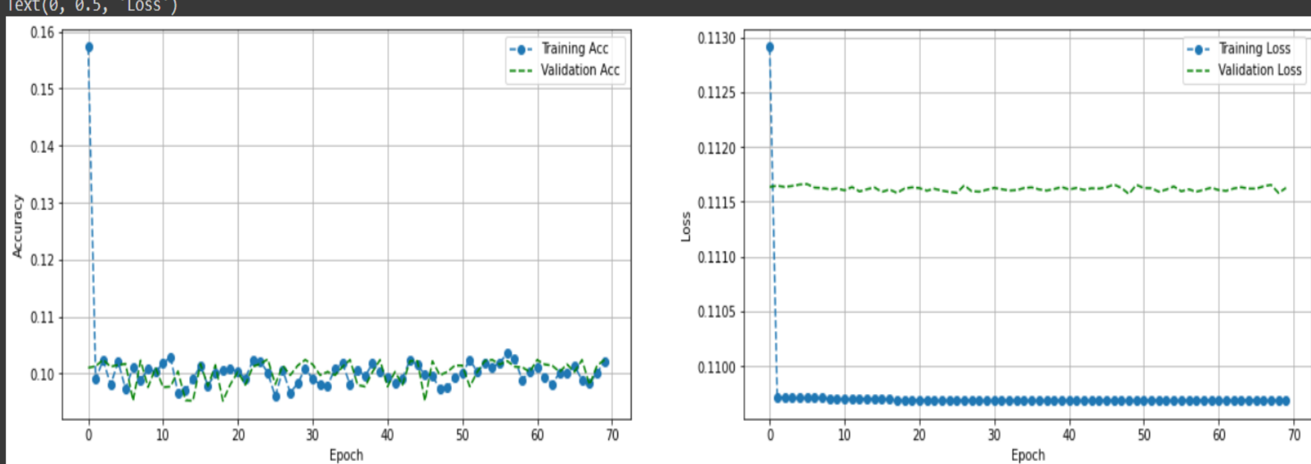
```
1429/1429 [==============================] - 33s 23ms/step - loss: 0.1097 - accuracy: 0.0982 - val_loss: 0.1116 - val_accuracy: 0.1014
Epoch 64/70
1429/1429 [==============================] - 33s 23ms/step - loss: 0.1097 - accuracy: 0.1001 - val_loss: 0.1116 - val_accuracy: 0.1003
Epoch 65/70
1429/1429 [==============================] - 33s 23ms/step - loss: 0.1097 - accuracy: 0.1001 - val_loss: 0.1116 - val_accuracy: 0.1016
Epoch 66/70
1429/1429 [==============================] - 33s 23ms/step - loss: 0.1097 - accuracy: 0.1013 - val_loss: 0.1116 - val_accuracy: 0.1003
Epoch 67/70
1429/1429 [==============================] - 33s 23ms/step - loss: 0.1097 - accuracy: 0.0988 - val_loss: 0.1116 - val_accuracy: 0.1024
Epoch 68/70
1429/1429 [==============================] - 33s 23ms/step - loss: 0.1097 - accuracy: 0.0984 - val_loss: 0.1117 - val_accuracy: 0.0980
Epoch 69/70
1429/1429 [==============================] - 33s 23ms/step - loss: 0.1097 - accuracy: 0.1001 - val_loss: 0.1116 - val_accuracy: 0.1014
Epoch 70/70
1429/1429 [==============================] - 33s 23ms/step - loss: 0.1097 - accuracy: 0.1022 - val_loss: 0.1116 - val_accuracy: 0.1024
```

```
[17] test_accuracy = model.evaluate(test_images_norm,test_labels)

313/313 [==============================] - 2s 7ms/step - loss: 0.1101 - accuracy: 0.1000
```

Here 0.1022 training accuracy and 0.1024 validation accuracy were achieved. Both accuracies were almost same. The training loss was .1097 and validation loss was 0.1116. Also, the test

```
[18]
Text(0, 0.5, 'Loss')
```



accuracy was almost similar to both training and validation accuracy, which was 0.1000. Test loss was 0.1101. Here also the consistency between training and validation was seen.

Optimizer: SGD

Loss: mean squared logarithmic error

```
[19] model.compile(
         optimizer = keras.optimizers.SGD(learning_rate=0.001),
         loss = keras.losses.mean_squared_logarithmic_error ,
         metrics = ['accuracy']
     )

[20] h3 = model.fit(x = train_images_norm, y = train_labels, epochs = 70, batch_size = 28, validation_split = 0.2)
```

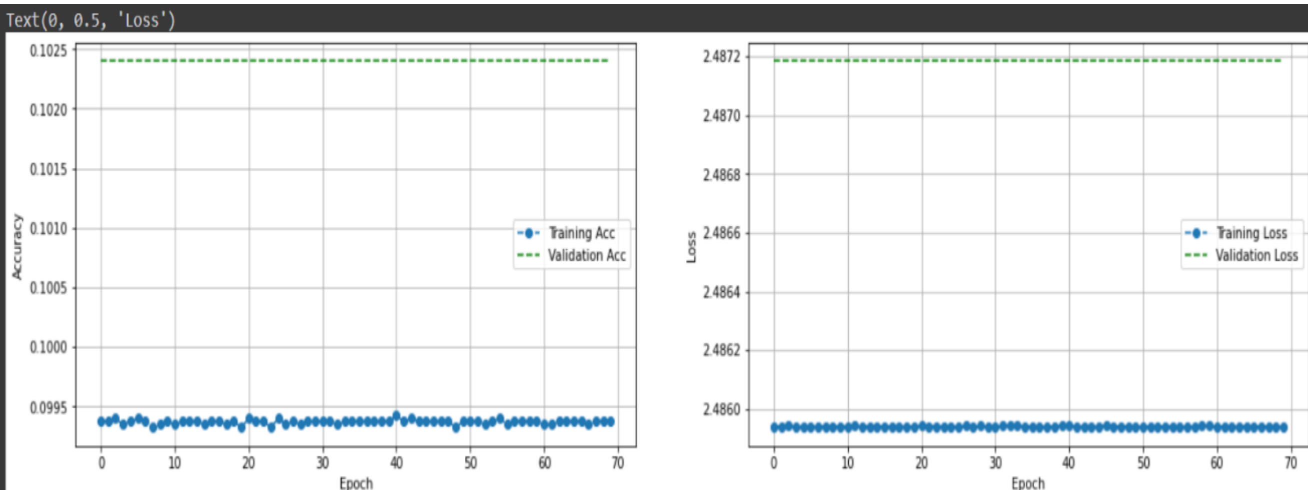
```
1429/1429 [==============================] - 24s 16ms/step - loss: 2.4859 - accuracy: 0.0994 - val_loss: 2.4872 - val_accuracy: 0.1024
Epoch 66/70
1429/1429 [==============================] - 24s 16ms/step - loss: 2.4859 - accuracy: 0.0994 - val_loss: 2.4872 - val_accuracy: 0.1024
Epoch 67/70
1429/1429 [==============================] - 23s 16ms/step - loss: 2.4859 - accuracy: 0.0993 - val_loss: 2.4872 - val_accuracy: 0.1024
Epoch 68/70
1429/1429 [==============================] - 23s 16ms/step - loss: 2.4859 - accuracy: 0.0994 - val_loss: 2.4872 - val_accuracy: 0.1024
Epoch 69/70
1429/1429 [==============================] - 24s 17ms/step - loss: 2.4859 - accuracy: 0.0994 - val_loss: 2.4872 - val_accuracy: 0.1024
Epoch 70/70
1429/1429 [==============================] - 24s 17ms/step - loss: 2.4859 - accuracy: 0.0994 - val_loss: 2.4872 - val_accuracy: 0.1024

test_accuracy = model.evaluate(test_images_norm,test_labels)

313/313 [==============================] - 2s 7ms/step - loss: 2.4862 - accuracy: 0.1000
```

Here the both training and validation accuracies had slightly higher difference between them. But the losses were almost same 2.4859 training loss and 2.4872 validation loss. Also, both

```
Text(0, 0.5, 'Loss')
```



accuracies and losses had consistency with test accuracy and loss which were according to 0.1000 and 2.4862.

Conclusion:

After observing the three models, according to my opinion third model was the best among the two models. The training results and validation results have some consistency in the third model, which is missing in the other two models. So it can be said, the third model is almost more perfect than the other two models.