In [1]:
```python
# Import Packages
import timeit
import sys
import numpy as np
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
from keras import callbacks
import matplotlib.pyplot as plt
```

```
Using TensorFlow backend.
```

In [2]:
```python
# Load Train and Test CIFAR10 dataset
(X_train, y_train), (X_test, y_test) = datasets.cifar10.load_data()
```

In [3]:
```python
# Check the Train and Test Dataset
print("Shape of X_train Set: ", X_train.shape)
print("Shape of y_train set: ", y_train.shape)
print("Shape of X_test Set: ", X_test.shape)
print("Shape of y_test set: ", y_test.shape)
```

```
Shape of X_train Set:  (50000, 32, 32, 3)
Shape of y_train set:  (50000, 1)
Shape of X_test Set:  (10000, 32, 32, 3)
Shape of y_test set:  (10000, 1)
```

As we can see that there are 50000 training samples and 10000 test samples.

In [4]:
```python
# Normalization of the Dataset to bring the pixel values between 0 and 1
X_train = X_train/255.0
X_test = X_test/255.0
```

In [5]:
```python
# Check the shape of image Dataset
print("Shape of X_train Set: ", X_train.shape)
print("Shape of X_test Set: ", X_test.shape)
```

```
Shape of X_train Set:  (50000, 32, 32, 3)
Shape of X_test Set:  (10000, 32, 32, 3)
```

In [6]:
```python
from sklearn.model_selection import train_test_split
X_train_gar, X_train_20per, y_train_gar, y_train_20per = train_test_split(X_train, y_train, test_size=0.20, random_state = 42)
```

As required, the 20% of the train dataset is randomly select to form a new train dataset.

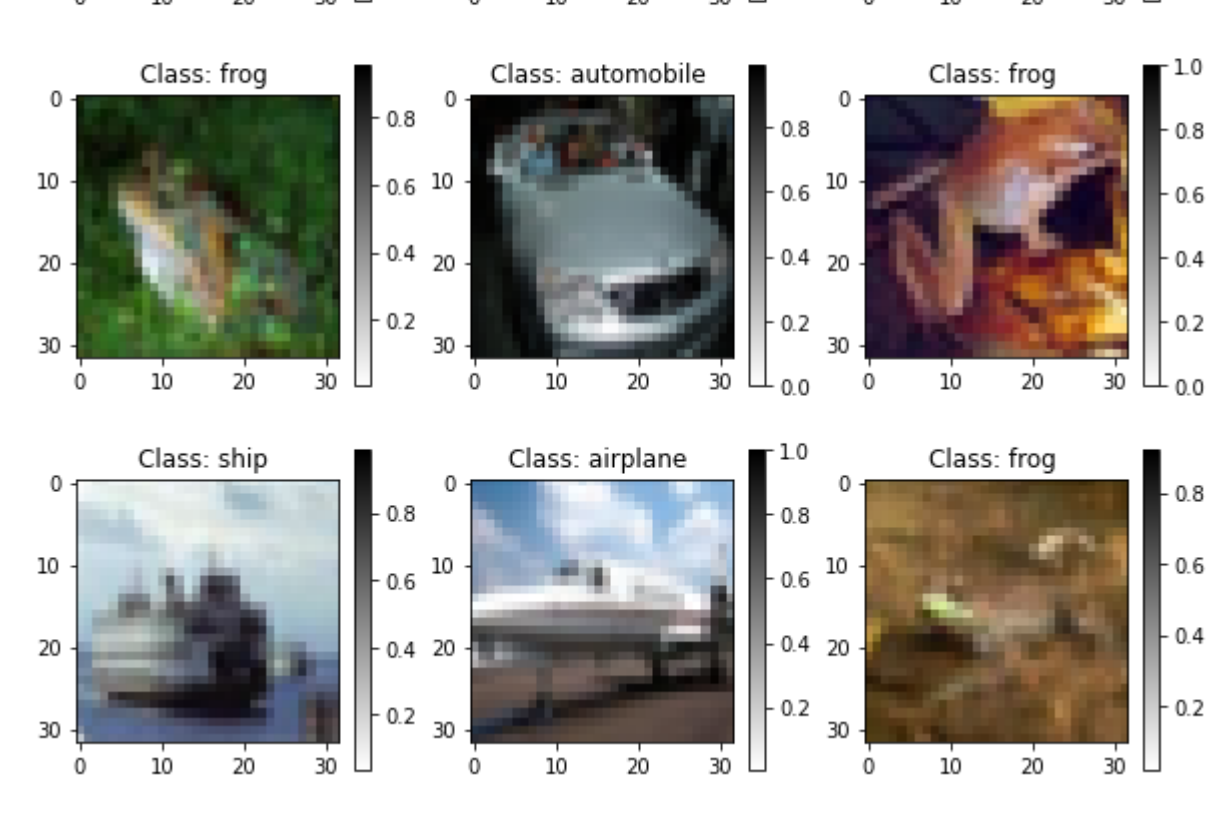
In [7]:
```python
# 10 different Classes in the Dataset
labels = ['airplane', 'automobile', 'bird', 'cat', 'deer',
              'dog', 'frog', 'horse', 'ship', 'truck']
```

In [8]:
```python
# Idea is from Tensorflow Website regarding subplot but modifications are made.
# Plot function for the Images
def Plot_CIFAR10Images(X_train, y_train):
  """ Plot for the Images to confirm
  Parameter:
  -----------
  X_train: Train Dataset
  y_train: Test Dataset

  """
  plt.figure(figsize=(10,10), edgecolor='g')
  for image_index in range(9):
    plt.subplot(3, 3, image_index + 1)
    plt.imshow(X_train[image_index], cmap = plt.cm.binary)
    plt.colorbar()
    plt.title("Class: {0} ".format(labels[y_train[image_index][0]]))

  plt.show()
```

In [9]:
```python
# Plot for the Dataset
Plot_CIFAR10Images(X_train_20per, y_train_20per)
```



Test for the Imported Dataset

The above plot is mainly done to show that the dataset is correctly imported and labels match with the picture as this is important to verify before training the Neural Network

In [10]:
```python
# Loss Evaluation Function for the Model
def Model_loss_Evaluation(history_model, model_eva):
  """ Loss Plot for Model
  Parameter:
  -----------
  history_model: history of the model
  model_eva: Name of the Model

  """
  plt.plot(history_model.history['loss'], color = 'm', label = 'Training Loss')
  plt.plot(history_model.history['val_loss'], color = 'r', label = "Val Loss")
  plt.xlabel('Epoch')
  plt.ylabel('Loss')
  plt.title('Model Evaluation for {0}'.format(model_eva))
  plt.legend(loc = 'best')
  plt.show()
```

In [11]:
```python
# Accuracy Evaluation for the Model
def Model_Accuracy_Evaluation(history_model, model_eva):
  """ Accracy Plot for the Model
  Parameter:
  -----------
  history_model: history for the model
  model_eva: Name of the Model
  """
  plt.plot(history_model.history['accuracy'], color = 'k', label = " Accuracy ")
  plt.plot(history_model.history['val_accuracy'], color = 'g', label = 'Val Accuracy')
  plt.xlabel('Epoch')
  plt.ylabel('Accuracy Score ')
  plt.title('Model Accuracy Evaluation for {0}'.format(model_eva))
  plt.legend(loc = 'best')
  plt.show()
```

Preprocessing For Dataset

In the preprocessing step the pixels of the images are normalized between 0 and 1 by dividing with 255.

1. MLP1

In [12]:
```python
# Model Initialization for MLP1
model_mlp = models.Sequential()
model_mlp.add(layers.Flatten(input_shape = X_train_20per.shape[1:]))
model_mlp.add(layers.Dense(512, activation= 'sigmoid'))
model_mlp.add(layers.Dense(512, activation= 'sigmoid'))
model_mlp.add(layers.Dense(10, activation='softmax'))
```

In [13]:
```python
# Summary of the MLP1 Model
model_mlp.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 3072)              0
_________________________________________________________________
dense (Dense)                (None, 512)               1573376
_________________________________________________________________
dense_1 (Dense)              (None, 512)               262656
_________________________________________________________________
dense_2 (Dense)              (None, 10)                5130
=================================================================
Total params: 1,841,162
Trainable params: 1,841,162
Non-trainable params: 0
_________________________________________________________________
```

In [14]:
```python
# Initialization of the Model
Batch = 32

# Compile the Model
model_mlp.compile(optimizer='adam',
              loss = 'sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Train the Model
start = timeit.default_timer()
history_mlp = model_mlp.fit(X_train_20per, y_train_20per, epochs = 5, batch_size = Batch, shuffle=True, validation_data = (X_test, y_test))
stop = timeit.default_timer()
sys.stderr.write(" \n Training Time for MLP is {0} Seconds".format(stop - start))
sys.stderr.flush()
```

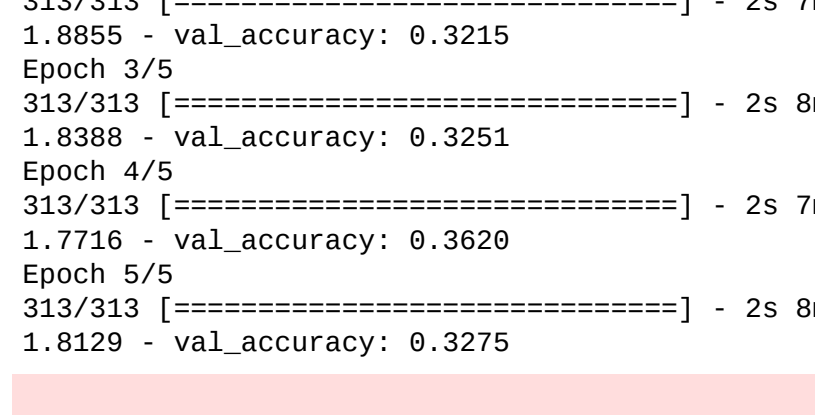
```
Epoch 1/5
313/313 [==============================] - 2s 8ms/step - loss: 2.0809 - accuracy: 0.2261 - val_loss: 1.8989 - val_accuracy: 0.3139
Epoch 2/5
313/313 [==============================] - 2s 7ms/step - loss: 1.9013 - accuracy: 0.3068 - val_loss: 1.8855 - val_accuracy: 0.3215
Epoch 3/5
313/313 [==============================] - 2s 8ms/step - loss: 1.8389 - accuracy: 0.3219 - val_loss: 1.8388 - val_accuracy: 0.3251
Epoch 4/5
313/313 [==============================] - 2s 7ms/step - loss: 1.8041 - accuracy: 0.3427 - val_loss: 1.7716 - val_accuracy: 0.3620
Epoch 5/5
313/313 [==============================] - 2s 8ms/step - loss: 1.7648 - accuracy: 0.3537 - val_loss: 1.8129 - val_accuracy: 0.3275

 Training Time for MLP is 12.605483563000234 Seconds
```

In [15]:
```python
# Test the Model on the Test Dataset
test_loss_mlp, test_acc_mlp = model_mlp.evaluate(X_test,  y_test, verbose=2)
```

```
313/313 - 1s - loss: 1.8129 - accuracy: 0.3275
```

In [16]:
```python
# Plot the Loss, Accuracy and Validation Accuracy
Model_loss_Evaluation(history_mlp, model_eva='MLP1')
```

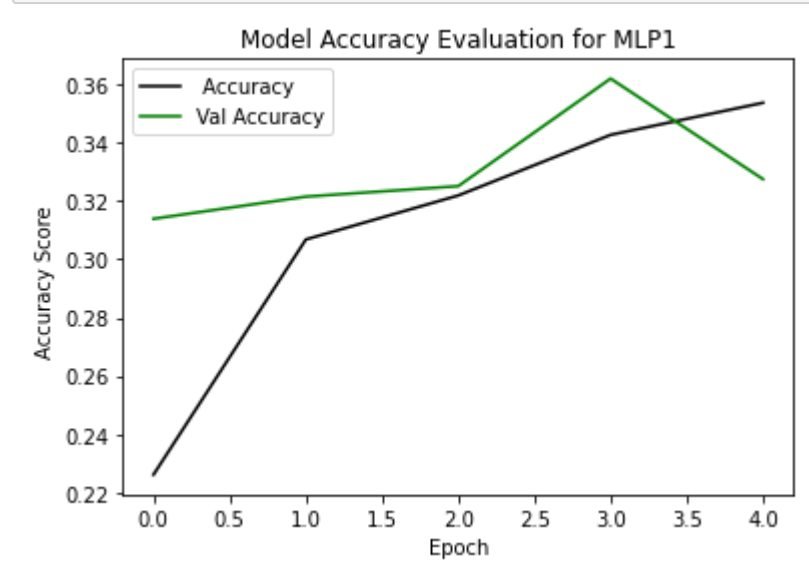


Loss Analysis for MLP1 Network

1. Architecture 1: 512x512x10 as per the Assignemnt pdf.

From the Loss Plot shown above, it is clear that the validation Loss and Trainig Loss decreases as the number of epochs increases. However, after Epoch 3 the validation loss becomes stagnant and is higher than Training loss after this point forward. This validation loss value reaches a low at 1.8129 for the 5th Epoch which is higher than the Training loss as shown by the graph. The Training loss decreases further and at Epoch 5 its value is 1.7648. When the number of Epoch is increased in the above case the Validation loss becomes stagnant where as the training loss decreases further which shows that the model is overfitting at higher epochs.

In [17]:
```python
# Plot the Accuracy  Score for MLP1
Model_Accuracy_Evaluation(history_mlp, model_eva='MLP1')
```



Accuracy Analysis for MLP1 Network

1. Architecture 1: 512x512x10 as per the Assignemnt pdf.

From the Accuracy Plot shown above it is clear that the validation accuracy increase as the number of epochs increases and this value reaches a high at 36.20% for the 4th Epoch which is higher than the Training accuracy as shown by the graph and then it decreases at Epoch 5. The training accuracy remains lower for each of the Epochs when compared with the validation accuracy until Epoch 4.5 and from there on the value increases above validation accuracy. When we increase the number of Epoch for this network the accuracy score oscillates between 35% to 39% but doesn't drastically increase that much.

The time taken to train this network is 12.61 seconds.

1.1. MLP2 for Testing with Increased Layers and Neurons Without Dropouts.

In [18]:
```python
# Model Initialization for MLP2
model_mlp2 = models.Sequential()
model_mlp2.add(layers.Flatten(input_shape = X_train_20per.shape[1:]))
model_mlp2.add(layers.Dense(3024, activation= 'sigmoid'))
model_mlp2.add(layers.Dense(2024, activation= 'sigmoid'))
model_mlp2.add(layers.Dense(1024, activation= 'sigmoid'))
model_mlp2.add(layers.Dense(512, activation= 'sigmoid'))
model_mlp2.add(layers.Dense(256, activation= 'sigmoid'))
model_mlp2.add(layers.Dense(128, activation= 'sigmoid'))
model_mlp2.add(layers.Dense(50, activation= 'sigmoid'))
model_mlp2.add(layers.Dense(10, activation='softmax'))
```

In [19]:
```python
model_mlp2.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_1 (Flatten)          (None, 3072)              0
_________________________________________________________________
dense_3 (Dense)              (None, 3024)              9292752
_________________________________________________________________
dense_4 (Dense)              (None, 2024)              6122600
_________________________________________________________________
dense_5 (Dense)              (None, 1024)              2073600
_________________________________________________________________
dense_6 (Dense)              (None, 512)               524800
_________________________________________________________________
dense_7 (Dense)              (None, 256)               131328
_________________________________________________________________
dense_8 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_9 (Dense)              (None, 50)                6450
_________________________________________________________________
dense_10 (Dense)             (None, 10)                510
=================================================================
Total params: 18,184,936
Trainable params: 18,184,936
Non-trainable params: 0
_________________________________________________________________
```

In [20]:
```python
# Initialization of the Model
Batch = 32

model_mlp2.compile(optimizer='adam',
              loss = 'sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Train the Model
start = timeit.default_timer()
history_mlp2 = model_mlp2.fit(X_train_20per, y_train_20per, epochs = 5, batch_size = Batch, shuffle=True, validation_data = (X_test, y_test))
stop = timeit.default_timer()
sys.stderr.write(" \n Training Time for MLP2 is {0} Seconds".format(stop - start))
sys.stderr.flush()
```

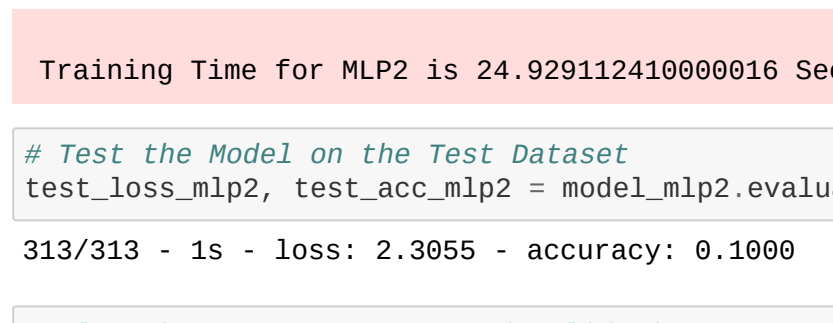
```
Epoch 1/5
313/313 [==============================] - 5s 16ms/step - loss: 2.3137 - accuracy: 0.0973 - val_loss: 2.3079 - val_accuracy: 0.1000
Epoch 2/5
313/313 [==============================] - 5s 15ms/step - loss: 2.3084 - accuracy: 0.0983 - val_loss: 2.3070 - val_accuracy: 0.1000
Epoch 3/5
313/313 [==============================] - 5s 15ms/step - loss: 2.3076 - accuracy: 0.0955 - val_loss: 2.3050 - val_accuracy: 0.1000
Epoch 4/5
313/313 [==============================] - 5s 15ms/step - loss: 2.3084 - accuracy: 0.0967 - val_loss: 2.3045 - val_accuracy: 0.1000
Epoch 5/5
313/313 [==============================] - 5s 15ms/step - loss: 2.3058 - accuracy: 0.0979 - val_loss: 2.3055 - val_accuracy: 0.1000

 Training Time for MLP2 is 24.929112410000016 Seconds
```

In [21]:
```python
# Test the Model on the Test Dataset
test_loss_mlp2, test_acc_mlp2 = model_mlp2.evaluate(X_test,  y_test, verbose=2)
```

```
313/313 - 1s - loss: 2.3055 - accuracy: 0.1000
```

In [22]:
```python
# Plot the Loss, Accuracy and Validation Accuracy
Model_loss_Evaluation(history_mlp2, model_eva='MLP2')
```

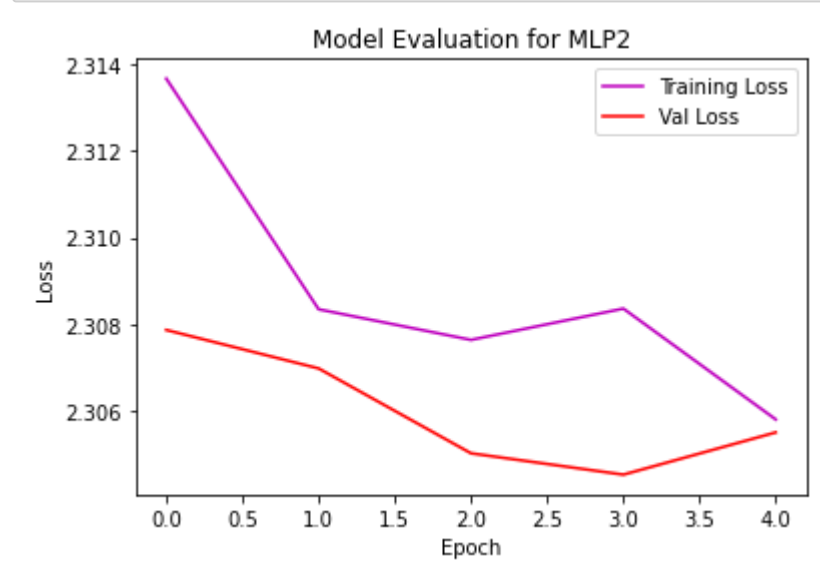


Loss Analysis for MLP2 Network

1. Architecture 2: Increased Number of Hidden Layers and Neurons.

From the above plot it is clear that the training loss decreases and at Epoch 5 it takes its lowest value of 2.3058. In contrast Validation loss first decreases at Epoch 1 and then it increases from Epoch 4 onwards until Epoch 5. In addition, When we increase the number of Epoch the validation loss oscillates which shows the overfitting at increased number of Epochs.

In [23]:
```python
# Plot the Loss, Accuracy and Validation Accuracy
Model_Accuracy_Evaluation(history_mlp2, model_eva='MLP2')
```



Accuracy Analysis for MLP2 Network

1. Architecture 2: Increased Hidden and Neurons.

From the Accuracy Plot shown above, it is clear that the validation accuracy remains stagnant at 0.1000 as the number of epochs increases. Even if we increase the number of Epochs there is no effect to the validation accuracy score of the Network. This shows the Overfitting clearly. In addition, the training accuracy first increases from epoch 1 to 2 and then decreases at Epoch 3. In short the training accuracy score oscillates and this also shows overfitting. In addition, When we increase the number of Epoch this result doesn't changes and when compared to MLP1 the results are Poorer.

In addition the Training Time for this network is 24.93 seconds which is higher than the MLP1 Network.

1.2. MLP3 for Testing with Increased Layers and Neurons With Dropouts.

In [24]:
```python
# Model Initialization for MLP3
model_mlp3 = models.Sequential()
model_mlp3.add(layers.Flatten(input_shape = X_train_20per.shape[1:]))
model_mlp3.add(layers.Dense(3024, activation= 'sigmoid'))
model_mlp3.add(layers.Dropout(0.2))
model_mlp3.add(layers.Dense(2024, activation= 'sigmoid'))
model_mlp3.add(layers.Dropout(0.2))
model_mlp3.add(layers.Dense(1024, activation= 'sigmoid'))
model_mlp3.add(layers.Dropout(0.2))
model_mlp3.add(layers.Dense(512, activation= 'sigmoid'))
model_mlp3.add(layers.Dropout(0.2))
model_mlp3.add(layers.Dense(256, activation= 'sigmoid'))
model_mlp3.add(layers.Dropout(0.2))
model_mlp3.add(layers.Dense(128, activation= 'sigmoid'))
model_mlp3.add(layers.Dropout(0.2))
model_mlp3.add(layers.Dense(50, activation= 'sigmoid'))
model_mlp3.add(layers.Dropout(0.2))
model_mlp3.add(layers.Dense(10, activation='softmax'))
```

In [25]:
```python
model_mlp3.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_2 (Flatten)          (None, 3072)              0
_________________________________________________________________
dense_11 (Dense)             (None, 3024)              9292752
_________________________________________________________________
dropout (Dropout)            (None, 3024)              0
_________________________________________________________________
dense_12 (Dense)             (None, 2024)              6122600
_________________________________________________________________
dropout_1 (Dropout)          (None, 2024)              0
_________________________________________________________________
dense_13 (Dense)             (None, 1024)              2073600
_________________________________________________________________
dropout_2 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_14 (Dense)             (None, 512)               524800
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_15 (Dense)             (None, 256)               131328
_________________________________________________________________
dropout_4 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_16 (Dense)             (None, 128)               32896
_________________________________________________________________
dropout_5 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_17 (Dense)             (None, 50)                6450
_________________________________________________________________
dropout_6 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_18 (Dense)             (None, 10)                510
=================================================================
Total params: 18,184,936
Trainable params: 18,184,936
Non-trainable params: 0
_________________________________________________________________
```

In [26]:
```python
# Initialization of the Model
Batch = 32

model_mlp3.compile(optimizer='adam',
              loss = 'sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Train the Model
start = timeit.default_timer()
history_mlp3 = model_mlp3.fit(X_train_20per, y_train_20per, epochs = 5, batch_size = Batch, shuffle=True, validation_data = (X_test, y_test))
stop = timeit.default_timer()
sys.stderr.write(" \n Training Time for MLP3 is {0} Seconds".format(stop - start))
sys.stderr.flush()
```

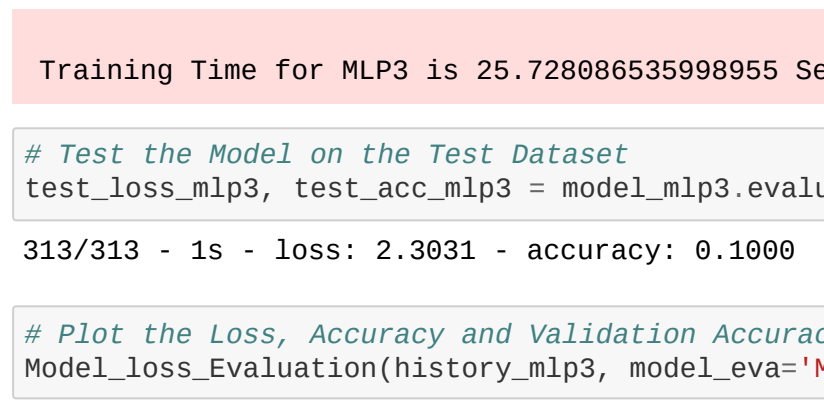
```
Epoch 1/5
313/313 [==============================] - 5s 16ms/step - loss: 2.3303 - accuracy: 0.1022 - val_loss: 2.3058 - val_accuracy: 0.1000
Epoch 2/5
313/313 [==============================] - 5s 16ms/step - loss: 2.3089 - accuracy: 0.0919 - val_loss: 2.3028 - val_accuracy: 0.1000
Epoch 3/5
313/313 [==============================] - 5s 16ms/step - loss: 2.3047 - accuracy: 0.0981 - val_loss: 2.3027 - val_accuracy: 0.1000
Epoch 4/5
313/313 [==============================] - 5s 16ms/step - loss: 2.3030 - accuracy: 0.0984 - val_loss: 2.3030 - val_accuracy: 0.1000
Epoch 5/5
313/313 [==============================] - 5s 16ms/step - loss: 2.3031 - accuracy: 0.0983 - val_loss: 2.3031 - val_accuracy: 0.1000

 Training Time for MLP3 is 25.728086535998955 Seconds
```

In [27]:
```python
# Test the Model on the Test Dataset
test_loss_mlp3, test_acc_mlp3 = model_mlp3.evaluate(X_test,  y_test, verbose=2)
```

```
313/313 - 1s - loss: 2.3031 - accuracy: 0.1000
```

In [28]:
```python
# Plot the Loss, Accuracy and Validation Accuracy
Model_loss_Evaluation(history_mlp3, model_eva='MLP3')
```

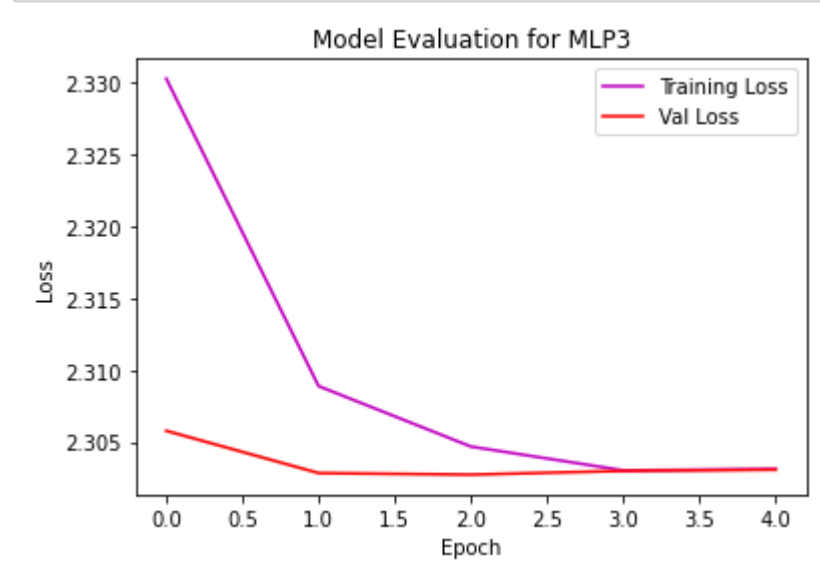


Loss Analysis for MLP3 Network

1. Architecture 2: Increased Number of Hidden Layers and Neurons with Dropouts.

From the above plot it is clear that the training loss decreases and at Epoch 5 it takes its lowest value of 2.3031. In contrast Validation loss first decreases slightly at Epoch 1 and then it remains stagnant further at the increase of Epoch. In addition, When we increase the number of Epoch the validation loss oscillates which shows the overfitting at increased number of Epochs.

In [29]:
```python
# Plot the Loss, Accuracy and Validation Accuracy
Model_Accuracy_Evaluation(history_mlp3, model_eva='MLP3')
```



Accuracy Analysis for MLP3 Network

1. Architecture 2: Increased Hidden and Neurons.

From the Accuracy Plot shown above, it is clear that the validation accuracy remains stagnant at 0.1000 as the number of epochs increases until Epoch 5. Even if we increase the number of Epochs there is no effect to the validation accuracy score of the Network. This shows the Overfitting clearly. In addition, the training accuracy first oscillates until Epochs 3 and then becomes stagnant. The Model Clearly tells us the case of Overfitting.

In addition the Training Time for this network is 25.73 seconds which is higher than the MLP1 Network.

1.3. MLP4 for Testing with decreased Hidden Layers and Neurons With Dropouts.

In [30]:
```python
# Model Initialization for MLP4
model_mlp4 = models.Sequential()
model_mlp4.add(layers.Flatten(input_shape = X_train_20per.shape[1:]))
model_mlp4.add(layers.Dense(1024, activation= 'sigmoid'))
model_mlp4.add(layers.Dropout(0.2))
model_mlp4.add(layers.Dense(512, activation= 'sigmoid'))
model_mlp4.add(layers.Dropout(0.2))
model_mlp4.add(layers.Dense(256, activation= 'sigmoid'))
model_mlp4.add(layers.Dropout(0.2))
model_mlp4.add(layers.Dense(128, activation= 'sigmoid'))
model_mlp4.add(layers.Dropout(0.2))
model_mlp4.add(layers.Dense(50, activation= 'sigmoid'))
model_mlp4.add(layers.Dropout(0.2))
model_mlp4.add(layers.Dense(10, activation='softmax'))
```

In [31]:
```python
# Summary of the Model MLP4
model_mlp4.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_3 (Flatten)          (None, 3072)              0
_________________________________________________________________
dense_19 (Dense)             (None, 1024)              3146752
_________________________________________________________________
dropout_7 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_20 (Dense)             (None, 512)               524800
_________________________________________________________________
dropout_8 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_21 (Dense)             (None, 256)               131328
_________________________________________________________________
dropout_9 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_22 (Dense)             (None, 128)               32896
_________________________________________________________________
dropout_10 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_23 (Dense)             (None, 50)                6450
_________________________________________________________________
dropout_11 (Dropout)         (None, 50)                0
_________________________________________________________________
dense_24 (Dense)             (None, 10)                510
=================================================================
Total params: 3,842,736
Trainable params: 3,842,736
Non-trainable params: 0
_________________________________________________________________
```

In [32]:
```python
# Initialization of the Model
Batch = 32

model_mlp4.compile(optimizer='adam',
              loss = 'sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Train the Model
start = timeit.default_timer()
history_mlp4 = model_mlp4.fit(X_train_20per, y_train_20per, epochs = 5, batch_size = Batch, shuffle=True, validation_data = (X_test, y_test))
stop = timeit.default_timer()
sys.stderr.write(" \n Training Time for MLP4 is {0} Seconds".format(stop - start))
sys.stderr.flush()
```

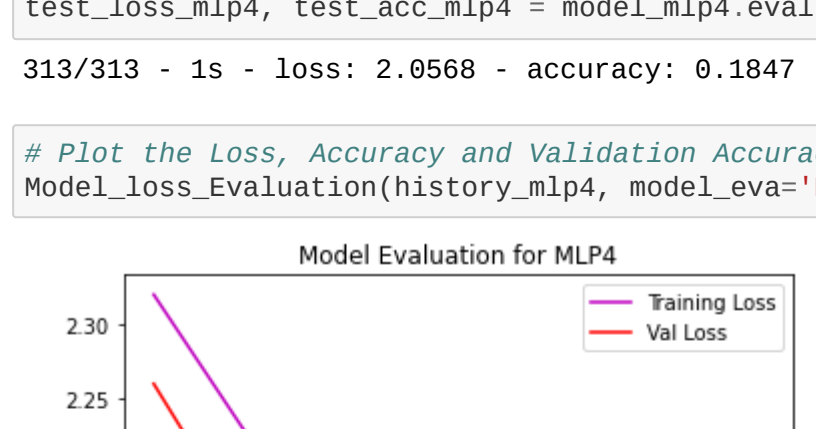
```
Epoch 1/5
313/313 [==============================] - 3s 8ms/step - loss: 2.3192 - accuracy: 0.1102 - val_loss: 2.2601 - val_accuracy: 0.1538
Epoch 2/5
313/313 [==============================] - 3s 8ms/step - loss: 2.1675 - accuracy: 0.1607 - val_loss: 2.0962 - val_accuracy: 0.1821
Epoch 3/5
313/313 [==============================] - 3s 8ms/step - loss: 2.1174 - accuracy: 0.1687 - val_loss: 2.0974 - val_accuracy: 0.1917
Epoch 4/5
313/313 [==============================] - 3s 8ms/step - loss: 2.0995 - accuracy: 0.1740 - val_loss: 2.0651 - val_accuracy: 0.1905
Epoch 5/5
313/313 [==============================] - 3s 8ms/step - loss: 2.0898 - accuracy: 0.1804 - val_loss: 2.0568 - val_accuracy: 0.1847

 Training Time for MLP4 is 13.634988903000703 Seconds
```

In [33]:
```python
# Test the Model on the Test Dataset
test_loss_mlp4, test_acc_mlp4 = model_mlp4.evaluate(X_test,  y_test, verbose =2)
```

```
313/313 - 1s - loss: 2.0568 - accuracy: 0.1847
```

In [34]:
```python
# Plot the Loss, Accuracy and Validation Accuracy
Model_loss_Evaluation(history_mlp4, model_eva='MLP4')
```

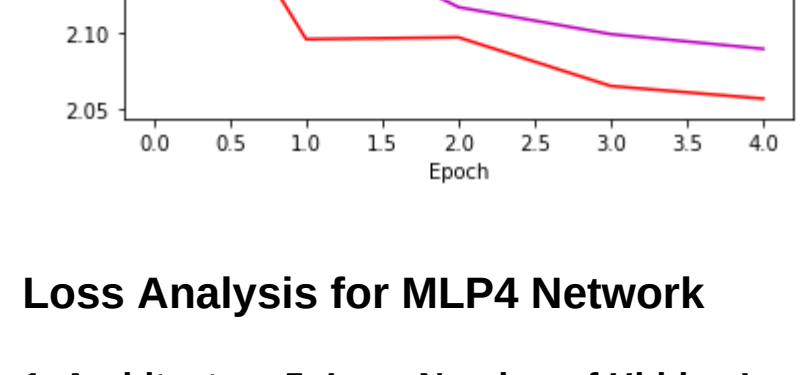


Loss Analysis for MLP4 Network

1. Architecture 5: Less Number of Hidden Layers and Neurons with Dropouts and Relu as Activation Function.

This architecture is considered to analyse how the perfomance varies when the number of hidden layer and neuron is reduced. From the above plot it is clear that the training loss decreases and this decrease is smoother than the previous two models. At Epoch 5 it takes its lowest value of 2.0898 and this loss value is lower than that of the previous model. In addition, the Validation loss follows a similar pattern to that of validation loss and the loss value is smaller thorought the Epochs which is better than the above two models. The Validation loss takes a values of 2.0568 which is lower than training loss. This model is an improvement as it doesn't overfitt.

In [35]:
```python
# Plot the Loss, Accuracy and Validation Accuracy
Model_Accuracy_Evaluation(history_mlp4, model_eva='MLP4')
```



Accuracy Analysis for MLP4 Network

1. Architecture 2: Hidden and Neurons with Dropouts.

From the Accuracy Plot shown above, it is clear that the validation accuracy remains higher than that of Training accuracy as the number of epochs increases until Epoch 5. This takes the highest value of 0.1917 at Epoch 3. The Training accuracy remains lower than that of Validation accuracy and it seems to oblige with the Loss graph. The models has clearly overcome the Overfitting problem seen in the models seen above.

In addition the Training Time for this network is 13.66 seconds which is higher than the MLP2 Network.

1.4. MLP5 for Testing with few Hidden Layers and Neurons With Dropouts.

In [36]:
```python
# Model Initialization for MLP5
model_mlp5 = models.Sequential()
model_mlp5.add(layers.Flatten(input_shape = X_train_20per.shape[1:]))
model_mlp5.add(layers.Dense(1024, activation= 'relu'))
model_mlp5.add(layers.Dropout(0.2))
model_mlp5.add(layers.Dense(512, activation= 'relu'))
model_mlp5.add(layers.Dropout(0.2))
model_mlp5.add(layers.Dense(256, activation= 'relu'))
model_mlp5.add(layers.Dropout(0.2))
model_mlp5.add(layers.Dense(128, activation= 'relu'))
model_mlp5.add(layers.Dropout(0.2))
model_mlp5.add(layers.Dense(50, activation= 'relu'))
model_mlp5.add(layers.Dropout(0.2))
model_mlp5.add(layers.Dense(10, activation='softmax'))
```

In [37]:
```python
# Model Summary
model_mlp5.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_4 (Flatten)          (None, 3072)              0
_________________________________________________________________
dense_25 (Dense)             (None, 1024)              3146752
_________________________________________________________________
dropout_12 (Dropout)         (None, 1024)              0
_________________________________________________________________
dense_26 (Dense)             (None, 512)               524800
_________________________________________________________________
dropout_13 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_27 (Dense)             (None, 256)               131328
_________________________________________________________________
dropout_14 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_28 (Dense)             (None, 128)               32896
_________________________________________________________________
dropout_15 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_29 (Dense)             (None, 50)                6450
_________________________________________________________________
dropout_16 (Dropout)         (None, 50)                0
_________________________________________________________________
dense_30 (Dense)             (None, 10)                510
=================================================================
Total params: 3,842,736
Trainable params: 3,842,736
Non-trainable params: 0
_________________________________________________________________
```

In [38]:

```
# Initialization of the Model
Batch = 32

model_mlp5.compile(optimizer='adam',
              loss = 'sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Train the Model
start = timeit.default_timer()
history_mlp5 = model_mlp5.fit(X_train_20per, y_train_20per, epochs = 5, batch_size = Batch, shuffle=
True, validation_data = (X_test, y_test))
stop = timeit.default_timer()
sys.stderr.write(" \n Training Time for MLP4 is {0} Seconds".format(stop - start))
sys.stderr.flush()
```

```
Epoch 1/5
313/313 [==============================] - 3s 10ms/step - loss: 2.2889 - accuracy: 0.1363 - val_loss:
2.1331 - val_accuracy: 0.1696
Epoch 2/5
313/313 [==============================] - 3s 10ms/step - loss: 2.1335 - accuracy: 0.1806 - val_loss:
2.0759 - val_accuracy: 0.2183
Epoch 3/5
313/313 [==============================] - 3s 10ms/step - loss: 2.0587 - accuracy: 0.2097 - val_loss:
1.9756 - val_accuracy: 0.2593
Epoch 4/5
313/313 [==============================] - 3s 10ms/step - loss: 2.0259 - accuracy: 0.2217 - val_loss:
1.9561 - val_accuracy: 0.2675
Epoch 5/5
313/313 [==============================] - 3s 10ms/step - loss: 1.9914 - accuracy: 0.2437 - val_loss:
1.9266 - val_accuracy: 0.2636
```

```
Training Time for MLP4 is 16.321659884000837 Seconds
```

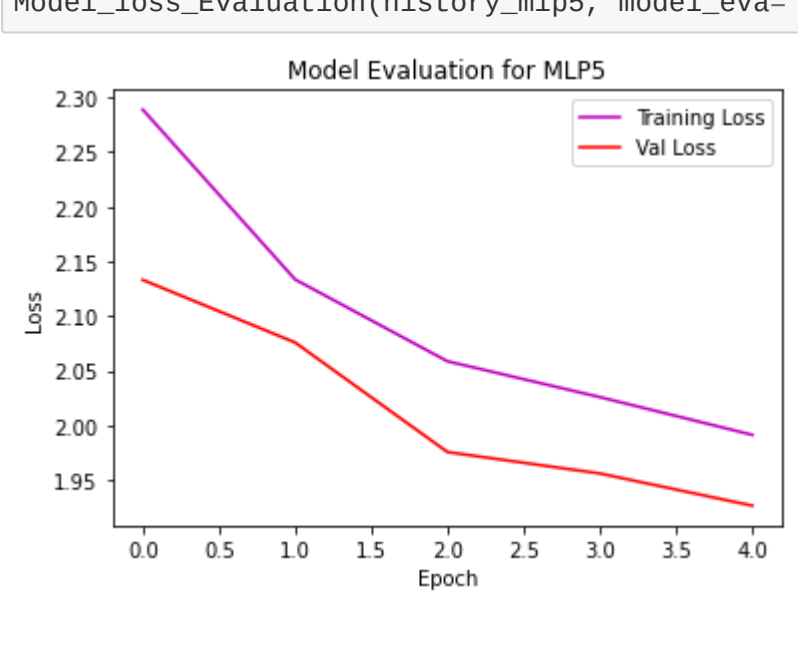
In [39]:

```
# Test the Model on the Test Dataset
test_loss_mlp5, test_acc_mlp5 = model_mlp5.evaluate(X_test,  y_test, verbose =2)
```

```
313/313 - 1s - loss: 1.9266 - accuracy: 0.2636
```

In [40]:

```
# Plot the Loss, Accuracy and Validation Accuracy
Model_loss_Evaluation(history_mlp5, model_eva='MLP5')
```



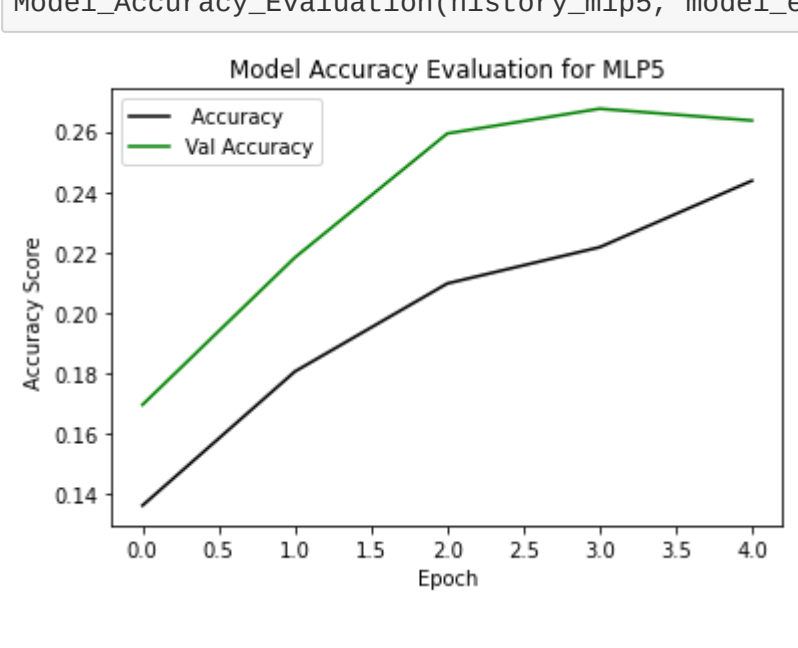
Loss Analysis for MLP5 Network

1. Architecture 5: Less Number of Hidden Layers and Neurons with Dropouts and Relu as Activation Function.

This architecture is considered to analyse how the perfomance varies when the number of hidden layer and neuron is reduced along with 'Relu' as Activation function. In the above plot, it is evident that the Validation loss is always lower than that of Training loss for the whole training process. In addition, the patter of decrease is similar for both the losses apart from small differences. The lowest loss value for Validation and Training loss is achieved at Epoch 5 which is 1.9144 and 1.9266 respectively. This loss values are smaller than the above 3 models.

In [41]:

```
# Plot the Loss, Accuracy and Validation Accuracy
Model_Accuracy_Evaluation(history_mlp5, model_eva='MLP5')
```



Accuracy Analysis for MLP5 Network

1. Architecture 5: Less Number of Hidden Layers and Neurons with Dropouts and Relu as Activation Function.

This architecture is considered to analyse how the perfomance varies when the number of hidden layer and neuron is reduced along with 'Relu' as Activation function. The Validation accuracy is always greater than Training Accuracy the highest value for Validation Accuracy is 0.2675 achieved at Epoch 5. The training accuracy is increases and at Epoch 5 it get the highest value of 0.2437 which is lower than that of Validation Accuracy. This is slightly better model as it doesn't overfitt.

2. CNN1

In [42]:

```
# Model for CNN1

model_cnn1 = models.Sequential()
model_cnn1.add(layers.Conv2D(64, (3,3), activation= 'relu', input_shape=(32, 32, 3)))
model_cnn1.add(layers.Conv2D(64, (3,3), activation= 'relu' ))
model_cnn1.add(layers.Flatten())
model_cnn1.add(layers.Dense(512, activation='sigmoid'))
model_cnn1.add(layers.Dense(512, activation='sigmoid'))
model_cnn1.add(layers.Dense(10, activation= 'softmax'))
```

In [43]:

```
# Model Summary
model_cnn1.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 30, 30, 64)        1792      
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 28, 28, 64)        36928     
_________________________________________________________________
flatten_5 (Flatten)          (None, 50176)             0         
_________________________________________________________________
dense_31 (Dense)             (None, 512)               25690624  
_________________________________________________________________
dense_32 (Dense)             (None, 512)               262656    
_________________________________________________________________
dense_33 (Dense)             (None, 10)                5130      
=================================================================
Total params: 25,997,130
Trainable params: 25,997,130
Non-trainable params: 0
_________________________________________________________________
```

In [44]:

```
# Initialization of the Model
Batch = 32

# Compile the Model
model_cnn1.compile(optimizer='adam',
              loss = 'sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Train the Model
start = timeit.default_timer()
history_cnn1 = model_cnn1.fit(X_train_20per, y_train_20per, epochs = 5, batch_size = 32, validation_
data = (X_test, y_test))
stop = timeit.default_timer()
sys.stderr.write(" \n Training Time for CNN1 is {0} Seconds".format(stop - start))
sys.stderr.flush()
```

```
Epoch 1/5
313/313 [==============================] - 9s 27ms/step - loss: 1.7284 - accuracy: 0.3616 - val_loss:
1.4953 - val_accuracy: 0.4512
Epoch 2/5
313/313 [==============================] - 8s 27ms/step - loss: 1.3182 - accuracy: 0.5271 - val_loss:
1.3228 - val_accuracy: 0.5230
Epoch 3/5
313/313 [==============================] - 8s 27ms/step - loss: 0.9673 - accuracy: 0.6595 - val_loss:
1.3613 - val_accuracy: 0.5335
Epoch 4/5
313/313 [==============================] - 8s 27ms/step - loss: 0.5694 - accuracy: 0.8122 - val_loss:
1.4072 - val_accuracy: 0.5538
Epoch 5/5
313/313 [==============================] - 8s 27ms/step - loss: 0.2088 - accuracy: 0.9434 - val_loss:
1.5956 - val_accuracy: 0.5542
```

```
Training Time for CNN1 is 43.25438329500139 Seconds
```

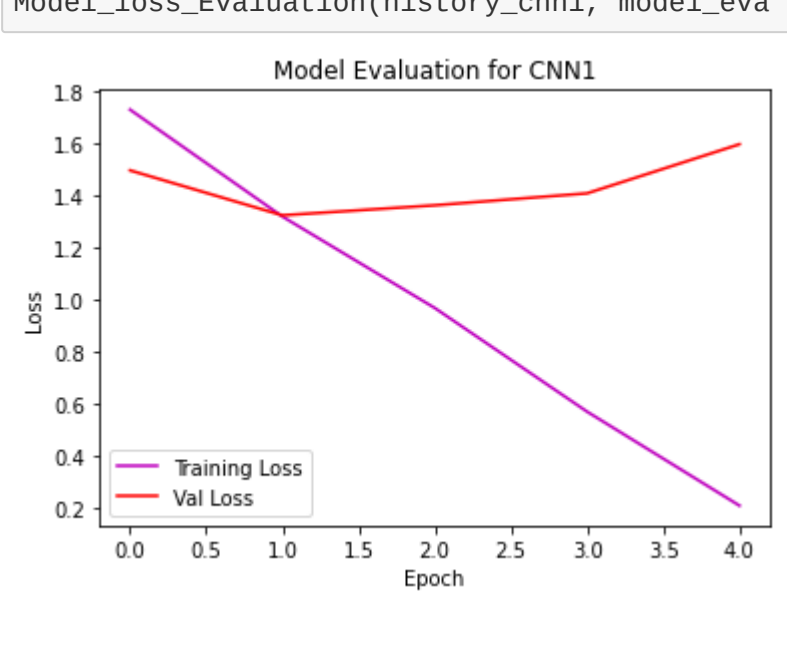
In [45]:

```
# Test the Model on the Test Dataset
test_loss, test_acc = model_cnn1.evaluate(X_test,  y_test, verbose=2)
```

```
313/313 - 2s - loss: 1.5956 - accuracy: 0.5542
```

In [46]:

```
# Evaluation Plot for CNN1
Model_loss_Evaluation(history_cnn1, model_eva = "CNN1")
```



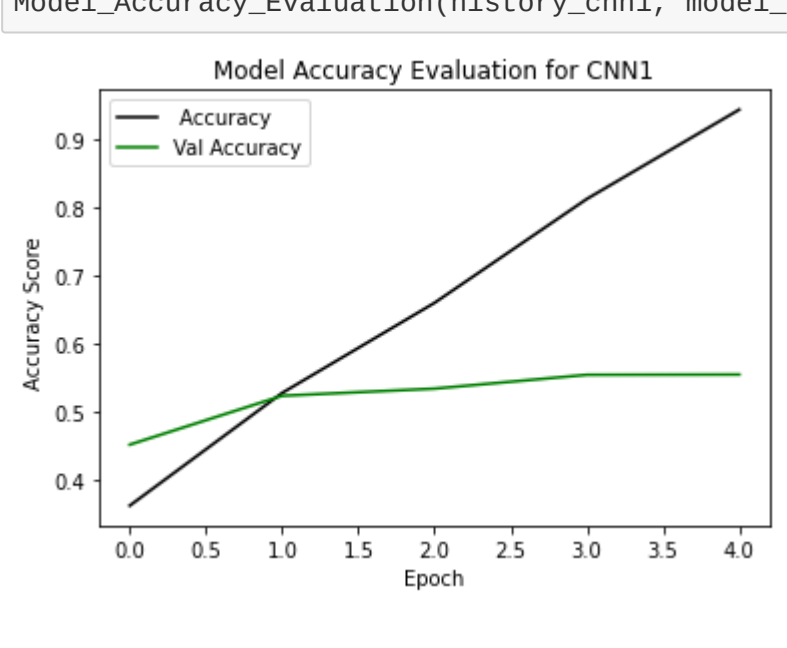
Loss Analysis for CNN1 Network

1. Architecture 1: 64x64x512x512x10 as per the Assignemnt pdf.

From the Loss Plot shown above, it is clear that the Training Loss decreases linearly as the number of epochs increases and takes the lowest value of 0.2088 at Epoch 5. The decrease in the Training loss is steep without any oscillations. The validation loss is higher than that of Training loss from Epoch 2 onwards as evident from the graph. The value of Validation loss increases after Epoch 2 which shows that the model overfits.

In [47]:

```
# Evaluation Plot for CNN1
Model_Accuracy_Evaluation(history_cnn1, model_eva = "CNN1")
```



Accuracy Analysis for CNN1 Network

2. Architecture 1: 64x64x512x512x10 as per the Assignemnt pdf.

From the Accuracy Plot shown above, it is clear that the Trainig accuracy increases almost linearly and it reach a value of 0.9434 at Epoch 5 whereas the accuracy score for validation set is quite low at 0.5542 at Epoch 5. In comparision to the MLPs the validation accuracy score is better by almost 20%. In addition, as the number of Epochs increases the validation score starts to oscillates and this value never crosses more than 62%. One important point to note is that the Training and validation accuracy at Epoch 5 differs by quite a margin which shows that the model is overfitting.

In addition, the time taken to train this model is 43.25 second which higher than that of MLP1.

3. CNN2

In [48]:

```
# Model for CNN2

model_cnn2 = models.Sequential()
model_cnn2.add(layers.Conv2D(64, (3,3), activation= 'relu', input_shape=(32, 32, 3)))
model_cnn2.add(layers.MaxPooling2D((2, 2)))
model_cnn2.add(layers.Conv2D(64, (3,3), activation= 'relu' ))
model_cnn2.add(layers.MaxPooling2D((2, 2)))
model_cnn2.add(layers.Flatten())
model_cnn2.add(layers.Dense(512, activation='sigmoid'))
model_cnn2.add(layers.Dropout(0.2))
model_cnn2.add(layers.Dense(512, activation='sigmoid'))
model_cnn2.add(layers.Dropout(0.2))
model_cnn2.add(layers.Dense(10, activation= 'softmax'))
```

In [49]:

```
# Summary of the CNN2 Model
model_cnn2.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_2 (Conv2D)            (None, 30, 30, 64)        1792      
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 15, 15, 64)        0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 13, 13, 64)        36928     
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 64)          0         
_________________________________________________________________
flatten_6 (Flatten)          (None, 2304)              0         
_________________________________________________________________
dense_34 (Dense)             (None, 512)               1180160   
_________________________________________________________________
dropout_17 (Dropout)         (None, 512)               0         
_________________________________________________________________
dense_35 (Dense)             (None, 512)               262656    
_________________________________________________________________
dropout_18 (Dropout)         (None, 512)               0         
_________________________________________________________________
dense_36 (Dense)             (None, 10)                5130      
=================================================================
Total params: 1,486,666
Trainable params: 1,486,666
Non-trainable params: 0
_________________________________________________________________
```

In [50]:

```
# Initialization of the Model
Batch = 32

# Compile the Model
model_cnn2.compile(optimizer='adam',
              loss = 'sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Train the Model
start = timeit.default_timer()
history_cnn2 = model_cnn2.fit(X_train_20per, y_train_20per, epochs = 5, batch_size = Batch, shuffle=
True, validation_data = (X_test, y_test))
stop = timeit.default_timer()
sys.stderr.write(" \n Training Time for CNN2 is {0} Seconds".format(stop - start))
sys.stderr.flush()
```

```
Epoch 1/5
313/313 [==============================] - 4s 11ms/step - loss: 1.9736 - accuracy: 0.2640 - val_loss:
1.6824 - val_accuracy: 0.3734
Epoch 2/5
313/313 [==============================] - 4s 11ms/step - loss: 1.5759 - accuracy: 0.4204 - val_loss:
1.6137 - val_accuracy: 0.4244
Epoch 3/5
313/313 [==============================] - 4s 12ms/step - loss: 1.4092 - accuracy: 0.4849 - val_loss:
1.4830 - val_accuracy: 0.4644
Epoch 4/5
313/313 [==============================] - 3s 11ms/step - loss: 1.2916 - accuracy: 0.5344 - val_loss:
1.3306 - val_accuracy: 0.5145
Epoch 5/5
313/313 [==============================] - 3s 11ms/step - loss: 1.1722 - accuracy: 0.5861 - val_loss:
1.2586 - val_accuracy: 0.5538
```

```
Training Time for CNN2 is 18.109348484000293 Seconds
```

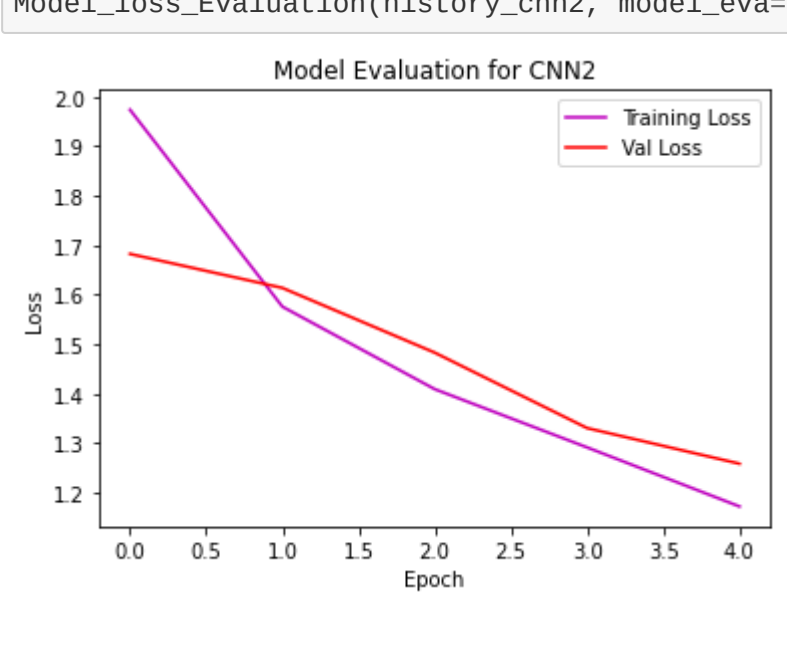
In [51]:

```
# Test the Model on the Test Dataset
test_loss_cnn2, test_acc_cnn2 = model_cnn2.evaluate(X_test,  y_test, verbose=2)
```

```
313/313 - 1s - loss: 1.2586 - accuracy: 0.5538
```

In [52]:

```
# Plot the Loss, Accuracy and Validation Accuracy
Model_loss_Evaluation(history_cnn2, model_eva='CNN2')
```



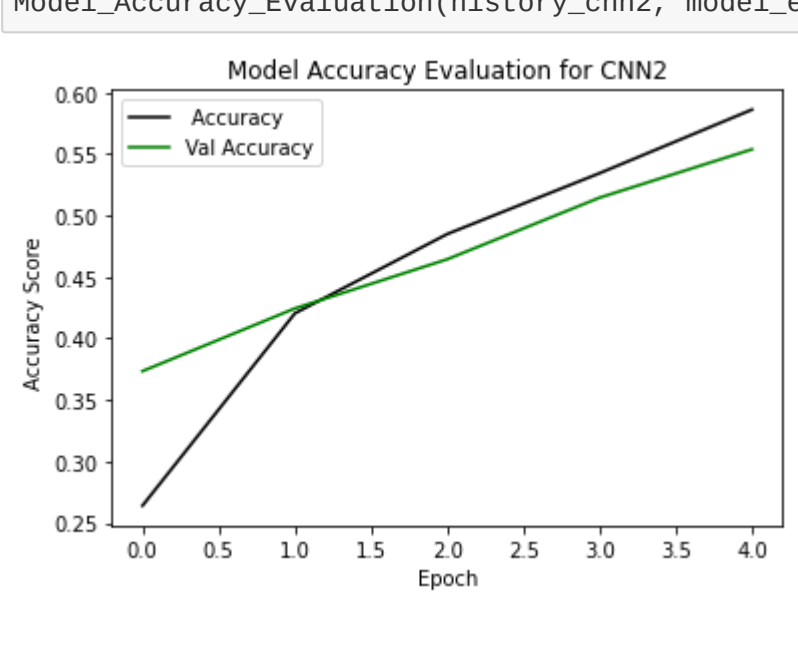
Loss Analysis for CNN2 Network

1. Architecture 2: 64x64x512x512x10 with Max pooling and dropouts as per the Assignemnt pdf.

From the Loss Plot shown above it is clear that the validation Loss and Trainig Loss decreases smoothly as the number of epochs increases. The Training loss initially higher than that of Validation Loss and as the number of Epoch increases the loss also decreases and after Epoch 2 the loss in both Training and Validation is quite similar which was not the case in that of CNN1. At Epoch 5 the Training and Validation loss is 1.1722 and 1.2586 respectively which is different from that of CNN1.

In [53]:

```
# Evaluation Plot for CNN2
Model_Accuracy_Evaluation(history_cnn2, model_eva='CNN2')
```



Accuracy Analysis for CNN2 Network

1. Architecture 2: 64x64x512x512x10 as per the Assignemnt pdf.

From the Accuracy plot it is clear that the accuracy score for both Training and Validation increases as the number of Epochs increases. The Training accuracy is approximately 0.5861 at Epoch 5 and the accuracy score of validation set is 0.5538 which is closer. This shows us that the model doesn't overfit. However, when the number of Epochs increases the Validation score increases and becomes stagnant after some point. The performance of this model is better than that of MLP1 and CNN1 in terms of Accuracy Score and Loss score.

The Training time for the network is 18.11 seconds which is lower than that of CNN1. Therefore it is a better model than CNN1.

In [53]:

Comments On the Results.

1. In the Preprocessing step the image pixels are divided by 255 to normalize the pixels of the images.

2. The output layer is a Dense Layer with 10 Neurons as the number of different Class labels are 10. In addition, Sparse Cross Entropy is used as loss function as there is no need to convert the class labels to one hot encoding and the classifications still works in a similar way. In addition, Sparse Encoding uses less time and saves memory as it uses single integer from the class rather than the whole vector in One hot encoding. Thus it Computationally efficient than that of Cross Entropy therefore in this design Sparse Cross Entropy is used.

3. When the Number of Hidden Layer, Neurons and Epochs is increased for the MLP as shown in MLP2, MLP3, MLP4 and MLP5 the Accuracy Score varies and depending on the network it changes. The performance of MLP2 is lower than that of MLP1 as it is evident from their respective graphs. In addition a detailed analysis is done above for the Models.

4. There are 7 Networks that are modeled above, The performance of all the MLPs is lower than both the CNNs (CNN1, CNN2). In drawing out the comparision between CNNs, CNN2 performs slightly better than that of CNN1. The major different between the two network is the addition of Pooling layer and the Dropout which seems to be affecting the accuracy score for the CNN2.

5. The plot along with comparision between Training and Validation loss/ accuracy is done above. In addition MLP1 training time is lower then CNN2 and CNN1 takes the longest to train. One reason for CNN2 to take lower training time than that of CNN1 is because CNN2 uses a Pooling layer that effects its training time for CNN2. If we increase the number of Epochs for each of the CNNs the accuracy Score increases very slowly for CNN2. The Training Accuracy is higher than that of Validation accuracy and after running for number of Epochs the Validation Accuracy become almost Stagnant. In Comparision, when the number of Epochs is increases for CNN1 the Training accuracy reached 90% but the Validation accuracy remains below 55% and doesn't increaes further even if the number of Epochs is increased. Therefore, it is clear that for both the networks if we keep the number of Epochs to be very large the accuracy score will reach a stagnant point.

6. In order to improve the CNNs more number of filter can be added to take more features from the images which will help in training the network and increase the Validation accuracy for the network. In addition we can experiment with different pooling layers and methods to see which works best for the network and the images. In addition, a experiment with different activation function can also give good result. This changes can help us understand the network more and significantly imporve the performance for the Network.