ECE 657 Assignment 3 – Group 80

Rumna Samanta – 20883387

rsamanta@uwaterloo.ca

Juhi Vasudev Bachani – 20979706

jvbachani@uwaterloo.ca

Frank Huang Huang – 20433010

g7huang@uwaterloo.ca

Problem 1:

CNN for Image Classification

Using the API we will load the dataset of the CIFAR10.

Importing Libraries

In [1]:
```
import tensorflow as tf
from keras.datasets import cifar10
from sklearn.model_selection import train_test_split
from tensorflow.keras import datasets, layers, models
from tensorflow.keras.layers import Dropout, Activation, Flatten, Conv2D, MaxPooling2D
import matplotlib.pyplot as plt
from keras import Sequential
```

Loading Dataset and normalizing pixel values between 0 to 1

In [2]:
```
(train_X, train_Y), (validation_X, validation_Y) = cifar10.load_data()
train_X, validation_X = train_X/255.0, validation_X/255.0         #Normalizing values between 0 to 1 rather than 0 to 255.
print("The shape of training set is :",train_X.shape)
print("The shape of Validation set is :",validation_X.shape)
#Using test set as validation.
```

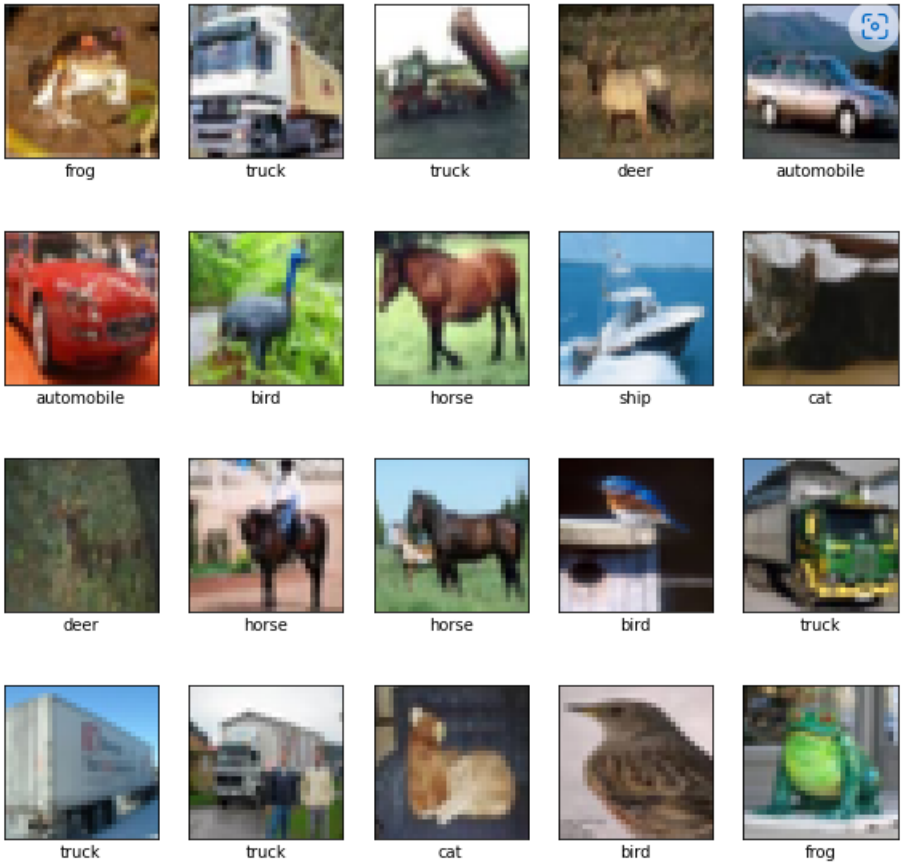
```
The shape of training set is : (50000, 32, 32, 3)
The shape of Validation set is : (10000, 32, 32, 3)
```

There are 10 classes names 'airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck' in CIFAR-10. There are 60,000 32x32 (pixels) colored images in CIFAR-10 dataset. Out of those 60,000 images, 50,000 are used as training dataset and 10,000 as testing dataset. We will plot the 20 Images to verify the dataset.

In [3]:
```
#10 classes are described below:
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer','dog', 'frog', 'horse', 'ship', 'truck']
```

In [4]:
```
#Generating the plot to verify the dataset.
plt.figure(figsize=(10,10))
for j in range(20):
    plt.subplot(4,5,j+1)
    plt.grid(False)
    plt.yticks([])
    plt.xticks([])
    plt.imshow(train_X[j])
    plt.xlabel(class_names[train_Y[j][0]])
plt.show()
```

Output:



We are Splitting the training dataset randomly in 80-20 Ratios and will use 20% as the new train dataset.

```
In [5]: #Randomly Splitting the train set in 80-20 ratio and will use the 20% as new train set.
        train_X1,test_X1,train_Y1,test_Y = train_test_split(train_X,train_Y,train_size = 0.2,random_state = 42)

In [6]: print("After split the shape of train set is:",train_X1.shape)

        After split the shape of train set is: (10000, 32, 32, 3)
```

1. MLP:

Multi-layer perceptron has 2 fully connected layers with 512 neurons each and is using sigmoid as an activation function. We will use output neurons as 10 in the last dense layer as we have 10 classes and using the SoftMax activation function as it can perform multiclass classification.

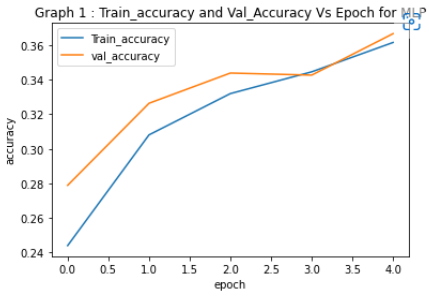
In [7]:
```
model_mlp = models.Sequential()
model_mlp.add(layers.Flatten(input_shape=(32, 32, 3)))
#Using Flatten layer to convert 3D data to 1D
model_mlp.add(layers.Dense(512,activation = 'sigmoid'))
model_mlp.add(layers.Dense(512,activation = 'sigmoid'))
model_mlp.add(layers.Dense(10,activation = 'softmax'))
#Using 10 neurons as output in last dense layer as we have 10 classes.
#Using a softmax function as an activation function because it can perform multiclass classification
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 [8]:
```
model_mlp.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
#Fitting the model using 32 batch size and 5 epoch as asked in question.
history_mlp = model_mlp.fit(train_X1, train_Y1, batch_size = 32, epochs=5, validation_data=(validation_X, validation_Y))
```

```
Epoch 1/5
313/313 [==============================] - 4s 12ms/step - loss: 2.0404 - accuracy: 0.2440 - val_loss: 1.9381 - val_accuracy:
0.2789
Epoch 2/5
313/313 [==============================] - 4s 12ms/step - loss: 1.8972 - accuracy: 0.3081 - val_loss: 1.8689 - val_accuracy:
0.3264
Epoch 3/5
313/313 [==============================] - 4s 12ms/step - loss: 1.8366 - accuracy: 0.3320 - val_loss: 1.8025 - val_accuracy:
0.3439
Epoch 4/5
313/313 [==============================] - 4s 12ms/step - loss: 1.7969 - accuracy: 0.3446 - val_loss: 1.8246 - val_accuracy:
0.3427
Epoch 5/5
313/313 [==============================] - 4s 12ms/step - loss: 1.7557 - accuracy: 0.3616 - val_loss: 1.7650 - val_accuracy:
0.3667
```

In [30]:
```
plt.plot(history_mlp.history['accuracy'], label = 'Train_accuracy')
plt.plot(history_mlp.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('epoch')
plt.ylabel('accuracy')
plt.title('Graph 1 : Train_accuracy and Val_Accuracy Vs Epoch for MLP')
plt.legend(loc = 'upper left')
plt.show()
```



In the initial epoch, Validation accuracy is more than training accuracy. But, the training accuracy increased with the increase in epoch and was almost similar to the validation accuracy in the last.

1.1) Changing the number of Layers:

We will be changing the number of hidden layers from 1 to 5 to see the impact on the training and validation accuracy. We are keeping the number of neurons fixed here to 512 in each layer.

In [10]:
```
layer_acc =[]
layer_val_acc = []

print("               1 Hidden Layer                 ")
layer1=models.Sequential()
layer1.add(layers.Flatten(input_shape=(32, 32, 3)))
layer1.add(layers.Dense(512,activation = 'sigmoid'))
layer1.add(layers.Dense(10,activation = 'softmax'))
layer1.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
h1 = layer1.fit(train_X1, train_Y1, batch_size = 32, epochs=5, validation_data=(validation_X, validation_Y))
loss,accuracy = layer1.evaluate(train_X1,train_Y1)
val_loss,val_accuracy = layer1.evaluate(validation_X,validation_Y)
layer_acc.append(accuracy)
layer_val_acc.append(val_accuracy)
```

```
               1 Hidden Layer
Epoch 1/5
313/313 [==============================] - 4s 11ms/step - loss: 2.0638 - accuracy: 0.2529 - val_loss: 1.8933 - val_accuracy:
0.3291
Epoch 2/5
313/313 [==============================] - 3s 10ms/step - loss: 1.8839 - accuracy: 0.3233 - val_loss: 1.8595 - val_accuracy:
0.3271
Epoch 3/5
313/313 [==============================] - 3s 10ms/step - loss: 1.8206 - accuracy: 0.3450 - val_loss: 1.8729 - val_accuracy:
0.3296
Epoch 4/5
313/313 [==============================] - 3s 10ms/step - loss: 1.7840 - accuracy: 0.3604 - val_loss: 1.7994 - val_accuracy:
0.3613
Epoch 5/5
313/313 [==============================] - 3s 10ms/step - loss: 1.7461 - accuracy: 0.3726 - val_loss: 1.7434 - val_accuracy:
0.3750
313/313 [==============================] - 1s 2ms/step - loss: 1.6813 - accuracy: 0.4003
313/313 [==============================] - 1s 2ms/step - loss: 1.7434 - accuracy: 0.3750
```

In [11]:
```
print("                2 Hidden Layer             ")
layer2=models.Sequential()
layer2.add(layers.Flatten(input_shape=(32, 32, 3)))
layer2.add(layers.Dense(512,activation = 'sigmoid'))
layer2.add(layers.Dense(512,activation = 'sigmoid'))
layer2.add(layers.Dense(10,activation = 'softmax'))
layer2.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
h1 = layer2.fit(train_X1, train_Y1, batch_size = 32, epochs=5, validation_data=(validation_X, validation_Y))
loss,accuracy = layer2.evaluate(train_X1,train_Y1)
val_loss,val_accuracy = layer2.evaluate(validation_X,validation_Y)
layer_acc.append(accuracy)
layer_val_acc.append(val_accuracy)
```

```
               2 Hidden Layer
Epoch 1/5
313/313 [==============================] - 4s 12ms/step - loss: 2.0797 - accuracy: 0.2304 - val_loss: 1.9529 - val_accuracy:
0.2868
Epoch 2/5
313/313 [==============================] - 4s 12ms/step - loss: 1.9295 - accuracy: 0.2965 - val_loss: 1.9660 - val_accuracy:
0.2776
Epoch 3/5
313/313 [==============================] - 4s 12ms/step - loss: 1.8684 - accuracy: 0.3222 - val_loss: 1.9339 - val_accuracy:
0.2824
Epoch 4/5
313/313 [==============================] - 4s 12ms/step - loss: 1.8150 - accuracy: 0.3375 - val_loss: 1.8178 - val_accuracy:
0.3661
Epoch 5/5
313/313 [==============================] - 4s 12ms/step - loss: 1.7720 - accuracy: 0.3556 - val_loss: 1.7926 - val_accuracy:
0.3473
313/313 [==============================] - 1s 3ms/step - loss: 1.7465 - accuracy: 0.3609
313/313 [==============================] - 1s 3ms/step - loss: 1.7926 - accuracy: 0.3473
```

In [12]:

```
print("                 3 Hidden Layer              ")
layer3=models.Sequential()
layer3.add(layers.Flatten(input_shape=(32, 32, 3)))
layer3.add(layers.Dense(512,activation = 'sigmoid'))
layer3.add(layers.Dense(512,activation = 'sigmoid'))
layer3.add(layers.Dense(512,activation = 'sigmoid'))
layer3.add(layers.Dense(10,activation = 'softmax'))
layer3.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
h1 = layer3.fit(train_X1, train_Y1, batch_size = 32, epochs=5, validation_data=(validation_X, validation_Y))
loss,accuracy = layer3.evaluate(train_X1,train_Y1)
val_loss,val_accuracy = layer3.evaluate(validation_X,validation_Y)
layer_acc.append(accuracy)
layer_val_acc.append(val_accuracy)
```

```
                 3 Hidden Layer
Epoch 1/5
313/313 [==============================] - 5s 15ms/step - loss: 2.1510 - accuracy: 0.1872 - val_loss: 1.9841 - val_accuracy:
0.2638
Epoch 2/5
313/313 [==============================] - 5s 15ms/step - loss: 1.9725 - accuracy: 0.2693 - val_loss: 1.9580 - val_accuracy:
0.2695
Epoch 3/5
313/313 [==============================] - 5s 15ms/step - loss: 1.9100 - accuracy: 0.2982 - val_loss: 1.8692 - val_accuracy:
0.3091
Epoch 4/5
313/313 [==============================] - 5s 15ms/step - loss: 1.8759 - accuracy: 0.3130 - val_loss: 1.9283 - val_accuracy:
0.2924
Epoch 5/5
313/313 [==============================] - 4s 14ms/step - loss: 1.8170 - accuracy: 0.3389 - val_loss: 1.9107 - val_accuracy:
0.3227
313/313 [==============================] - 1s 4ms/step - loss: 1.8745 - accuracy: 0.3306
313/313 [==============================] - 1s 3ms/step - loss: 1.9107 - accuracy: 0.3227
```

In [13]:

```
print("                 4 Hidden Layer              ")
layer4=models.Sequential()
layer4.add(layers.Flatten(input_shape=(32, 32, 3)))
layer4.add(layers.Dense(512,activation = 'sigmoid'))
layer4.add(layers.Dense(512,activation = 'sigmoid'))
layer4.add(layers.Dense(512,activation = 'sigmoid'))
layer4.add(layers.Dense(512,activation = 'sigmoid'))
layer4.add(layers.Dense(10,activation = 'softmax'))
layer4.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
h1 = layer4.fit(train_X1, train_Y1, batch_size = 32, epochs=5, validation_data=(validation_X, validation_Y))
loss,accuracy = layer4.evaluate(train_X1,train_Y1)
val_loss,val_accuracy = layer4.evaluate(validation_X,validation_Y)
layer_acc.append(accuracy)
layer_val_acc.append(val_accuracy)
```

```
                 4 Hidden Layer
Epoch 1/5
313/313 [==============================] - 5s 16ms/step - loss: 2.1818 - accuracy: 0.1554 - val_loss: 2.1458 - val_accuracy:
0.1751
Epoch 2/5
313/313 [==============================] - 5s 16ms/step - loss: 2.0893 - accuracy: 0.1838 - val_loss: 2.1068 - val_accuracy:
0.1903
Epoch 3/5
313/313 [==============================] - 6s 19ms/step - loss: 2.0335 - accuracy: 0.2308 - val_loss: 1.9930 - val_accuracy:
0.2286
Epoch 4/5
313/313 [==============================] - 7s 23ms/step - loss: 1.9558 - accuracy: 0.2680 - val_loss: 1.9233 - val_accuracy:
0.2846
Epoch 5/5
313/313 [==============================] - 7s 21ms/step - loss: 1.9261 - accuracy: 0.2813 - val_loss: 1.8851 - val_accuracy:
0.2906
313/313 [==============================] - 3s 9ms/step - loss: 1.8625 - accuracy: 0.2980
313/313 [==============================] - 2s 7ms/step - loss: 1.8851 - accuracy: 0.2906
```

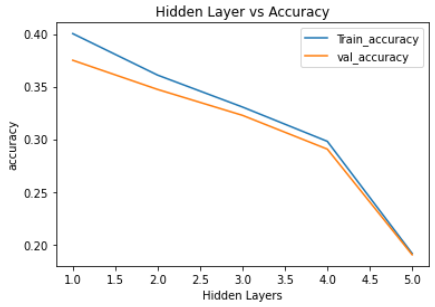
In [14]:
```
print("                5 Hidden Layer              ")
layer5=models.Sequential()
layer5.add(layers.Flatten(input_shape=(32, 32, 3)))
layer5.add(layers.Dense(512,activation = 'sigmoid'))
layer5.add(layers.Dense(512,activation = 'sigmoid'))
layer5.add(layers.Dense(512,activation = 'sigmoid'))
layer5.add(layers.Dense(512,activation = 'sigmoid'))
layer5.add(layers.Dense(512,activation = 'sigmoid'))
layer5.add(layers.Dense(10,activation = 'softmax'))
layer5.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
h1 = layer5.fit(train_X1, train_Y1, batch_size = 32, epochs=5, validation_data=(validation_X, validation_Y))
loss,accuracy = layer5.evaluate(train_X1,train_Y1)
val_loss,val_accuracy = layer5.evaluate(validation_X,validation_Y)
layer_acc.append(accuracy)
layer_val_acc.append(val_accuracy)
```

```
                  5 Hidden Layer
Epoch 1/5
313/313 [==============================] - 9s 26ms/step - loss: 2.2248 - accuracy: 0.1450 - val_loss: 2.0944 - val_accuracy:
0.1638
Epoch 2/5
313/313 [==============================] - 8s 25ms/step - loss: 2.1039 - accuracy: 0.1726 - val_loss: 2.1755 - val_accuracy:
0.1701
Epoch 3/5
313/313 [==============================] - 7s 23ms/step - loss: 2.0964 - accuracy: 0.1806 - val_loss: 2.0638 - val_accuracy:
0.2017
Epoch 4/5
313/313 [==============================] - 8s 25ms/step - loss: 2.0765 - accuracy: 0.1755 - val_loss: 2.0595 - val_accuracy:
0.2017
Epoch 5/5
313/313 [==============================] - 7s 21ms/step - loss: 2.0640 - accuracy: 0.1953 - val_loss: 2.0575 - val_accuracy:
0.1903
313/313 [==============================] - 2s 6ms/step - loss: 2.0457 - accuracy: 0.1915
313/313 [==============================] - 3s 10ms/step - loss: 2.0575 - accuracy: 0.1903
```

In [15]:
```
print("Training Accuracy with 1,2,3,4,5 Hidden Layer is :\n",layer_acc)
print("\nValidation Accuracy with 1,2,3,4,5 Hidden Layer is : \n:", layer_val_acc)
```

```
Training Accuracy with 1,2,3,4,5 Hidden Layer is :
 [0.400299996137619, 0.36090001463890076, 0.33059999346733093, 0.2980000078678131, 0.1914999932050705]

Validation Accuracy with 1,2,3,4,5 Hidden Layer is :
: [0.375, 0.3472999930381775, 0.32269999384880066, 0.2906000018119812, 0.19030000269412994]
```

In [16]:
```
plt.plot([1,2,3,4,5],layer_acc, label = 'Train_accuracy')
plt.plot([1,2,3,4,5],layer_val_acc,label = 'val_accuracy')
plt.title('Hidden Layer vs Accuracy')
plt.ylabel('accuracy')
plt.xlabel('Hidden Layers')
plt.legend(loc = 'upper right')
plt.show()
```



We can see that both training and Validation accuracy is dropping with an increase in Hidden layers. we are getting maximum accuracy for 1 hidden layer. We can also notice from the plot, that accuracy is dropping very sharply when we change 4 hidden layers to 5 hidden layers.

1.2) Impact of changing Neurons on train and validation accuracy:

We will change the number of neurons and see the impact on the training and validation set. We will keep the number of hidden layers fixed to 2.

In [17]:
```
neurons_acc = []
neurons_val_acc = []



print("                64 Neurons                 ")
n1 = models.Sequential()
n1.add(layers.Flatten(input_shape=(32, 32, 3)))
n1.add(layers.Dense(64,activation = 'sigmoid'))
n1.add(layers.Dense(64,activation = 'sigmoid'))
n1.add(layers.Dense(10,activation = 'softmax'))
n1.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
h2 = n1.fit(train_X1, train_Y1, batch_size = 32, epochs=5, validation_data=(validation_X, validation_Y))

loss,accuracy = n1.evaluate(train_X1,train_Y1)
val_loss,val_accuracy = n1.evaluate(validation_X,validation_Y)
neurons_acc.append(accuracy)
neurons_val_acc.append(val_accuracy)
```

```
                64 Neurons
Epoch 1/5
313/313 [==============================] - 1s 3ms/step - loss: 2.2012 - accuracy: 0.1712 - val_loss: 2.0859 - val_accuracy:
0.1783
Epoch 2/5
313/313 [==============================] - 1s 3ms/step - loss: 2.0417 - accuracy: 0.2364 - val_loss: 1.9890 - val_accuracy:
0.2864
Epoch 3/5
313/313 [==============================] - 1s 3ms/step - loss: 1.9716 - accuracy: 0.2751 - val_loss: 1.9864 - val_accuracy:
0.2716
Epoch 4/5
313/313 [==============================] - 1s 3ms/step - loss: 1.9284 - accuracy: 0.3055 - val_loss: 1.9108 - val_accuracy:
0.3042
Epoch 5/5
313/313 [==============================] - 1s 3ms/step - loss: 1.8927 - accuracy: 0.3192 - val_loss: 1.9030 - val_accuracy:
0.3091
313/313 [==============================] - 0s 2ms/step - loss: 1.8742 - accuracy: 0.3262
313/313 [==============================] - 0s 1ms/step - loss: 1.9030 - accuracy: 0.3091
```

In [18]:
```
print("                128 Neurons                 ")
n2 = models.Sequential()
n2.add(layers.Flatten(input_shape=(32, 32, 3)))
n2.add(layers.Dense(128,activation = 'sigmoid'))
n2.add(layers.Dense(128,activation = 'sigmoid'))
n2.add(layers.Dense(10,activation = 'softmax'))
n2.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
h2 = n2.fit(train_X1, train_Y1, batch_size = 32, epochs=5, validation_data=(validation_X, validation_Y))

loss,accuracy = n2.evaluate(train_X1,train_Y1)
val_loss,val_accuracy = n2.evaluate(validation_X,validation_Y)
neurons_acc.append(accuracy)
neurons_val_acc.append(val_accuracy)
```

```
                128 Neurons
Epoch 1/5
313/313 [==============================] - 2s 5ms/step - loss: 2.1217 - accuracy: 0.2125 - val_loss: 1.9981 - val_accuracy:
0.2463
Epoch 2/5
313/313 [==============================] - 1s 4ms/step - loss: 1.9475 - accuracy: 0.2957 - val_loss: 1.9192 - val_accuracy:
0.2882
Epoch 3/5
313/313 [==============================] - 1s 4ms/step - loss: 1.8782 - accuracy: 0.3238 - val_loss: 1.8587 - val_accuracy:
0.3358
Epoch 4/5
313/313 [==============================] - 1s 4ms/step - loss: 1.8521 - accuracy: 0.3313 - val_loss: 1.8605 - val_accuracy:
0.3179
Epoch 5/5
313/313 [==============================] - 1s 4ms/step - loss: 1.8110 - accuracy: 0.3495 - val_loss: 1.8280 - val_accuracy:
0.3349
313/313 [==============================] - 1s 2ms/step - loss: 1.7868 - accuracy: 0.3530
313/313 [==============================] - 1s 2ms/step - loss: 1.8280 - accuracy: 0.3349
```

In [19]:
```
print("                256 Neurons                ")
n3 = models.Sequential()
n3.add(layers.Flatten(input_shape=(32, 32, 3)))
n3.add(layers.Dense(256,activation = 'sigmoid'))
n3.add(layers.Dense(256,activation = 'sigmoid'))
n3.add(layers.Dense(10,activation = 'softmax'))
n3.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
h2 = n3.fit(train_X1, train_Y1, batch_size = 32, epochs=5, validation_data=(validation_X, validation_Y))

loss,accuracy = n3.evaluate(train_X1,train_Y1)
val_loss,val_accuracy = n3.evaluate(validation_X,validation_Y)
neurons_acc.append(accuracy)
neurons_val_acc.append(val_accuracy)
```

```
                256 Neurons
Epoch 1/5
313/313 [==============================] - 2s 7ms/step - loss: 2.0708 - accuracy: 0.2282 - val_loss: 1.9350 - val_accuracy:
0.2882
Epoch 2/5
313/313 [==============================] - 2s 6ms/step - loss: 1.9106 - accuracy: 0.3052 - val_loss: 1.8871 - val_accuracy:
0.3205
Epoch 3/5
313/313 [==============================] - 2s 6ms/step - loss: 1.8567 - accuracy: 0.3261 - val_loss: 1.8633 - val_accuracy:
0.3150
Epoch 4/5
313/313 [==============================] - 2s 6ms/step - loss: 1.8140 - accuracy: 0.3424 - val_loss: 1.8015 - val_accuracy:
0.3454
Epoch 5/5
313/313 [==============================] - 2s 6ms/step - loss: 1.7720 - accuracy: 0.3589 - val_loss: 1.8994 - val_accuracy:
0.3372
313/313 [==============================] - 1s 2ms/step - loss: 1.8399 - accuracy: 0.3487
313/313 [==============================] - 1s 2ms/step - loss: 1.8994 - accuracy: 0.3372
```

In [20]:
```
print("                512 Neurons                ")
n4 = models.Sequential()
n4.add(layers.Flatten(input_shape=(32, 32, 3)))
n4.add(layers.Dense(512,activation = 'sigmoid'))
n4.add(layers.Dense(512,activation = 'sigmoid'))
n4.add(layers.Dense(10,activation = 'softmax'))
n4.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
h4 = n4.fit(train_X1, train_Y1, batch_size = 32, epochs=5, validation_data=(validation_X, validation_Y))

loss,accuracy = n4.evaluate(train_X1,train_Y1)
val_loss,val_accuracy = n4.evaluate(validation_X,validation_Y)
neurons_acc.append(accuracy)
neurons_val_acc.append(val_accuracy)
```

```
                512 Neurons
Epoch 1/5
313/313 [==============================] - 5s 15ms/step - loss: 2.0765 - accuracy: 0.2388 - val_loss: 1.9422 - val_accuracy:
0.2988
Epoch 2/5
313/313 [==============================] - 5s 15ms/step - loss: 1.9226 - accuracy: 0.3021 - val_loss: 1.8554 - val_accuracy:
0.3151
Epoch 3/5
313/313 [==============================] - 4s 14ms/step - loss: 1.8621 - accuracy: 0.3202 - val_loss: 1.8200 - val_accuracy:
0.3410
Epoch 4/5
313/313 [==============================] - 5s 15ms/step - loss: 1.8130 - accuracy: 0.3414 - val_loss: 1.8207 - val_accuracy:
0.3271
Epoch 5/5
313/313 [==============================] - 5s 17ms/step - loss: 1.7672 - accuracy: 0.3572 - val_loss: 1.7781 - val_accuracy:
0.3545
313/313 [==============================] - 2s 5ms/step - loss: 1.7281 - accuracy: 0.3706
313/313 [==============================] - 2s 5ms/step - loss: 1.7781 - accuracy: 0.3545
```

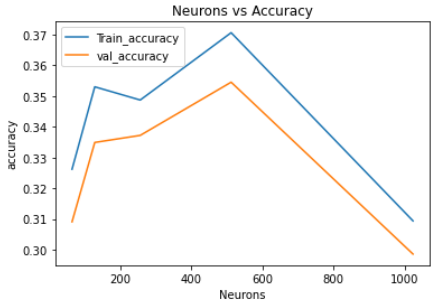
In [21]:
```
print("                1024 Neurons                ")
n5 = models.Sequential()
n5.add(layers.Flatten(input_shape=(32, 32, 3)))
n5.add(layers.Dense(1024,activation = 'sigmoid'))
n5.add(layers.Dense(1024,activation = 'sigmoid'))
n5.add(layers.Dense(10,activation = 'softmax'))
n5.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
h5 = n5.fit(train_X1, train_Y1, batch_size = 32, epochs=5, validation_data=(validation_X, validation_Y))

loss,accuracy = n5.evaluate(train_X1,train_Y1)
val_loss,val_accuracy = n5.evaluate(validation_X,validation_Y)
neurons_acc.append(accuracy)
neurons_val_acc.append(val_accuracy)
```

```
                1024 Neurons
Epoch 1/5
313/313 [==============================] - 9s 27ms/step - loss: 2.1244 - accuracy: 0.2189 - val_loss: 1.9910 - val_accuracy:
0.2602
Epoch 2/5
313/313 [==============================] - 9s 29ms/step - loss: 1.9364 - accuracy: 0.2947 - val_loss: 1.9259 - val_accuracy:
0.2988
Epoch 3/5
313/313 [==============================] - 9s 29ms/step - loss: 1.8861 - accuracy: 0.3141 - val_loss: 1.8367 - val_accuracy:
0.3424
Epoch 4/5
313/313 [==============================] - 9s 28ms/step - loss: 1.8236 - accuracy: 0.3367 - val_loss: 1.8688 - val_accuracy:
0.2920
Epoch 5/5
313/313 [==============================] - 9s 27ms/step - loss: 1.8029 - accuracy: 0.3392 - val_loss: 1.8821 - val_accuracy:
0.2986
313/313 [==============================] - 2s 6ms/step - loss: 1.8440 - accuracy: 0.3094
313/313 [==============================] - 2s 6ms/step - loss: 1.8821 - accuracy: 0.2986
```

In [22]:
```
print("The impact of changing neurons 64,128,256,512,1024 on Train_Accuracy is :\n",neurons_acc)
print("\nThe impact of changing neurons 64,128,256,512,1024 on Validation_Accuracy is :\n",neurons_val_acc)
```

```
The impact of changing neurons 64,128,256,512,1024 on Train_Accuracy is :
 [0.326200008392334, 0.3529999852180481, 0.34869998693466187, 0.37059998512268066, 0.3093999922275543]

The impact of changing neurons 64,128,256,512,1024 on Validation_Accuracy is :
 [0.3091000020503998, 0.33489999175071716, 0.33719998598098755, 0.3544999957084656, 0.2985999882221222]
```

In [23]:
```
plt.plot([64,128,256,512,1024],neurons_acc, label = 'Train_accuracy')
plt.plot([64,128,256,512,1024],neurons_val_acc,label = 'val_accuracy')
plt.title('Neurons vs Accuracy')
plt.ylabel('accuracy')
plt.xlabel('Neurons')
plt.legend(loc = 'upper left')
plt.show()
```



So, the plot shows that we are getting maximum accuracy for both train and validation at 512 neurons. Also, after 512 neurons, the accuracy is reduced.

2) CNN1:

CNN1 has 2 Conv Layer which has 64 filters, size of 3*3, and ReLu activation function each. The shape of the input is (image height, image width, color channels). There are 3 color channels that refer to R, G, and B (R=Red, G=Green, B=Blue), so the input shape is (32,32,3). before Applying the dense layer, we will flatten the 3D data to 1D and the will apply 2 dense layers with 512 neurons and sigmoid activation function each. the last dense layer has 10 outputs as we have 10 classes in the CIFAR-10 Dataset.

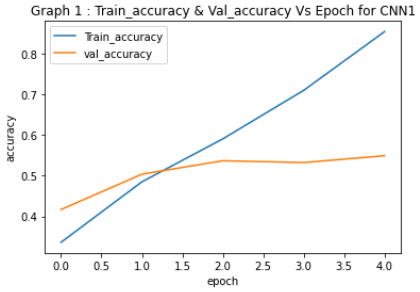
In [24]:
```
model = models.Sequential()
model.add(layers.Conv2D(64, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten())
#Using Flatten layer to convert 3D data to 1D
model.add(layers.Dense(512,activation = 'sigmoid'))
model.add(layers.Dense(512,activation = 'sigmoid'))
model.add(layers.Dense(10,activation = 'softmax'))
#Using 10 neurons as output in last dense layer as we have 10 classes.
#Using a softmax function as an activation function because it can perform multiclass classification
model.summary()
```

```
Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 64)        1792

 conv2d_1 (Conv2D)           (None, 28, 28, 64)        36928

 flatten_11 (Flatten)        (None, 50176)             0

 dense_38 (Dense)            (None, 512)               25690624

 dense_39 (Dense)            (None, 512)               262656

 dense_40 (Dense)            (None, 10)                5130

=================================================================
Total params: 25,997,130
Trainable params: 25,997,130
Non-trainable params: 0
_________________________________________________________________
```

In [25]:
```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
#Fitting the model using 32 batch size and 5 epoch as asked in question.
history = model.fit(train_X1, train_Y1, batch_size = 32, epochs=5, validation_data=(validation_X, validation_Y))
```

```
Epoch 1/5
313/313 [==============================] - 74s 237ms/step - loss: 1.8120 - accuracy: 0.3364 - val_loss: 1.5723 - val_accurac
y: 0.4170
Epoch 2/5
313/313 [==============================] - 72s 231ms/step - loss: 1.4066 - accuracy: 0.4851 - val_loss: 1.3705 - val_accurac
y: 0.5038
Epoch 3/5
313/313 [==============================] - 75s 241ms/step - loss: 1.1284 - accuracy: 0.5906 - val_loss: 1.3121 - val_accurac
y: 0.5369
Epoch 4/5
313/313 [==============================] - 75s 238ms/step - loss: 0.8165 - accuracy: 0.7098 - val_loss: 1.3921 - val_accurac
y: 0.5324
Epoch 5/5
313/313 [==============================] - 76s 242ms/step - loss: 0.4584 - accuracy: 0.8542 - val_loss: 1.4876 - val_accurac
y: 0.5492
```

In [26]:
```
plt.plot(history.history['accuracy'], label = 'Train_accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('epoch')
plt.ylabel('accuracy')
plt.title('Graph 1 : Train_accuracy & Val_accuracy Vs Epoch for CNN1')
plt.legend(loc = 'upper left')
plt.show()
```



After applying the CNN1 model, it is evident from the graph that the accuracy of the training dataset is increasing with each epoch but that is not the case with validation accuracy. Validation accuracy

hardly has any changes after 2 Nd epoch. We can see that compare to MLP, CNN1 has better accuracy performance for training but not for validation. We can see that CNN1 is showing overfitting of data.

3) CNN2:

CNN2 uses 2 Conv Layer with 64 filters and Relu activation Layer each layer followed by 2x2 Max pooling layer and then output of max-pooling layer goes to 2 Hidden layers followed by dropout layer with 0.2 dropout rate.

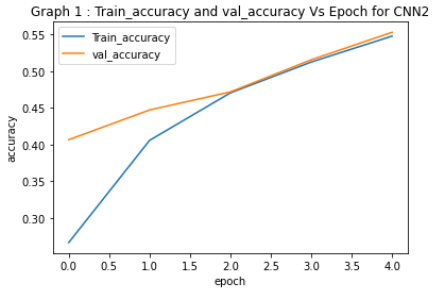
In [27]:
```
model2 = models.Sequential()
model2.add(layers.Conv2D(64, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model2.add(MaxPooling2D(pool_size=(2,2)))
model2.add(layers.Conv2D(64, (3, 3), activation='relu'))
model2.add(MaxPooling2D(pool_size=(2,2)))
model2.add(layers.Flatten())        #Flattening 3D to 1D
model2.add(layers.Dense(512,activation = 'sigmoid'))
model2.add(Dropout(0.2))            # Using DropOut with 0.2 dropout rate to avoid Overfitting
model2.add(layers.Dense(512,activation = 'sigmoid'))
model2.add(Dropout(0.2))
model2.add(layers.Dense(10,activation = 'softmax'))   #Using 10 as output because we have 10 Classes.
model2.summary()
```

```
Model: "sequential_12"
_________________________________________________________________
 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 (MaxPooling  (None, 6, 6, 64)         0
 2D)

 flatten_12 (Flatten)        (None, 2304)              0

 dense_41 (Dense)            (None, 512)               1180160

 dropout (Dropout)           (None, 512)               0

 dense_42 (Dense)            (None, 512)               262656

 dropout_1 (Dropout)         (None, 512)               0

 dense_43 (Dense)            (None, 10)                5130

=================================================================
Total params: 1,486,666
Trainable params: 1,486,666
Non-trainable params: 0
_________________________________________________________________
```

In [28]:
```
model2.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history2 = model2.fit(train_X1, train_Y1, batch_size = 32, epochs=5, validation_data=(validation_X, validation_Y))
```

```
Epoch 1/5
313/313 [==============================] - 16s 49ms/step - loss: 1.9852 - accuracy: 0.2665 - val_loss: 1.6275 - val_accurac
y: 0.4066
Epoch 2/5
313/313 [==============================] - 16s 52ms/step - loss: 1.6217 - accuracy: 0.4057 - val_loss: 1.5122 - val_accurac
y: 0.4471
Epoch 3/5
313/313 [==============================] - 16s 51ms/step - loss: 1.4608 - accuracy: 0.4704 - val_loss: 1.4391 - val_accurac
y: 0.4717
Epoch 4/5
313/313 [==============================] - 17s 53ms/step - loss: 1.3466 - accuracy: 0.5124 - val_loss: 1.3270 - val_accurac
y: 0.5155
Epoch 5/5
313/313 [==============================] - 16s 52ms/step - loss: 1.2393 - accuracy: 0.5477 - val_loss: 1.2427 - val_accurac
y: 0.5529
```

In [29]:
```
plt.plot(history2.history['accuracy'], label = 'Train_accuracy')
plt.plot(history2.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('epoch')
plt.ylabel('accuracy')
plt.title('Graph 1 : Train_accuracy and val_accuracy Vs Epoch for CNN2')
plt.legend(loc = 'upper left')
plt.show()
```



After applying the CNN2 model to our data, we can see that it is giving better accuracy than MLP and CNN1 as well as it resolves the problem of overfitting as we are using 2 pooling layers and 2 drop-out layers which help in regularizing the network. Initially, the validation accuracy is more than the training accuracy but later we are getting almost the same accuracy for train and testing with higher epoch.

General Observations:

- CNN1 model was working better than the MLP model for the training dataset but not for the validation set. So in our case, CNN was a better classifier than MLP for our dataset.
- CNN1 was having the problem of overfitting which was resolved by adding pooling layers and dropout layers in the CNN2 Model.
- If we use more number of epochs the accuracy will get better as we have 10 classes having more epochs could improve accuracy.
- Keeping number of hidden layers less (maybe 1) and the number of neurons 512 can help our current model perform better as we saw in the plot the accuracy was high with 1 hidden layer. Also accuracy was seen highest at 512 neurons while changing the number of neurons.
- In general we have used 10 neurons in the output layer as we have 10 classes and have used a SoftMax function as an activation function because it can perform multiclass classification for all 3 models.
- We can see in the above code that the CNN1 model is taking more time per epoch around 240 ms/step but CNN2 takes only around 40 ms/step.
- While the MLP model is only taking 10ms/step for each epoch. Also, we noticed that as number of hidden layer is increasing the time to train is epoch is also increasing same

happened with neurons too as number of neurons are increasing, time to train data is also increasing.

- To keep less training time we can also make sure to keep less number of neurons and hidden layer without compromising the accuracy.

Question 2:

Explanation of how you created your dataset

First, we loaded the dataset as Pandas dataframe, sorted it by date in descending order, and prepared the labels by assigning the "Open" column to the dataframe with a different name. Then we converted the dataframe into numpy and we used a loop to iterate through the matrix and get past three days' Volume, Open, High, and Low for each day with slicing operation, while iterating, each slice is reshaped from a 3x4 matrix to 1 x 12 matrix, that gives us the 3 day-worth of features in a vector. Once we had the feature matrix ready, we removed the first row of the feature matrix as well as the last element of the label column so that each feature vector can be mapped to Open price on the next day after horizontal concatenation. Lastly the matrix is shuffled along with dates and we splitted it and keep 70% of the data for training and 30% for testing.

Any preprocessing steps you followed

All features and labels are normalized using MinMaxScaler before plugging into the model. Each value there gets transformed into a number between -1 and 1.

All design steps you went through in finding the best network in your report and how you chose your final design.

We chose RNN over LSTM and GRU because the number of layers we picked is relatively small (2) so gradient vanishing is not going to have a big impact on model performance. In addition, RNN has relatively simpler architecture compared to the other two networks so it will be faster to train. The input dimension of the RNN layers is determined by the number of features we have in training data. Since there are 12 features, the input dimension is set to 12. The output is the open price of the stock for the next day, so the output dimension of the fully connected layer is set to 1.

Architecture of your final network, number of epochs, batch size (if needed), loss function, training algorithm, etc

The model is built with two RNN layers with 12 input dimensions and 200 hidden nodes, and one fully connected layer that takes 200 hidden nodes from the last RNN layer and outputs a scalar.

```
RNN(
  (rnn): RNN(12, 200, num_layers=2, batch_first=True)
  (fc): Linear(in_features=200, out_features=1, bias=True)
)
```

Number of epochs: 200
Batch Size: Not needed
Loss Function: MSE
Training algorithm: Adam
Learning Rate: 0.01

Output of the training loop with comments on your output

Please refer to `./data/train_pred.csv` for the predictions on training set
After 200 epochs, the best MSE on scale training set is 0.0005889208405278623

```
Epoch  191 MSE:  0.0005987018230371177
Epoch  192 MSE:  0.0005974542582407594
Epoch  193 MSE:  0.0005962137947790325
Epoch  194 MSE:  0.0005949809565208852
Epoch  195 MSE:  0.0005937542882747948
Epoch  196 MSE:  0.0005925358855165541
Epoch  197 MSE:  0.0005913234199397266
Epoch  198 MSE:  0.000590119103435427
Epoch  199 MSE:  0.0005889208405278623
```

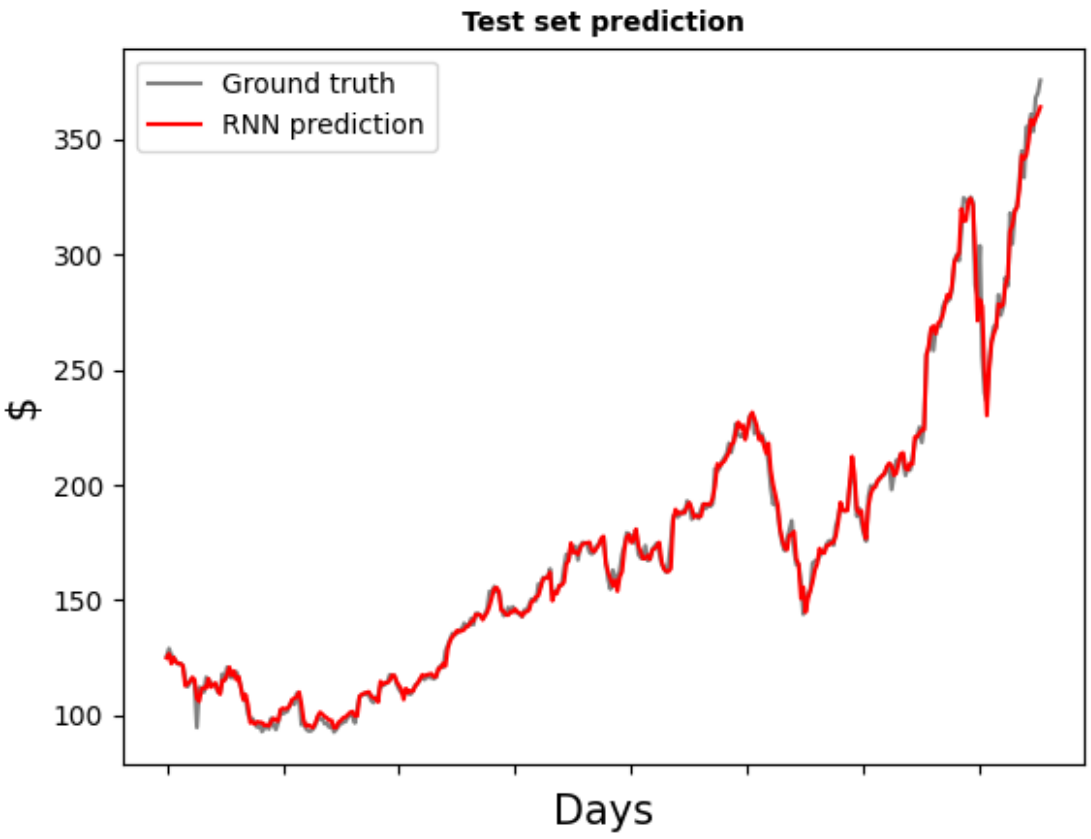
Output from testing, including the final plot and your comment on it

Please refer to `./data/test_pred.csv` for the predictions on testing set

The loss for normalized labels is 0.0006873089
The loss for unnormalized labels is 14.125634

```
/home/quartic/Quartic/virtualenvs/py3.8-jup
loss after scaling 0.0006873089
loss before scaling 14.125634

Process finished with exit code 0
```

Overall, the predictions on the test set are fairly accurate by looking at the loss and the trend the in the plot below



What would happen if you used more days for features

The model will have more information for making predictions but may not necessarily give us better results because the stock market tends to be very volatile and dynamic and what happened in the past is unlikely to occur in future, so the features from older days will have less to contribute to the prediction. Another issue with a bigger window is that dimensionality would rapidly increase. The number of features is increased by 4 for every day we are adding and that may cause the model to underfit.

```python
import torch
from sklearn.preprocessing import MinMaxScaler
import pandas as pd
import numpy as np
import torch.nn as nn
import pickle


def split_data(df, feat_names, target_name, window):
    data = df[feat_names]
    date_series = df["Date"]
    data[f"{target_name}_target"] = df[target_name]
    data = data.to_numpy()

    # bundle past window days features for each day
    windows = []
    for i in range(len(data) - window):
        windows.append(data[i: i + window, :-1].reshape(1, -1))

    windows = np.array(windows)
    dt = date_series.values[:len(data) - window].reshape(-1, 1)[1:, :]
    x = windows[1:, :]
    y = data[:len(data) - window - 1, -1]

    # horizontally concat datetime column, features and labels
    xy = np.concatenate([dt, x.reshape(x.shape[0], -1), y.reshape(-1, 1)], axis=1)

    # shuffle with fixed seed for reproducibility
    np.random.seed(123)
    np.random.shuffle(xy)

    # 70 % goes to training set 30 % goes to testing set
    split = 0.7
    train_set_size = int(xy.shape[0] * split)
    dt_train = xy[:train_set_size, :1]
    x_train = xy[:train_set_size, 1:-1].reshape(train_set_size, 1, -1)
    y_train = xy[:train_set_size, -1:]
```

```python
class RNN(nn.Module):
    def __init__(self, input_dim, hidden_dim, num_layers, output_dim, scalars):
        super(RNN, self).__init__()
        self.hidden_dim = hidden_dim
        self.num_layers = num_layers
        self.rnn = nn.RNN(input_dim, hidden_dim, num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_dim, output_dim)
        self.scalars = scalars

    def forward(self, x):
        h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_dim).requires_grad_()
        out, hn = self.rnn(x, h0.detach())
        out = self.fc(out[:, -1, :])
        return out


def to_tensor(np_array):
    return torch.from_numpy(np_array).type(torch.Tensor)


if __name__ == "__main__":
    # load raw data
    # df = pd.read_csv("./q2_dataset.csv")
    #
    # df.columns = df.columns.str.strip()
    #
    # feats = ['Volume', 'Open', 'High', 'Low']
    # target = 'Open'
    #
    # window = 3

    # Shift target to
    # dt_train, x_train, y_train, dt_test, x_test, y_test = split_data(df, feats, target, window)
    # train_data = np.concatenate([dt_train, x_train.reshape(x_train.shape[0], -1), y_train], axis=1)
    # test_data = np.concatenate([dt_test, x_test.reshape(x_test.shape[0], -1), y_test], axis=1)
    #
```

```
# window = 3

# Shift target to
# dt_train, x_train, y_train, dt_test, x_test, y_test = split_data(df, feats, target, window)
# train_data = np.concatenate([dt_train, x_train.reshape(x_train.shape[0], -1), y_train], axis=1)
# test_data = np.concatenate([dt_test, x_test.reshape(x_test.shape[0], -1), y_test], axis=1)
#
# np.savetxt("./data/train_data_RNN.csv", train_data, delimiter=",", fmt='%s')
# np.savetxt("./data/test_data_RNN.csv", test_data, delimiter=",", fmt='%s')

# read training data
train_data = pd.read_csv("./data/train_data_RNN.csv", header=None, index_col=None)
train_data = train_data[train_data.columns[1:]]

train_pred = pd.DataFrame({"train_pred": train_data[train_data.columns[-1]]})

# scale all feature between -1 and 1
scalers = {}
for column in train_data.columns:
    scalers[column] = MinMaxScaler(feature_range=(-1, 1))
    train_data[column] = scalers[column].fit_transform(train_data[column].values.reshape(-1, 1))

x_train = to_tensor(train_data.values[:, :-1].reshape(len(train_data), 1, -1))
y_train_lstm = to_tensor(train_data.values[:, -1].reshape(-1, 1))

input_dim = 12
hidden_dim = 200
num_layers = 2
output_dim = 1
num_epochs = 200

# initialize model
model = RNN(input_dim=input_dim, hidden_dim=hidden_dim, output_dim=output_dim, num_layers=num_layers,
            scalars=scalers)
mse = torch.nn.MSELoss(reduction='mean')
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
```

```
110
111     losses = []
112
113     # training loop
114     for t in range(num_epochs):
115         y_train_pred = model(x_train)
116         loss = mse(y_train_pred, y_train_lstm)
117         print("Epoch ", t, "MSE: ", loss.item())
118
119         optimizer.zero_grad()
120         loss.backward()
121         optimizer.step()
122         losses.append(loss)
123
124     print(model)
125     torch.save(model, "./models/20433010_RNN_model.torch")
126     with open("./models/scalar.pk", "wb") as f:
127         pickle.dump(scalers, f)
128
```

```python
import torch
from train_RNN import RNN, to_tensor
import pickle
from sklearn.preprocessing import MinMaxScaler
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

if __name__ == "__main__":
    model = torch.load("./models/20433010_RNN_model.torch")
    model.eval()
    with open("./models/scalar.pk", "rb") as f:
        scalars = pickle.load(f)

    test_df = pd.read_csv("./data/test_data_RNN.csv", header=None)
    test_df[test_df.columns[0]] = pd.to_datetime(test_df[test_df.columns[0]])
    test_data = test_df.sort_values(by=[test_df.columns[0]])
    test_data = test_data[test_data.columns[1:]]
    # label before normalization
    y_test_raw = np.array(test_data[test_data.columns[-1]].values)

    for col in test_data.columns:
        test_data[col] = scalars[col].transform(test_data[col].values.reshape(-1, 1))

    scaled_test_data = test_data.values

    x_test = to_tensor(scaled_test_data[:, :-1].reshape(scaled_test_data.shape[0], 1, -1))
    y_test = scaled_test_data[:, -1:]

    y_test_pred = model(x_test)

    mse = torch.nn.MSELoss(reduction='mean')
    loss = mse(y_test_pred, to_tensor(y_test))
    print("loss after scaling", loss.detach().numpy())

    # mse calculated with reverse normalized predictions and labels
    target_scalar = scalars[test_df.columns[-1]]
```

```python
# mse calculated with reverse normalized predictions and labels
target_scalar = scalars[test_df.columns[-1]]
y_test_pred_reversed = target_scalar.inverse_transform(y_test_pred.detach().numpy())

loss = mse(to_tensor(y_test_pred_reversed), to_tensor(y_test_raw.reshape(-1, 1)))
print("loss before scaling", loss.detach().numpy())

test_pred = pd.DataFrame({"y_test_pred": y_test_pred_reversed.reshape(-1), "y_test": y_test_raw.reshape(-1)})

# save predictions and ground truth for test set
test_pred.to_csv("./data/test_pred.csv", index=False)

fig = plt.figure()

ax = sns.lineplot(x=test_df.index, y=y_test_raw.reshape(-1), label="Ground truth", color='grey')
ax = sns.lineplot(x=test_df.index, y=y_test_pred_reversed.reshape(-1), label="RNN prediction", color='red')
ax.set_title('Test set prediction', size=10, fontweight='bold')
ax.set_xlabel("Days", size=15)
ax.set_ylabel("$", size=15)
ax.set_xticklabels('', size=10)

plt.show()
```

Question 3:

Problem 3: Sentiment Analysis on IMDB review dataset

```python
# Importing the required libraries
import tensorflow as tf
import tarfile
import os
import numpy as np
import re
import random
import tensorflow as tf
from tensorflow import keras
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
from tensorflow.python.keras import Sequential
from tensorflow.python.keras.layers import Dense, LSTM, Dropout, Embedding
import nltk
from nltk.corpus import stopwords
import matplotlib.pyplot as plt
import pickle
```

[5] Python

```
# Downloading the stopwords for data preprocessing
#nltk.download('stopwords')
```
[66] Python

```
# Function to download the data from the website and extracting it
def load_data():
  Imdb_Dataset_raw = tf.keras.utils.get_file(
      fname = "aclImdb_v1.tar",
      origin = "https://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz",
      extract = True
  )
  Imdb_Dataset_extracted = tarfile.open(Imdb_Dataset_raw)
  Imdb_Dataset_extracted.extractall('./data/')
  Imdb_Dataset_extracted.close()
```
[6] Python

```
# Call load_data() to load the data
#load_data()
```
[7] Python

After loading the dataset directly from the website, positive and negative data are loaded from the directory and concateneted. train_X contains a sequece of text and train_y contains a sequence of ones and zeros for positive and negative reviews respectively.

```
# Code to load train dataset
pos_path = './data/aclImdb/train/pos'
neg_path = './data/aclImdb/train/neg'

train_data_pos = [open(os.path.join(pos_path, f), errors='ignore').read() for f in os.listdir(pos_path)]

train_data_neg = [open(os.path.join(neg_path, f), errors='ignore').read() for f in os.listdir(neg_path)]

train_X = np.concatenate((train_data_pos, train_data_neg), axis = 0)
train_y = np.hstack((np.ones(len(train_data_pos)), np.zeros(len(train_data_neg))))
```
[8] Python

```
# Dimension of the train dataset
train_X.shape, train_y.shape
```
[9] Python

```
((25000,), (25000,))
```

```
# Print the text from the train dataset
print(train_X[1])
```
[10] Python

```
Homelessness (or Houselessness as George Carlin stated) has been an issue for years but never a plan to help those on the street that were once considered human who did everything from going to school, work, or vote for the matter. Most people think of the homeless as just a lost cause while worrying about things such as racism, the war on Iraq, pressuring kids to succeed, technology, the elections, inflation, or worrying if they'll be next to end up on the streets.<br /><br />But what if you were given a bet to live on the streets for a month without the luxuries you once had from a home, the entertainment sets, a bathroom, pictures on the wall, a computer, and everything you once treasure to see what it's like to be homeless? That is Goddard Bolt's lesson.<br /><br />Mel Brooks (who directs) who stars as Bolt plays a rich man who has everything in the world until deciding to make a bet with a sissy rival (Jeffery Tambor) to see if he can live in the streets for thirty days without the luxuries; if Bolt succeeds, he can do what he wants with a future project of making more buildings. The bet's on where Bolt is thrown on the street with a bracelet on his leg to monitor his every move where he can't step off the sidewalk. He's given the nickname Pepto by a vagrant after it's written on his forehead where Bolt meets other characters including a woman by the name of Molly (Lesley Ann Warren) an ex-dancer who got divorce before losing her home, and her pals Sailor (Howard Morris) and Fumes (Teddy Wilson) who are already used to the streets. They're survivors. Bolt isn't. He's not used to reaching mutual agreements like he once did when being rich where it's fight or flight, kill or be killed.<br /><br />While the love connection between Molly and Bolt wasn't necessary to plot, I found "Life Stinks" to be one of Mel Brooks' observant films where prior to being a comedy, it shows a tender side compared to his slapstick work such as Blazing Saddles, Young Frankenstein, or Spaceballs for the matter, to show what it's like having something valuable before losing it the next day or on the other hand making a stupid bet like all rich people do when they don't know what to do with their money. Maybe they should give it to the homeless instead of using it like Monopoly money.<br /><br />Or maybe this film will inspire you to help others.
```

From this text it is visible that the texts are having some noise like '.', "",html tags and some special characters. In order to process these text properly, we need to remove these noise from the text.

The following function takes care of removing noise. Also, after removing the noise, we are also removing stopwords from the text. Removing the stopsworde could improve the performance of the model. For removing the stopwords we are using stop words in the corpus module in nltk library.

```
# Function to clean the data
def data_cleanup(data_raw):
  tokens = re.compile("[.;:!#\'?,\"()\[\]]|(<br\s*/><br\s*/>)|(\-)|(\/)")
  res = [tokens.sub("", d.lower()) for d in data_raw]

  stop_words = set(nltk.corpus.stopwords.words("english"))

  data_cleaned = [r for r in res if r not in stop_words]

  return res
```

[11] Python

```
train_X_cleaned = data_cleanup(train_X)
```

[12] Python

Cleaned train data

```
# Print the cleaned data
print(train_X_cleaned[1])
```

[13] Python

```
homelessness or houselessness as george carlin stated has been an issue for years but never a plan to help those on the street that were once considered human who did everything from going to school work or vote for the matter most people think of the homeless as just a lost cause while worrying about things such as racism the war on iraq pressuring kids to succeed technology the elections inflation or worrying if theyll be next to end up on the streetsbut what if you were given a bet to live on the streets for a month without the luxuries you once had from a home the entertainment sets a bathroom pictures on the wall a computer and everything you once treasure to see what its like to be homeless that is goddard bolts lessonmel brooks who directs who stars as bolt plays a rich man who has everything in the world until deciding to make a bet with a sissy rival jeffery tambor to see if he can live in the streets for thirty days without the luxuries if bolt succeeds he can do what he wants with a future project of making more buildings the bets on where bolt is thrown on the street with a bracelet on his leg to monitor his every move where he cant step off the sidewalk hes given the nickname pepto by a vagrant after its written on his forehead where bolt meets other characters including a woman by the name of molly lesley ann warren an exdancer who got divorce before losing her home and her pals sailor howard morris and fumes teddy wilson who are already used to the streets theyre survivors bolt isnt hes not used to reaching mutual agreements like he once did when being rich where its fight or flight kill or be killedwhile the love connection between molly and bolt wasnt necessary to plot i found life stinks to be one of mel brooks observant films where prior to being a comedy it shows a tender side compared to his slapstick work such as blazing saddles young frankenstein or spaceballs for the matter to show what its like having something valuable before losing it the next day or on the other hand making a stupid bet like all rich people do when they dont know what to do with their money maybe they should give it to the homeless instead of using it like monopoly moneyor maybe this film will inspire you to help others
```

From the above data it is noticable that all the noise is removed from the data.

In the below code, we have initialized tokenizer and fitted it in the clead dataset. Then we have vectorize the text corpus and changed it to a sequence of integers.

```
tokenizer = keras.preprocessing.text.Tokenizer()
tokenizer.fit_on_texts(train_X_cleaned)
train_X = tokenizer.texts_to_sequences(train_X_cleaned)

# code to pad the sequence to maximun length of 1100 to ensure that all the sequence is having same length
train_X = keras.preprocessing.sequence.pad_sequences(train_X,padding='post',maxlen=1100)
```

[15] Python

```
# Code to split the data set into 80-20 ratio
train_X, X_validate , train_y, y_validate = train_test_split(train_X, train_y, test_size=0.2, random_state=42)
```

[16] Python

Below is the design of our model to perform the sentiment analysis. We chose a model of sequential CNN model. Here CNN is useful in extracting features from text. The architecture is as follows:

- First there is an embedding layer that transfroms integers to a line of embedding weights matrix
- Then there is a convolution network with filter size 8 and kernel size as 2 and activation function as relu
- Next we have added a Global Average pooling layer is used to better represent the vector
- Then we have added one Dropout layer to avoid overfitting
- Lastly, we have added activation functions like 'relu' and 'sigmoid' also one Dropout layer to avoid overfitting of the model

[93]
```python
# Model design
vocab_size = len(tokenizer.word_index)+1

model = keras.Sequential()
model.add(keras.layers.Embedding(vocab_size, 8))
model.add(keras.layers.Conv1D(filters=8,kernel_size=2,padding='valid',activation='relu'))
model.add(keras.layers.GlobalAveragePooling1D())
model.add(keras.layers.Dropout(0.1))
model.add(keras.layers.Dense(16, activation='relu'))
model.add(keras.layers.Dropout(0.1))
model.add(keras.layers.Dense(1, activation='sigmoid'))

model.summary()
```
Python

```
Model: "sequential_19"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_19 (Embedding)    (None, None, 8)           1142200

 conv1d_19 (Conv1D)          (None, None, 8)           136

 global_average_pooling1d_19  (None, 8)                0
  (GlobalAveragePooling1D)

 dropout_38 (Dropout)        (None, 8)                 0

 dense_38 (Dense)            (None, 16)                144

 dropout_39 (Dropout)        (None, 16)                0

 dense_39 (Dense)            (None, 1)                 17

=================================================================
Total params: 1,142,497
Trainable params: 1,142,497
Non-trainable params: 0
_________________________________________________________________
```

```
# Code to compile the model
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['acc'])
```

[94] Python

```
# Code to fit the model
history = model.fit(train_X, train_y,
                    epochs=10,
                    validation_data=(X_validate, y_validate),
                    verbose=1,
                    batch_size=200)
```

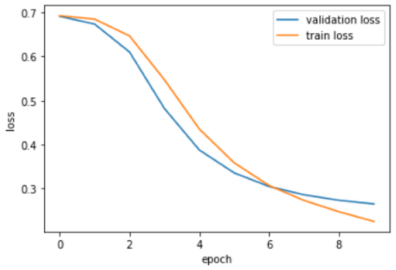
[95] Python

```
Epoch 1/10
100/100 [==============================] - 5s 42ms/step - loss: 0.6928 - acc: 0.5151 - val_loss: 0.6913 - val_acc: 0.6086
Epoch 2/10
100/100 [==============================] - 5s 53ms/step - loss: 0.6849 - acc: 0.6004 - val_loss: 0.6737 - val_acc: 0.6604
Epoch 3/10
100/100 [==============================] - 5s 54ms/step - loss: 0.6469 - acc: 0.6762 - val_loss: 0.6103 - val_acc: 0.7056
Epoch 4/10
100/100 [==============================] - 5s 48ms/step - loss: 0.5473 - acc: 0.7737 - val_loss: 0.4820 - val_acc: 0.8346
Epoch 5/10
100/100 [==============================] - 5s 48ms/step - loss: 0.4354 - acc: 0.8348 - val_loss: 0.3878 - val_acc: 0.8586
Epoch 6/10
100/100 [==============================] - 5s 45ms/step - loss: 0.3586 - acc: 0.8641 - val_loss: 0.3358 - val_acc: 0.8724
Epoch 7/10
100/100 [==============================] - 4s 45ms/step - loss: 0.3071 - acc: 0.8863 - val_loss: 0.3049 - val_acc: 0.8864
Epoch 8/10
100/100 [==============================] - 4s 45ms/step - loss: 0.2733 - acc: 0.9000 - val_loss: 0.2860 - val_acc: 0.8890
Epoch 9/10
100/100 [==============================] - 4s 44ms/step - loss: 0.2476 - acc: 0.9098 - val_loss: 0.2736 - val_acc: 0.8954
Epoch 10/10
100/100 [==============================] - 5s 47ms/step - loss: 0.2256 - acc: 0.9180 - val_loss: 0.2653 - val_acc: 0.8958
```

We have trained the model with 10 epoch cycles and with a bact size of 200. From the accuracy reporting, we can see that the model has attained the validation accuracy as 89.5% with a loss of 22.5%. If we try to train the model with more epoch cycles then it overfits the model. Thus, we had trained the model upto 10 epoch cycles only.

```
# Output graph
plt.plot(history.history['val_loss'],label="validation loss")
plt.plot(history.history['loss'],label="train loss")
plt.xlabel('epoch')
plt.ylabel('loss')

plt.legend()
plt.show()
```

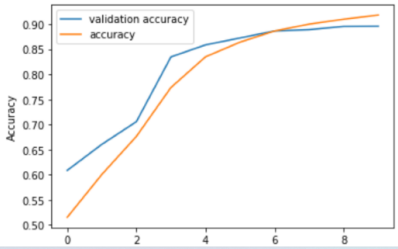
[98] Python



It is evident from the graph that training and validation loss decreases over epoch cycles

```
# Accuracy reporting graph
plt.plot(history.history['val_acc'],label="validation accuracy")
plt.plot(history.history['acc'],label="accuracy")
plt.xlabel('epoch')
plt.ylabel('Accuracy')

plt.legend()
plt.show()
```

[99] Python



The graph shows that the accuracy of the model increases over the epoch cycles and the validation accuracy of the model converges with the training accuracy of the model.

```
# Code to dump the model in the directory
model.save(os.path.join('models','Group_80_NLP_model'))
with open('models/Group_80_NLP_model/NLP_token.pkl', 'wb') as handle:
    pickle.dump(tokenizer, handle, protocol=pickle.HIGHEST_PROTOCOL )
model.save('models/Group_80_NLP_model')
```

[100] Python

Validation with test dataset:

After loading the dataset directly from the website, positive and negative data are loaded from the directory and concateneted. test_X contains a sequece of text and test_y contains a sequence of ones and zeros for positive and negative reviews respectively.

```
# Code to load test dataset
pos_path = './data/aclImdb/test/pos'
neg_path = './data/aclImdb/test/neg'

test_data_pos = [open(os.path.join(pos_path, f), errors='ignore').read() for f in os.listdir(pos_path)]

test_data_neg = [open(os.path.join(neg_path, f), errors='ignore').read() for f in os.listdir(neg_path)]

test_X = np.concatenate((test_data_pos, test_data_neg), axis = 0)
test_y = np.hstack((np.ones(len(test_data_pos)), np.zeros(len(test_data_neg))))
```

8] ✓ 1m 40.4s Python

```
# Code to load train dataset
pos_path = './data/aclImdb/train/pos'
neg_path = './data/aclImdb/train/neg'

train_data_pos = [open(os.path.join(pos_path, f), errors='ignore').read() for f in os.listdir(pos_path)]

train_data_neg = [open(os.path.join(neg_path, f), errors='ignore').read() for f in os.listdir(neg_path)]

train_X = np.concatenate((train_data_pos, train_data_neg), axis = 0)
train_y = np.hstack((np.ones(len(train_data_pos)), np.zeros(len(train_data_neg))))
```

[3] ✓ 2m 7.1s Python

Print the test dataset

```
print(test_X[0])
```

19] ✓ 0.1s Python

```
I went and saw this movie last night after being coaxed to by a few friends of mine. I'll admit that I was reluctant to see it because from what I knew of Ashton Kutcher he was only able to do comedy. I was wrong. Kutcher played the character of Jake Fischer very well, and Kevin Costner played Ben Randall with such professionalism. The sign of a good movie is that it can toy with our emotions. This one did exactly that. The entire theater (which was sold out) was overcome by laughter during the first half of the movie, and were moved to tears during the second half. While exiting the theater I not only saw many women in tears, but many full grown men as well, trying desperately not to let anyone see them crying. This movie was great, and I suggest that you go see it before you judge.
```

Below are the code to clean the test dataset same as the train dataset

```
# Function to clean the data
def data_cleanup(data_raw):
    tokens = re.compile("[.;:!#\'?,\"()\[\]]|(<br\s*/><br\s*/>)|(\-)|(\/)")
    res = [tokens.sub("", d.lower()) for d in data_raw]

    stop_words = set(nltk.corpus.stopwords.words("english"))

    data_cleaned = [r for r in res if r not in stop_words]

    return data_cleaned
```

4] ✓ 0.5s Python

```
# Cleaning up the test data
test_X_cleaned = data_cleanup(test_X)
train_X_cleaned = data_cleanup(train_X)
```

5] ✓ 5.3s Python

Print the cleaned dataset

+ Code + Markdown

```
print(test_X_cleaned[0])
```

[33] Python

```
i went and saw this movie last night after being coaxed to by a few friends of mine ill admit that i was reluctant to see it because from what i knew of ashton kutcher he was only able to do comedy i was wrong kutcher played the character of jake fischer very well and kevin costner played ben randall with such professionalism the sign of a good movie is that it can toy with our emotions this one did exactly that the entire theater which was sold out was overcome by laughter during the first half of the movie and were moved to tears during the second half while exiting the theater i not only saw many women in tears but many full grown men as well trying desperately not to let anyone see them crying this movie was great and i suggest that you go see it before you judge
```

Code to vectorize the text corpus and changed it to a sequence of integers same as the train data

+ Code + Markdown

```
tokenizer = keras.preprocessing.text.Tokenizer()
tokenizer.fit_on_texts(train_X_cleaned)
train_X = tokenizer.texts_to_sequences(test_X_cleaned)
test_X = tokenizer.texts_to_sequences(test_X_cleaned)
```

[20] ✓ 7.5s Python

```
# code to pad the sequence to maximun length of 1100 to ensure that all the sequence is having same length
test_X = keras.preprocessing.sequence.pad_sequences(test_X,padding='post',maxlen=1100)

# Code to split the data set into 80-20 ratio
test_X, X_validate, test_y, y_validate = train_test_split(test_X, test_y, test_size=0.2, random_state=42)
```

[21] ✓ 0.3s Python

```
# Code to load the model from the directory
model = tf.keras.models.load_model('models/Group_80_NLP_model')
```

[22] ✓ 0.4s Python

```
test_loss, test_acc = model.evaluate(test_X, test_y)
print("Test dataset result")
print("Loss: ", test_loss)
print("Model accuracy: ", test_acc)
```

[28] ✓ 2.5s Python

```
625/625 [==============================] - 2s 3ms/step - loss: 0.2930 - acc: 0.8863
Test dataset result
Loss:  0.2929551899433136
Model accuracy:  0.8863000273704529
```

Conclusion : We are able to achieve accuracy of 88.6% on the test dataset wich is very close to the accuracy we got in training model (89.5%). So, we can conclude that our model is efficient enough to correct sentiment analysis on the dataset. Also, we can conclude that in this particular case, sequential CNN model is very useful to perform sentiment analysis on the text. Also, layering the model with Dropout has helped the model to avoid the issues of overfitting.

References:

- https://medium.com/@pyashpq56/sentiment-analysis-on-imdb-movie-review-d004f3e470bd
- https://towardsdatascience.com/sentiment-analysis-using-lstm-and-glove-embeddings-99223a87fe8e