Convolutional Neural Network on Images dataset

- SaiTejaswini Thumpala

Introduction:

This project shows how a Convolutional neural network is built, trained, and tested on Images' dataset. After building the network, I worked on the way to improve the performance i.e the accuracy value of the Convolutional neural network by adding more convolutional layers and more pooling layers. The process of uploading the CIFAR_10 dataset to the remote VM server, is also explained.

Part-1: A Dataset of Images

Dataset:

CIFAR-10 dataset consists of 60,000 images within the 10 number of classes (6000 images per class). This dataset consists of training dataset and test dataset. The training dataset consists of 50,000 images and the test dataset consists of 10,000 images. These images are of 32*32 pixels.

The training datasets are divided into five sub-datasets, each one containing 10,000 images and testing is under one sub-dataset. Testing sub-dataset contains 1000 randomly selected images from each class.

Part-2: Obtain CIFAR-10 Dataset

Steps in accessing and uploading the CIFAR-10 dataset:

- The CIFAR-10 dataset is taken from canvas module and all the available seven files are downloaded.
- In Google cloud console, open VM instances and then open the Remote VM server.
- Create a folder by name 'JPTR_NTBK' and create a new sub folder under it with the name 'CIFAR_10_DATA'. Open the folder JPTR_NTBK by using the command 'cd JPTR_NTBK' and then create a sub-folder by using the command 'mkdir CIFAR_10_DATA'.
- All the files need to be uploaded to the new sub folder in the remote VM instance through the upload file option.

Part-3: Build, Train, and Test CNN on CIFAR-10 Dataset

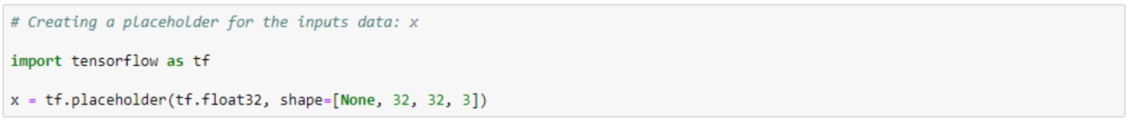
A Convolutional neural network is a type of artificial neural network that is used in Image recognition and processing. This feed-forward network takes the input image and then assign weights to it by splitting them into various objects. Convolutional neural networks take fixed size of inputs and generates fixed size of outputs.

Design of the Convolutional Neural Network:

1. Two sets of convolution layers and pooling layers are used to obtain the features from the images in the dataset.
2. Flattening layer is used to convert the feature map into vector data.
3. The Fully connected layer is applied along with the activation function to act as a deep learning neural network model.
4. The Dropout layer removes the output. It acts as input to the fully connected layer and gives the labels as output.
5. The loss function is used to calculate the probability error in the discrete classification tasks in which the classes are mutually exclusive.
6. Neural network can be improved by the optimizer and can be trained by the AI deep learning trainer.

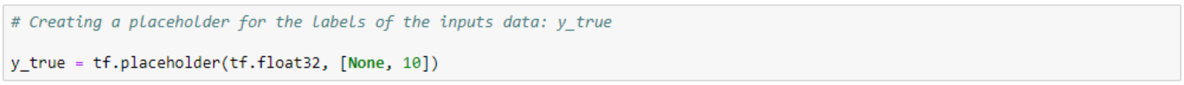
Building the Convolutional neural network:

1. A place holder 'x' has been created for input data and this place holder is a 2D array which can hold any number of records or rows. The rows in this array are vectors which are used to hold the 2D images.



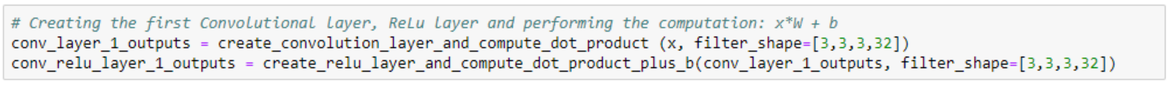
```
# Creating a placeholder for the inputs data: x

import tensorflow as tf

x = tf.placeholder(tf.float32, shape=[None, 32, 32, 3])
```

2. A place holder 'y' has been created for labels of the input data and this place holder is a 2D array which can hold any number of records or rows.



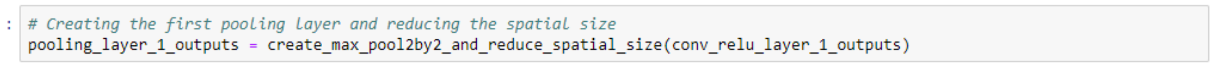
```
# Creating a placeholder for the labels of the inputs data: y_true

y_true = tf.placeholder(tf.float32, [None, 10])
```

3. The first Convolutional layer and ReLu layer are created, and the computation is performed.



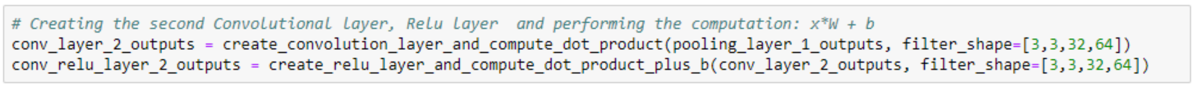
```
# Creating the first Convolutional layer, ReLu layer and performing the computation: x*W + b
conv_layer_1_outputs = create_convolution_layer_and_compute_dot_product (x, filter_shape=[3,3,3,32])
conv_relu_layer_1_outputs = create_relu_layer_and_compute_dot_product_plus_b(conv_layer_1_outputs, filter_shape=[3,3,3,32])
```

4. The first pooling layer is created, and the spatial size is reduced.



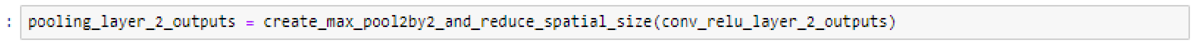
```
: # Creating the first pooling layer and reducing the spatial size
  pooling_layer_1_outputs = create_max_pool2by2_and_reduce_spatial_size(conv_relu_layer_1_outputs)
```

5. The second Convolutional layer and ReLu layer are created, and the computation is performed.



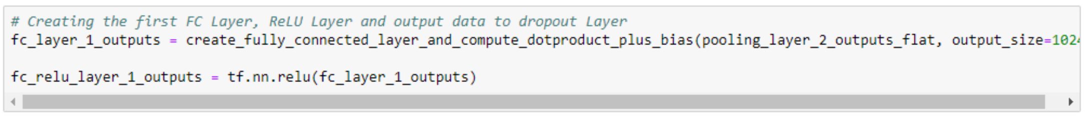
```
# Creating the second Convolutional layer, Relu layer  and performing the computation: x*W + b
conv_layer_2_outputs = create_convolution_layer_and_compute_dot_product(pooling_layer_1_outputs, filter_shape=[3,3,32,64])
conv_relu_layer_2_outputs = create_relu_layer_and_compute_dot_product_plus_b(conv_layer_2_outputs, filter_shape=[3,3,32,64])
```

6. The second pooling layer is also created, and the spatial size is reduced.



```
: pooling_layer_2_outputs = create_max_pool2by2_and_reduce_spatial_size(conv_relu_layer_2_outputs)
```

7. Outputs obtained from the second pooling layer have been reshaped and flattened and that data is made ready to fit into the first fully connected layer.
8. The first Fully Connected layer and ReLu layer are created, and the output data is then sent to the dropout layer.



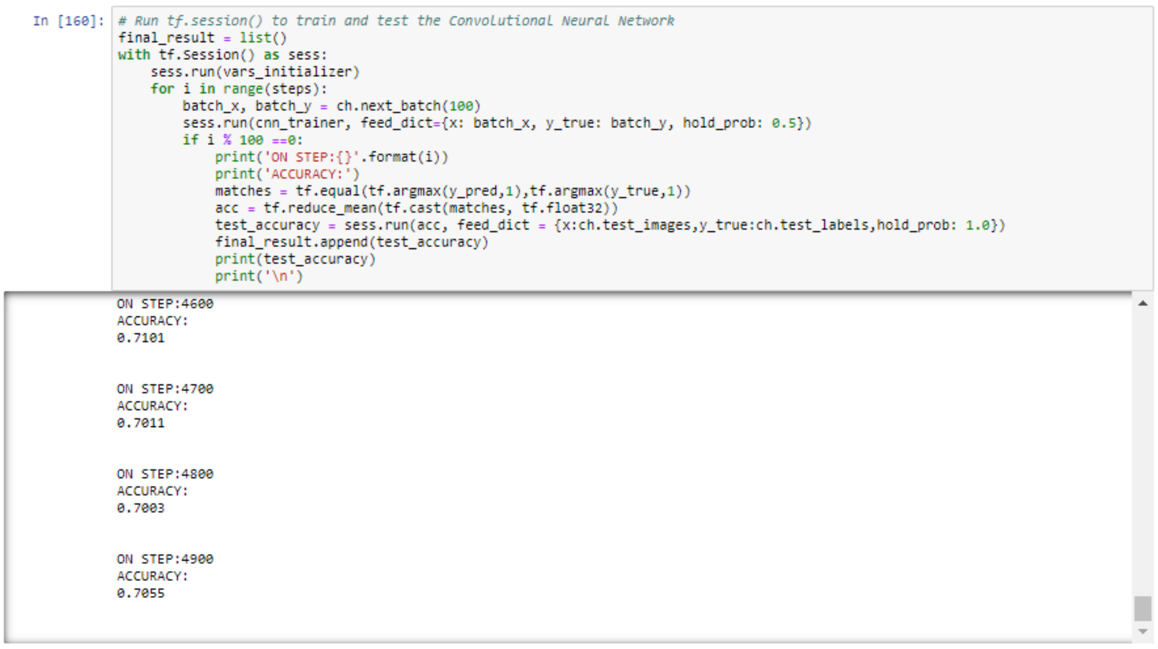
```
# Creating the first FC Layer, ReLU Layer and output data to dropout Layer
fc_layer_1_outputs = create_fully_connected_layer_and_compute_dotproduct_plus_bias(pooling_layer_2_outputs_flat, output_size=102

fc_relu_layer_1_outputs = tf.nn.relu(fc_layer_1_outputs)
```

9. The dropout layer is created, and the fraction of outputs are sent to the output layer.
10. The final output layer is created, and the outputs will be produced.

Training and testing the model:

1. A trainer is created to train the model.
2. Variables are initialized by using the variable initializer.
3. The step count is set to 5000.
4. tf.session() has been created to train and test the Convolutional neural network.

Results:



```
In [160]: # Run tf.session() to train and test the Convolutional Neural Network
          final_result = list()
          with tf.Session() as sess:
              sess.run(vars_initializer)
              for i in range(steps):
                  batch_x, batch_y = ch.next_batch(100)
                  sess.run(cnn_trainer, feed_dict={x: batch_x, y_true: batch_y, hold_prob: 0.5})
                  if i % 100 ==0:
                      print('ON STEP:{}'.format(i))
                      print('ACCURACY:')
                      matches = tf.equal(tf.argmax(y_pred,1),tf.argmax(y_true,1))
                      acc = tf.reduce_mean(tf.cast(matches, tf.float32))
                      test_accuracy = sess.run(acc, feed_dict = {x:ch.test_images,y_true:ch.test_labels,hold_prob: 1.0})
                      final_result.append(test_accuracy)
                      print(test_accuracy)
                      print('\n')
```

```
ON STEP:4600
ACCURACY:
0.7101


ON STEP:4700
ACCURACY:
0.7011


ON STEP:4800
ACCURACY:
0.7003


ON STEP:4900
ACCURACY:
0.7055
```

As we can say from the above, CNN is constructed. Adam optimizer is used to train the model with a step count of 5000. From the output above, we can see that the accuracy is 0.7055. This means that 70.55% of the images in the dataset, are recognized.

Part-4: Improve Convolutional Neural Network Performance

The Performance of the Convolutional neural network for CIFAR_10 dataset is 70.55%, which is not much preferred. This indicates that the images in the dataset are not properly identified. To improve the performance of Convolutional neural network, here two more convolutional layers and two more pooling layers are added and then the accuracy value of the neural network on CIFAR_10 dataset is tested again.

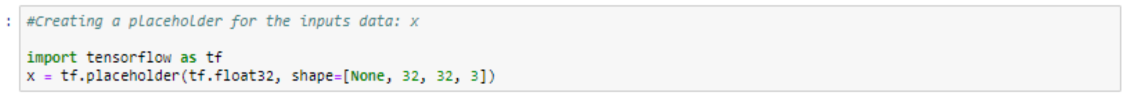
Design of the Convolutional neural network:

1. Four sets of convolution layers and pooling layers are used to obtain the features from the images in the dataset.
2. Flattening layer is used to convert the feature map into vector data.
3. The Fully connected layer is applied along with the activation function to act as a deep learning neural network model.
4. The Dropout layer removes the output. It acts as input to the fully connected layer and gives the labels as output.

5. The loss function is used to calculate the probability error in the discrete classification tasks in which the classes are mutually exclusive.
6. Neural network can be improved by the optimizer and can be trained by the AI deep learning trainer.

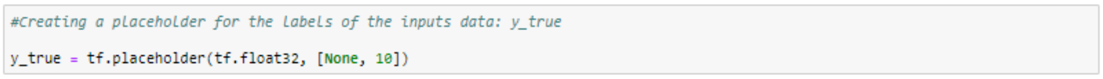
Building the Convolutional neural network:

1. A place holder ‘x’ has been created for input data and this place holder is a 2D array. which can hold any number of records or rows. The rows in this array are vectors which are used to hold the 2D images.



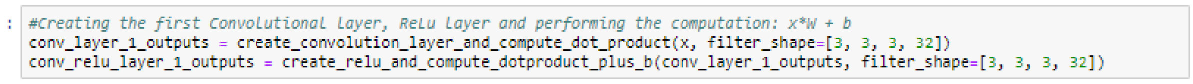
```
#Creating a placeholder for the inputs data: x

import tensorflow as tf
x = tf.placeholder(tf.float32, shape=[None, 32, 32, 3])
```

2. A place holder ‘y’ has been created for labels of the input data and this place holder is a 2D array which can hold any number of records or rows.



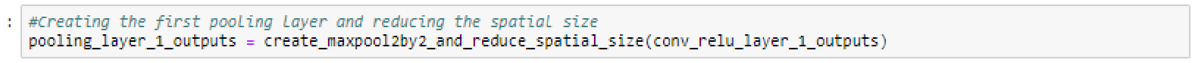
```
#Creating a placeholder for the labels of the inputs data: y_true

y_true = tf.placeholder(tf.float32, [None, 10])
```

3. The first Convolutional layer and ReLu layer are created, and the computation is performed.



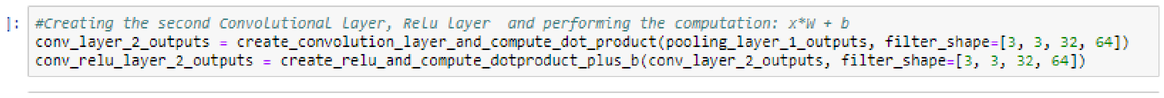
```
#Creating the first Convolutional Layer, ReLu Layer and performing the computation: x*W + b
conv_layer_1_outputs = create_convolution_layer_and_compute_dot_product(x, filter_shape=[3, 3, 3, 32])
conv_relu_layer_1_outputs = create_relu_and_compute_dotproduct_plus_b(conv_layer_1_outputs, filter_shape=[3, 3, 3, 32])
```

4. The first pooling layer is created, and the spatial size is reduced.



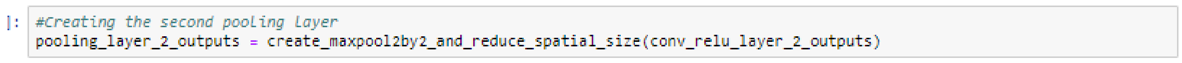
```
#Creating the first pooling Layer and reducing the spatial size
pooling_layer_1_outputs = create_maxpool2by2_and_reduce_spatial_size(conv_relu_layer_1_outputs)
```

5. The second Convolutional layer and ReLu layer are created, and the computation is performed.



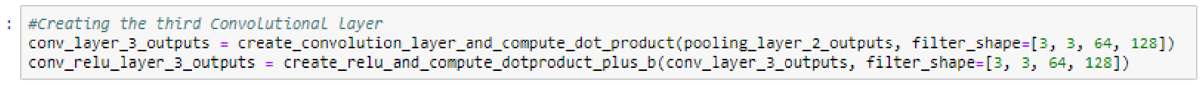
```
#Creating the second Convolutional Layer, ReLu Layer  and performing the computation: x*W + b
conv_layer_2_outputs = create_convolution_layer_and_compute_dot_product(pooling_layer_1_outputs, filter_shape=[3, 3, 32, 64])
conv_relu_layer_2_outputs = create_relu_and_compute_dotproduct_plus_b(conv_layer_2_outputs, filter_shape=[3, 3, 32, 64])
```

6. The second pooling layer is created, and the spatial size is reduced.



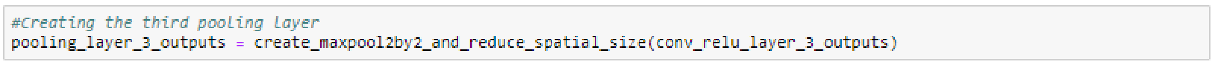
```python
#Creating the second pooling layer
pooling_layer_2_outputs = create_maxpool2by2_and_reduce_spatial_size(conv_relu_layer_2_outputs)
```

7. The third Convolutional layer and ReLu layer are created, and the computation is performed.



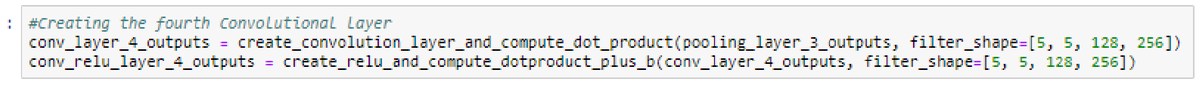
```python
#Creating the third Convolutional Layer
conv_layer_3_outputs = create_convolution_layer_and_compute_dot_product(pooling_layer_2_outputs, filter_shape=[3, 3, 64, 128])
conv_relu_layer_3_outputs = create_relu_and_compute_dotproduct_plus_b(conv_layer_3_outputs, filter_shape=[3, 3, 64, 128])
```

8. The third pooling layer is created, and the spatial size is reduced.



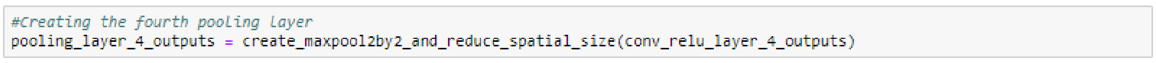
```python
#Creating the third pooling Layer
pooling_layer_3_outputs = create_maxpool2by2_and_reduce_spatial_size(conv_relu_layer_3_outputs)
```

9. The fourth Convolutional layer and ReLu layer are created, and the computation is performed.



```python
#Creating the fourth Convolutional Layer
conv_layer_4_outputs = create_convolution_layer_and_compute_dot_product(pooling_layer_3_outputs, filter_shape=[5, 5, 128, 256])
conv_relu_layer_4_outputs = create_relu_and_compute_dotproduct_plus_b(conv_layer_4_outputs, filter_shape=[5, 5, 128, 256])
```

10. The fourth pooling layer is created, and the spatial size is reduced.



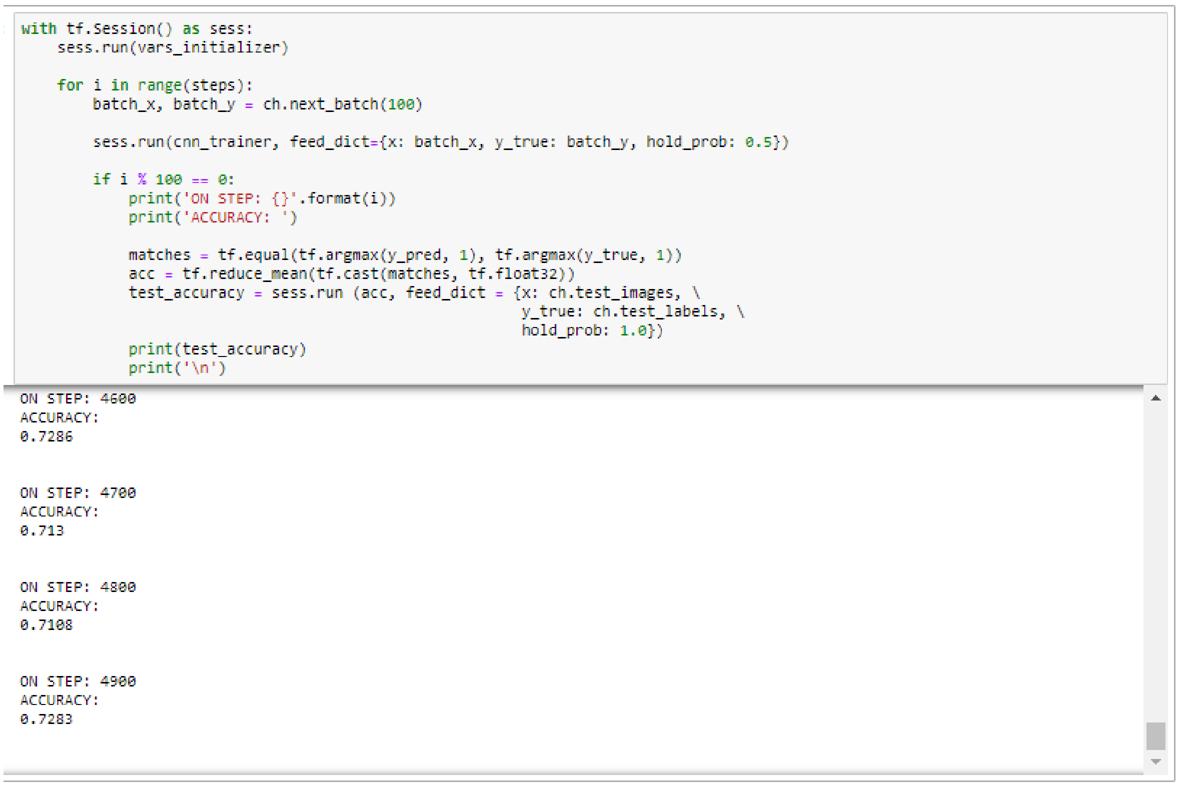
```python
#Creating the fourth pooling Layer
pooling_layer_4_outputs = create_maxpool2by2_and_reduce_spatial_size(conv_relu_layer_4_outputs)
```

11. Outputs obtained from the fourth pooling layer have been reshaped and flattened and that data is made ready to fit into the first fully connected layer.
12. The first Fully Connected layer and ReLu layer are created, and the output data is then sent to the dropout layer.
13. The dropout layer is created, and the fraction of outputs are sent to the output layer.
14. The final output layer is created, and the outputs will be produced.

Training and testing the model:

1. A trainer is created to train the model.
2. Variables are initialized by using the variable initializer.
3. The step count is set to 5000.
4. tf.session() has been created to train and test the updated Convolutional neural network.

Results:



```
with tf.Session() as sess:
    sess.run(vars_initializer)

    for i in range(steps):
        batch_x, batch_y = ch.next_batch(100)

        sess.run(cnn_trainer, feed_dict={x: batch_x, y_true: batch_y, hold_prob: 0.5})

        if i % 100 == 0:
            print('ON STEP: {}'.format(i))
            print('ACCURACY: ')

            matches = tf.equal(tf.argmax(y_pred, 1), tf.argmax(y_true, 1))
            acc = tf.reduce_mean(tf.cast(matches, tf.float32))
            test_accuracy = sess.run (acc, feed_dict = {x: ch.test_images, \
                                                        y_true: ch.test_labels, \
                                                        hold_prob: 1.0})

            print(test_accuracy)
            print('\n')
```

```
ON STEP: 4600
ACCURACY:
0.7286


ON STEP: 4700
ACCURACY:
0.713


ON STEP: 4800
ACCURACY:
0.7108


ON STEP: 4900
ACCURACY:
0.7283
```

As we can say from the above, CNN is constructed. Adam optimizer is used to train the model with a step count of 5000 and used softmax cross entropy to decrease the gap between the predicted and the obtained outputs. From the output above, we can see that the accuracy is 0.7283. This means that 72.83% of the images in the dataset, are recognized.

To get improvement in the accuracy value of CNN, I have added two more convolutional layers and two more pooling layers. After performing the training and the testing of the model, the accuracy came to be 72.83% which is 2% more than the previous accuracy value from Part-4. Therefore, the accuracy i.e., the performance of CNN has been increased.

Conclusion:

In this project, in the first part, I have explained about the data structure and the critical information of the dataset which contains images of fruits and vegetables. In the second part, I have described how to upload CIFAR_10 dataset to remote VM server. From the third part, I have built, trained, and tested CNN for CIFAR_10 dataset with an accuracy value of 70.55%. In the fourth part, I have compared the accuracy values of CIFAR_10 dataset with the accuracy values of MNIST dataset. For MNIST dataset, the accuracy value is 98.99% which is very higher than that of CIFAR_10 dataset. This is due to the reason that MNIST dataset contains black and white images which are easily identified whereas the CIFAR_10 dataset has colour images which are comparatively harder to identify. In the fifth part, the CNN model is improved by adding two more convolutional layers and two more pooling layers. The updated CNN model has the accuracy value of 72.83% which is almost 2% more than the previous one.

YouTube links:

https://youtu.be/nPSLlA33cME

https://youtu.be/H2y5deLgEfY

https://youtu.be/7NjTJpnAz9s