CV Assignemnt 4
Image Segmentation with CNN

Shivani Chepuri
2018122004

Scope: Implement a CNN which takes images with a single object in them and classifies them.
*The parameters/weights after training have been stored in a file named **'keras_cifar10_trained_model.h5'**. It is uploaded on Google Drive. Please find it in this link.
https://drive.google.com/open?id=13MlUwwEp8Fz4Xbirij-jYSfegtp4T_aD

*Please find other results also in the above link.
* All the training has been done on my own laptop, using python .py and .ipynb. No GPU is used.

The implementation is done in 2 files → cnn1.py, cnn2.py
Training and storing the model, Model evaluation is done in cnn1.py

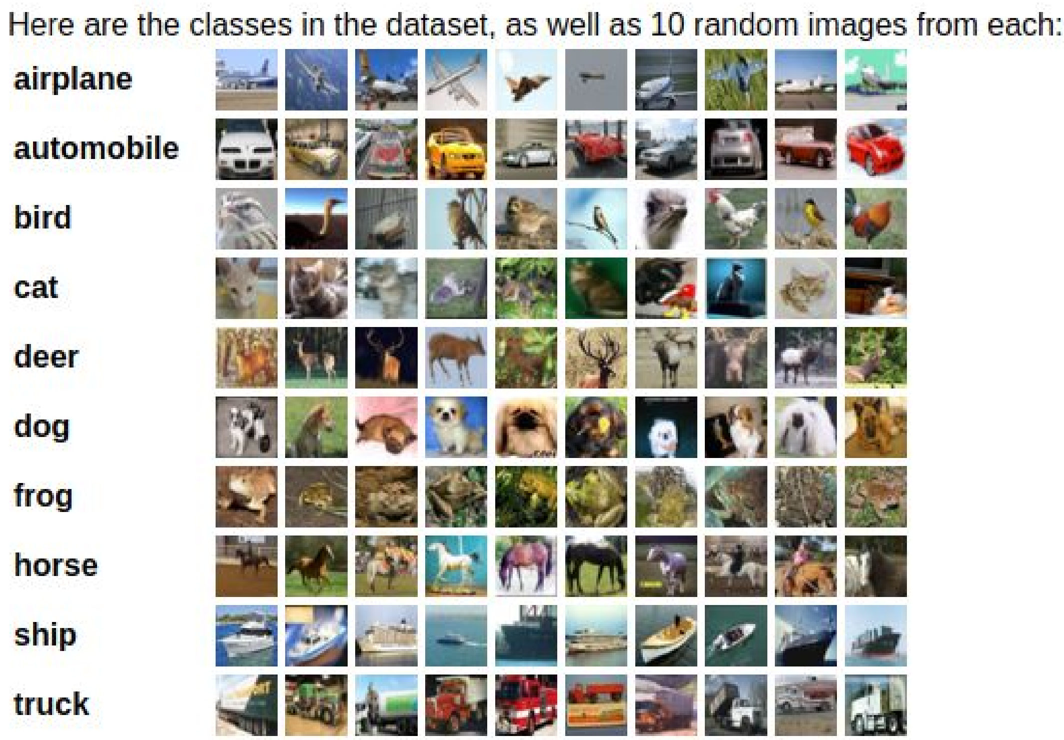
Testing the cifar dataset i.e, its test-data batch, (predicting and printing all the class of each of the 10,000 images in the cifar dataset) is done in cnn2.py

Please note → 3 images from the internet, out of cifar dataset are also tested and classes are predicted.
Please find these images also in the above link. I.e,
https://drive.google.com/open?id=13MlUwwEp8Fz4Xbirij-jYSfegtp4T_aD

Implementation details:
Basic Structure of the CNN built:

Layers	Layers Parameter	Activation Function
Conv2D	32,size=(3,3)	Relu
Conv2D	32,size=(3,3)	Relu
Conv2D	32,size=(3,3)	Relu
Maxpooling2D	Size=(2,2)	
Dropout	0.25	
Conv2D	64,size=(3,3)	Relu
Conv2D	64,size=(3,3)	Relu
Conv2D	64,size=(3,3)	Relu
Maxpooling2D	Size=(2,2)	
Dropout	0.25	
Dense	512	Relu
Dropout	0.5	
Dense	10	Softmax

The DL module used: **Tensor flow and Keras**
Dataset: cifar-10-python
Number of classes in the dataset: **10**



The CIFAR-10 dataset consists of 60000 32x32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

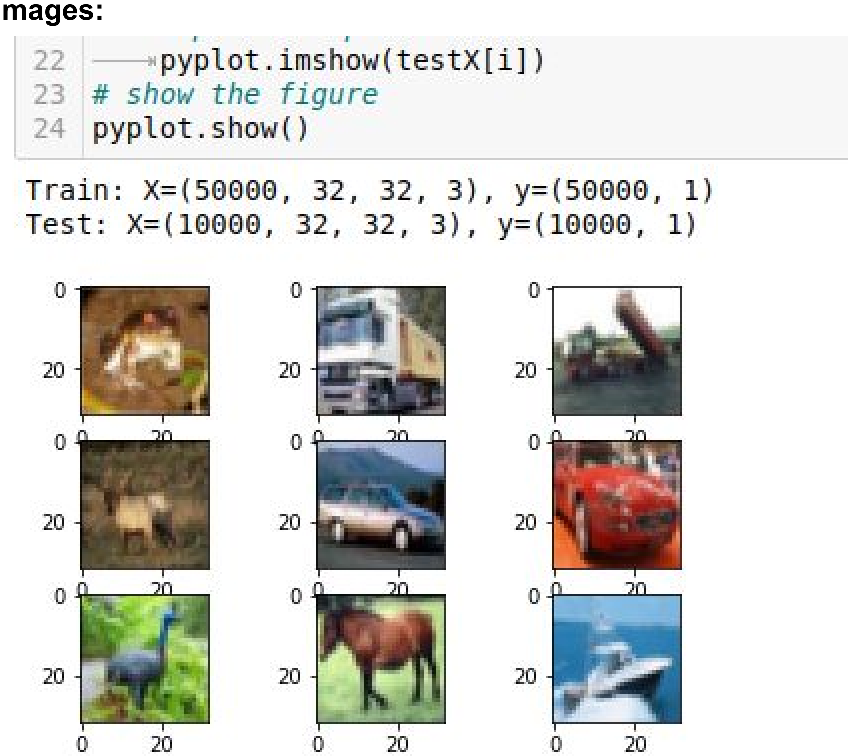
Visualizing the dataset:
The following is the code that downloads and loads the cifar dataset in anaconda/dist-packages. It basically structures the data into TEST and TRAIN sets.

```
# load dataset
(trainX, trainy), (testX, testy) = cifar10.load_data()

print('Train: X=%s, y=%s' % (trainX.shape, trainy.shape))
print('Test: X=%s, y=%s' % (testX.shape, testy.shape))
```

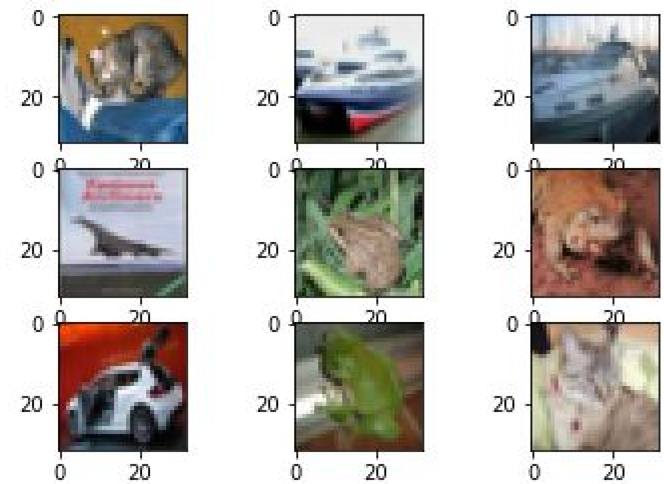
```
#--------------data visualisation-----------------------------
# plot first few images of the training dataset
for i in range(9):
	pyplot.subplot(330 + 1 + i)
	pyplot.imshow(trainX[i])
pyplot.show()

# plot first few images of the testing dataset
for i in range(9):
	pyplot.subplot(330 + 1 + i)
	pyplot.imshow(testX[i])
pyplot.show()
```

Results (Screenshots):

Train Images:



Test Images:



Parameters used and changed:
Batches = 32 to 4 varied(changed)
Optimizer used: RMSProp(changed)
Loss: categorical_crossentropy(changed)
Epochs = 100 to 5 varied(changed)

Data Augmentation: Implemented. If data augmentation is enabled, the model will work in real-time. Instead of aiming for RCNN, RPN or multi-object classification, etc, I have tried to make the model work in real-time.

Code Explanation:
The following code in the notebook is where the model is being trained **cnn1.py**

```
from __future__ import print_function
import keras
from keras.datasets import cifar10
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D
import os
```

Parameters - These are changed many times to get best performance

```
batch_size = 4
num_classes = 10
```

```
epochs = 14
data_augmentation = False
num_predictions = 20
save_dir = os.path.join(os.getcwd(), 'saved_models')
model_name = 'keras_cifar10_trained_model_1.h5'

# train test split
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
```

Make data amenable to keras library

```
# Convert class vectors to binary class matrices.
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

Design of the Neural Network

```
model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same',
            input_shape=x_train.shape[1:]))
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(256))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))
```

Optimizing

```
# initiate RMSprop optimizer
```

opt = keras.optimizers.RMSprop(learning_rate=0.001, decay=1e-4)

 # Let's train the model using RMSprop
Compiling the model, with the trained parameters
```
model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
```

Normal data fitting
```
if not data_augmentation:
    print('Not using data augmentation.')
    history = model.fit(x_train, y_train,
              batch_size=batch_size,
              epochs=epochs,
              validation_data=(x_test, y_test),
              shuffle=True)
Else:
```
DataGenerator for Real time classification
```
    print('Using real-time data augmentation.')
    #preprocessing fro realtime data augmentation:
    datagen = ImageDataGenerator(
        featurewise_center=False, samplewise_center=False,
featurewise_std_normalization=False,
        samplewise_std_normalization=False, zca_whitening=False, zca_epsilon=1e-06,
        rotation_range=0, width_shift_range=0.1, height_shift_range=0.1, shear_range=0.,
        zoom_range=0., channel_shift_range=0., fill_mode='nearest', cval=0., horizontal_flip=True,
        vertical_flip=False, rescale=None, preprocessing_function=None, data_format=None,
validation_split=0.0)

    #feature-wise normalization.
    datagen.fit(x_train)
```
Save the model and results to plot accuracy and loss
```
    # create models by datagen.flow()
    # then fit the models with the pre-trained parameters
    history = model.fit_generator(datagen.flow(x_train, y_train,
                                  batch_size=batch_size),
                        epochs=epochs,
                        validation_data=(x_test, y_test),
```

```
                        workers=4)
```

Save the model parameters to the file ‘keras_cifar10_trained_model.h5'

```
# Save model and weights
if not os.path.isdir(save_dir):
    os.makedirs(save_dir)
model_path = os.path.join(save_dir, model_name)
model.save(model_path)
print('Saved trained model at %s ' % model_path)

# Score trained model.
scores = model.evaluate(x_test, y_test, verbose=1)

print('Test accuracy in percentage: ', scores[1]*100)
print('Test loss: ', scores[0])
```

Plot Accuracy

```
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

Plot Loss

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

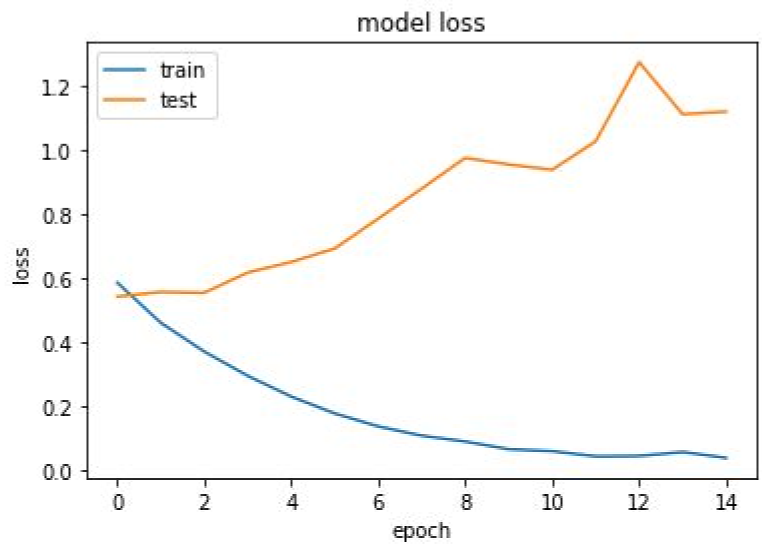
The following code in the notebook is where the model is being tested **cnn2.py**

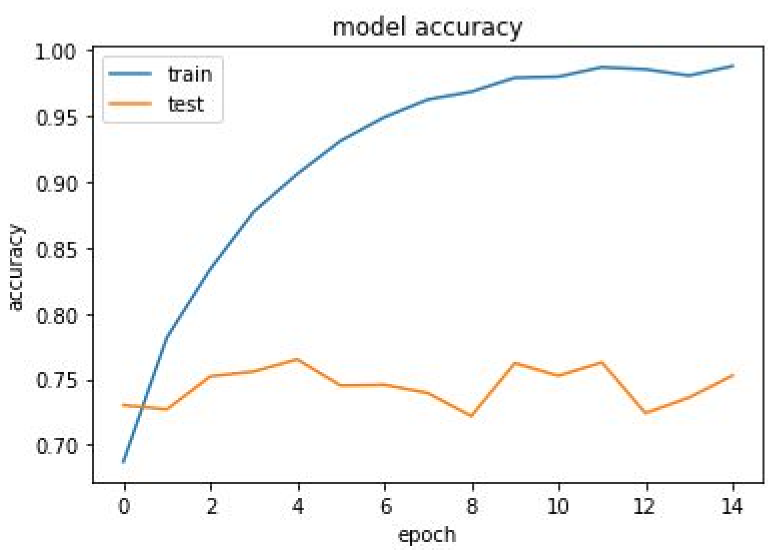
```
import math
from matplotlib import pyplot
from keras.datasets import cifar10
from keras.models import load_model
from keras.preprocessing import image
import numpy as np
import matplotlib.pyplot as plt
from numpy import linalg as LA
```

```
# load dataset
(trainX, trainy), (testX, testy) = cifar10.load_data()

print('Train: X=%s, y=%s' % (trainX.shape, trainy.shape))
print('Test: X=%s, y=%s' % (testX.shape, testy.shape))

#--------------data visualisation------------------------------
# plot first few images of the training dataset
for i in range(9):
	pyplot.subplot(330 + 1 + i)
	pyplot.imshow(trainX[i])
pyplot.show()

# plot first few images of the testing dataset
for i in range(9):
	pyplot.subplot(330 + 1 + i)
	pyplot.imshow(testX[i])
pyplot.show()

#load the test_data batch of cifar10
def unpickle(file):
    import pickle
    with open(file, 'rb') as fo:
        dict = pickle.load(fo, encoding='bytes')
    return dict

file1 = file1 = '/home/shivani/cifar-10-batches-py/test_batch'
batch1 = unpickle(file1)
data = []
labels = []
data.append(batch1[b'data'])
labels.append(batch1[b'labels'])
data = np.array(data)
labels = np.array(labels)
#rgb_data = np.concatenate(data)
#batch_labels = np.concatenate(labels)

# dimensions of our images
img_width, img_height = 32, 32

# load the model that is pretrained i.e, 'keras_cifar10_trained_model_1.h5'

model = load_model('/home/shivani/saved_models/keras_cifar10_trained_model_1.h5')
```

```
model.compile(loss='binary_crossentropy',
            optimizer='rmsprop',
            metrics=['accuracy'])
#-------------------------random test images other than those in cifar10, (from internet))
s = ['/home/shivani/Downloads/car1.jpg', '/home/shivani/Downloads/bird1.jpg',
'/home/shivani/Downloads/ship1.jpg']

# predicting images
images = []
for i in range(3):
	img = image.load_img(s[i], target_size = (img_width, img_height))
	x = image.img_to_array(img)
	x = np.expand_dims(x, axis=0)

	images = np.vstack([x])
	#images = np.concatenate(images,x, axis = 1)
	classes = model.predict_classes(images, batch_size=10)
	#print(images.shape)
	#print (classes[0])

data = data[0]

classes_test = []
for i in range(10000):
	d = data[i,:].reshape((1,32,32,3))
	classes_test.append(model.predict_classes(d, batch_size=10))
	#print(len(d))
	#print("\n")
	#print(d)

print(len(classes_test))
print(classes_test)

#print (classes)
#print (classes[0])
#print (classes[0][0])
```

Plots for Model Accuracy and Model Loss for epoch = 14

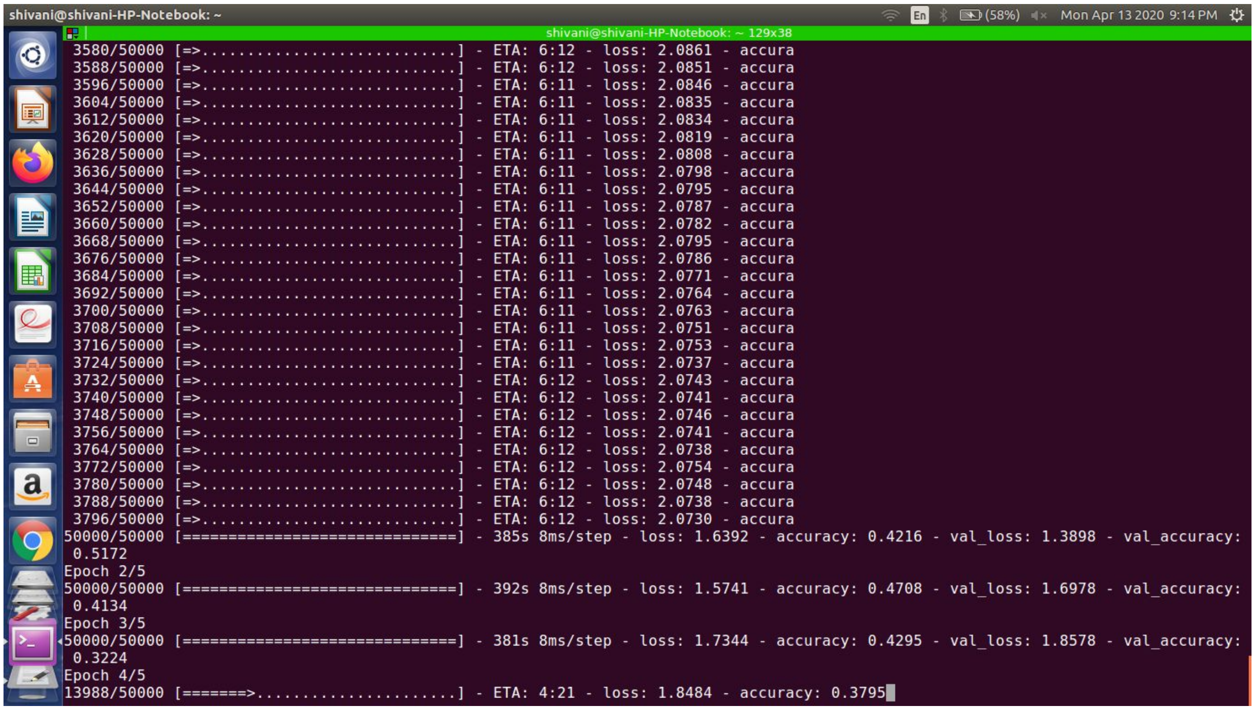




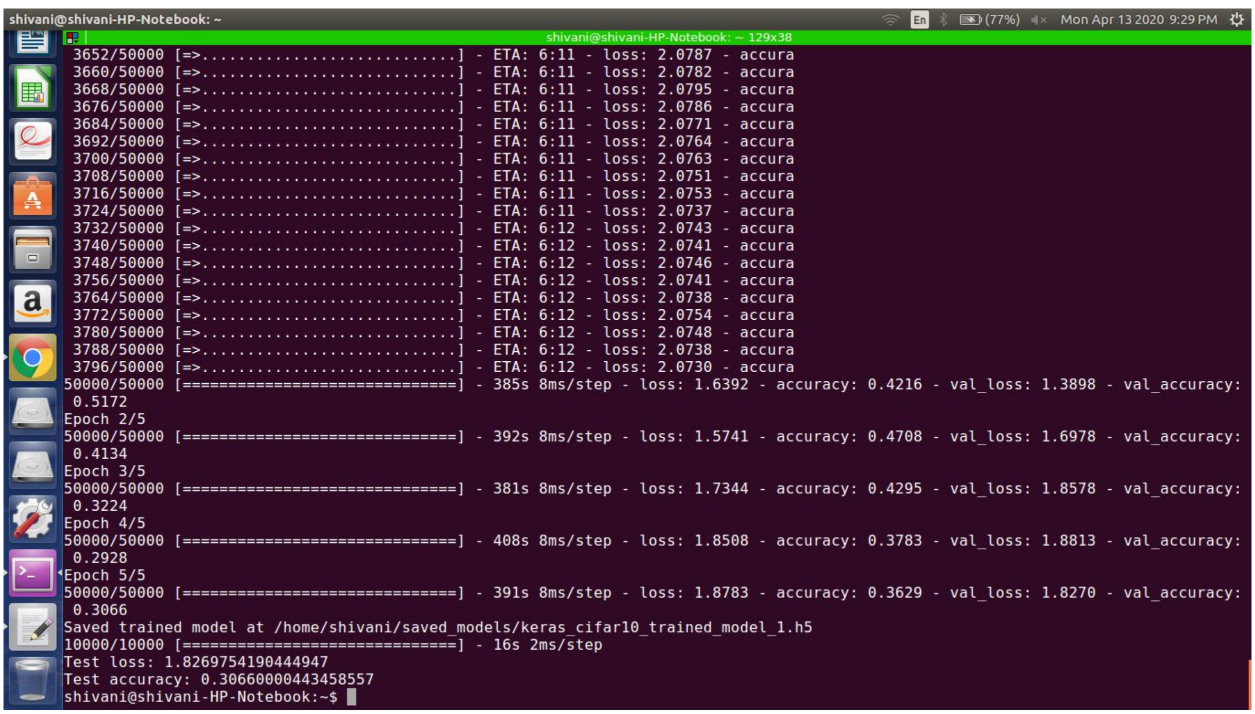
The time for training, with respect to different parameters:
Epoch 14 -- 2 hrs, epoch 5 -- 30 min
Change (Improvement) of accuracy with epoch: Best accuracy is got with epoch around 14.

For the optimizer, the experimented parameters are:
learning_rate=0.001, decay=1e-4
Observation: Lower learning rate → convergence is slow. Takes too many epochs to converge
Higher Learning rate → Converge is fast but should be taken care that oscillation won't happen

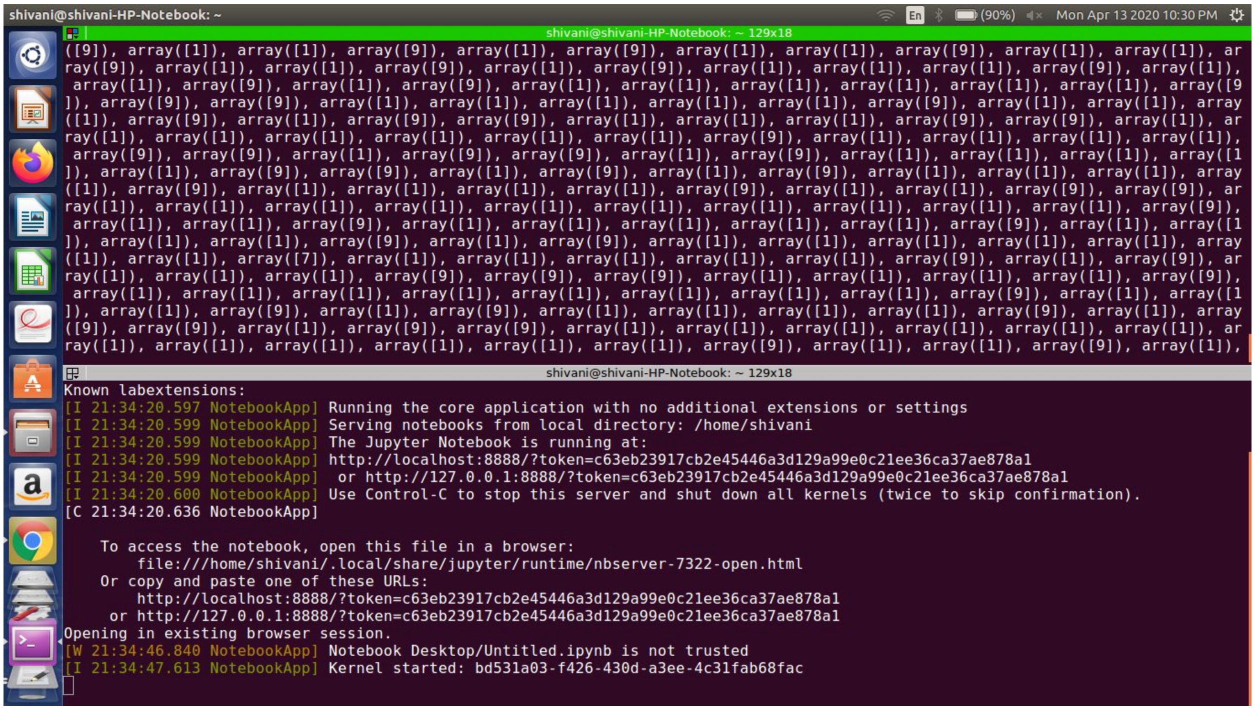
Results:
Training(epoch = 5). Therefore, the accuracy is very low. Around 30%
Training(epoch = 14). Therefore, the accuracy is very good. Around 70%



Results of classification:



Predicted classes -- epoch 5


shivani@shivani-HP-Notebook: ~
En (90%) Mon Apr 13 2020 10:30 PM
shivani@shivani-HP-Notebook: ~ 129x18
([9]), array([1]), array([1]), array([9]), array([1]), array([9]), array([1]), array([1]), array([9]), array([1]), array([1]), ar
ray([9]), array([1]), array([1]), array([9]), array([1]), array([9]), array([1]), array([1]), array([1]), array([9]), array([1]),
array([1]), array([9]), array([1]), array([9]), array([1]), array([1]), array([1]), array([1]), array([1]), array([1]), array([9
]), array([9]), array([9]), array([1]), array([1]), array([1]), array([1]), array([1]), array([9]), array([1]), array([1]), array
([1]), array([9]), array([1]), array([9]), array([9]), array([1]), array([1]), array([1]), array([9]), array([9]), array([1]), ar
ray([1]), array([1]), array([1]), array([1]), array([1]), array([1]), array([9]), array([1]), array([1]), array([1]), array([1]),
array([9]), array([9]), array([1]), array([9]), array([9]), array([1]), array([9]), array([1]), array([1]), array([1]), array([1
]), array([1]), array([9]), array([9]), array([1]), array([9]), array([1]), array([9]), array([1]), array([1]), array([1]), array
([1]), array([9]), array([1]), array([1]), array([1]), array([1]), array([9]), array([1]), array([1]), array([9]), array([9]), ar
ray([1]), array([1]), array([1]), array([1]), array([1]), array([1]), array([1]), array([1]), array([1]), array([1]), array([9]),
array([1]), array([1]), array([9]), array([1]), array([1]), array([1]), array([1]), array([1]), array([9]), array([1]), array([1
]), array([1]), array([1]), array([9]), array([1]), array([9]), array([1]), array([1]), array([1]), array([1]), array([1]), array
([1]), array([1]), array([7]), array([1]), array([1]), array([1]), array([1]), array([1]), array([9]), array([1]), array([9]), ar
ray([1]), array([1]), array([1]), array([9]), array([9]), array([9]), array([1]), array([1]), array([1]), array([1]), array([9]),
array([1]), array([1]), array([1]), array([1]), array([1]), array([1]), array([1]), array([1]), array([9]), array([1]), array([1
]), array([1]), array([9]), array([1]), array([9]), array([1]), array([1]), array([1]), array([1]), array([9]), array([1]), array
([9]), array([9]), array([1]), array([9]), array([9]), array([1]), array([1]), array([1]), array([1]), array([1]), array([1]), ar
ray([1]), array([1]), array([1]), array([1]), array([1]), array([1]), array([9]), array([1]), array([1]), array([9]), array([1]),
shivani@shivani-HP-Notebook: ~ 129x18
Known labextensions:
[I 21:34:20.597 NotebookApp] Running the core application with no additional extensions or settings
[I 21:34:20.599 NotebookApp] Serving notebooks from local directory: /home/shivani
[I 21:34:20.599 NotebookApp] The Jupyter Notebook is running at:
[I 21:34:20.599 NotebookApp] http://localhost:8888/?token=c63eb23917cb2e45446a3d129a99e0c21ee36ca37ae878a1
[I 21:34:20.599 NotebookApp] or http://127.0.0.1:8888/?token=c63eb23917cb2e45446a3d129a99e0c21ee36ca37ae878a1
[I 21:34:20.600 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 21:34:20.636 NotebookApp]
To access the notebook, open this file in a browser:
file:///home/shivani/.local/share/jupyter/runtime/nbserver-7322-open.html
Or copy and paste one of these URLs:
http://localhost:8888/?token=c63eb23917cb2e45446a3d129a99e0c21ee36ca37ae878a1
or http://127.0.0.1:8888/?token=c63eb23917cb2e45446a3d129a99e0c21ee36ca37ae878a1
Opening in existing browser session.
[W 21:34:46.840 NotebookApp] Notebook Desktop/Untitled.ipynb is not trusted
[I 21:34:47.613 NotebookApp] Kernel started: bd531a03-f426-430d-a3ee-4c31fab68fac