Python and Deep Learning Programming Lab 4

Lab ID:14

Student ID: 20

Student Name : Raju Nekadi .

Student ID: 19

Student Name: Sneha Mishra

Github Link:

https://github.com/rnekadi/CSEE5590_PYTHON_DEEPLEARNING_FALL2018/tree/master/Lab4

Video Link:

https://youtu.be/KUsG7yV8oz0

Introduction:

To apply CNN, LSTM and RNN models on text and image datasets for Text and Image Classification.

Objective:

Implementing the Text Classification using the CNN Model on text dataset.

Approaches:

For Text Classification using CNN we have used the Positive and Negative Polarity datasets.First of all we have loaded preprocessed polarity data from files, split the data into words and generate labels and returns vectors, labels, vocabulary, and inverse vocabulary.

Split the processed data into train and test for our model.

The setup the CNN Model using Embedding of Size 256 using the vocabulary we created. We have set 3 convolutional layers of filter size 3 4 5 ,3 MaxPool layer, Relu activation function and then compile our model using Adam optimizer. Perform Model fitting on train data and evaluated the model score and accuracy.

Also plotted the Loss and Accuracy on graph. The detailed explanation given below in workflow section.

Workflow:

Fist of all loaded the preprocessed file and extracted the input vectors, labels, vocabulary, and inverse vocabulary using the data_helprs.py file.

data_helpers.py

```python
    return padded_sentences


def build_vocab(sentences):
    """
    Builds a vocabulary mapping from word to index based on the sentences.
    Returns vocabulary mapping and inverse vocabulary mapping.
    """
    # Build vocabulary
    word_counts = Counter(itertools.chain(*sentences))
    # Mapping from index to word
    vocabulary_inv = [x[0] for x in word_counts.most_common()]
    vocabulary_inv = list(sorted(vocabulary_inv))
    # Mapping from word to index
    vocabulary = {x: i for i, x in enumerate(vocabulary_inv)}
    return [vocabulary, vocabulary_inv]


def build_input_data(sentences, labels, vocabulary):
    """
    Maps sentences and labels to vectors based on a vocabulary.
    """
    x = np.array([[vocabulary[word] for word in sentence] for sentence in sentences])
    y = np.array(labels)
    return [x, y]


def load_data():
    """
    Loads and preprocessed data for the dataset.
    Returns input vectors, labels, vocabulary, and inverse vocabulary.
    """
    # Load and preprocess data
    sentences, labels = load_data_and_labels()
    sentences_padded = pad_sentences(sentences)
    vocabulary, vocabulary_inv = build_vocab(sentences_padded)
    x, y = build_input_data(sentences_padded, labels, vocabulary)
    return [x, y, vocabulary, vocabulary_inv]

```

Train and Test Data Creation

```python
# cnn_model.py
from keras.layers import Reshape, Flatten, Dropout, Concatenate
from keras.callbacks import ModelCheckpoint
from keras.optimizers import Adam
from keras.models import Model
from sklearn.model_selection import train_test_split
from data_helpers import load_data
import matplotlib.pyplot as plt

print('Loading data')
x, y, vocabulary, vocabulary_inv = load_data()

# x.shape -> (10662, 56)
# y.shape -> (10662, 2)
# len(vocabulary) -> 18765
# len(vocabulary_inv) -> 18765

X_train, X_test, y_train, y_test = train_test_split( x, y, test_size=0.2, random_state=42)
```

Creating Model , adding the Embedding CONV2D,MaxPool and Compile the model.

```python
# cnn_model.py
sequence_length = x.shape[1] # 56
vocabulary_size = len(vocabulary_inv) # 18765
embedding_dim = 256
filter_sizes = [3, 4, 5]
num_filters = 512
drop = 0.5

epochs = 20
batch_size = 80

# this returns a tensor
print("Creating Model...")
inputs = Input(shape=(sequence_length,), dtype='int32')
embedding = Embedding(input_dim=vocabulary_size, output_dim=embedding_dim, input_length=sequence_length)(inputs)
reshape = Reshape((sequence_length, embedding_dim, 1))(embedding)

conv_0 = Conv2D(num_filters, kernel_size=(filter_sizes[0], embedding_dim), padding='valid', kernel_initializer='normal', activation='relu')
conv_1 = Conv2D(num_filters, kernel_size=(filter_sizes[1], embedding_dim), padding='valid', kernel_initializer='normal', activation='relu')
conv_2 = Conv2D(num_filters, kernel_size=(filter_sizes[2], embedding_dim), padding='valid', kernel_initializer='normal', activation='relu')

maxpool_0 = MaxPool2D(pool_size=(sequence_length - filter_sizes[0] + 1, 1), strides=(1, 1), padding='valid')(conv_0)
maxpool_1 = MaxPool2D(pool_size=(sequence_length - filter_sizes[1] + 1, 1), strides=(1, 1), padding='valid')(conv_1)
maxpool_2 = MaxPool2D(pool_size=(sequence_length - filter_sizes[2] + 1, 1), strides=(1, 1), padding='valid')(conv_2)

concatenated_tensor = Concatenate(axis=1)([maxpool_0, maxpool_1, maxpool_2])
flatten = Flatten()(concatenated_tensor)
dropout = Dropout(drop)(flatten)
output = Dense(units=2, activation='softmax')(dropout)

# this creates a model that includes
model = Model(inputs=inputs, outputs=output)

checkpoint = ModelCheckpoint('weights.{epoch:03d}-{val_acc:.4f}.hdf5', monitor='val_acc', verbose=1, save_best_only=True, mode='auto')
adam = Adam(lr=1e-4, beta_1=0.9, beta_2=0.999, epsilon=1e-08, decay=0.0)

model.compile(optimizer=adam, loss='binary_crossentropy', metrics=['accuracy'])
```

Fitting the Model on train data and train it . Calculated the accuracy and score and plot loss and accuracy on graph.

cnn_model.py

```python
model.compile(optimizer=adam, loss='binary_crossentropy', metrics=['accuracy'])


print("Simplified convolutional neural network")
model.summary()

print("Traning Model...")
history = model.fit(X_train, y_train, batch_size=batch_size, epochs=epochs, verbose=1, callbacks=[checkpoint],
                             validation_data=(X_test, y_test))  # starts training

print("Evaluating the Model")

score, accuracy = model.evaluate(X_test, y_test, batch_size=batch_size, verbose=1)

print('Score: %.2f' %(score))
print('Validation Accuracy: %.2f' % (accuracy))

# Plot The CNN LOSS
fig1 = plt.figure()
plt.plot(history.history['loss'], 'r', linewidth=3.0)
plt.plot(history.history['val_loss'], 'b', linewidth=3.0)
plt.legend(['Training loss', 'Validation Loss'], fontsize=18)
plt.xlabel('Epochs ', fontsize=16)
plt.ylabel('Loss', fontsize=16)
plt.title('Loss Curves :CNN', fontsize=16)
fig1.savefig('loss_cnn.png')


# Plot the CNN Accuracy

fig2=plt.figure()
plt.plot(history.history['acc'], 'r', linewidth=3.0)
plt.plot(history.history['val_acc'], 'b', linewidth=3.0)
plt.legend(['Training Accuracy', 'Validation Accuracy'], fontsize=18)
plt.xlabel('Epochs ', fontsize=16)
plt.ylabel('Accuracy', fontsize=16)
plt.title('Accuracy Curves : CNN', fontsize=16)
fig2.savefig('accuracy_cnn.png')
plt.show()
```

This concludes our workflow.

Dataset:

http://www.cs.cornell.edu/people/pabo/movie-review-data/

Parameters:

None

Evaluation:

After running the model for the for 20 Epochs, batch size of 80 and dropout of 50% we got the got the score and accuracy of 72% and 75% respectively.

cnn_model

```
  80/2133 [>.............................] - ETA: 4s
 160/2133 [=>............................] - ETA: 4s
 240/2133 [==>...........................] - ETA: 4s
 320/2133 [===>..........................] - ETA: 4s
 400/2133 [====>.........................] - ETA: 4s
 480/2133 [=====>........................] - ETA: 3s
 560/2133 [======>.......................] - ETA: 3s
 640/2133 [========>.....................] - ETA: 3s
 720/2133 [=========>....................] - ETA: 3s
 800/2133 [==========>...................] - ETA: 3s
 880/2133 [===========>..................] - ETA: 2s
 960/2133 [============>.................] - ETA: 2s
1040/2133 [=============>................] - ETA: 2s
1120/2133 [==============>...............] - ETA: 2s
1200/2133 [===============>..............] - ETA: 2s
1280/2133 [=================>............] - ETA: 1s
1360/2133 [==================>...........] - ETA: 1s
1440/2133 [===================>..........] - ETA: 1s
1520/2133 [====================>.........] - ETA: 1s
1600/2133 [=====================>........] - ETA: 1s
1680/2133 [======================>.......] - ETA: 1s
1760/2133 [=======================>......] - ETA: 0s
1840/2133 [========================>.....] - ETA: 0s
1920/2133 [==========================>...] - ETA: 0s
2000/2133 [===========================>..] - ETA: 0s
2080/2133 [============================>.] - ETA: 0s
2133/2133 [==============================] - 5s 2ms/step
Score: 0.72
Validation Accuracy: 0.75
```

Conclusion:

CNN Model is very good for Text Classification.

Objective:

Implementing the Text Classification using the LSTM Model on text dataset.

Approaches:

For Text Classification using CNN we have used the Positive and Negative Polarity datasets.First of all we have loaded preprocessed polarity data from files, split the data into words and generate labels and returns vectors, labels, vocabulary, and inverse vocabulary.

Split the processed data into train and test for our model.

The setup the LSTM Model using Embedding. Add the LSTM layer of 100 with dropout 20% .Added the Dense of size 2 and activation used is sigmoid . Compile model using adam optimizer Perform Model fitting on train data and evaluated the model score and accuracy.

Also plotted the Loss and Accuracy on graph. The detailed explanation given below in workflow section.

Workflow:

Fist of all loaded the preprocessed file and extracted the input vectors, labels, vocabulary, and inverse vocabulary using the data_helprs.py file.

data_helpers.py

```python
    return padded_sentences


def build_vocab(sentences):
    """
    Builds a vocabulary mapping from word to index based on the sentences.
    Returns vocabulary mapping and inverse vocabulary mapping.
    """
    # Build vocabulary
    word_counts = Counter(itertools.chain(*sentences))
    # Mapping from index to word
    vocabulary_inv = [x[0] for x in word_counts.most_common()]
    vocabulary_inv = list(sorted(vocabulary_inv))
    # Mapping from word to index
    vocabulary = {x: i for i, x in enumerate(vocabulary_inv)}
    return [vocabulary, vocabulary_inv]


def build_input_data(sentences, labels, vocabulary):
    """
    Maps sentences and labels to vectors based on a vocabulary.
    """
    x = np.array([[vocabulary[word] for word in sentence] for sentence in sentences])
    y = np.array(labels)
    return [x, y]


def load_data():
    """
    Loads and preprocessed data for the dataset.
    Returns input vectors, labels, vocabulary, and inverse vocabulary.
    """
    # Load and preprocess data
    sentences, labels = load_data_and_labels()
    sentences_padded = pad_sentences(sentences)
    vocabulary, vocabulary_inv = build_vocab(sentences_padded)
    x, y = build_input_data(sentences_padded, labels, vocabulary)
    return [x, y, vocabulary, vocabulary_inv]

```

Train and Test Data Creation

lstm_model.py

```python
from keras.layers import Dense, Embedding, LSTM
from keras.optimizers import Adam
from keras.models import Model
from keras.models import Sequential
from sklearn.model_selection import train_test_split
from data_helpers import load_data
import matplotlib.pyplot as plt

print('Loading data')
x, y, vocabulary, vocabulary_inv = load_data()

# x.shape -> (10662, 56)
# y.shape -> (10662, 2)
# len(vocabulary) -> 18765
# len(vocabulary_inv) -> 18765

X_train, X_test, y_train, y_test = train_test_split( x, y, test_size=0.2, random_state=42)
```

Creating model, Embedding, Adding LSTM Layer Dense layer and compiling using adam optimizer.

lstm_model.py

```python
epochs = 20
batch_size = 80

# Creating the Model
print("Creating Model...")


model = Sequential()
model.add(Embedding(20000, 100, input_length=56))
model.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(2, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

model.summary()
```

We fitted the model on train data and train it . Calculated the accuracy and score and plot loss and accuracy on graph.

lstm_model.py

```python


## Fit the model

history = model.fit(X_train, y_train, validation_data=(X_test, y_test), batch_size=batch_size, epochs=epochs, verbose=1)


# Evaluate the model

score, accuracy = model.evaluate(X_test, y_test, batch_size=batch_size, verbose=1)

print('LSTM Score: %.2f' %(score))
print('LSTM Validation Accuracy: %.2f' % (accuracy))


# Plot The LSTM LOSS
fig1 = plt.figure()
plt.plot(history.history['loss'], 'r', linewidth=3.0)
plt.plot(history.history['val_loss'], 'b', linewidth=3.0)
plt.legend(['Training loss', 'Validation Loss'], fontsize=18)
plt.xlabel('Epochs ', fontsize=16)
plt.ylabel('Loss', fontsize=16)
plt.title('Loss Curves :LSTM', fontsize=16)
fig1.savefig('loss_lstm.png')


# Plot the LSTM Accuracy

fig2=plt.figure()
plt.plot(history.history['acc'], 'r', linewidth=3.0)
plt.plot(history.history['val_acc'], 'b', linewidth=3.0)
plt.legend(['Training Accuracy', 'Validation Accuracy'], fontsize=18)
plt.xlabel('Epochs ', fontsize=16)
plt.ylabel('Accuracy', fontsize=16)
plt.title('Accuracy Curves : LSTM', fontsize=16)
fig2.savefig('accuracy_lstm.png')
plt.show()
```

This Concludes our workflow.

Dataset:

http://www.cs.cornell.edu/people/pabo/movie-review-data/

Parameters:

None

Evaluation:

After running the model for the for 20 Epochs, batch size of 80 and dropout of 50% we got the got the score and accuracy of 69% and 50% respectively.

```
rnn_model
8529/8529 [==============================] - 28s 3ms/step - loss: 0.6932 - acc: 0.4991 - val_loss: 0.6933 - val_acc: 0.4979

  80/2133 [>.............................] - ETA: 1s
 160/2133 [=>............................] - ETA: 1s
 240/2133 [==>...........................] - ETA: 1s
 320/2133 [===>..........................] - ETA: 1s
 400/2133 [====>.........................] - ETA: 1s
 480/2133 [=====>........................] - ETA: 1s
 560/2133 [======>.......................] - ETA: 1s
 640/2133 [========>.....................] - ETA: 1s
 720/2133 [=========>....................] - ETA: 1s
 800/2133 [==========>...................] - ETA: 1s
 880/2133 [===========>..................] - ETA: 1s
 960/2133 [============>.................] - ETA: 0s
1040/2133 [=============>................] - ETA: 0s
1120/2133 [==============>...............] - ETA: 0s
1200/2133 [===============>..............] - ETA: 0s
1280/2133 [=================>............] - ETA: 0s
1360/2133 [==================>...........] - ETA: 0s
1440/2133 [===================>..........] - ETA: 0s
1520/2133 [====================>.........] - ETA: 0s
1600/2133 [=====================>........] - ETA: 0s
1680/2133 [======================>.......] - ETA: 0s
1760/2133 [=======================>......] - ETA: 0s
1840/2133 [========================>.....] - ETA: 0s
1920/2133 [==========================>...] - ETA: 0s
2000/2133 [===========================>..] - ETA: 0s
2080/2133 [============================>.] - ETA: 0s
2133/2133 [==============================] - 2s 852us/step
LSTM Score: 0.69
LSTM Validation Accuracy: 0.50

Process finished with exit code 0
```

Conclusion:

LSTM Model gives average performance for Text Classification as compared to CNN.

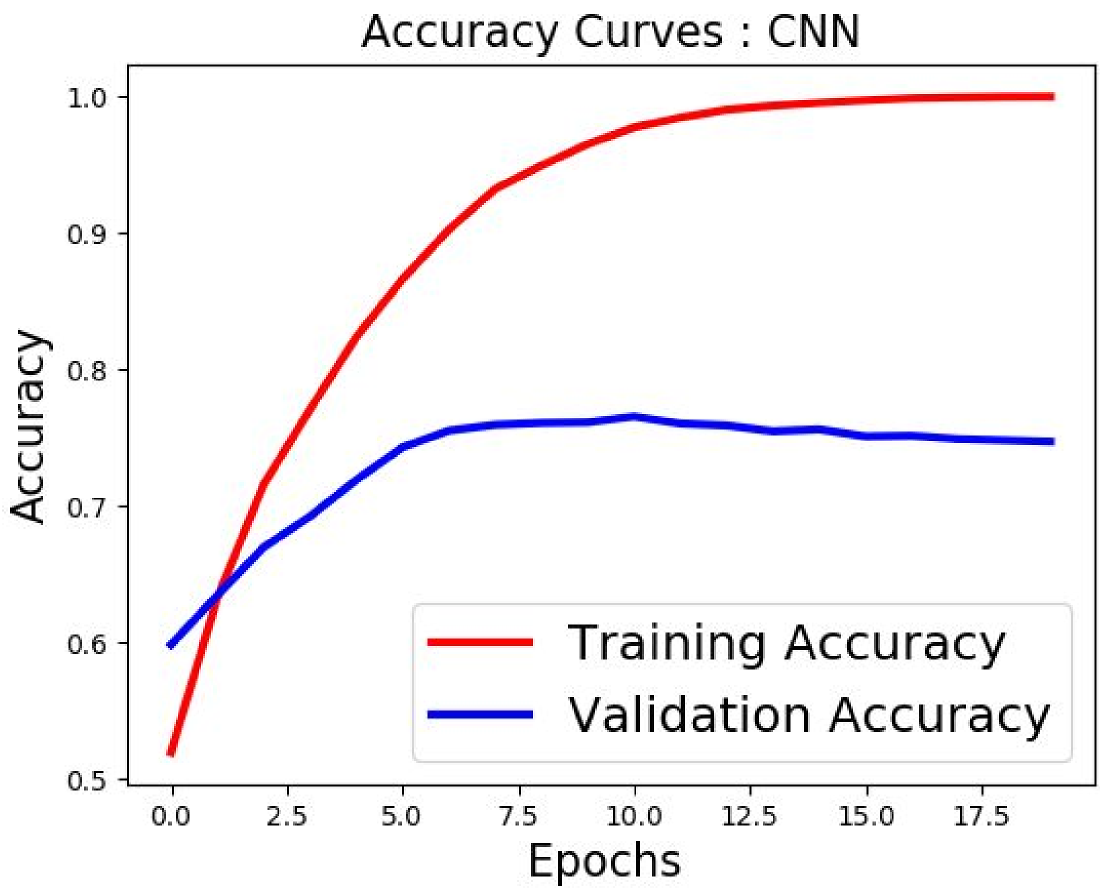
Objective:

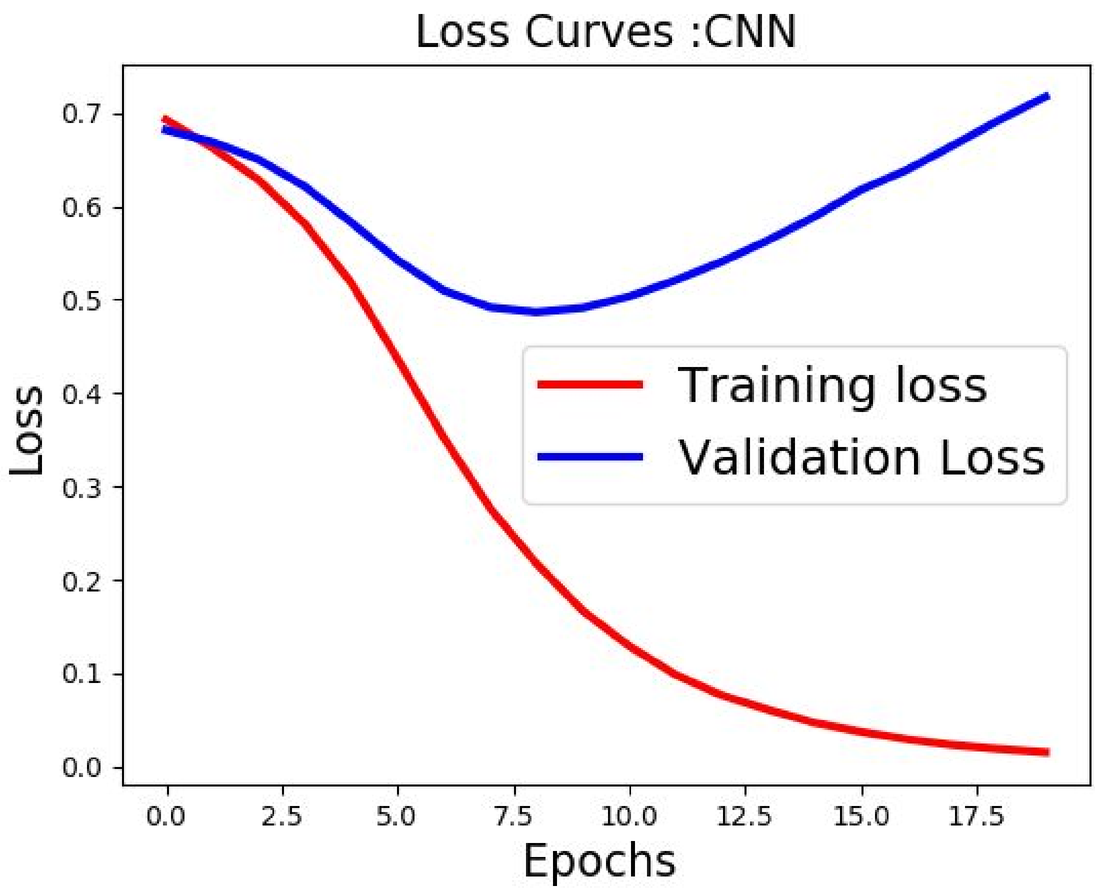
Comparing the LSTM and CNN Model Using above models.

Comparison Results:

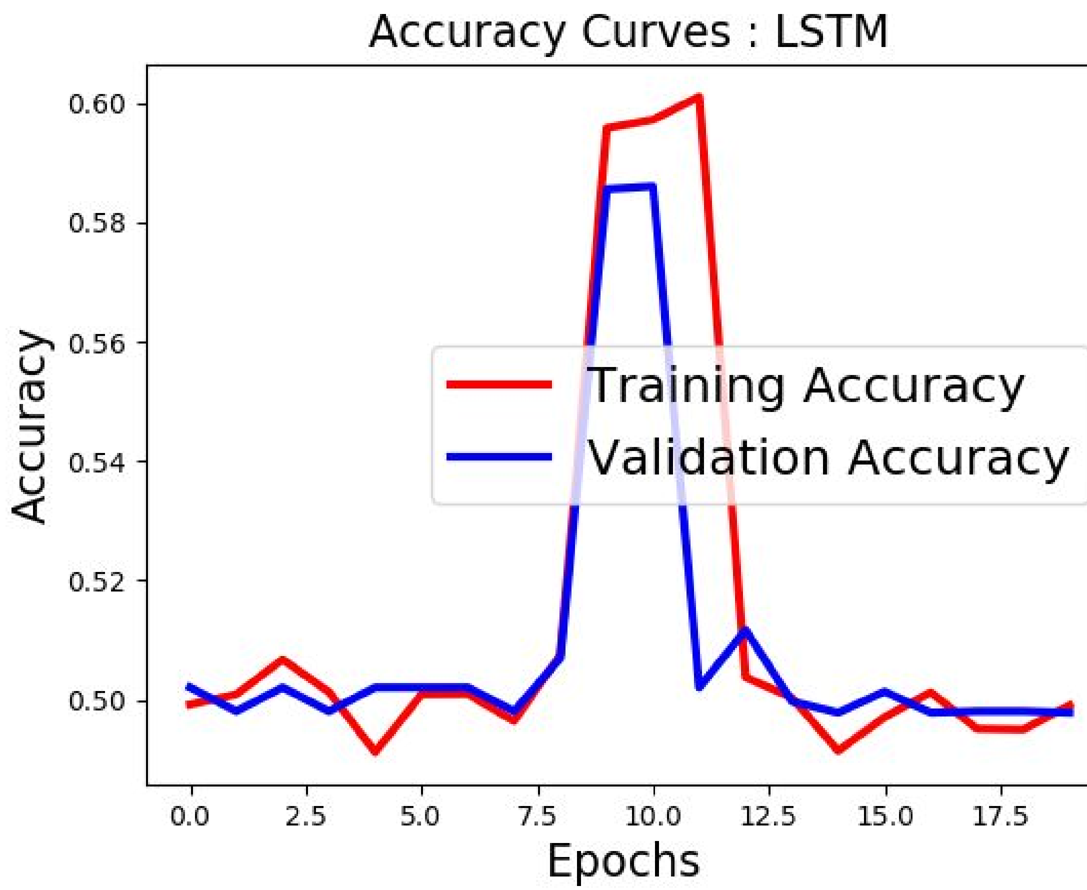
From the above models we got below Accuracy and Loss plots.

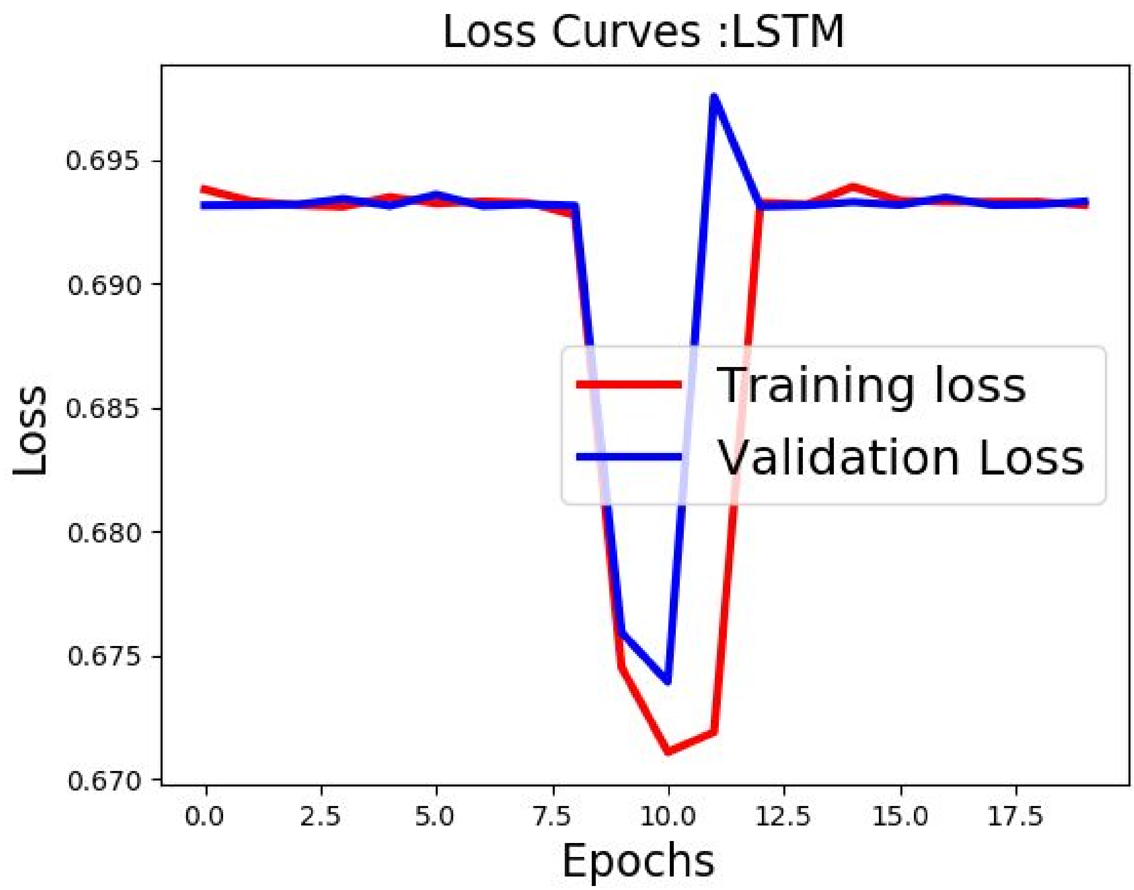
CNN LOSS and ACCURACY


Accuracy Curves : CNN
Accuracy
1.0
0.9
0.8
0.7
0.6
0.5
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epochs
Training Accuracy
Validation Accuracy



LSTM LOSS AND ACCURACY


Accuracy Curves : LSTM
Accuracy
Epochs
Training Accuracy
Validation Accuracy
0.60
0.58
0.56
0.54
0.52
0.50
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5



Observations:

- CNN has achieved good validation accuracy with high consistency , LSTM gave average performance but not consistent as CNN.
- LSTM model architecture is not for deployment in production .
- CNN outperformed LSTM in training time.

Objective:

To apply CNN Model on Image Dataset.

Approaches:

For CNN model training and testing. I had used the small CFIR-10 dataset. CNN Model take input image for classification and follow it definitions.

We used the keras and Tensorflow as backend . Loaded the inbuilt CIFR-10 and created the CNN model with multiple Conv2D layer, Maxpooling and dropout layer.

Compile the model with loss categorical_crossentropy and optimizer rmsprop. Fitted the model on tain and test data for training. Calculated score and accuracy plotted same on graph.

Workflow:

First of all we have loaded the cifr-10 .The data, split between train and test sets,converts the vectors into binary class matrices.

image_classification_cnn.py

```python
from __future__ import print_function
import keras
from keras.datasets import cifar10

from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D
import matplotlib.pyplot as plt

batch_size = 32
num_classes = 10
epochs = 15
num_predictions = 20


# The data, split between train and test sets:
(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')

# 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)

```

Created and Compiled the model.

image_classification_cnn.py

```python
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(512))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))

# initiate RMSprop optimizer
opt = keras.optimizers.rmsprop(lr=0.0001, decay=1e-6)

# Let's train the model using RMSprop
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

# Model Summary
model.summary()
```

Fitted the model on train and test. Calculated the score, accuracy and plot the graph for loss and accuracy.

```
# Model Fitting
history = model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_data=(x_test, y_test),shuffle=True)


# Score trained model.
scores = model.evaluate(x_test, y_test, verbose=1)
print('Test loss:', scores[0])
print('Test accuracy:', scores[1])

# Plot The CFIR-10 CNN LOSS
fig1 = plt.figure()
plt.plot(history.history['loss'], 'r', linewidth=3.0)
plt.plot(history.history['val_loss'], 'b', linewidth=3.0)
plt.legend(['Training loss', 'Validation Loss'], fontsize=18)
plt.xlabel('Epochs ', fontsize=16)
plt.ylabel('Loss', fontsize=16)
plt.title('Loss Curves : CIFR-10 CNN', fontsize=16)
fig1.savefig('cifr_loss_lstm.png')


# Plot the CIFR-10 CNN Accuracy

fig2=plt.figure()
plt.plot(history.history['acc'], 'r', linewidth=3.0)
plt.plot(history.history['val_acc'], 'b', linewidth=3.0)
plt.legend(['Training Accuracy', 'Validation Accuracy'], fontsize=18)
plt.xlabel('Epochs ', fontsize=16)
plt.ylabel('Accuracy', fontsize=16)
plt.title('Accuracy Curves : CIFR-10 CNN', fontsize=16)
fig2.savefig('cifr_accuracy_lstm.png')
plt.show()
```

This Concludes the workflow.

Dataset:

Used inbuilt dataset from

https://www.cs.toronto.edu/~kriz/cifar.html

Parameters:

None

Evaluation:

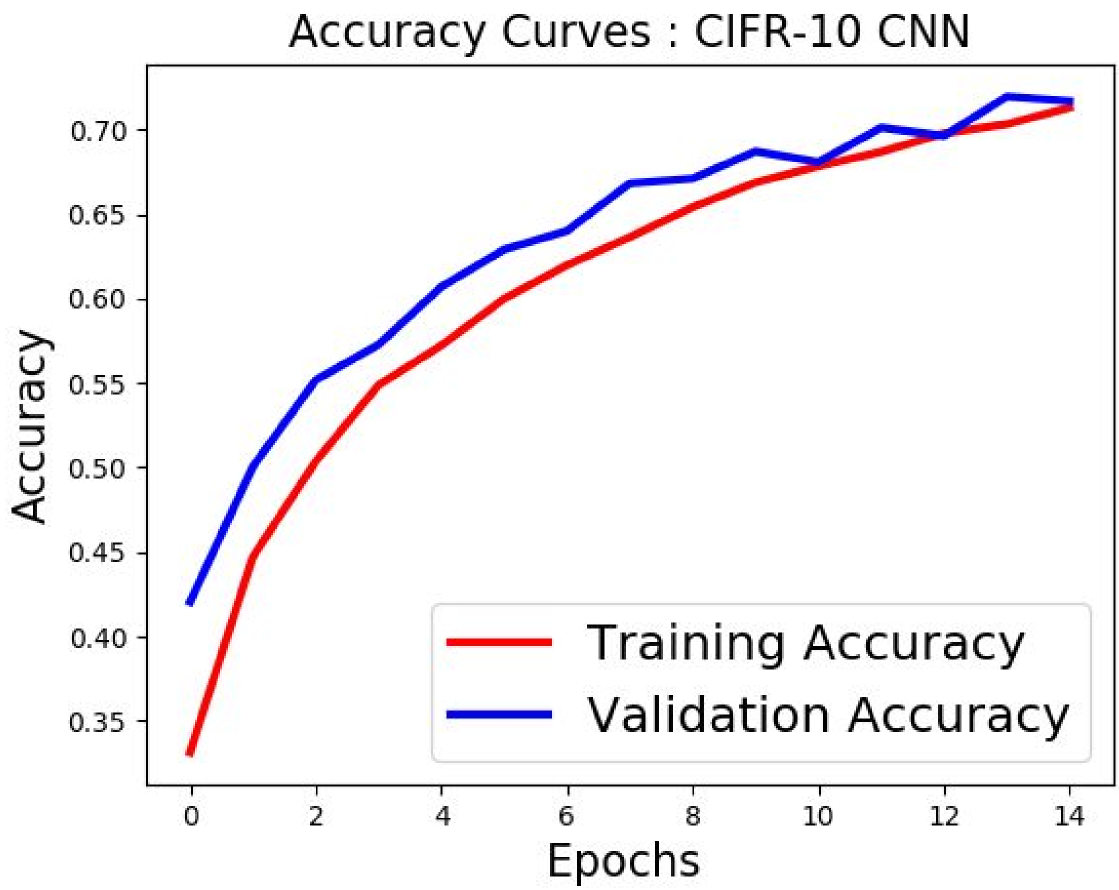
With 15 Epoch and batch size 32 we will give accuracy of 71 percent. We can improve the accuracy with more epochs.

```
image_classification_cnn
 8768/10000 [=========================>....] - ETA: 0s
 8864/10000 [=========================>....] - ETA: 0s
 8960/10000 [=========================>....] - ETA: 0s
 9024/10000 [==========================>...] - ETA: 0s
 9088/10000 [==========================>...] - ETA: 0s
 9152/10000 [==========================>...] - ETA: 0s
 9248/10000 [==========================>...] - ETA: 0s
 9344/10000 [===========================>..] - ETA: 0s
 9440/10000 [===========================>..] - ETA: 0s
 9536/10000 [===========================>..] - ETA: 0s
 9632/10000 [===========================>..] - ETA: 0s
 9728/10000 [============================>.] - ETA: 0s
 9824/10000 [============================>.] - ETA: 0s
 9920/10000 [============================>.] - ETA: 0s
10000/10000 [==============================] - 8s 781us/step
Test loss: 0.8282874810218811
Test accuracy: 0.717
```

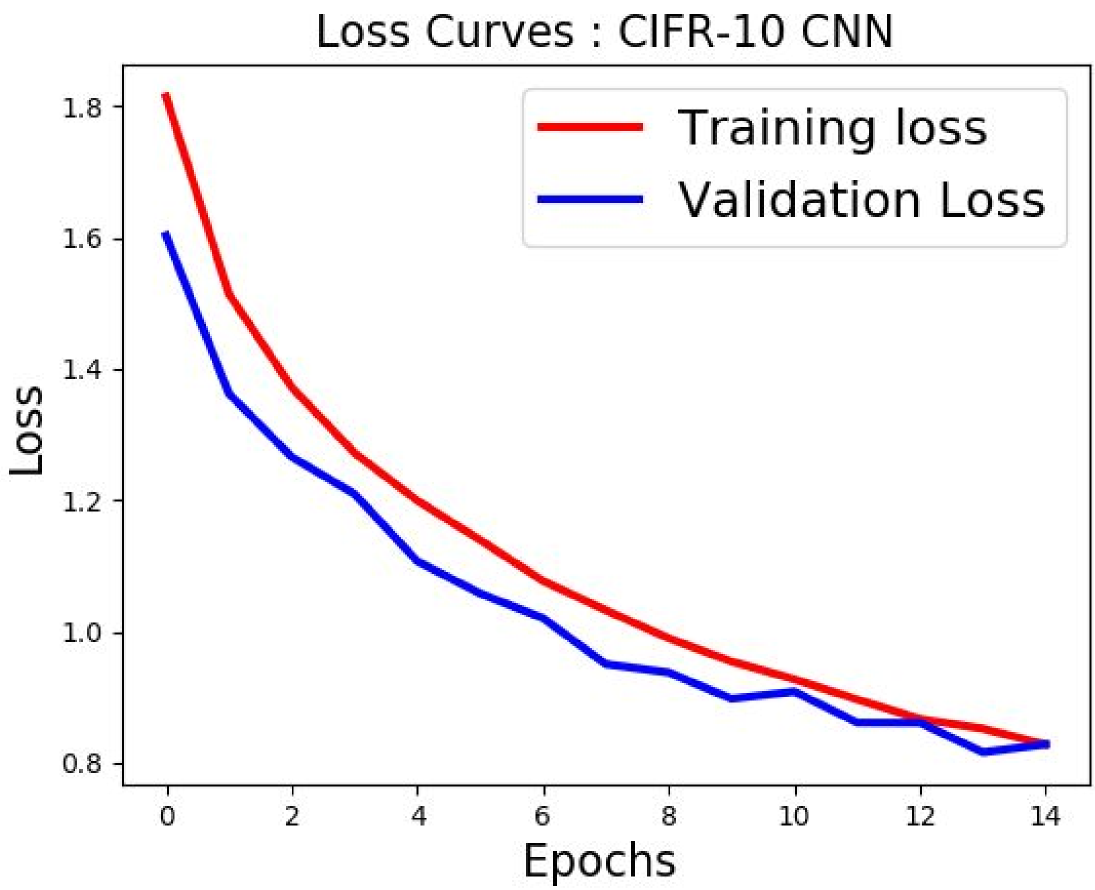
Conclusion:

We can see from below loss and Accuracy graph CNN performs very well on image datasets.

Accuracy:


Accuracy Curves : CIFR-10 CNN
Accuracy
Epochs
Training Accuracy
Validation Accuracy
0.35
0.40
0.45
0.50
0.55
0.60
0.65
0.70
0
2
4
6
8
10
12
14

Loss



References:

https://medium.com/@ksusorokina/image-classification-with-convolutional-neural-networks-496815db12a8

https://towardsdatascience.com/recurrent-neural-networks-and-lstm-4b601dd822a5

https://medium.com/@sabber/classifying-yelp-review-comments-using-lstm-and-word-embeddings-part-1-eb2275e4066b