ADITYA

COLLEGE OF ENGINEERING

Aditya Nagar, ADB Road, Surampalem - 533437

Department of

Name : **Roll No. :** | | | | | | | | | | |

Certified that this is the bonafide record of practical work done by

Mr. /Ms. .....

a student of .....with PIN No. .....

in the ..... Laboratory during the year .....

No. of Practicals Conducted : ☐ No. of Practicals Attended : ☐

Signature - Faculty Incharge **Signature - Head of the Department**

Submitted for the Practical examinationheld on .....

EXAMINER - 1 **EXAMINER - 2**

VISION & MISSION OF THE INSTITUTE

VISION

To induce higher planes of learning by imparting technical education with

- International standards
- Applied research
- Creative Ability
- Value based instruction and to emerge as a premiere institute.

MISSION

Achieving academic excellence by providing globally acceptable technical education by forecasting technology through

- Innovative Research and development
- Industry Institute Interaction
- Empowered Manpower

VISION & MISSION OF THE DEPARTMENT

VISION

To be a recognized Computer Science and Engineering hub striving to meet the growing needs of the Industry and Society.

MISSION

M1: Imparting Quality Education through state-of-the-art infrastructure with industry Collaboration

M2: Enhance Teaching Learning Process to disseminate knowledge.

M3: Organize Skill based, Industrial and Societal Events for overall Development.

Pointer

S No	Date	Name of the Experiment	Page No	Remark

Pointer

S No	Date	Name of the Experiment	Page No	Remark

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY: KAKINADA
KAKINADA – 533 003, Andhra Pradesh, India

DEPARTMENT OF CSE - ARTIFICIAL INTELLIGENCE & MACHINE LEARNING

III B Tech II Sem		L	T	P	C
		0	0	3	1.5
DEEP LEARNING WITH TENSORFLOW					

Course Outcomes:
On completion of this course, the student will be able to

- Implement deep neural networks to solve real world problems
- Choose appropriate pre-trained model to solve real time problem
- Interpret the results of two different deep learning models

Software Packages required:

- Keras
- Tensorflow
- PyTorch

List of Experiments:

1. Implement multilayer perceptron algorithm for MNIST Hand written Digit Classification.
2. Design a neural network for classifying movie reviews (Binary Classification) using IMDB dataset.
3. Design a neural Network for classifying news wires (Multi class classification) using Reuters dataset.
4. Design a neural network for predicting house prices using Boston Housing Price dataset.
5. Build a Convolution Neural Network for MNIST Hand written Digit Classification.
6. Build a Convolution Neural Network for simple image (dogs and Cats) Classification
7. Use a pre-trained convolution neural network (VGG16) for image classification.
8. Implement one hot encoding of words or characters.
9. Implement word embeddings for IMDB dataset.
10. Implement a Recurrent Neural Network for IMDB movie review classification problem.

Text Books:

1. Reza Zadeh and BharathRamsundar, “Tensorflow for Deep Learning”, O’Reilly publishers, 2018

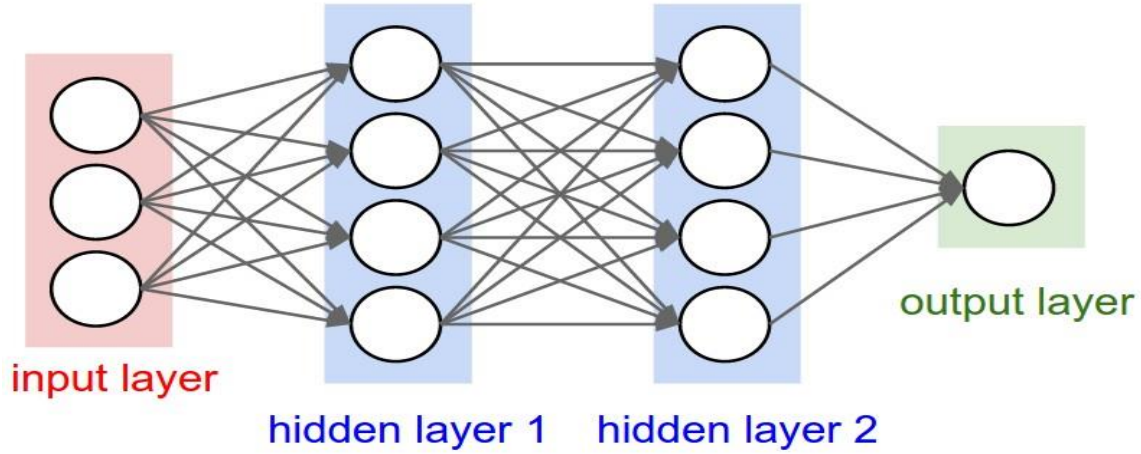
References:

1. https://github.com/fchollet/deep-learning-with-python-notebooks

Introduction: Artificial intelligence, machine learning, and deep learning

- **Artificial intelligence** is a broad term that refers to the ability of machines to perform tasks that are typically associated with human intelligence, such as learning, reasoning, and problem-solving.
- **Machine learning** is a subset of AI that involves the development of algorithms that can learn from data without being explicitly programmed. Machine learning algorithms are trained on large datasets, and they can then be used to make predictions or decisions about new data.
- **Deep learning** is a subset of machine learning that uses artificial neural networks to learn from data. Neural networks are inspired by the human brain, and they can be used to solve complex problems that would be difficult or impossible to solve with traditional machine learning algorithms.

- **Artificial neural networks:** are built on the principles of the structure and operation of human neurons. It is also known as neural networks or neural nets. An artificial neural network's input layer, which is the first layer, receives input from external sources and passes it on to the hidden layer, which is the second layer. Each neuron in the hidden layer gets information from the neurons in the previous layer, computes the weighted total, and then transfers it to the neurons in the next layer.



- **Keras**, Keras is a high-level, deep learning API developed by Google for implementing neural networks. It is written in Python and is used to make the implementation of neural networks easy. It also supports multiple backend neural network computation. Keras is relatively easy to learn and work with because it provides a python frontend with a high level of abstraction while having the option of multiple back-ends for computation purposes. This makes Keras slower than other deep learning frameworks, but extremely beginner-friendly.

Keras allows you to switch between different back ends. The frameworks supported by Keras are:

- Tensorflow
- Theano

- PlaidML
- MXNet
- CNTK (Microsoft Cognitive Toolkit)

- **TensorFlow**: Is an open-source library developed by Google primarily for deep learning applications. It also supports traditional machine learning. Tensor Flow was originally developed for large numerical computations without keeping deep learning in mind.

Process of running the project in Deep Learning:

1. Import the libraries and load the dataset

First, we are going to import all the modules that we are going to need for training our model. The Keras library already contains some datasets and MNIST is one of them. So we can easily import the dataset and start working with it. The **mnist.load_data()** method returns us the training data, its labels and also the testing data and its labels.

2. Preprocess the data

The image data cannot be fed directly into the model so we need to **perform some operations and process the data** to make it ready for our neural network. The dimension of the training data is (60000,28,28). The CNN model will require one more dimension so we reshape the matrix to shape (60000,28,28,1).

3. Create the model

Now we will **create our CNN model** in Python data science project. A CNN model generally consists of convolutional and pooling layers. It works better for data that are represented as grid structures, this is the reason why CNN works well for image classification problems. The dropout layer is used to deactivate some of the neurons and while training, it reduces offer fitting of the model. We will then compile the model with the Adadelta optimizer.

4. Train the model

The **model.fit() function** of Keras will start the training of the model. It **takes the training data, validation data, epochs, and batch size.**

It takes some time to train the model. After training, we save the weights and model definition in the 'mnist.h5' file.

5. Evaluate the model

We have 10,000 images in our dataset which will be used to **evaluate how good our model works**. The testing data was not involved in the training of the data therefore, it is new data for our model. The MNIST dataset is well balanced so we can get around 99% accuracy.

6. Create GUI to predict digits

Now for the GUI, we have created a new file in which we **build an interactive window to draw digits on canvas** and with a button, we can recognize the digit. The Tkinter library comes in the Python standard library. We have created a function **predict_digit()** that takes the image as input and then uses the trained model to predict the digit.

Then we **create the App class** which is responsible for building the GUI for our app. We create a canvas where we can draw by capturing the mouse event and with a button, we trigger the predict_digit() function and display the results.

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical

# Load and preprocess the MNIST dataset
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images = train_images.reshape((60000, 28, 28, 1)).astype('float32') / 255
test_images = test_images.reshape((10000, 28, 28, 1)).astype('float32') / 255

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

# Build the MLP model
model = models.Sequential()
model.add(layers.Flatten(input_shape=(28, 28, 1)))
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

# Compile the model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
model.fit(train_images, train_labels, epochs=5, batch_size=64, validation_split=0.2)

# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f'Test accuracy: {test_acc * 100:.2f}%')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
Epoch 1/5
750/750 [==============================] - 6s 7ms/step - loss: 0.3091 - accuracy: 0.9115 - val_loss: 0.1560 - val_accuracy: 0.9557
Epoch 2/5
750/750 [==============================] - 3s 5ms/step - loss: 0.1309 - accuracy: 0.9617 - val_loss: 0.1102 - val_accuracy: 0.9676
Epoch 3/5
750/750 [==============================] - 3s 4ms/step - loss: 0.0920 - accuracy: 0.9717 - val_loss: 0.1015 - val_accuracy: 0.9672
Epoch 4/5
750/750 [==============================] - 3s 4ms/step - loss: 0.0681 - accuracy: 0.9787 - val_loss: 0.0941 - val_accuracy: 0.9703
Epoch 5/5
750/750 [==============================] - 2s 3ms/step - loss: 0.0523 - accuracy: 0.9836 - val_loss: 0.0981 - val_accuracy: 0.9719
313/313 [==============================] - 1s 2ms/step - loss: 0.0829 - accuracy: 0.9741
Test accuracy: 97.41%
```

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing import sequence

# Load the IMDB dataset
max_features = 10000  # consider only the top 10,000 words in the dataset
maxlen = 200  # truncate reviews to a maximum length of 200 words
batch_size = 32

(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)
x_train = sequence.pad_sequences(x_train, maxlen=maxlen)
x_test = sequence.pad_sequences(x_test, maxlen=maxlen)

# Define the neural network model
model = models.Sequential()

# Add an Embedding layer to convert integer-encoded words to dense vectors of fixed size
model.add(layers.Embedding(max_features, 128, input_length=maxlen))

# Add an LSTM layer for sequence processing
model.add(layers.LSTM(64, dropout=0.2, recurrent_dropout=0.2))

# Add a Dense layer for binary classification
model.add(layers.Dense(1, activation='sigmoid'))

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

# Print the model summary
model.summary()

# Train the model
model.fit(x_train, y_train, epochs=5, batch_size=batch_size, validation_split=0.2)

# Evaluate the model on the test set
accuracy = model.evaluate(x_test, y_test)[1]
print(f'Test Accuracy: {accuracy * 100:.2f}%')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 [==============================] - 0s 0us/step
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 200, 128)          1280000

 lstm (LSTM)                 (None, 64)                49408

 dense (Dense)               (None, 1)                 65

=================================================================
Total params: 1329473 (5.07 MB)
Trainable params: 1329473 (5.07 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/5
625/625 [==============================] - 198s 311ms/step - loss: 0.4599 - accuracy: 0.7779 - val_loss: 0.3589 - val_accuracy: 0.84
Epoch 2/5
625/625 [==============================] - 190s 304ms/step - loss: 0.2587 - accuracy: 0.8986 - val_loss: 0.3900 - val_accuracy: 0.85
Epoch 3/5
625/625 [==============================] - 189s 303ms/step - loss: 0.1915 - accuracy: 0.9250 - val_loss: 0.3289 - val_accuracy: 0.86
Epoch 4/5
112/625 [====>.....................] - ETA: 2:23 - loss: 0.1199 - accuracy: 0.9584
```

```
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.text import text_to_word_sequence
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras import models
from tensorflow.keras import layers
from tensorflow.keras import losses
from tensorflow.keras import metrics
from tensorflow.keras import optimizers
from tensorflow.keras.utils import plot_model

import numpy as np
import matplotlib.pyplot as plt

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split


from collections import Counter
from pathlib import Path
import os
import numpy as np
import re
import string
import nltk
nltk.download('punkt')
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
nltk.download('stopwords')
from nltk.stem.porter import PorterStemmer
from nltk.stem import WordNetLemmatizer
nltk.download('wordnet')
from nltk.corpus import wordnet
import unicodedata
import html
stop_words = stopwords.words('english')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
```

```
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words=10000)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 [==============================] - 0s 0us/step
```

```
train_data[0]
```

```
 224,
 92,
 25,
 104,
 4,
 226,
 65,
 16,
 38,
 1334,
 88,
 12,
 16,
 283,
 5,
 16,
 4472,
 113,
 103,
 32,
 15,
 16,
 5345,
 19,
 178,
 32]
```

```
train_labels[0]
```

```
1
```

```
max([max(sequence) for sequence in train_data])
```

```
9999
```

```
word_index = imdb.get_word_index()
reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])
decoded_review = ' '.join([reverse_word_index.get(i - 3, '?') for i in train_data[0]])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb_word_index.json
1641221/1641221 [==============================] - 0s 0us/step
```

```
decoded_review
```

```
'? this film was just brilliant casting location scenery story direction everyone's
really suited the part they played and you could just imagine being there robert ? i
s an amazing actor and now the same being director ? father came from the same scott
ish island as myself so i loved the fact there was a real connection with this film
the witty remarks throughout the film were great it was just brilliant so much that
i bought the film as soon as it was released for ? and would recommend it to everyon
e to watch and the fly fishing was amazing really cried at the end it was so sad and
you know what they say if you cry at a film it must have been good and this definite
```

```
def vectorize_sequences(sequences, dimension=10000):
    # Create an all-zero matrix of shape (len(sequences), dimension)
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.  # set specific indices of results[i] to 1s
    return results

# Our vectorized training data
x_train = vectorize_sequences(train_data)
# Our vectorized test data
x_test = vectorize_sequences(test_data)
```

```
x_train.shape
```

```
(25000, 10000)
```

```
x_test.shape
```

```
(25000, 10000)
```

```
x_train[0]
```

```
array([0., 1., 1., ..., 0., 0., 0.])
```

```
y_train = np.asarray(train_labels).astype('float32')
y_test = np.asarray(test_labels).astype('float32')
```

```
from tensorflow.keras import models
from tensorflow.keras import layers

model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))


model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 16)                160016

 dense_1 (Dense)             (None, 16)                272

 dense_2 (Dense)             (None, 1)                 17

=================================================================
Total params: 160305 (626.19 KB)
Trainable params: 160305 (626.19 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

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


from tensorflow.keras import losses
from tensorflow.keras import metrics

model.compile(optimizer=optimizers.RMSprop(lr=0.001),
              loss=losses.binary_crossentropy,
              metrics=[metrics.binary_accuracy])
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers
```

```
x_val = x_train[:10000]
partial_x_train = x_train[10000:]

y_val = y_train[:10000]
partial_y_train = y_train[10000:]


history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

```
Epoch 1/20
30/30 [==============================] - 4s 103ms/step - loss: 0.5477 - binary_accuracy: 0.7715 - val_loss: 0.4334 - val_binary_accu
Epoch 2/20
30/30 [==============================] - 2s 54ms/step - loss: 0.3468 - binary_accuracy: 0.8915 - val_loss: 0.3396 - val_binary_accur
Epoch 3/20
30/30 [==============================] - 1s 44ms/step - loss: 0.2585 - binary_accuracy: 0.9158 - val_loss: 0.2948 - val_binary_accur
Epoch 4/20
30/30 [==============================] - 2s 53ms/step - loss: 0.2087 - binary_accuracy: 0.9311 - val_loss: 0.2771 - val_binary_accur
Epoch 5/20
30/30 [==============================] - 1s 43ms/step - loss: 0.1724 - binary_accuracy: 0.9434 - val_loss: 0.2907 - val_binary_accur
Epoch 6/20
30/30 [==============================] - 2s 57ms/step - loss: 0.1456 - binary_accuracy: 0.9530 - val_loss: 0.3213 - val_binary_accur
Epoch 7/20
30/30 [==============================] - 2s 66ms/step - loss: 0.1257 - binary_accuracy: 0.9610 - val_loss: 0.2907 - val_binary_accur
Epoch 8/20
30/30 [==============================] - 2s 55ms/step - loss: 0.1103 - binary_accuracy: 0.9668 - val_loss: 0.3371 - val_binary_accur
Epoch 9/20
30/30 [==============================] - 2s 53ms/step - loss: 0.0952 - binary_accuracy: 0.9723 - val_loss: 0.3244 - val_binary_accur
Epoch 10/20
30/30 [==============================] - 2s 54ms/step - loss: 0.0818 - binary_accuracy: 0.9777 - val_loss: 0.3362 - val_binary_accur
Epoch 11/20
30/30 [==============================] - 2s 55ms/step - loss: 0.0717 - binary_accuracy: 0.9808 - val_loss: 0.3722 - val_binary_accur
Epoch 12/20
30/30 [==============================] - 2s 63ms/step - loss: 0.0607 - binary_accuracy: 0.9849 - val_loss: 0.3635 - val_binary_accur
Epoch 13/20
30/30 [==============================] - 2s 64ms/step - loss: 0.0530 - binary_accuracy: 0.9875 - val_loss: 0.3864 - val_binary_accur
Epoch 14/20
30/30 [==============================] - 2s 64ms/step - loss: 0.0424 - binary_accuracy: 0.9922 - val_loss: 0.4201 - val_binary_accur
Epoch 15/20
30/30 [==============================] - 1s 48ms/step - loss: 0.0388 - binary_accuracy: 0.9924 - val_loss: 0.4320 - val_binary_accur
Epoch 16/20
```

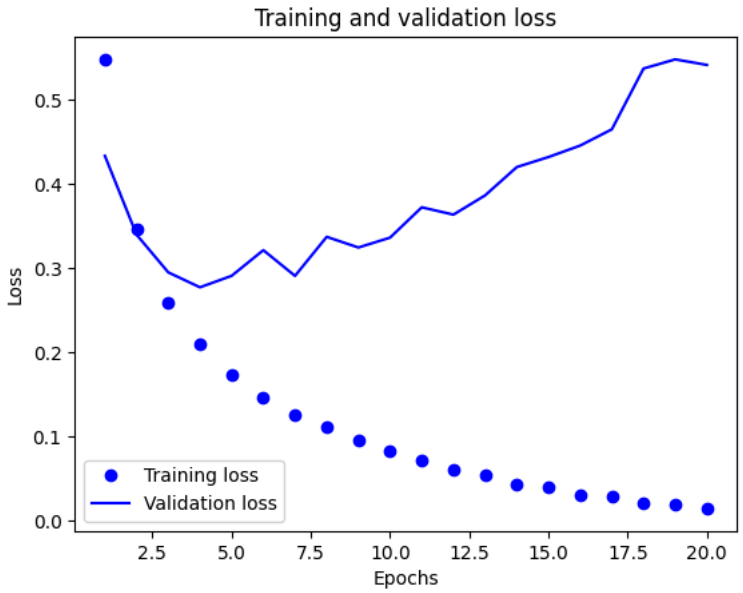
```
30/30 [==============================] - 2s 52ms/step - loss: 0.0303 - binary_accuracy: 0.9957 - val_loss: 0.4456 - val_binary_accur
Epoch 17/20
30/30 [==============================] - 1s 43ms/step - loss: 0.0279 - binary_accuracy: 0.9948 - val_loss: 0.4650 - val_binary_accur
Epoch 18/20
30/30 [==============================] - 2s 53ms/step - loss: 0.0203 - binary_accuracy: 0.9973 - val_loss: 0.5373 - val_binary_accur
Epoch 19/20
30/30 [==============================] - 2s 53ms/step - loss: 0.0193 - binary_accuracy: 0.9975 - val_loss: 0.5482 - val_binary_accur
Epoch 20/20
30/30 [==============================] - 2s 53ms/step - loss: 0.0141 - binary_accuracy: 0.9988 - val_loss: 0.5416 - val_binary_accur
```

```
history_dict = history.history
history_dict.keys()
```

```
dict_keys(['loss', 'binary_accuracy', 'val_loss', 'val_binary_accuracy'])
```

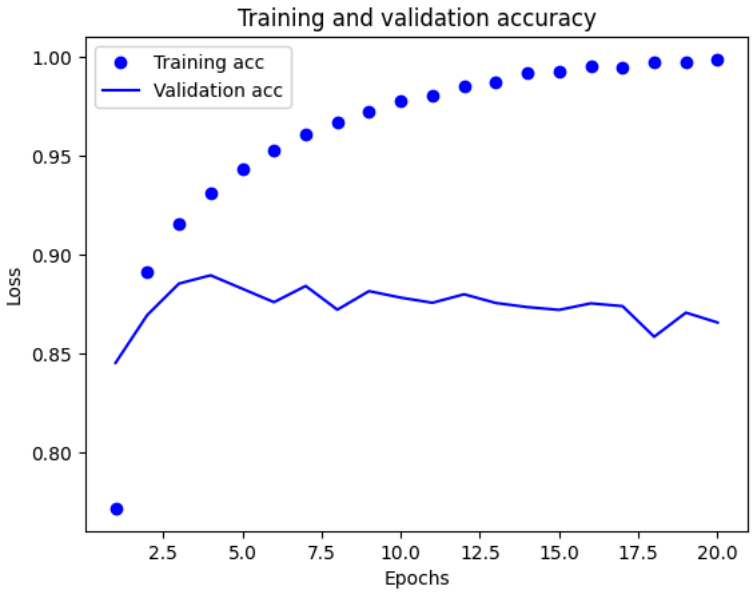
```
import matplotlib.pyplot as plt

acc = history.history['binary_accuracy']
val_acc = history.history['val_binary_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(acc) + 1)

# "bo" is for "blue dot"
plt.plot(epochs, loss, 'bo', label='Training loss')
# b is for "solid blue line"
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7d03c5eba2f0>
```




```
plt.show()
```

```
plt.clf()   # clear figure
acc_values = history_dict['binary_accuracy']
val_acc_values = history_dict['val_binary_accuracy']

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```



Start coding or generate with AI.

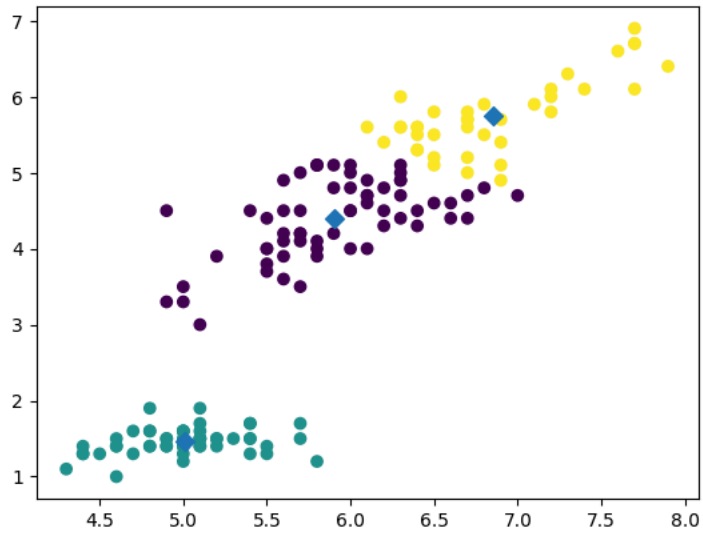
```
import numpy as np
import pandas as pd
import os
```

```
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
```

```
iris=load_iris()
samples=iris.data
model=KMeans(n_clusters=3)
model.fit(samples)
labels=model.predict(samples)
print(labels)
```

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 2 2 2 2 0 2 2 2 2
 2 2 0 0 2 2 2 2 2 0 2 0 2 0 2 2 0 0 2 2 2 2 2 0 2 2 2 2 0 2 2 2 2 0 2 2 2 2
 2 0]
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change fro
  warnings.warn(
```

```
xs=samples[:,0]
ys=samples[:,2]
plt.scatter(xs,ys,c=labels)
centroids=model.cluster_centers_
centroids_x=centroids[:,0]
centroids_y=centroids[:,2]
plt.scatter(centroids_x,centroids_y,marker='D',s=50)
plt.show()
```




```
from google.colab import files
uploaded=files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

Saving HousingData.csv to HousingData.csv

```
import io
```

```
df = pd.read_csv(io.BytesIO(uploaded['HousingData.csv']))
```

```
print(df)
```

```
        CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD  TAX  \
0    0.00632  18.0   2.31   0.0  0.538  6.575  65.2  4.0900    1  296
1    0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671    2  242
2    0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671    2  242
3    0.03237   0.0   2.18   0.0  0.458  6.998  45.8  6.0622    3  222
4    0.06905   0.0   2.18   0.0  0.458  7.147  54.2  6.0622    3  222
..       ...   ...    ...   ...    ...    ...   ...     ...  ...  ...
501  0.06263   0.0  11.93   0.0  0.573  6.593  69.1  2.4786    1  273
502  0.04527   0.0  11.93   0.0  0.573  6.120  76.7  2.2875    1  273
```

```
503  0.06076   0.0  11.93   0.0  0.573  6.976  91.0  2.1675    1  273
504  0.10959   0.0  11.93   0.0  0.573  6.794  89.3  2.3889    1  273
505  0.04741   0.0  11.93   0.0  0.573  6.030   NaN  2.5050    1  273

     PTRATIO       B  LSTAT  MEDV
0       15.3  396.90   4.98  24.0
1       17.8  396.90   9.14  21.6
2       17.8  392.83   4.03  34.7
3       18.7  394.63   2.94  33.4
4       18.7  396.90    NaN  36.2
..       ...     ...    ...   ...
501     21.0  391.99    NaN  22.4
502     21.0  396.90   9.08  20.6
503     21.0  396.90   5.64  23.9
504     21.0  393.45   6.48  22.0
505     21.0  396.90   7.88  11.9

[506 rows x 14 columns]
```

```
df_dict={
    0:'INDUS',
    1:'AGE',
    2:'TAX'
}
df_list=df['TAX'].map(df_dict).tolist()
```

```
df=df.drop(['TAX'],axis=1)
df.head()
```

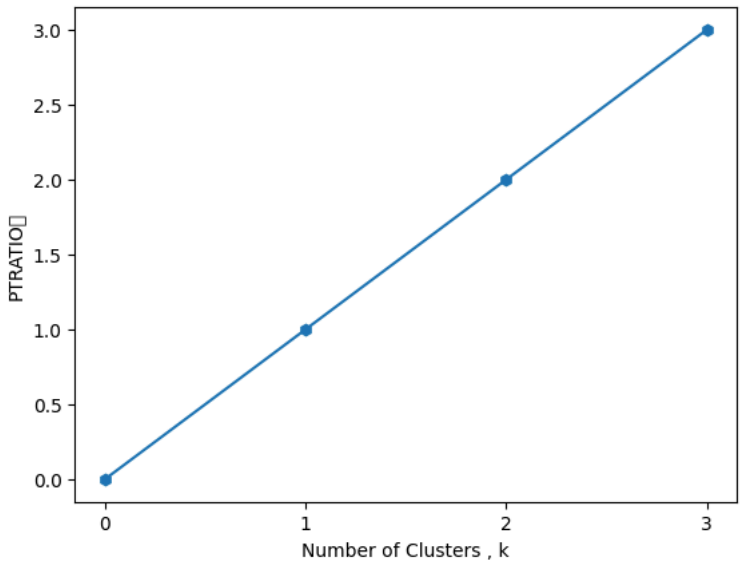
	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3	18.7	396.90	NaN	36.2

```
b=range(0,4)
PTRATIO =[]
```

```
for k in b:
    model=KMeans(n_clusters=k)
```

```
plt.plot(b,'-h')
plt.xlabel('Number of Clusters , k')
plt.ylabel('PTRATIO ')
plt.xticks(b)
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 9 (  ) missing from current font.
  fig.canvas.print_figure(bytes_io, **kw)
```



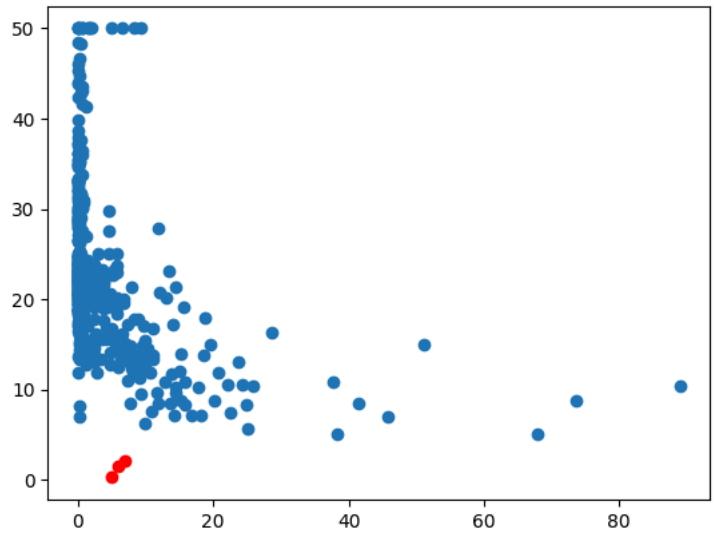
```
df.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3	18.7	396.90	NaN	36.2

```
model=KMeans(n_clusters=3)
centroids
```

```
array([[5.9016129 , 2.7483871 , 4.39354839, 1.43387097],
       [5.006     , 3.428     , 1.462     , 0.246     ],
       [6.85      , 3.07368421, 5.74210526, 2.07105263]])
```

```
xs_ZN=df.iloc[:,0]
ys_AGE=df.iloc[:,-1]
centroids_xs_ZN=centroids[:,0]
centroids_ys_AGE=centroids[:,-1]
plt.scatter(xs_ZN,ys_AGE)
plt.scatter(centroids_xs_ZN,centroids_ys_AGE,c='red')
plt.show()
```

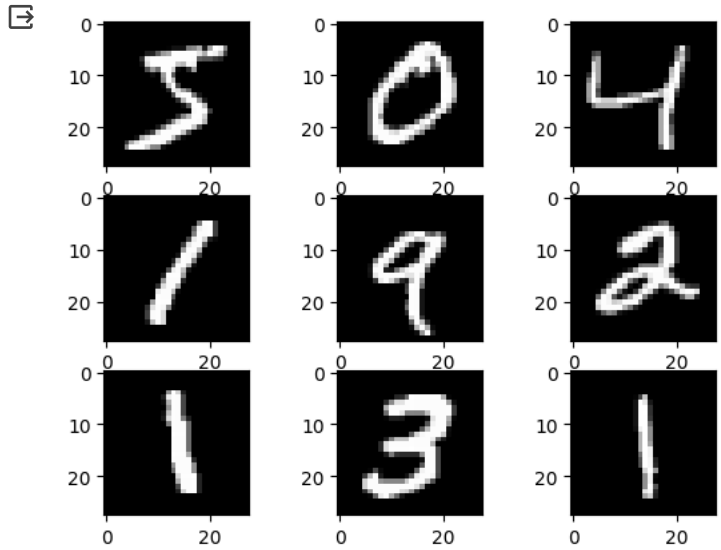


```
import numpy as np
import os
```

```
from tensorflow.keras.datasets import mnist
from matplotlib import pyplot as plt
(trainX, trainy), (testX, testy) = mnist.load_data()
print('Train: X=%s, y=%s' % (trainX.shape, trainy.shape))
print('Test: X=%s, y=%s' % (testX.shape, testy.shape))
```

```
Train: X=(60000, 28, 28), y=(60000,)
Test: X=(10000, 28, 28), y=(10000,)
```

```
for i in range(9):
  plt.subplot(330 + 1 + i)
  plt.imshow(trainX[i], cmap=plt.get_cmap('gray'))
data = ...
```



```
import matplotlib.pyplot as plt
import seaborn as sns
#import cv2
from PIL import Image
import tensorflow as tf
tf.random.set_seed(3)
from tensorflow import keras
from keras.datasets import mnist
from tensorflow.math import confusion_matrix
import random
```

```
(X_train,Y_train),(X_test,Y_test) = mnist.load_data()
```

```
type(X_train)
```

```
numpy.ndarray
```

```
print(X_train.shape,Y_train.shape,X_test.shape,Y_test.shape)
```

```
(60000, 28, 28) (60000,) (10000, 28, 28) (10000,)
```

```
print(X_train[9])
```

```
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0 189 190   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

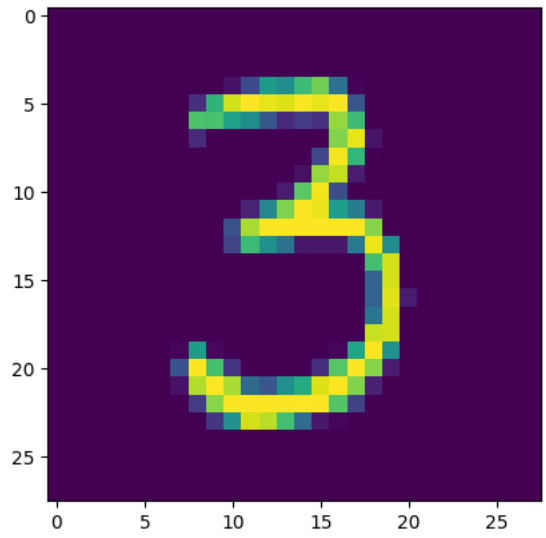
```
   0   0   0 143 247 153   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0 136 247 242  86   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0 192 252 187   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0  62 185  18   0   0   0
   0  89 236 217  47   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0 216 253  60   0   0   0
   0 212 255  81   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0 206 252  68   0   0   0
  48 242 253  89   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0 131 251 212  21   0   0  11
 167 252 197   5   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0  29 232 247  63   0   0   0 153
 252 226   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0  45 219 252 143   0   0   0 116 249
 252 103   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   4  96 253 255 253 200 122   7  25 201 250
 158   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0  92 252 252 253 217 252 252 200 227 252 231
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0  87 251 247 231  65  48 189 252 252 253 252 251
 227  35   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0 190 221  98   0   0   0  42 196 252 253 252 252
 162   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0 111  29   0   0   0   0  62 239 252  86  42  42
  14   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0  15 148 253 218   0   0   0
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0 121 252 231  28   0   0   0
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0  31 221 251 129   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0 218 252 160   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0 122 252  82   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0]]
```

```
print(X_train[9].shape)
```

```
(28, 28)
```

```
plt.imshow(X_train[50])
plt.show
print(Y_train[50])
```

```
3
```




```
print(Y_train.shape,Y_test.shape)
```

```
(60000,) (10000,)
```

```
# print unique value in Y_train
print(np.unique(Y_train))

# print unique value in Y_test
print(np.unique(Y_test))
```

```
[0 1 2 3 4 5 6 7 8 9]
[0 1 2 3 4 5 6 7 8 9]
```

```
X_train = X_train/255
X_test = X_test/255
```

```
print(X_train[9])
```

```
[[0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.74117647 0.74509804
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.56078431 0.96862745 0.6
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.53333333 0.96862745 0.94901961 0.3372549
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.75294118 0.98823529 0.73333333 0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.24313725 0.7254902  0.07058824 0.         0.         0.
  0.         0.34901961 0.9254902  0.85098039 0.18431373 0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.84705882 0.99215686 0.23529412 0.         0.         0.
  0.         0.83137255 1.         0.31764706 0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.80784314 0.98823529 0.26666667 0.         0.         0.
  0.18823529 0.94901961 0.99215686 0.34901961 0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.51372549
  0.98431373 0.83137255 0.08235294 0.         0.         0.04313725
```

```
model = keras.Sequential([
    keras.layers.Flatten(input_shape = (28,28)),
    keras.layers.Dense(50, activation = 'relu'),
    keras.layers.Dense(50, activation = 'relu'),
    keras.layers.Dense(10, activation = 'sigmoid')
])
```

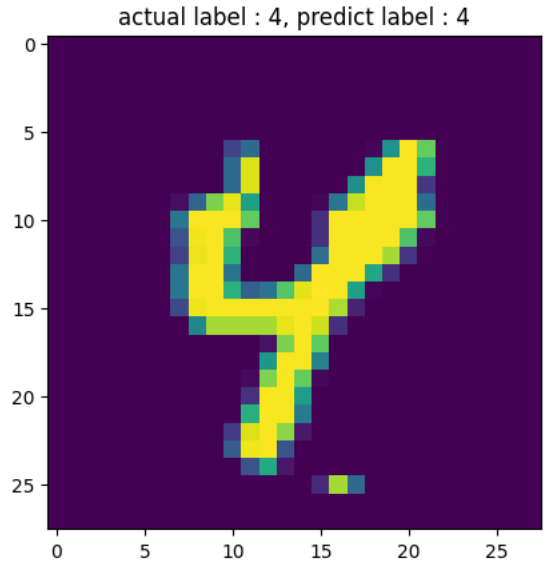
```
model.compile(optimizer = 'adam',
             loss = 'sparse_categorical_crossentropy',
             metrics = ['accuracy'])
```

```
model.fit(X_train, Y_train,epochs = 10)
```

```
Epoch 1/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2876 - accuracy: 0.9163
Epoch 2/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1363 - accuracy: 0.9594
Epoch 3/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1021 - accuracy: 0.9693
Epoch 4/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0837 - accuracy: 0.9743
Epoch 5/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0678 - accuracy: 0.9790
Epoch 6/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0594 - accuracy: 0.9812
Epoch 7/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0510 - accuracy: 0.9836
Epoch 8/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0470 - accuracy: 0.9848
Epoch 9/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0401 - accuracy: 0.9868
Epoch 10/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0339 - accuracy: 0.9889
<keras.src.callbacks.History at 0x7deb6d454340>
```

```
INDEX = random.randint(0,Y_test.shape[0])
actual = Y_test[INDEX]
pred = np.argmax(model.predict(np.array([X_test[INDEX]]))[0])
pred,actual
plt.imshow(X_test[INDEX])
plt.title(f'actual label : {actual}, predict label : {pred}')
plt.show()
```

```
1/1 [==============================] - 0s 96ms/step
```



```
!wget --no-check-certificate \
    https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip \
    -O /tmp/cats_and_dogs_filtered.zip
```

```
--2023-04-02 14:30:36--  https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip
Resolving storage.googleapis.com (storage.googleapis.com)... 142.250.141.128, 74.125.137.128, 142.251.2.128, ...
Connecting to storage.googleapis.com (storage.googleapis.com)|142.250.141.128|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 68606236 (65M) [application/zip]
Saving to: ‘/tmp/cats_and_dogs_filtered.zip’

/tmp/cats_and_dogs_ 100%[===================>]  65.43M   188MB/s    in 0.3s

2023-04-02 14:30:36 (188 MB/s) - ‘/tmp/cats_and_dogs_filtered.zip’ saved [68606236/68606236]
```

```
import os
import zipfile

local_zip = '/tmp/cats_and_dogs_filtered.zip'
zip_ref = zipfile.ZipFile(local_zip, 'r')
zip_ref.extractall('/tmp')
zip_ref.close()


base_dir = '/tmp/cats_and_dogs_filtered'
train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')

# Directory with our training cat pictures
train_cats_dir = os.path.join(train_dir, 'cats')

# Directory with our training dog pictures
train_dogs_dir = os.path.join(train_dir, 'dogs')

# Directory with our validation cat pictures
validation_cats_dir = os.path.join(validation_dir, 'cats')

# Directory with our validation dog pictures
validation_dogs_dir = os.path.join(validation_dir, 'dogs')


train_cat_fnames = os.listdir(train_cats_dir)
print(train_cat_fnames[:10])

train_dog_fnames = os.listdir(train_dogs_dir)
train_dog_fnames.sort()
print(train_dog_fnames[:10])
```

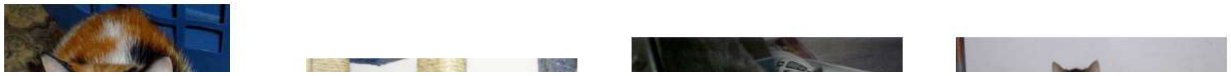
```
['cat.111.jpg', 'cat.723.jpg', 'cat.721.jpg', 'cat.58.jpg', 'cat.325.jpg', 'cat.525.jpg', 'cat.565.jpg', 'cat.212.jpg', 'cat.567.jpg',
['dog.0.jpg', 'dog.1.jpg', 'dog.10.jpg', 'dog.100.jpg', 'dog.101.jpg', 'dog.102.jpg', 'dog.103.jpg', 'dog.104.jpg', 'dog.105.jpg', 'dog.
```

```
print('total training cat images:', len(os.listdir(train_cats_dir)))
print('total training dog images:', len(os.listdir(train_dogs_dir)))
print('total validation cat images:', len(os.listdir(validation_cats_dir)))
print('total validation dog images:', len(os.listdir(validation_dogs_dir)))
```

```
total training cat images: 1000
total training dog images: 1000
total validation cat images: 500
total validation dog images: 500
```

```
%matplotlib inline

import matplotlib.pyplot as plt
import matplotlib.image as mpimg

# Parameters for our graph; we'll output images in a 4x4 configuration
nrows = 4
ncols = 4

# Index for iterating over images
pic_index = 0
```

```
# Set up matplotlib fig, and size it to fit 4x4 pics
fig = plt.gcf()
fig.set_size_inches(ncols * 4, nrows * 4)

pic_index += 8
next_cat_pix = [os.path.join(train_cats_dir, fname)
                for fname in train_cat_fnames[pic_index-8:pic_index]]
next_dog_pix = [os.path.join(train_dogs_dir, fname)
                for fname in train_dog_fnames[pic_index-8:pic_index]]

for i, img_path in enumerate(next_cat_pix+next_dog_pix):
  # Set up subplot; subplot indices start at 1
  sp = plt.subplot(nrows, ncols, i + 1)
  sp.axis('Off') # Don't show axes (or gridlines)

  img = mpimg.imread(img_path)
  plt.imshow(img)

plt.show()
```



```
from tensorflow.keras import layers
from tensorflow.keras import Model
```

```
# Our input feature map is 150x150x3: 150x150 for the image pixels, and 3 for
# the three color channels: R, G, and B
img_input = layers.Input(shape=(150, 150, 3))

# First convolution extracts 16 filters that are 3x3
# Convolution is followed by max-pooling layer with a 2x2 window
x = layers.Conv2D(16, 3, activation='relu')(img_input)
x = layers.MaxPooling2D(2)(x)

# Second convolution extracts 32 filters that are 3x3
# Convolution is followed by max-pooling layer with a 2x2 window
x = layers.Conv2D(32, 3, activation='relu')(x)
x = layers.MaxPooling2D(2)(x)

# Third convolution extracts 64 filters that are 3x3
# Convolution is followed by max-pooling layer with a 2x2 window
x = layers.Conv2D(64, 3, activation='relu')(x)
x = layers.MaxPooling2D(2)(x)
```

```
# Flatten feature map to a 1-dim tensor so we can add fully connected layers
x = layers.Flatten()(x)

# Create a fully connected layer with ReLU activation and 512 hidden units
x = layers.Dense(512, activation='relu')(x)

# Create output layer with a single node and sigmoid activation
output = layers.Dense(1, activation='sigmoid')(x)

# Create model:
# input = input feature map
# output = input feature map + stacked convolution/maxpooling layers + fully
# connected layer + sigmoid output layer
model = Model(img_input, output)
```

```
from tensorflow.keras.optimizers import RMSprop

model.compile(loss='binary_crossentropy',
              optimizer=RMSprop(lr=0.001),
              metrics=['acc'])
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.leg
```

```
from keras.preprocessing.image import ImageDataGenerator
# All images will be rescaled by 1./255
train_datagen = ImageDataGenerator(rescale=1./255)
val_datagen = ImageDataGenerator(rescale=1./255)

# Flow training images in batches of 20 using train_datagen generator
train_generator = train_datagen.flow_from_directory(
        train_dir,  # This is the source directory for training images
        target_size=(150, 150),  # All images will be resized to 150x150
        batch_size=20,
        # Since we use binary_crossentropy loss, we need binary labels
        class_mode='binary')

# Flow validation images in batches of 20 using val_datagen generator
validation_generator = val_datagen.flow_from_directory(
        validation_dir,
        target_size=(150, 150),
        batch_size=20,
        class_mode='binary')
```

```
Found 2000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
```

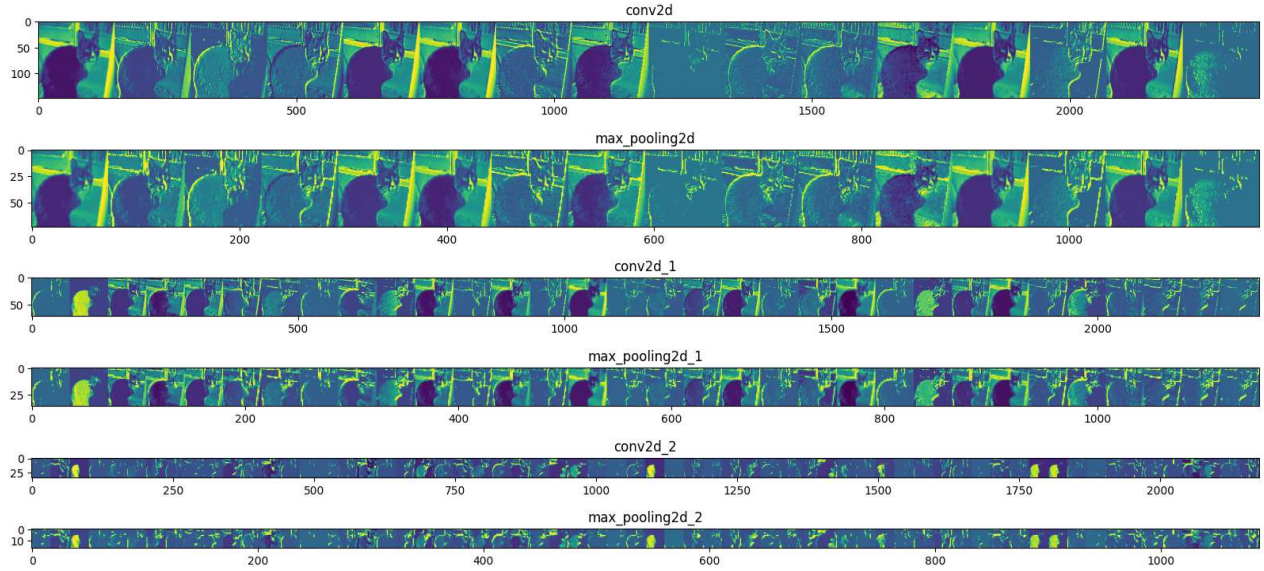
```
history = model.fit_generator(
      train_generator,
      steps_per_epoch=100,  # 2000 images = batch_size * steps
      epochs=15,
      validation_data=validation_generator,
      validation_steps=50,  # 1000 images = batch_size * steps
      verbose=2)
```

```
Epoch 1/15
<ipython-input-23-91d346d1b720>:1: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use
  history = model.fit_generator(
100/100 - 68s - loss: 0.5799 - acc: 0.7080 - val_loss: 0.5662 - val_acc: 0.7100 - 68s/epoch - 679ms/step
Epoch 2/15
100/100 - 68s - loss: 0.5108 - acc: 0.7465 - val_loss: 0.6089 - val_acc: 0.6730 - 68s/epoch - 682ms/step
Epoch 3/15
100/100 - 71s - loss: 0.4536 - acc: 0.7795 - val_loss: 0.5700 - val_acc: 0.7220 - 71s/epoch - 708ms/step
Epoch 4/15
100/100 - 69s - loss: 0.3941 - acc: 0.8125 - val_loss: 0.6064 - val_acc: 0.7420 - 69s/epoch - 691ms/step
Epoch 5/15
100/100 - 69s - loss: 0.3115 - acc: 0.8590 - val_loss: 0.7212 - val_acc: 0.7030 - 69s/epoch - 691ms/step
Epoch 6/15
100/100 - 69s - loss: 0.2525 - acc: 0.8895 - val_loss: 0.7653 - val_acc: 0.7210 - 69s/epoch - 693ms/step
Epoch 7/15
100/100 - 70s - loss: 0.1781 - acc: 0.9245 - val_loss: 0.7806 - val_acc: 0.7230 - 70s/epoch - 704ms/step
Epoch 8/15
100/100 - 67s - loss: 0.1186 - acc: 0.9560 - val_loss: 0.8062 - val_acc: 0.7390 - 67s/epoch - 671ms/step
Epoch 9/15
100/100 - 69s - loss: 0.0704 - acc: 0.9750 - val_loss: 1.3526 - val_acc: 0.6960 - 69s/epoch - 694ms/step
Epoch 10/15
100/100 - 69s - loss: 0.0425 - acc: 0.9865 - val_loss: 1.2689 - val_acc: 0.7310 - 69s/epoch - 686ms/step
Epoch 11/15
100/100 - 70s - loss: 0.1008 - acc: 0.9745 - val_loss: 1.0499 - val_acc: 0.7290 - 70s/epoch - 695ms/step
Epoch 12/15
100/100 - 69s - loss: 0.0202 - acc: 0.9955 - val_loss: 1.4985 - val_acc: 0.7260 - 69s/epoch - 688ms/step
Epoch 13/15
100/100 - 70s - loss: 0.0376 - acc: 0.9895 - val_loss: 1.6226 - val_acc: 0.7320 - 70s/epoch - 696ms/step
Epoch 14/15
100/100 - 69s - loss: 0.0292 - acc: 0.9920 - val_loss: 1.6459 - val_acc: 0.7350 - 69s/epoch - 690ms/step
Epoch 15/15
100/100 - 69s - loss: 0.0193 - acc: 0.9930 - val_loss: 2.2054 - val_acc: 0.6940 - 69s/epoch - 694ms/step
```

```
import numpy as np
import random
from tensorflow.keras.preprocessing.image import img_to_array, load_img

# Let's define a new Model that will take an image as input, and will output
# intermediate representations for all layers in the previous model after
# the first.
successive_outputs = [layer.output for layer in model.layers[1:]]
visualization_model = Model(img_input, successive_outputs)

# Let's prepare a random input image of a cat or dog from the training set.
cat_img_files = [os.path.join(train_cats_dir, f) for f in train_cat_fnames]
dog_img_files = [os.path.join(train_dogs_dir, f) for f in train_dog_fnames]
img_path = random.choice(cat_img_files + dog_img_files)

img = load_img(img_path, target_size=(150, 150))  # this is a PIL image
x = img_to_array(img)  # Numpy array with shape (150, 150, 3)
x = x.reshape((1,) + x.shape)  # Numpy array with shape (1, 150, 150, 3)

# Rescale by 1/255
x /= 255

# Let's run our image through our network, thus obtaining all
# intermediate representations for this image.
successive_feature_maps = visualization_model.predict(x)

# These are the names of the layers, so can have them as part of our plot
layer_names = [layer.name for layer in model.layers[1:]]

# Now let's display our representations
for layer_name, feature_map in zip(layer_names, successive_feature_maps):
  if len(feature_map.shape) == 4:
    # Just do this for the conv / maxpool layers, not the fully-connected layers
    n_features = feature_map.shape[-1]  # number of features in feature map
    # The feature map has shape (1, size, size, n_features)
    size = feature_map.shape[1]
    # We will tile our images in this matrix
```

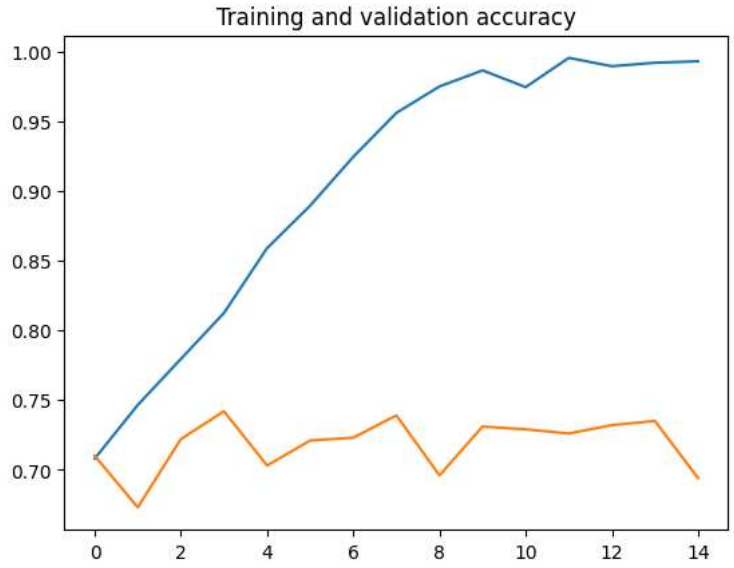
```
    display_grid = np.zeros((size, size * n_features))
    for i in range(n_features):
      # Postprocess the feature to make it visually palatable
      x = feature_map[0, :, :, i]
      x -= x.mean()
      x /= x.std()
      x *= 64
      x += 128
      x = np.clip(x, 0, 255).astype('uint8')
      # We'll tile each filter into this big horizontal grid
      display_grid[:, i * size : (i + 1) * size] = x
    # Display the grid
    scale = 20. / n_features
    plt.figure(figsize=(scale * n_features, scale))
    plt.title(layer_name)
    plt.grid(False)
    plt.imshow(display_grid, aspect='auto', cmap='viridis')
```

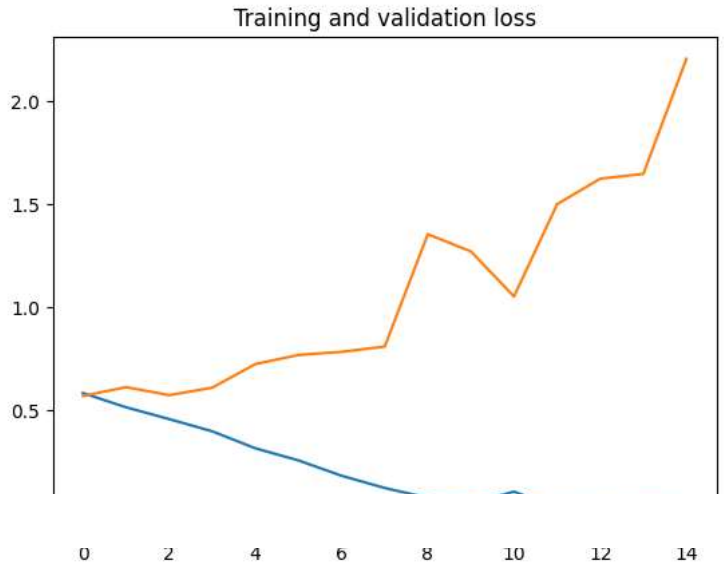
```
 1/1 [==============================] - 0s 198ms/step
```




```
# Retrieve a list of accuracy results on training and validation data
# sets for each training epoch
acc = history.history['acc']
val_acc = history.history['val_acc']

# Retrieve a list of list results on training and validation data
# sets for each training epoch
loss = history.history['loss']
val_loss = history.history['val_loss']

# Get number of epochs
epochs = range(len(acc))

# Plot training and validation accuracy per epoch
plt.plot(epochs, acc)
plt.plot(epochs, val_acc)
plt.title('Training and validation accuracy')

plt.figure()

# Plot training and validation loss per epoch
plt.plot(epochs, loss)
plt.plot(epochs, val_loss)
plt.title('Training and validation loss')
```

```
Text(0.5, 1.0, 'Training and validation loss')
```



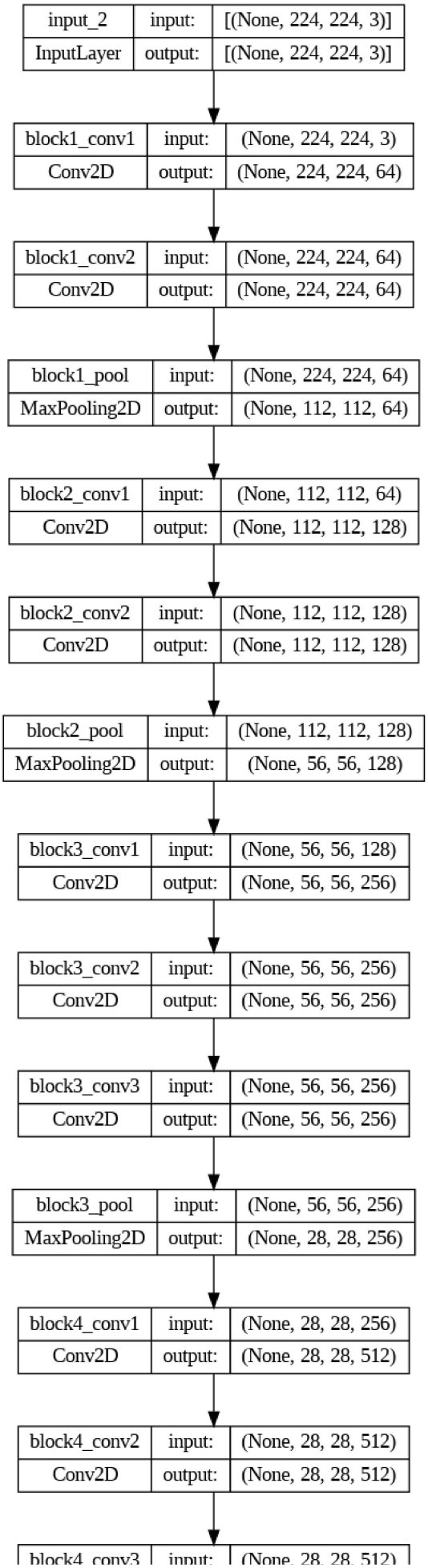


✓ 0s completed at 8:59 PM

```
from tensorflow.keras.applications.vgg16 import VGG16
model = VGG16()


print(model.summary())
```

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 flatten (Flatten)           (None, 25088)             0

 fc1 (Dense)                 (None, 4096)              102764544

 fc2 (Dense)                 (None, 4096)              16781312

 predictions (Dense)         (None, 1000)              4097000

=================================================================
Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0
_________________________________________________________________
None
```

```
from tensorflow.keras.utils import plot_model
plot_model(model,show_layer_names=True,show_shapes=True)
```


input_2 | input: | [(None, 224, 224, 3)]
InputLayer | output: | [(None, 224, 224, 3)]
block1_conv1 | input: | (None, 224, 224, 3)
Conv2D | output: | (None, 224, 224, 64)
block1_conv2 | input: | (None, 224, 224, 64)
Conv2D | output: | (None, 224, 224, 64)
block1_pool | input: | (None, 224, 224, 64)
MaxPooling2D | output: | (None, 112, 112, 64)
block2_conv1 | input: | (None, 112, 112, 64)
Conv2D | output: | (None, 112, 112, 128)
block2_conv2 | input: | (None, 112, 112, 128)
Conv2D | output: | (None, 112, 112, 128)
block2_pool | input: | (None, 112, 112, 128)
MaxPooling2D | output: | (None, 56, 56, 128)
block3_conv1 | input: | (None, 56, 56, 128)
Conv2D | output: | (None, 56, 56, 256)
block3_conv2 | input: | (None, 56, 56, 256)
Conv2D | output: | (None, 56, 56, 256)
block3_conv3 | input: | (None, 56, 56, 256)
Conv2D | output: | (None, 56, 56, 256)
block3_pool | input: | (None, 56, 56, 256)
MaxPooling2D | output: | (None, 28, 28, 256)
block4_conv1 | input: | (None, 28, 28, 256)
Conv2D | output: | (None, 28, 28, 512)
block4_conv2 | input: | (None, 28, 28, 512)
Conv2D | output: | (None, 28, 28, 512)
block4_conv3 | input: | (None, 28, 28, 512)

```
from google.colab import files
uploaded = files.upload()
```

Choose Files dog.jpg

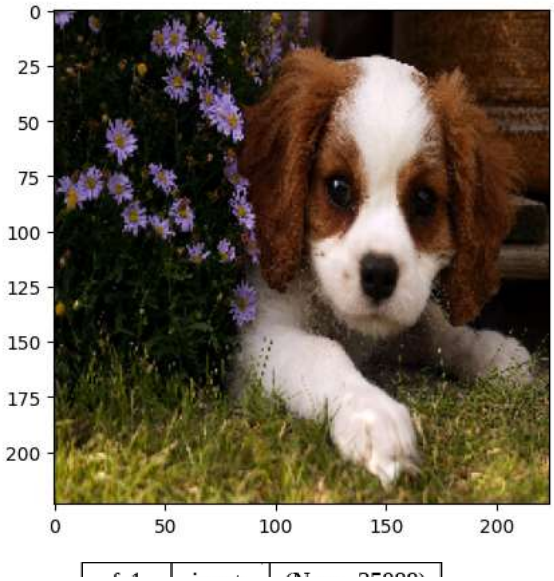
- **dog.jpg**(image/jpeg) - 248164 bytes, last modified: 4/9/2023 - 100% done

```
Saving dog.jpg to dog.jpg
```

```
from keras.utils.image_dataset import load_image
from tensorflow.keras.preprocessing.image import load_img
image = load_img('dog.jpg',target_size=(224,224))
```

```
import matplotlib.pyplot as plt
plt.imshow(image)
```

```
<matplotlib.image.AxesImage at 0x7f459456f4c0>
```




```
from tensorflow.keras.preprocessing.image import img_to_array
image= img_to_array(image)
```

```
image = image.reshape((1,image.shape[0],image.shape[1],image.shape[2]))
```

```
from tensorflow.keras.applications.vgg16 import preprocess_input
image = preprocess_input(image)
```

```
yhat = model.predict(image)
```

```
1/1 [==============================] - 1s 1s/step
```

```
print(yhat.shape)
```

```
(1, 1000)
```

```
import numpy as np
np.set_printoptions(formatter = {'float':'{: 0.3f}'.format})
print(yhat)
```

```
[[ 0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
   0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
   0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
```

```
0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
```

```
from tensorflow.keras.applications.vgg16 import decode_predictions
```

```
label= decode_predictions(yhat)
label= label[0][0]
print('%s (%.2f%%)' % (label[1],label[2]*100))
```

```
Blenheim_spaniel (76.40%)
```

```
import os
import sys
from tempfile import NamedTemporaryFile
from urllib.request import urlopen
from urllib.parse import unquote, urlparse
from urllib.error import HTTPError
from zipfile import ZipFile
import tarfile
import shutil

CHUNK_SIZE = 40960
DATA_SOURCE_MAPPING = ':https%3A%2F%2Fstorage.googleapis.com%2Fkaggle-data-sets%2F22169%2F30047%2Fbundle%2Farchive.zip%3FX-Goog-Algorithm

KAGGLE_INPUT_PATH='/kaggle/input'
KAGGLE_WORKING_PATH='/kaggle/working'
KAGGLE_SYMLINK='kaggle'

!umount /kaggle/input/ 2> /dev/null
shutil.rmtree('/kaggle/input', ignore_errors=True)
os.makedirs(KAGGLE_INPUT_PATH, 0o777, exist_ok=True)
os.makedirs(KAGGLE_WORKING_PATH, 0o777, exist_ok=True)

try:
  os.symlink(KAGGLE_INPUT_PATH, os.path.join("..", 'input'), target_is_directory=True)
except FileExistsError:
  pass
try:
  os.symlink(KAGGLE_WORKING_PATH, os.path.join("..", 'working'), target_is_directory=True)
except FileExistsError:
  pass

for data_source_mapping in DATA_SOURCE_MAPPING.split(','):
    directory, download_url_encoded = data_source_mapping.split(':')
    download_url = unquote(download_url_encoded)
    filename = urlparse(download_url).path
    destination_path = os.path.join(KAGGLE_INPUT_PATH, directory)
    try:
        with urlopen(download_url) as fileres, NamedTemporaryFile() as tfile:
            total_length = fileres.headers['content-length']
            print(f'Downloading {directory}, {total_length} bytes compressed')
            dl = 0
            data = fileres.read(CHUNK_SIZE)
            while len(data) > 0:
                dl += len(data)
                tfile.write(data)
                done = int(50 * dl / int(total_length))
                sys.stdout.write(f"\r[{'=' * done}{' ' * (50-done)}] {dl} bytes downloaded")
                sys.stdout.flush()
                data = fileres.read(CHUNK_SIZE)
            if filename.endswith('.zip'):
              with ZipFile(tfile) as zfile:
                zfile.extractall(destination_path)
            else:
              with tarfile.open(tfile.name) as tarfile:
                tarfile.extractall(destination_path)
            print(f'\nDownloaded and uncompressed: {directory}')
    except HTTPError as e:
        print(f'Failed to load (likely expired) {download_url} to path {destination_path}')
        continue
    except OSError as e:
        print(f'Failed to load {download_url} to path {destination_path}')
        continue

print('Data source import complete.')
```

```
Downloading , 334104 bytes compressed
[==================================================] 334104 bytes downloaded
Downloaded and uncompressed:
Data source import complete.
```

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import RandomizedSearchCV

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras import layers
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
import os
# print(os.listdir("../input"))
plt.style.use('ggplot')


filepath_dict = {'yelp':   '../input/sentiment labelled sentences/sentiment labelled sentences/yelp_labelled.txt',
                 'amazon': '../input/sentiment labelled sentences/sentiment labelled sentences/amazon_cells_labelled.txt',
                 'imdb':   '../input/sentiment labelled sentences/sentiment labelled sentences/imdb_labelled.txt'}

df_list = []
for source, filepath in filepath_dict.items():
    df = pd.read_csv(filepath, names=['sentence', 'label'], sep='\t')
    df['source'] = source
    df_list.append(df)


df_list
```

```
[                                          sentence  label source
 0                              Wow... Loved this place.      1   yelp
 1                                    Crust is not good.      0   yelp
 2             Not tasty and the texture was just nasty.      0   yelp
 3    Stopped by during the late May bank holiday of...      1   yelp
 4    The selection on the menu was great and so wer...      1   yelp
 ..                                                 ...    ...    ...
 995  I think food should have flavor and texture an...      0   yelp
 996                            Appetite instantly gone.      0   yelp
 997  Overall I was not impressed and would not go b...      0   yelp
 998  The whole experience was underwhelming, and I ...      0   yelp
 999  Then, as if I hadn't wasted enough of my life ...      0   yelp

 [1000 rows x 3 columns],
                                          sentence  label  source
 0    So there is no way for me to plug it in here i...      0  amazon
 1                          Good case, Excellent value.      1  amazon
 2                               Great for the jawbone.      1  amazon
 3    Tied to charger for conversations lasting more...      0  amazon
 4                                    The mic is great.      1  amazon
 ..                                                 ...    ...     ...
 995  The screen does get smudged easily because it ...      0  amazon
 996  What a piece of junk.. I lose more calls on th...      0  amazon
 997                       Item Does Not Match Picture.      0  amazon
 998  The only thing that disappoint me is the infra...      0  amazon
 999  You can not answer calls with the unit, never ...      0  amazon

 [1000 rows x 3 columns],
                                          sentence  label source
 0    A very, very, very slow-moving, aimless movie ...      0   imdb
 1    Not sure who was more lost - the flat characte...      0   imdb
 2    Attempting artiness with black & white and cle...      0   imdb
 3         Very little music or anything to speak of.        0   imdb
 4    The best scene in the movie was when Gerardo i...      1   imdb
 ..                                                 ...    ...    ...
 743  I just got bored watching Jessice Lange take h...      0   imdb
 744  Unfortunately, any virtue in this film's produ...      0   imdb
 745                  In a word, it is embarrassing.        0   imdb
 746                               Exceptionally bad!        0   imdb
 747  All in all its an insult to one's intelligence...      0   imdb

 [748 rows x 3 columns]]
```

```
df = pd.concat(df_list)
df.iloc[0]
```

```
sentence    Wow... Loved this place.
label                              1
source                          yelp
Name: 0, dtype: object
```

```
df.head()
```

	sentence	label	source
0	Wow... Loved this place.	1	yelp
1	Crust is not good.	0	yelp
2	Not tasty and the texture was just nasty.	0	yelp
3	Stopped by during the late May bank holiday of...	1	yelp
4	The selection on the menu was great and so wer...	1	yelp

Next steps: Generate code with `df` View recommended plots

```
df.tail()
```

	sentence	label	source
743	I just got bored watching Jessice Lange take h...	0	imdb
744	Unfortunately, any virtue in this film's produ...	0	imdb
745	In a word, it is embarrassing.	0	imdb
746	Exceptionally bad!	0	imdb
747	All in all its an insult to one's intelligence...	0	imdb

```
sentences = ['Rashmi likes ice cream', 'Rashmi hates chocolate.']


vectorizer = CountVectorizer(min_df=0, lowercase=False)
vectorizer.fit(sentences)
vectorizer.vocabulary_
```

```
{'Rashmi': 0, 'likes': 5, 'ice': 4, 'cream': 2, 'hates': 3, 'chocolate': 1}
```

```
vectorizer.transform(sentences).toarray()
```

```
array([[1, 0, 1, 0, 1, 1],
       [1, 1, 0, 1, 0, 0]])
```

```
df_yelp = df[df['source'] == 'yelp']
sentences = df_yelp['sentence'].values
y = df_yelp['label'].values

sentences_train, sentences_test, y_train, y_test = train_test_split(sentences, y, test_size=0.25, random_state=1000)


vectorizer = CountVectorizer()
vectorizer.fit(sentences_train)

X_train = vectorizer.transform(sentences_train)
X_test  = vectorizer.transform(sentences_test)


X_train
```

```
<750x1714 sparse matrix of type '<class 'numpy.int64'>'
        with 7368 stored elements in Compressed Sparse Row format>
```

```
classifier = LogisticRegression()
classifier.fit(X_train, y_train)
score = classifier.score(X_test, y_test)

print("Accuracy:", score)
```

```
Accuracy: 0.796
```

```
for source in df['source'].unique():
    df_source = df[df['source'] == source]
    sentences = df_source['sentence'].values
    y = df_source['label'].values

    sentences_train, sentences_test, y_train, y_test = train_test_split(
        sentences, y, test_size=0.25, random_state=1000)

    vectorizer = CountVectorizer()
    vectorizer.fit(sentences_train)
    X_train = vectorizer.transform(sentences_train)
```

```
X_test  = vectorizer.transform(sentences_test)

classifier = LogisticRegression()
classifier.fit(X_train, y_train)
score = classifier.score(X_test, y_test)
print('Accuracy for {} data: {:.4f}'.format(source, score))
```

```
 Accuracy for yelp data: 0.7960
 Accuracy for amazon data: 0.7960
 Accuracy for imdb data: 0.7487
```

```
from keras import Sequential
from keras.layers import Dense,SimpleRNN


model = Sequential()
model.add(SimpleRNN(3,input_shape=(4,5))) #simpleRNN layer with 3 nodes
model.add(Dense(1,activation='sigmoid')) #dense layer with one node
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn (SimpleRNN)      (None, 3)                 27

 dense (Dense)               (None, 1)                 4

=================================================================
Total params: 31 (124.00 Byte)
Trainable params: 31 (124.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
print(model.get_weights()[0].shape)
model.get_weights()[0]
```

```
(5, 3)
array([[ 0.06470662, -0.2154178 ,  0.5621187 ],
       [ 0.11821812,  0.3900357 ,  0.79684955],
       [-0.5363575 ,  0.363586  ,  0.625747  ],
       [-0.12909883, -0.5009439 ,  0.08297819],
       [-0.0636968 ,  0.6303415 , -0.52596885]], dtype=float32)
```

```
print(model.get_weights()[1].shape)
model.get_weights()[1]
```

```
(3, 3)
array([[ 0.27629387,  0.469978  , -0.83832115],
       [ 0.95624393, -0.22176677,  0.19083244],
       [ 0.0962247 ,  0.85436535,  0.5106865 ]], dtype=float32)
```

```
print(model.get_weights()[2].shape)
model.get_weights()[2]
```

```
(3,)
array([0., 0., 0.], dtype=float32)
```

```
print(model.get_weights()[3].shape)
model.get_weights()[3]
```

```
(3, 1)
array([[-0.0814203],
       [ 1.1457287],
       [ 1.1718167]], dtype=float32)
```

```
print(model.get_weights()[4].shape)
model.get_weights()[4]
```

```
(1,)
array([0.], dtype=float32)
```

Double-click (or enter) to edit

```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras.layers import Input, Dense, Concatenate
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
```

```
import numpy as np
from keras.preprocessing.text import Tokenizer


docs = ['go manralai',
    'manralai manralai',
    'yo yo ai',
    'manralai aggregating content for data science',
    'manralai will win',
    'ai ai',
    'dl dl',
    'ml ml',
    'dl is changing world',
    'cv helps in vision tasks']
docs
```

```
['go manralai',
 'manralai manralai',
 'yo yo ai',
 'manralai aggregating content for data science',
 'manralai will win',
 'ai ai',
 'dl dl',
 'ml ml',
 'dl is changing world',
 'cv helps in vision tasks']
```

```
tokenizer = Tokenizer(oov_token='mukeshmanral') #out of vocab token is mukeshmanral
tokenizer.fit_on_texts(docs)


print(tokenizer.word_index,end=' ')
```

```
{'mukeshmanral': 1, 'manralai': 2, 'ai': 3, 'dl': 4, 'yo': 5, 'ml': 6, 'go': 7, 'aggregating': 8, 'content': 9, 'for': 10, 'data': 11, '
```

```
print(tokenizer.word_counts,end=' ')
```

```
OrderedDict([('go', 1), ('manralai', 5), ('yo', 2), ('ai', 3), ('aggregating', 1), ('content', 1), ('for', 1), ('data', 1), ('science',
```

```
tokenizer.document_count
```

```
10
```

```
sequences = tokenizer.texts_to_sequences(docs)
sequences
```

```
[[7, 2],
 [2, 2],
 [5, 5, 3],
 [2, 8, 9, 10, 11, 12],
 [2, 13, 14],
 [3, 3],
 [4, 4],
 [6, 6],
 [4, 15, 16, 17],
 [18, 19, 20, 21, 22]]
```

```
from keras.utils import pad_sequences

sequences = pad_sequences(sequences,padding='post')
sequences
```

```
array([[ 7,  2,  0,  0,  0,  0],
       [ 2,  2,  0,  0,  0,  0],
       [ 5,  5,  3,  0,  0,  0],
       [ 2,  8,  9, 10, 11, 12],
       [ 2, 13, 14,  0,  0,  0],
       [ 3,  3,  0,  0,  0,  0],
       [ 4,  4,  0,  0,  0,  0],
       [ 6,  6,  0,  0,  0,  0],
       [ 4, 15, 16, 17,  0,  0],
       [18, 19, 20, 21, 22,  0]], dtype=int32)
```

```
docs = ['go manralai',
        'manralai manralai',
        'yo yo ai',
        'manralai aggregating content for data science',
        'manralai will win',
        'ai ai',
        'dl dl',
        'ml ml',
        'dl is changing world',
        'cv helps in vision tasks']


from keras.datasets import imdb
from keras import Sequential
from keras.layers import Dense,SimpleRNN,Flatten


(X_train,y_train),(X_test,y_test) = imdb.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 [==============================] - 0s 0us/step
```

```
X_train.shape, X_test.shape
```

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

```
print(X_train[0],end='')
```

```
[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4, 173, 36, 256, 5, 25, 100, 43, 838, 112, 50, 670, 22665, 9, 35, 480
```

```
X_train = pad_sequences(X_train,padding='post',maxlen=30) #padding='post' | pading zeroes will be implemented latter
X_test = pad_sequences(X_test,padding='post',maxlen=30) #taking only first 30 words of every review (training time will be reduced for now)


# 25000 reviews each having 30 words in them
X_train.shape
```

```
(25000, 30)
```

```
X_train[10]
```

```
array([    4,   277,   199,   166,   281,     5,  1030,     8,    30,
         179,  4442,   444, 13772,     9,     6,   371,    87,   189,
          22,     5,    31,     7,     4,   118,     7,     4,  2068,
         545,  1178,   829], dtype=int32)
```

```
model = Sequential()

# we only want output in last | return sequences False means there will be nothing going out of the RNN layer
model.add(SimpleRNN(32,input_shape=(30,1),return_sequences=False)) #in every review there are 30 words, so 30 time stamps # 32 nodes in RRN
model.add(Dense(1,activation='sigmoid'))

model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn_1 (SimpleRNN)    (None, 32)                1088

 dense_1 (Dense)             (None, 1)                 33

=================================================================
Total params: 1121 (4.38 KB)
Trainable params: 1121 (4.38 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

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

history=model.fit(X_train,y_train,epochs=10,validation_data=(X_test,y_test))
```

```
Epoch 1/10
782/782 [==============================] - 5s 6ms/step - loss: 0.6994 - accuracy: 0.5028 - val_loss: 0.6946 - val_accuracy: 0.5028
```

```
Epoch 2/10
782/782 [==============================] - 5s 6ms/step - loss: 0.6935 - accuracy: 0.5001 - val_loss: 0.6944 - val_accuracy: 0.5062
Epoch 3/10
782/782 [==============================] - 4s 5ms/step - loss: 0.6933 - accuracy: 0.5068 - val_loss: 0.6930 - val_accuracy: 0.5064
Epoch 4/10
782/782 [==============================] - 6s 7ms/step - loss: 0.6930 - accuracy: 0.5046 - val_loss: 0.6931 - val_accuracy: 0.5034
Epoch 5/10
782/782 [==============================] - 4s 5ms/step - loss: 0.6926 - accuracy: 0.5108 - val_loss: 0.6931 - val_accuracy: 0.5050
Epoch 6/10
782/782 [==============================] - 5s 7ms/step - loss: 0.6929 - accuracy: 0.5102 - val_loss: 0.6997 - val_accuracy: 0.5020
Epoch 7/10
782/782 [==============================] - 4s 5ms/step - loss: 0.6927 - accuracy: 0.5030 - val_loss: 0.6931 - val_accuracy: 0.5037
Epoch 8/10
782/782 [==============================] - 4s 5ms/step - loss: 0.6925 - accuracy: 0.5061 - val_loss: 0.6933 - val_accuracy: 0.5069
Epoch 9/10
782/782 [==============================] - 6s 8ms/step - loss: 0.6927 - accuracy: 0.5074 - val_loss: 0.6930 - val_accuracy: 0.5046
Epoch 10/10
782/782 [==============================] - 4s 6ms/step - loss: 0.6926 - accuracy: 0.5042 - val_loss: 0.6930 - val_accuracy: 0.5046
```

Sentiment analysis of IMDB movie reviews using LSTM networks

In this notebook, we are using LSTM recurrent neural networks to classify IMDB movie reviews positive or negative.

Prerequisites:

- **Basic knowledge of Deep Learning and Recurrent Neural Networks.**

Libraries:

- **Keras:** A library for creating neural networks in few lines.
- **Matplotlib:** Matplotlib is a plotting library. You can use it to draw different types of graphs, like line, scattered, bar, etc.

Introduction

We are using a dataset of 25,000 movies reviews from IMDB, labeled by sentiment (positive/negative). Reviews have been preprocessed, and each review is encoded as a sequence of word indexes (integers). Words are indexed by overall frequency in the dataset, so that for instance the integer "3" encodes the 3rd most frequent word in the data. This allows for quick filtering operations such as: "only consider the top 10,000 most common words, but eliminate the top 20 most common words".

As a convention, "0" does not stand for a specific word, but instead is used to encode any unknown word.

Preprocessing

```
from keras.models import Sequential
from keras.layers import CuDNNLSTM, Embedding, Dense
from keras.datasets import imdb
from keras.preprocessing import sequence

import matplotlib.pyplot as plt


max_word_range = 10000

(input_train, output_train), (input_test, output_test) = imdb.load_data(num_words=max_word_range)

print(len(input_train), 'train sequences.')
print(len(input_test), 'test sequences.')
```

```
25000 train sequences.
25000 test sequences.
```

```
max_word_amount = 500

input_train = sequence.pad_sequences(input_train, maxlen=max_word_amount)
input_test = sequence.pad_sequences(input_test, maxlen=max_word_amount)

print('input_train shape:', input_train.shape)
print('input_test shape:', input_test.shape)
```

```
input_train shape: (25000, 500)
input_test shape: (25000, 500)
```

Model

Now, let's create our model. We are initiating our sequential object with a list that contains the layers we want to use

Now, let's create our model. We are initiating our sequential object with a list that contains the layers we want to use.

- **Embedding:** Turns positive integers (indexes) into dense vectors of fixed size. This layer can only be used as the first layer in a model.
- **CuDNNLSTM:** Fast LSTM implementation with CuDNN. Can only be run on GPU, with the TensorFlow backend.
- **Dense (Fully Connected):** They are the simple layers that have connections between every input and output neuron.

We should use Sigmoid as our output activation function since there are two classes to choose from. Use Softmax in multi-class problems.

```
model = Sequential([
  Embedding(max_word_range, 32, input_length=max_word_amount),
  CuDNNLSTM(32),
  Dense(1, activation='sigmoid')
])

model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, 500, 32)           320000
_________________________________________________________________
cu_dnnlstm_3 (CuDNNLSTM)     (None, 32)                8448
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 33
=================================================================
Total params: 328,481
Trainable params: 328,481
Non-trainable params: 0
_________________________________________________________________
```

Now we have to decide which optimizer and loss function to use for our model. Since this is a binary classification problem and since we are using Sigmoid for the last activation function, Binary Crossentropy should be our choice.

And for our optimizer, we can pick RMSProp. It works really well on recurrent neural networks.

Loss Function: Binary Crossentropy

For more information about Keras loss functions click here.

Optimizer: RMSProp

For more information about Keras optimizers click here.

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

Training

Let's train our model. Additional to loss, accuracy will be calculated and printed out on every epoch.

```
history = model.fit(input_train, output_train,
                    epochs=10,
                    batch_size=128,
                    validation_split=0.2)
```

```
Train on 20000 samples, validate on 5000 samples
Epoch 1/10
20000/20000 [==============================] - 10s 522us/step - loss: 0.5081 - acc: 0.7506 - val_loss: 0.3448 - val_acc: 0
Epoch 2/10
20000/20000 [==============================] - 10s 477us/step - loss: 0.3101 - acc: 0.8754 - val_loss: 0.3024 - val_acc: 0
Epoch 3/10
```

```
20000/20000 [==============================] - 10s 478us/step - loss: 0.2520 - acc: 0.9050 - val_loss: 0.2810 - val_acc: 0
Epoch 4/10
20000/20000 [==============================] - 10s 477us/step - loss: 0.2129 - acc: 0.9196 - val_loss: 0.3027 - val_acc: 0
Epoch 5/10
20000/20000 [==============================] - 10s 479us/step - loss: 0.1863 - acc: 0.9308 - val_loss: 0.2921 - val_acc: 0
Epoch 6/10
20000/20000 [==============================] - 10s 478us/step - loss: 0.1690 - acc: 0.9410 - val_loss: 0.3083 - val_acc: 0
Epoch 7/10
20000/20000 [==============================] - 10s 477us/step - loss: 0.1530 - acc: 0.9450 - val_loss: 0.5410 - val_acc: 0
Epoch 8/10
20000/20000 [==============================] - 9s 465us/step - loss: 0.1368 - acc: 0.9517 - val_loss: 0.3697 - val_acc: 0.
Epoch 9/10
20000/20000 [==============================] - 9s 474us/step - loss: 0.1314 - acc: 0.9551 - val_loss: 0.3180 - val_acc: 0.
Epoch 10/10
20000/20000 [==============================] - 10s 477us/step - loss: 0.1183 - acc: 0.9584 - val_loss: 0.3393 - val_acc: 0
```

We can see how training went using our history object.

```
loss = history.history['loss']
acc = history.history['acc']
val_loss = history.history['val_loss']
val_acc = history.history['val_acc']
epochs = range(len(loss))

f, axarr = plt.subplots(1, 2, figsize=(12, 6))
p0 = axarr[0]
p1 = axarr[1]

p0.set_title("Training and Validation Loss")
p1.set_title("Training and Validation Accuracy")

p0l0 = p0.plot(epochs, loss, "-b", label="Training Loss")
p0l1 = p0.plot( epochs, val_loss, "-r", label="Validation Loss")

p1l0 = p1.plot(epochs, acc, "-b", label="Training Accuracy")
p1l1 = p1.plot(epochs, val_acc, "-r", label="Validation Accuracy")

legend0 = p0.legend()
legend1 = p1.legend()
```

Evaluation

Finally the training process is done. Now we can evaluate our model's performance on the test set.

```
score = model.evaluate(input_test, output_test)

print('Test Loss:', score[0])
print('Test Accuracy:', score[1])
```

```
25000/25000 [==============================] - 23s 926us/step
Test Loss: 0.38157773265838624
Test Accuracy: 0.8678
```

As you can see, our model's predictions are 86% accurate on the test set.

Let's predict couple of examples.

Side note: You can't predict single examples with "model.predict()". But you can reshape the data to have an extra batch size dimention to fix it.

```
predictions·=·model.predict(input_test[0:5])

true_labels·=·output_test[0:5]

print("0·is·a·negative·comment,·1·is·a·positive·comment.")

for·i·in·range(len(predictions)):
··print("Prediction:",·int(round(predictions[i][0])),·",",
·····""True·Label:",·true_labels[i])
```

```
0 is a negative comment, 1 is a positive comment.
Prediction: 0 , True Label: 0
Prediction: 1 , True Label: 1
Prediction: 1 , True Label: 1
Prediction: 1 , True Label: 0
Prediction: 1 , True Label: 1
```