Assignment 1

1. Name: Harsh Chaudhari
2. Batch: P-10
3. Roll No.: 43215

Problem Statement :

Study of Deep learning Packages: Tensorflow, Keras, Theano and PyTorch. Document the distinct features and functionality of the packages.

In [2]:

```
import numpy as np
```

1. Tensorflow In

[3]:

```
import tensorflow as tf
```

In [4]:

```
print(tf.__version__)
```

2.10.0 **2.**

Keras

In [5]:

```
from keras import datasets #
Load MNIST datasets from keras
(train_images, train_labels), (test_images, test_labels) = datasets.mnist.load_data
```

In [6]:

```
train_images.shape
```

Out[6]:

(60000, 28, 28)

In [7]:

```
test_images.shape
```

Out[7]:

(10000, 28, 28)

3. Theano

In [14]:

```
conda install Theano
```

```
Collecting package metadata (current_repodata.json): ...working... d
one
Solving environment: ...working... done

## Package Plan ##

  environment location: D:\anaconda

  added / updated specs:
    - theano


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    anaconda-2022.10           |          py310_0          13 KB
    ca-certificates-2022.10.11 |       haa95532_0         125 KB
    certifi-2022.9.24          |   py39haa95532_0         154 KB
    libgpuarray 0 7 6          |        h2bbff1b 1         255 KB
```

In [8]:

```python
import theano.tensor as T
from theano import function
```

In [10]:

```python
# Declaring 2 variables
x = T.dscalar('x')
y = T.dscalar('y')
```

In [11]:

```python
# Summing up the 2 numbers
z = x + y
```

In [12]:

```python
# Converting it to a callable object so that it takes matrix as parameters

f = function([x, y], z)
```

In [13]:

```python
f(5, 7)
```

Out[13]:

```
array(12.)
```

4. PyTorch

In [14]:

```
!pip3 install torch torchvision torchaudio --extra-index-url https://download.pytor
```

```
Looking in indexes: https://pypi.org/simple, (https://pypi.org/simpl
e,) https://download.pytorch.org/whl/cu115 (https://download.pytorch.o
rg/whl/cu115)
Requirement already satisfied: torch in d:\anaconda\lib\site-packages
 (1.13.0)
Requirement already satisfied: torchvision in d:\anaconda\lib\site-pac
kages (0.14.0)
Requirement already satisfied: torchaudio in d:\anaconda\lib\site-pack
ages (0.13.0)
Requirement already satisfied: typing_extensions in d:\anaconda\lib\si
te-packages (from torch) (4.3.0)
Requirement already satisfied: numpy in d:\anaconda\lib\site-packages
 (from torchvision) (1.21.5)
Requirement already satisfied: requests in d:\anaconda\lib\site-packag
es (from torchvision) (2.28.1)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in d:\anaconda\li
b\site-packages (from torchvision) (9.2.0)
Requirement already satisfied: idna<4,>=2.5 in d:\anaconda\lib\site-pa
ckages (from requests->torchvision) (3.3)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in d:\anaconda\li
b\site-packages (from requests->torchvision) (1.26.11)
Requirement already satisfied: certifi>=2017.4.17 in d:\anaconda\lib\s
ite-packages (from requests->torchvision) (2022.9.24)
Requirement already satisfied: charset-normalizer<3,>=2 in d:\anaconda
\lib\site-packages (from requests->torchvision) (2.0.4)
```

In [15]:

```python
import torch
import torch.nn as nn
```

In [16]:

```python
print(torch.__version__)
```

```
1.13.0
```

In [17]:

```python
torch.cuda.is_available()
```

Out[17]:

```
False
```

In [ ]:

Assignment 2

1. Name: Harsh Chaudhari
2. Batch: P-10
3. Roll No.: 43215

Problem Statement :

    Implementing Feedforward neural networks with Keras and TensorFlow

a. Import necessary packages

In [8]:

```
from sklearn.preprocessing import LabelBinarizer
from sklearn.metrics import classification_report
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense from
tensorflow.keras.optimizers import SGD from
tensorflow.keras.datasets import mnist from
tensorflow.keras import backend as K import
matplotlib.pyplot as plt import numpy as np
```

b. Load the training and testing data (MNIST/CIFAR10)

Grabbing the mnist dataset

In [10]:

```
((X_train, Y_train), (X_test, Y_test)) = mnist.load_data()
X_train = X_train.reshape((X_train.shape[0], 28 * 28 * 1))
X_test = X_test.reshape((X_test.shape[0], 28 * 28 * 1))
X_train = X_train.astype("float32") / 255.0
X_test = X_test.astype("float32") / 255.0
```

In [11]:

```
lb = LabelBinarizer()
Y_train = lb.fit_transform(Y_train)
Y_test = lb.transform(Y_test)
```

c. Define the network architecture using Keras

Building the model

In
[12]:

```
model = Sequential()
model.add(Dense(128, input_shape=(784,), activation="sigmoid"))
model.add(Dense(64, activation="sigmoid")) model.add(Dense(10,
activation="softmax"))
```

d. Train the model using SGD

In [13]:

```
sgd = SGD(0.01)
epochs=10
model.compile(loss="categorical_crossentropy", optimizer=sgd,metrics=["accuracy"])
H = model.fit(X_train, Y_train, validation_data=(X_test, Y_test),epochs=epochs, bat
```

```
Epoch 1/10
469/469 [==============================] - 2s 3ms/step - loss: 2.308
9 - accuracy: 0.1916 - val_loss: 2.2612 - val_accuracy: 0.2154
Epoch 2/10
469/469 [==============================] - 1s 3ms/step - loss: 2.238
5 - accuracy: 0.3252 - val_loss: 2.2115 - val_accuracy: 0.4688
Epoch 3/10
469/469 [==============================] - 1s 2ms/step - loss: 2.182
4 - accuracy: 0.4775 - val_loss: 2.1447 - val_accuracy: 0.5425
Epoch 4/10
469/469 [==============================] - 1s 2ms/step - loss: 2.103
8 - accuracy: 0.5598 - val_loss: 2.0488 - val_accuracy: 0.6049
Epoch 5/10
469/469 [==============================] - 1s 2ms/step - loss: 1.990
1 - accuracy: 0.6038 - val_loss: 1.9119 - val_accuracy: 0.6428
Epoch 6/10
469/469 [==============================] - 1s 2ms/step - loss: 1.834
0 - accuracy: 0.6388 - val_loss: 1.7330 - val_accuracy: 0.6669
Epoch 7/10
469/469 [==============================] 1s 2ms/step loss: 1 645
```

e.

Evaluate the network Making the predictions

[6]:

```
predictions = model.predict(X_test, batch_size=128)
print(classification_report(Y_test.argmax(axis=1),predictions.argmax(axis=1),target
```

	precision	recall	f1-score	support
0	0.79	0.96	0.87	980
1	0.78	0.98	0.87	1135
2	0.83	0.74	0.79	1032

```
In
 3    0.64      0.82      0.72      1010
 4    0.66      0.79      0.72       982
 5    0.84      0.45      0.59       892
 6    0.82      0.87      0.85       958
 7    0.80      0.86      0.83      1028
 8    0.84      0.55      0.67       974           9       0.68      0.52      0.59
 1009

   accuracy                              0.76     10000
macro avg       0.77      0.75      0.75     10000 weighted
avg       0.77      0.76      0.75     10000
```

f. Plot the training loss and accuracy

In [17]:

```
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, epochs), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, epochs), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, epochs), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, epochs), H.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy") plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy") plt.legend() plt.plot()
```

Out[17]:

```
[]
```

In [ ]:

Assignment 3

1. Name: Harsh Chaudhari
2. Batch: P-10
3. Roll No.: 43215

Problem Statement :

Build the Image classification model

Importing the libraries In

[1]:

```
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
```

a. Loading and preprocessing the image data

Grabbing CIFAR10 dataset

In [2]:

```
(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_da
train_images, test_images = train_images / 255.0, test_images / 255.0 In [4]:
```

```
type(train_images)
```

Out[4]:

numpy.ndarray

Showing images of mentioned categories

In [5]:

```
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer','dog', 'frog', 'hors

plt.figure(figsize=(10,10))
for i in range(10):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i])
    plt.xlabel(class_names[train_labels[i][0]])
plt.show()
```

b. Defining the model's architecture

Building CNN model

[6]:

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10))
 model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 32)        896
 max_pooling2d (MaxPooling2D  (None, 15, 15, 32)       0
 )
```

In

```
 conv2d_1 (Conv2D)           (None, 13, 13, 64)        18496
 max_pooling2d_1 (MaxPooling  (None, 6, 6, 64)         0
 2D)
 conv2d_2 (Conv2D)           (None, 4, 4, 64)          36928
 flatten (Flatten)           (None, 1024)              0
 dense (Dense)               (None, 64)                65600
 dense_1 (Dense)             (None, 10)                650
=================================================================
Total params: 122,570
Trainable params: 122,570
Non-trainable params: 0
```

c.

Training the model

Model compilation

In [7]:

```
model.compile(optimizer='adam',loss=tf.keras.losses.SparseCategoricalCrossentropy(f
```

d. Estimating the model's performance

[9]:

```
epochs = 10
h = model.fit(train_images, train_labels, epochs=epochs, validation_data=(test_imag
```

```
Epoch 1/10
1563/1563 [==============================] - 66s 42ms/step - loss: 1.1
581 - accuracy: 0.5890 - val_loss: 1.0580 - val_accuracy: 0.6277 Epoch
2/10
1563/1563 [==============================] - 69s 44ms/step - loss: 1.0
107 - accuracy: 0.6472 - val_loss: 0.9943 - val_accuracy: 0.6511 Epoch
3/10
1563/1563 [==============================] - 66s 43ms/step - loss: 0.9
166 - accuracy: 0.6772 - val_loss: 0.9544 - val_accuracy: 0.6693 Epoch
4/10
1563/1563 [==============================] - 65s 41ms/step - loss: 0.8
453 - accuracy: 0.7075 - val_loss: 0.9113 - val_accuracy: 0.6789 Epoch
5/10
1563/1563 [==============================] - 66s 42ms/step - loss: 0.7
```

In

```
916 - accuracy: 0.7241 - val_loss: 0.9141 - val_accuracy: 0.6884 Epoch
6/10
1563/1563 [==============================] - 65s 42ms/step - loss: 0.7
393 - accuracy: 0.7418 - val_loss: 0.8678 - val_accuracy: 0.7065 Epoch
7/10
1563/1563 [==============================] - 66s 42ms/step - loss: 0.6
958 - accuracy: 0.7573 - val_loss: 0.9028 - val_accuracy: 0.6945 Epoch
8/10
1563/1563 [==============================] - 66s 42ms/step - loss: 0.6
597 - accuracy: 0.7692 - val_loss: 0.8688 - val_accuracy: 0.7020 Epoch
9/10
1563/1563 [==============================] - 65s 42ms/step - loss: 0.6
243 - accuracy: 0.7818 - val_loss: 0.8780 - val_accuracy: 0.7071 Epoch
10/10
1563/1563 [==============================] - 65s 42ms/step - loss: 0.5
905 - accuracy: 0.7920 - val_loss: 0.8929 - val_accuracy: 0.7151
```

In [ ]:

Assignment 4

```
1. Name: Harsh Chaudhari
2. Batch: P-10
3. Roll No.: 43215
```

Problem Statement :

```
ECG Anomaly detection using Autoencoders
```

a. Import required libraries

In [10]:

```
import numpy as np import pandas as pd import
tensorflow as tf import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score from
tensorflow.keras.optimizers import Adam from
sklearn.preprocessing import MinMaxScaler from
tensorflow.keras import Model, Sequential from
tensorflow.keras.layers import Dense, Dropout from
sklearn.model_selection import train_test_split
from tensorflow.keras.losses import MeanSquaredLogarithmicError
```

b. Upload / access the dataset

In [11]:

```
PATH_TO_DATA = 'http://storage.googleapis.com/download.tensorflow.org/data/ecg.csv'
data = pd.read_csv(PATH_TO_DATA, header=None) data.head() Out[11]:
```

	0	1	2	3	4	5	6	7
0	-0.112522 1.818286	-2.827204 -1.250	-3.773897	-4.349751	-4.376041	-3.474986	-2.181408	-
1	-1.100878 0.992258	-3.996840 -0.754	-4.285843	-4.506579	-4.022377	-3.234368	-1.566126	-
2	-0.567088 1.490659	-2.593450 -1.183	-3.874230	-4.584095	-4.187449	-3.151462	-1.742940	-
3	0.490473 1.671131	-1.914407 -1.333	-3.616364	-4.318823	-4.268016	-3.881110	-2.993280	-
4	0.800232 1.783423	-0.874252 -1.594	-2.384761	-3.973292	-4.338224	-3.802422	-2.534510	-

5 rows × 141 columns

In

Finding shape of the dataset

[2]:

```
data.shape
```

Out[2]:

```
(4998, 141)
```

Splitting training and testing dataset

In [3]:

```
features = data.drop(140, axis=1)
target = data[140]
x_train, x_test, y_train, y_test = train_test_split(
    features, target, test_size=0.2, stratify=target
)
train_index = y_train[y_train == 1].index
train_data = x_train.loc[train_index]
```

Scaling the data using MinMaxScaler

In [12]:

```
min_max_scaler = MinMaxScaler(feature_range=(0, 1))
x_train_scaled = min_max_scaler.fit_transform(train_data.copy())
x_test_scaled = min_max_scaler.transform(x_test.copy())
```

c. Encoder converts it into latent representation

d. Decoder networks convert it back to the original input

In

Creating autoencoder subclass by extending Model class from keras

[13]:

```
class AutoEncoder(Model):
  def __init__(self, output_units, ldim=8):
    super().__init__()
    self.encoder = Sequential([
      Dense(64, activation='relu'),
      Dropout(0.1),
      Dense(32, activation='relu'),
      Dropout(0.1),
      Dense(16, activation='relu'),
      Dropout(0.1),
      Dense(ldim, activation='relu')
    ])
    self.decoder = Sequential([
      Dense(16, activation='relu'),
      Dropout(0.1),
      Dense(32, activation='relu'),
      Dropout(0.1),
      Dense(64, activation='relu'),
      Dropout(0.1),
      Dense(output_units, activation='sigmoid')
    ])

  def call(self, inputs):
    encoded = self.encoder(inputs)
    decoded = self.decoder(encoded)
    return decoded
```

e. Compile the models with Optimizer, Loss, and Evaluation Metrics

Model configuration

[14]:

```
model = AutoEncoder(output_units=x_train_scaled.shape[1])
model.compile(loss='msle', metrics=['mse'], optimizer='adam')
epochs = 20

history = model.fit(
    x_train_scaled,
    x_train_scaled,
    epochs=epochs,
    batch_size=512,
    validation_data=(x_test_scaled, x_test_scaled)
)
```

```
Epoch 1/20
5/5 [==============================] - 1s 43ms/step - loss: 0.0110 - m
se: 0.0243 - val_loss: 0.0132 - val_mse: 0.0301
Epoch 2/20
5/5 [==============================] - 0s 12ms/step - loss: 0.0106 - m
se: 0.0234 - val_loss: 0.0129 - val_mse: 0.0295
Epoch 3/20
```

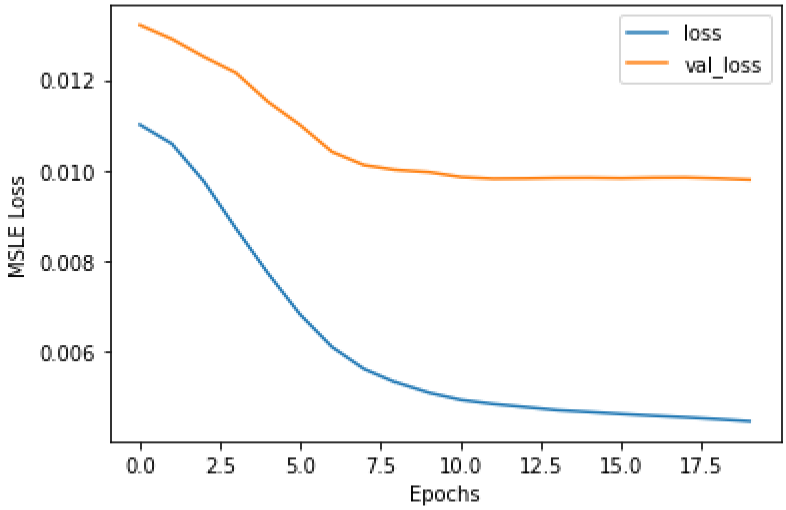
In

```
5/5 [==============================] - 0s 11ms/step - loss: 0.0098 - m
se: 0.0215 - val_loss: 0.0125 - val_mse: 0.0286
Epoch 4/20
5/5 [==============================] - 0s 15ms/step - loss: 0.0087 - m
se: 0.0193 - val_loss: 0.0122 - val_mse: 0.0278
Epoch 5/20
5/5 [==============================] - 0s 15ms/step - loss: 0.0077 - m
se: 0.0171 - val_loss: 0.0115 - val_mse: 0.0264
Epoch 6/20
5/5 [==============================] - 0s 11ms/step - loss: 0.0068 - m
se: 0.0151 - val_loss: 0.0110 - val_mse: 0.0252
Epoch 7/20
5/5 [==============================] - 0s 12ms/step - loss: 0.0061 - m
se: 0.0135 - val_loss: 0.0104 - val_mse: 0.0239
Epoch 8/20
5/5 [==============================] - 0s 11ms/step - loss: 0.0056 - m
se: 0.0124 - val_loss: 0.0101 - val_mse: 0.0233
Epoch 9/20
5/5 [==============================] - 0s 11ms/step - loss: 0.0053 - m
se: 0.0118 - val_loss: 0.0100 - val_mse: 0.0230
Epoch 10/20
5/5 [==============================] - 0s 12ms/step - loss: 0.0051 - m
se: 0.0113 - val_loss: 0.0100 - val_mse: 0.0229
Epoch 11/20
5/5 [==============================] - 0s 12ms/step - loss: 0.0049 - m
se: 0.0109 - val_loss: 0.0099 - val_mse: 0.0227
Epoch 12/20
5/5 [==============================] - 0s 12ms/step - loss: 0.0049 - m
se: 0.0108 - val_loss: 0.0098 - val_mse: 0.0226
Epoch 13/20
5/5 [==============================] - 0s 13ms/step - loss: 0.0048 - m
se: 0.0106 - val_loss: 0.0098 - val_mse: 0.0227
Epoch 14/20
5/5 [==============================] - 0s 12ms/step - loss: 0.0047 - m
se: 0.0105 - val_loss: 0.0098 - val_mse: 0.0227
Epoch 15/20
5/5 [==============================] - 0s 11ms/step - loss: 0.0047 - m
se: 0.0104 - val_loss: 0.0098 - val_mse: 0.0227
Epoch 16/20
5/5 [==============================] - 0s 11ms/step - loss: 0.0046 - m
```

```
se: 0.0103 - val_loss: 0.0098 - val_mse: 0.0227
Epoch 17/20
5/5 [==============================] - 0s 12ms/step - loss: 0.0046 - m
se: 0.0102 - val_loss: 0.0098 - val_mse: 0.0227
Epoch 18/20
5/5 [==============================] - 0s 13ms/step - loss: 0.0046 - m
se: 0.0101 - val_loss: 0.0098 - val_mse: 0.0227
Epoch 19/20
5/5 [==============================] - 0s 12ms/step - loss: 0.0045 - m
se: 0.0100 - val_loss: 0.0098 - val_mse: 0.0227
Epoch 20/20
5/5 [==============================] - 0s 11ms/step - loss: 0.0045 - m

se: 0.0099 - val_loss: 0.0098 - val_mse: 0.0226 In [15]:
```

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.xlabel('Epochs')
plt.ylabel('MSLE Loss')
plt.legend(['loss', 'val_loss'])
plt.show()
```



Finding threshold for anomaly and doing predictions

In [17]:

```
def find_threshold(model, x_train_scaled):
  reconstructions = model.predict(x_train_scaled)
  reconstruction_errors = tf.keras.losses.msle(reconstructions, x_train_scaled)
threshold = np.mean(reconstruction_errors.numpy()) \
   + np.std(reconstruction_errors.numpy())
return threshold
 def get_predictions(model, x_test_scaled,
threshold):
  predictions = model.predict(x_test_scaled)
  errors = tf.keras.losses.msle(predictions, x_test_scaled)
anomaly_mask = pd.Series(errors) > threshold
  preds = anomaly_mask.map(lambda x: 0.0 if x == True else 1.0)
return preds

threshold = find_threshold(model, x_train_scaled)
print(f"Threshold: {threshold}")
```

```
Threshold: 0.009662778406606926
```

Getting accuracy score

In [18]:

```
predictions = get_predictions(model, x_test_scaled, threshold)
accuracy_score(predictions, y_test)
```

Out[18]:

```
0.951
```

In [ ]:

Assignment 5

1. Name: Harsh Chaudhari
2. Batch: P-10
3. Roll No.: 43215

Problem Statement :

Implement the Continuous Bag of Words (CBOW) Model

Importing libraries

In [1]:

```
from keras.preprocessing import text
from keras.utils import np_utils
from keras.preprocessing import sequence
from keras_preprocessing.sequence import pad_sequences
import numpy as np
import pandas as pd
```

Taking random sentences as data

In [2]:

```
data = """Deep learning (also known as deep structured learning) is part of a broad
Deep-learning architectures such as deep neural networks, deep belief networks, dee
"""
dl_data = data.split()
```

a. Data preparation

Tokenization

In [3]:

```
tokenizer = text.Tokenizer()
tokenizer.fit_on_texts(dl_data)
word2id = tokenizer.word_index

word2id['PAD'] = 0
id2word = {v:k for k, v in word2id.items()}
wids = [[word2id[w] for w in text.text_to_word_sequence(doc)] for doc in dl_data]

vocab_size = len(word2id)
embed_size = 100
window_size = 2

print('Vocabulary  Size:',  vocab_size)
print('Vocabulary Sample:', list(word2id.items())[:10])
```

```
Vocabulary Size: 75
```

```
Vocabulary Sample: [('learning', 1), ('deep', 2), ('networks', 3), ('n
eural', 4), ('and', 5), ('as', 6), ('of', 7), ('machine', 8), ('superv
ised', 9), ('have', 10)]
```

b. Generate training data

Generating (context word, target/label word) pairs

In [4]:

```
def generate_context_word_pairs(corpus, window_size, vocab_size):
    context_length = window_size*2
for words in corpus:
        sentence_length = len(words)
 for index, word in enumerate(words):
context_words = []              label_word
= []                         start = index -
window_size              end = index +
window_size + 1

            context_words.append([words[i]
                                 for i in range(start, end)
if 0 <= i < sentence_length
and i != index])              label_word.append(word)

            x = pad_sequences(context_words, maxlen=context_length)
y = np_utils.to_categorical(label_word, vocab_size)
yield (x, y)
            i
= 0
for x, y in generate_context_word_pairs(corpus=wids, window_size=window_size, vocab
if 0 not in x[0]:
        print('Context (X):', [id2word[w] for w in x[0]], '-> Target (Y):', id2word
            if i ==
10:
break          i +=
1
```

c. Train model

Model building

In [5]:

```
import keras.backend as K from
keras.models import Sequential
from keras.layers import Dense, Embedding, Lambda

cbow = Sequential()
cbow.add(Embedding(input_dim=vocab_size, output_dim=embed_size, input_length=window
cbow.add(Lambda(lambda x: K.mean(x, axis=1), output_shape=(embed_size,)))
cbow.add(Dense(vocab_size, activation='softmax'))
cbow.compile(loss='categorical_crossentropy', optimizer='rmsprop')

print(cbow.summary())
```

```
# from IPython.display import SVG
# from keras.utils.vis_utils import model_to_dot

# SVG(model_to_dot(cbow, show_shapes=True, show_layer_names=False, rankdir='TB').cr
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 4, 100)            7500
 lambda (Lambda)             (None, 100)               0
 dense (Dense)               (None, 75)                7575
=================================================================
Total params: 15,075
Trainable params: 15,075
Non-trainable params: 0
_________________________________________________________________ None
```

In [6]:

```
for epoch in range(1, 6):
    loss = 0.
    i = 0
    for x, y in generate_context_word_pairs(corpus=wids, window_size=window_size, v
        i += 1
        loss += cbow.train_on_batch(x, y)
        if i % 100000 == 0:
            print('Processed {} (context, word) pairs'.format(i))

    print('Epoch:', epoch, '\tLoss:', loss)
    print()
```

```
Epoch: 1        Loss: 433.61818504333496

Epoch: 2        Loss: 428.8695614337921

Epoch: 3        Loss: 425.2637906074524

Epoch: 4        Loss: 421.93233609199524

Epoch: 5        Loss: 419.51635098457336
```

In [7]:

```
weights = cbow.get_weights()[0]
weights = weights[1:]
print(weights.shape)

pd.DataFrame(weights, index=list(id2word.values())[1:]).head()
```

```
(74, 100)
```

Out[7]:

	0	1	2	3	4	5	6	7
deep	-0.034130	0.024219	0.032866	0.053057	-0.015359	0.024081	-0.027761	0.015272
networks	-0.015954	-0.040530	0.016333	0.065731	-0.064257	-0.008575	-0.043316	0.004140
neural	-0.015125	-0.016590	-0.026489	-0.046720	0.038668	-0.012035	-0.045278	0.046965
and	0.029279	0.033890	0.049657	-0.037406	-0.049706	-0.005566	0.040193	0.014699
as	-0.009610	0.026094	-0.016352	0.039663	0.004246	-0.007173	-0.008121	-0.004822

5 rows × 100 columns

d. Output

In [8]:

```
from sklearn.metrics.pairwise import euclidean_distances

distance_matrix = euclidean_distances(weights)
print(distance_matrix.shape)

similar_words = {search_term: [id2word[idx] for idx in distance_matrix[word2id[sear
                   for search_term in ['deep']}

similar_words
```

```
(74, 74)
```

Out[8]:

```
{'deep': ['material', 'based', 'can', 'of',
'reinforcement']}
```

In [ ]:

Assignment 6

1. Name: Harsh Chaudhari
2. Batch: P-10
3. Roll No.: 43215

Problem Statement :

Object detection using Transfer Learning of CNN architectures

Importing the libraries

In [1]:

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
tf.__version__
```

Out[1]:

```
'2.8.0'
```

Preprocessing for dataset

In [2]:

```
 img_generator = tf.keras.preprocessing.image.ImageDataGenerator(#rotation_range=90,
                                                                 brightness_range=((
                                                                   #shear_range=0.2,
                                                                 #zoom_range=0.2,
channel_shift_range
horizontal_flip=Tru
vertical_flip=True,
rescale=1./255,
validation_split=0.
```

[3]:

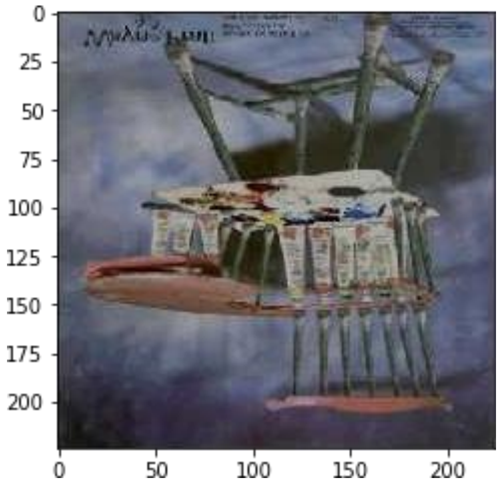
```
root_dir = '101_ObjectCategories'
img_generator_flow_train = img_generator.flow_from_directory(
    directory=root_dir,
    target_size=(224, 224),
    batch_size=32,
    shuffle=True,
    subset="training")

img_generator_flow_valid = img_generator.flow_from_directory(
    directory=root_dir,
    target_size=(224, 224),
    batch_size=32,
    shuffle=True,
    subset="validation")
```

```
Found 6444 images belonging to 102 classes.
Found 2700 images belonging to 102 classes.
```

In
In [4]:

```
imgs, labels = next(iter(img_generator_flow_train))
for img, label in zip(imgs, labels):
    plt.imshow(img)
    plt.show()
```



a. Load in a pretrained model (InceptionV3)

In [5]:

```
base_model = tf.keras.applications.InceptionV3(input_shape=(224,224,3),
                                                include_top=False,
                                               weights = "imagenet"
                                           )
```

b. Freeze parameters (weights) in model's lower convolutional layers

[6]:

```
base_model.trainable = False
```

c. Add custom classifier with several layers of trainable parameters to model

In [7]:

```
model = tf.keras.Sequential([
    base_model,
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(102, activation="softmax")
])
```

In [8]:

In

```
model.summary()
```

```
Model: "sequential"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 inception_v3 (Functional)   (None, 5, 5, 2048)        21802784
 max_pooling2d_4 (MaxPooling  (None, 2, 2, 2048)       0
 2D)
 flatten (Flatten)           (None, 8192)              0
 dense (Dense)               (None, 102)               835686
=================================================================
Total params: 22,638,470
Trainable params: 835,686
Non-trainable params: 21,802,784
_________________________________________________________________
```

d. Train classifier layers on training data available for task

In [9]:

```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate = 0.001),
loss = tf.keras.losses.CategoricalCrossentropy(),                metrics
= [tf.keras.metrics.CategoricalAccuracy()])
```

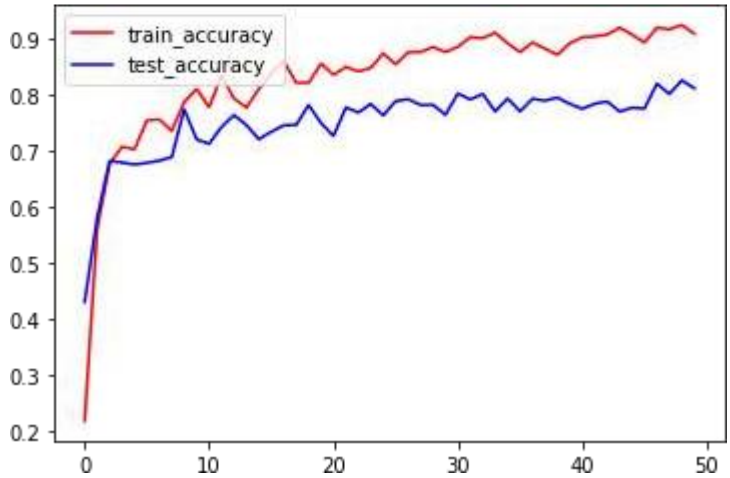
[10]: model.fit(img_generator_flow_train,

```
validation_data=img_generator_flow_valid, steps
```

```
Epoch 1/50
20/20 [==============================] - 299s 15s/step - loss: 9.578
0 - categorical_accuracy: 0.2188 - val_loss: 4.9710 - val_categorica
l_accuracy: 0.4307
Epoch 2/50
20/20 [==============================] - 281s 15s/step - loss: 3.323
5 - categorical_accuracy: 0.5609 - val_loss: 2.7433 - val_categorica
l_accuracy: 0.5800
Epoch 3/50
20/20 [==============================] - 277s 14s/step - loss: 2.033
5 - categorical_accuracy: 0.6766 - val_loss: 2.1982 - val_categorica
l_accuracy: 0.6822
Epoch 4/50
20/20 [==============================] - 280s 15s/step - loss: 2.223
7 - categorical_accuracy: 0.7078 - val_loss: 2.1987 - val_categorica
l_accuracy: 0.6796
Epoch 5/50
20/20 [==============================] - 280s 15s/step - loss: 1.995
8 - categorical_accuracy: 0.7031 - val_loss: 2.3371 - val_categorica
l accuracy: 0 6759
```

In
In [11]:

```
# Visualise train / Valid Accuracy
plt.plot(model.history.history["categorical_accuracy"], c="r", label="train_accurac
plt.plot(model.history.history["val_categorical_accuracy"], c="b", label="test_accu
plt.legend(loc="upper left")
plt.show()
```



e. Fine-tune hyper parameters and unfreeze more layers as needed

In [12]:

```
base_model.trainable = True

model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate = 0.001),
loss = tf.keras.losses.CategoricalCrossentropy(),                metrics
= [tf.keras.metrics.CategoricalAccuracy()])
```

[ ]: model.fit(img_generator_flow_train,

```
validation_data=img_generator_flow_valid, steps
```

```
Epoch 1/50
20/20 [==============================] - 474s 24s/step - loss: 3.9578 -
categorical_accuracy: 0.4359 - val_loss: 126.2205 - val_categorical_
accuracy: 0.0163
Epoch 2/50
20/20 [==============================] - 466s 24s/step - loss: 3.9319 -
categorical_accuracy: 0.2875 - val_loss: 13552.6338 - val_categorica
l_accuracy: 0.0481
Epoch 3/50
20/20 [==============================] - 462s 24s/step - loss: 3.5396 -
categorical_accuracy: 0.3250 - val_loss: 65.9670 - val_categorical_a
ccuracy: 0.0663
Epoch 4/50
20/20 [==============================] - 461s 24s/step - loss: 3.1368 -
categorical_accuracy: 0.3812 - val_loss: 62.4981 - val_categorical_a
ccuracy: 0.0722
Epoch 5/50
```

In

```
20/20 [==============================] - 463s 24s/step - loss: 2.8024 -
categorical_accuracy: 0.4109 - val_loss: 4.9331 - val_categorical_ac
curacy: 0.2341
Epoch 6/50
20/20 [==============================] - 457s 23s/step - loss: 2.5176 -
categorical_accuracy: 0.4328 - val_loss: 4.8107 - val_categorical_ac
curacy: 0.2370
Epoch 7/50
20/20 [==============================] - 459s 23s/step - loss: 2.0832 -
categorical_accuracy: 0.5250 - val_loss: 3.3150 - val_categorical_ac
curacy: 0.4315
Epoch 8/50
20/20 [==============================] - 458s 24s/step - loss: 1.6358 -
categorical_accuracy: 0.6047 - val_loss: 2.4610 - val_categorical_ac
curacy: 0.4459
Epoch 9/50
20/20 [==============================] - 460s 24s/step - loss: 1.6275 -
categorical_accuracy: 0.5953 - val_loss: 2.1677 - val_categorical_ac
curacy: 0.5081
Epoch 10/50
20/20 [==============================] - ETA: 0s - loss: 1.6000 - cate
gorical_accuracy: 0.5938
```

In [ ]: