Sinhgad Technical Education Society's

SINHGAD COLLEGE OF ENGINEERING, Vadgaon 411041

NAAC Accredited with 'A' Grade

Sinhgad Institutes

ACADEMIC YEAR:2022 - 23

DEPARTMENT:INFORMATION TECHNOLOGY

CLASS:B.E. **SEMESTER:I**

SUBJECT: 414447: Lab Practice IV

LAB EXPT.NO	PROBLEMSTATEMENT
1.	Study of Deep learning Packages: Tensorflow, Keras, Theano and PyTorch.Document the distinct features and functionality of the packages.
2.	Implementing Feedforward neural networks with Keras and TensorFlow a. Import the necessary packages b. Load the training and testing data (MNIST/CIFAR10) c. Define the network architecture using Keras d. Train the model using SGD e. Evaluate the network f. Plot the training loss and accuracy
3.	Build the Image classification model by dividing the model into following 4 stages: a. Loading and preprocessing the image data b. Defining the model's architecture c. Training the model d. Estimating the model's performance
4.	Use Autoencoder to implement anomaly detection. Build the model by using: a. Import required libraries b. Upload / access the dataset c. Encoder converts it into latent representation d. Decoder networks convert it back to the original input e. Compile the models with Optimizer, Loss, and Evaluation Metrics
5.	Implement the Continuous Bag of Words (CBOW) Model. Stages can be: a. Data preparation b. Generate training data c. Train model d. Output
6.	Object detection using Transfer Learning of CNN architectures a. Load in a pre-trained CNN model trained on a large dataset b. Freeze parameters (weights) in model's lower convolutional layers c. Add custom classifier with several layers of trainable parameters to model d. Train classifier layers on training data available for task Fine-tune hyper parameters and unfreeze more layers as needed

Assignment No.1

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

Aim: Study and installation of following Deep learning Packages :
i. Tensor Flow
ii. Keras
iii. Theno
iV . PyTorch

Theory : 1) What is Deep learning ?
2)What are various packages in python for supporting Machine Learning libraries and which are mainly used for Deep Learning ?
3)Compare Tensorflow / Keras/Theno and PyTorch on following points(make a table) :
i. Available Functionality
ii. GUI status
iii. Versions.
iv. Features
v. Compatibilty with other enviornments.
vi. Specific Applictaion domains.
4) Enlist the Models Datasets and pretrained models, Libraaries and Extensions , Tools related to Tensorflow also discuss any two casestudies like (PayPal, Intel, Etc.) related to Tensor Flow. [Ref:https://www.tensorflow.org/about]
5) Explain the Keras Ecosystem.(kerastuner,kerasNLP,kerasCV,Autokeras and Modeloptimization.) Also explain following concepts related to keras : 1. Developing sequential Model 2. Training and validation using the inbuilt functions 3. Parameter Optimization. [Ref: https://keras.io/]
6) Explain simple Theano program.
7) Explain PyTorch Tensors . And also explain Uber's Pyro, Tesala Autopilot.[https://pytorch.org/]

Steps/ Algorithm
Installation of Tensorflow On Ubntu:
1. 1. Install the Python Development Environment:
You need to download Python, the PIP package, and a virtual environment. If these packages are already installed, you can skip this step.

You can download and install what is needed by visiting the following links:

https://www.python.org/

https://pip.pypa.io/en/stable/installing/

https://docs.python.org/3/library/venv.html

To install these packages, run the following commands in the terminal:

sudo apt update

sudo apt install python3-dev python3-pip python3-venv

2. Create a Virtual Environment

Navigate to the directory where you want to store your Python 3.0 virtual environment. It can be in your home directory, or any other directory where your user can read and write permissions.

mkdir tensorflow_files

cd tensorflow_files

Now, you are inside the directory. Run the following command to create a virtual environment:

python3 -m venv virtualenv

The command above creates a directory named virtualenv. It contains a copy of the Python binary, the PIP package manager, the standard Python library, and other supporting files.

3. Activate the Virtual Environment

source virtualenv/bin/activate

Once the environment is activated, the virtual environment's bin directory will be added to the beginning of the $PATH variable. Your shell's prompt will alter, and it will show the name of the virtual environment you are currently using, i.e. virtualenv.

4. Update PIP

pip install --upgrade pip

5. 5. Install TensorFlow

The virtual environment is activated, and it's up and running. Now, it's time to install the TensorFlow package.

pip install -- upgrade TensorFlow

Installation of Keras on Ubntu :
Prerequisite : Python version 3.5 or above.

STEP 1: Install and Update Python3 and Pip
Skip this step if you already have Python3 and Pip on your machine**.**

sudo apt install python3 python3.pip

sudo pip3 install —upgrade pip

STEP 2: Upgrade Setuptools

pip3 install —upgrade setuptools

STEP 3: Install TensorFlow

pip3 install tensorflow

Verify the installation was successful by checking the software package information:
pip3 show tensorflow

STEP 4: Install Keras

pip3 install keras
Verify the installation by displaying the package information:

pip3 show keras
[https://phoenixnap.com/kb/how-to-install-keras-on-linux]

Installation of Theano on Ubuntu:
Step 1: First of all, we will install Python3 on our Linux Machine. Use the following command in the terminal to install Python3.

sudo apt-get install python3
Step 2: Now, install the pip module
*s*udo apt install python3-pip
Step 3: Now, install the Theano
Verifying Theano package Installation on Linux using PIP
python3 -m pip show theano
Installation of PyTorch

First, check if you are using python's latest version or not.Because PyGame requires python 3.7 or a higher version

python3 –version

pip3 –version

pip3 install torch==1.8.1+cpu torchvision==0.9.1+cpu torchaudio==0.8.1 -f https://download.pytorch.org/whl/torch_stable.html

[Ref : https://www.geeksforgeeks.org/install-pytorch-on-linux/]

Python Libraries and functions required

1. Tensorflow,keras

numpy : NumPy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices. NumPy stands for Numerical Python. To import numpy use

import numpy as np

pandas: pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language. To import pandas use

import pandas as pd

sklearn : Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python. This library, which is largely written in Python, is built upon NumPy, SciPy and Matplotlib. For importing train_test_ split use

from sklearn.model_selection import train_test_split

2. For TheaonRequirements:
- Python3
- Python3-pip
- NumPy

- SciPy
- BLAS

Sample Code with comments

1. Tensorflow Test program:

```
import tensorflow as tf

print(tf.__version__)

2.1.0

print(tf.reduce_sum(tf.random.normal([1000, 1000])))

tf.Tensor(-505.04108, shape=(), dtype=float32)
```

2. Keras Test Program:

```
from tensorflow import keras
from keras import datasets

#
# Load MNIST data
#
(train_images, train_labels), (test_images, test_labels) = datasets.mnist.load_data()
#
# Check the dataset loaded
#
train_images.shape, test_images.shape
```

3. Theano test program

```
# Python program showing
# addition of two scalars

# Addition of two scalars
import numpy
import theano.tensor as T
from theano import function

# Declaring two variables
x = T.dscalar('x')
y = T.dscalar('y')

# Summing up the two numbers
z = x + y

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

```
f = function([x, y], z)
f(5, 7)
```

4. Test program for PyTorch

```
## The usual imports
import torch
import torch.nn as nn

## print out the pytorch version used
print(torch._version_)
```

Conclusion :

Tensorflow , PyTorch,Keras and Theano all these packages are installed and ready for Deep learning applications . As per application domain and dataset we can choose the appropriate package and build required type of Neural Network.

Output:-

Tensorflow :
2.8.0
tf.Tensor(-1356.2506, shape=(), dtype=float32)

Keras:
((60000, 28, 28), (10000, 28, 28))

Theano:
array(12)

torch:
1.11.0+cpu

Assignment No.2

Title : Implementing Feedforward neural networks

Aim: Implementing Feedforward neural networks with Keras and TensorFlow

a. Import the necessary packages

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

c. Define the network architecture using Keras

d. Train the model using SGD

e. Evaluate the network

f. Plot the training loss and accuracy

Theory : 1) What is Feedforward Neural Network ?
2) How the Feedforward Neural Network Works ?
3) Enlist atleast three Real time scenarios where Feedforward Neural Network is used.
4) Explain the components of Feedforward Neural Network.
5) What is costf unction in Feedforward Neural Network.
6) Define mean square error cost function.
7) What is Loss function in Feedforward Neural Network.
8) What is cross entropy loss.
9) What is kernel concept related to Feedforrward Neural Network.
10) Describe MNIST and CIFAR 10 Dataset.
11) Explain use and parameter setting related to feedforward network implementation for following libraries : SKlearn : i) LabelBinarizer (sklearn.preprocessing) ii) classification_report (sklearn.metrics) and tensorflow.keras : models , layers,optimizers,datasets ,baclend and set to respective values.
12) What is mean by flattening the dataset and why it is needed related to standard neural network implementation .
13) Explain difference between Sigmoid and Softmax activation function.14)
What is significance of optimizer in training model.
15) What is Epochs in fit command in training .

Steps/ Algorithm
1. Dataset link and libraries :
Dataset : MNIST or CIFAR 10 : kaggel.com
You can download dataset from above mentioned website.

Libraries required :

Pandas and Numpy for data manipulation

Tensorflow/Keras for Neural Networks

Scikit-learn library for splitting the data into train-test samples, and for some basic model evaluation

https://pyimagesearch.com/2021/05/06/implementing-feedforward-neural-networks-with-keras-and-tensorflow/

a) Import following libraries from SKlearn : i) LabelBinarizer (sklearn.preprocessing) ii) classification_report (sklearn.metrics) .

b) Import Following libraries from tensorflow.keras : models , layers,optimizers,datasets ,baclend and set to respective values.

c) Grab the MNIST dataset or required dataset.

d) Flatten the dataset.

e) If required do the normalization of data .

f) Convert the labels from integers to vectors.(specially for one hot coding)

g) Decide the Neural Network Architecture : i) Select model (Sequential recommended)

   ii) Activation function (sigmoid recommended) iii) Select the input shape iv) see the weights in the output layer

h) Train the model : i) Select optimizer (SGD recommended) ii) use model that .fit to start training ii) Set Epochs and batch size

i) Call model.predict for class prediction.

j) Plot training and loss accuracy

k) Calculate Precision, Recall, F1-score, Support

l) Repeat for CIFAR dataset.

Conclusion : Should be based on Evaluation model parameters and plots**.**

Input code:

```
import tensorflow as tf
from tensorflow import keras
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import random
get_ipython().run_line_magic("matplotlib","inline")
```

```
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
len(x_train)
len(x_test)
x_train.shape
x_test.shape
x_train[0]
```

```
plt.matshow(x_train[11]) #we can change it by changing the argument
```

```
x_train = x_train/255
x_test = x_test/255
```

```
x_train[11]
```

```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(10, activation='softmax')
])

model.summary()
```

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

```
history=model.fit(x_train, y_train,validation_data=(x_test,y_test),epochs=10)
```

```
test_loss, test_acc=model.evaluate(x_test,y_test)
print("Loss=%.3f" %test_loss)
print("Accuracy=%.3f" %test_acc)
```

```
n=random.randint(0,9999)
plt.imshow(x_test[n])
plt.show()
```

```
predicted_value=model.predict(x_test)
print("Handwritten nuber in the image is= %d" %np.argmax(predicted_value))
```

```
get_ipython().run_line_magic('pinfo2','history.history')
history.history.keys()
```

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```

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

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Training Loss and accuracy')
plt.ylabel('accuracy/Loss')
plt.xlabel('epoch')
plt.legend(['accuracy', 'val_accuracy','loss','val_loss'])
plt.show()
```

```
keras_model_path="/content/sample_data"
model.save(keras_model_path)

restored_keras_model = tf.keras.models.load_model(keras_model_path)
```

Output:-

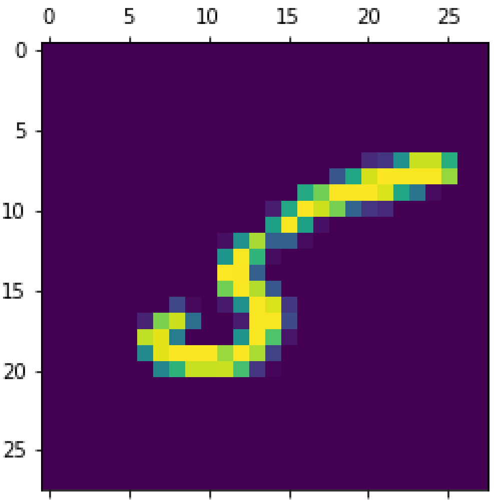
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz 11490434/11490434 [==============================] - 0s 0us/step

60000

10000

(10000, 28, 28)

array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 18, 18, 18, 126, 136, 175, 26, 166, 255, 247, 127, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 30, 36, 94, 154, 170, 253, 253, 253, 253, 253, 225, 172, 253, 242, 195, 64, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 49, 238, 253, 253, 253, 253, 253, 253, 253, 253, 251, 93, 82, 82, 56, 39, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 18, 219, 253, 253, 253, 253, 253, 198, 182, 247, 241, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 80, 156, 107, 253, 253, 205, 11, 0, 43, 154, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 14, 1, 154, 253, 90, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 139, 253, 190, 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, 11, 190, 253, 70, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 35, 241, 225, 160, 108, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 81, 240, 253, 253, 119, 25, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 45, 186, 253, 253, 150, 27, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 16, 93, 252, 253, 187, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 249, 253, 249, 64, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 46, 130, 183, 253, 253, 207, 2, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 39, 148, 229, 253, 253, 253, 250, 182, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 24, 114, 221, 253, 253, 253, 253, 201, 78, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 23, 66, 213, 253, 253, 253, 253, 198, 81, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 18, 171, 219, 253, 253, 253, 253, 195, 80, 9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 55, 172, 226, 253, 253, 253, 253, 244, 133, 11, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 136, 253, 253, 253, 212, 135, 132, 16, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]], dtype=uint8)



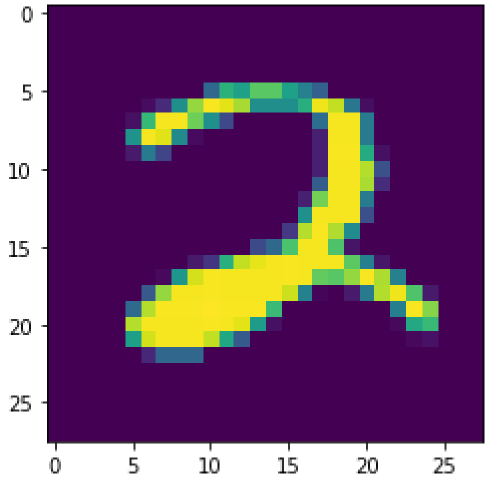
array([[0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.], [0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.], [0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.], [0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.], [0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.], [0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.], [0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.], [0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.

, 0. , 0. , 0. , 0. , 0.12156863, 0.15686275, 0.50588235, 0.91764706, 0.91764706, 0.62352941, 0. , 0.], [0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.26666667, 0.58823529, 0.9372549 , 0.99607843, 0.99215686, 0.99215686, 0.99215686, 0.84313725, 0. , 0.], [0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.61176471, 0.78823529, 0.99607843, 0.99607843, 0.99607843, 0.94509804, 0.58823529, 0.38431373, 0.03137255, 0. , 0. , 0.], [0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.0745098 , 0.60392157, 0.99607843, 0.9254902 , 0.79607843, 0.3254902 , 0.15294118, 0.11764706, 0. , 0. , 0. , 0. , 0. , 0.], [0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.56470588, 0.99215686, 0.56862745, 0.04705882, 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.], [0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.03921569, 0.50588235, 0.87058824, 0.30588235, 0.30980392, 0.03137255, 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.], [0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.5254902 , 0.99215686, 0.65490196, 0.03137255, 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.], [0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 1. , 0.99607843, 0.30588235, 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.], [0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.78823529, 0.99215686, 0.88627451, 0.27058824, 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.], [0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.21568627, 0.02352941, 0. , 0.07058824, 0.50196078, 0.99215686, 0.94509804, 0.16078431, 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.], [0. , 0. , 0. , 0. , 0. , 0. , 0.09803922, 0.80392157, 0.92156863, 0.36078431, 0. , 0. , 0.07843137, 0.99215686, 0.99215686, 0.22745098, 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.], [0. , 0. , 0. , 0. , 0. , 0. , 0.90588235, 0.96078431, 0.42352941, 0. , 0. , 0. , 0.51764706, 0.99215686, 0.7254902 , 0.05490196, 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.], [0. , 0. , 0. , 0. , 0. , 0. , 0.4745098 , 0.96078431, 0.99607843, 0.99607843, 0.99607843, 0.85098039, 0.99607843, 0.8745098 , 0.19607843, 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.], [0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.45490196, 0.64705882, 0.91372549, 0.91372549, 0.91764706, 0.70588235, 0.15294118, 0.01176471, 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.], [0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.], [0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.], [0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.], [0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.], [0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.], [0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.]])

Model: "sequential" ____ Layer (type) Output Shape Param # === flatten (Flatten) (None, 784) 0 dense (Dense) (None, 128) 100480 dense_1 (Dense) (None, 10) 1290 === Total params: 101,770 Trainable params: 101,770 Non-trainable params: 0

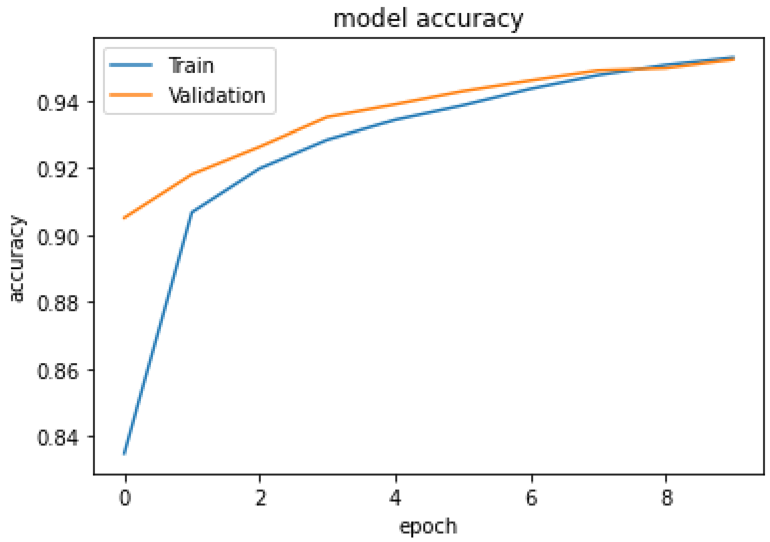
Epoch 1/10 1875/1875 [==============================] - 6s 3ms/step - loss: 0.6578 - accuracy: 0.8348 - val_loss: 0.3575 - val_accuracy: 0.9051 Epoch 2/10 1875/1875 [==============================] - 5s 3ms/step - loss: 0.3340 - accuracy: 0.9067 - val_loss: 0.2884 - val_accuracy: 0.9181 Epoch 3/10 1875/1875 [==============================] - 9s 5ms/step - loss: 0.2846 - accuracy: 0.9198 - val_loss: 0.2567 - val_accuracy: 0.9263 Epoch 4/10 1875/1875 [==============================] - 8s 4ms/step - loss: 0.2545 - accuracy: 0.9284 - val_loss: 0.2334 - val_accuracy: 0.9351 Epoch 5/10 1875/1875 [==============================] - 5s 3ms/step - loss: 0.2328 - accuracy: 0.9343 - val_loss: 0.2164 - val_accuracy: 0.9389 Epoch 6/10 1875/1875 [==============================] - 5s 2ms/step - loss: 0.2155 - accuracy: 0.9386 - val_loss: 0.2025 - val_accuracy: 0.9428 Epoch 7/10 1875/1875 [==============================] - 5s 2ms/step - loss: 0.2006 - accuracy: 0.9435 - val_loss: 0.1908 - val_accuracy: 0.9460 Epoch 8/10 1875/1875 [==============================] - 5s 2ms/step - loss: 0.1874 - accuracy: 0.9476 - val_loss: 0.1815 - val_accuracy: 0.9490 Epoch 9/10 1875/1875 [==============================] - 5s 2ms/step - loss: 0.1760 - accuracy: 0.9506 - val_loss: 0.1741 - val_accuracy: 0.9497 Epoch 10/10 1875/1875 [==============================] - 4s 2ms/step - loss: 0.1662 - accuracy: 0.9529 - val_loss: 0.1632 - val_accuracy: 0.9523

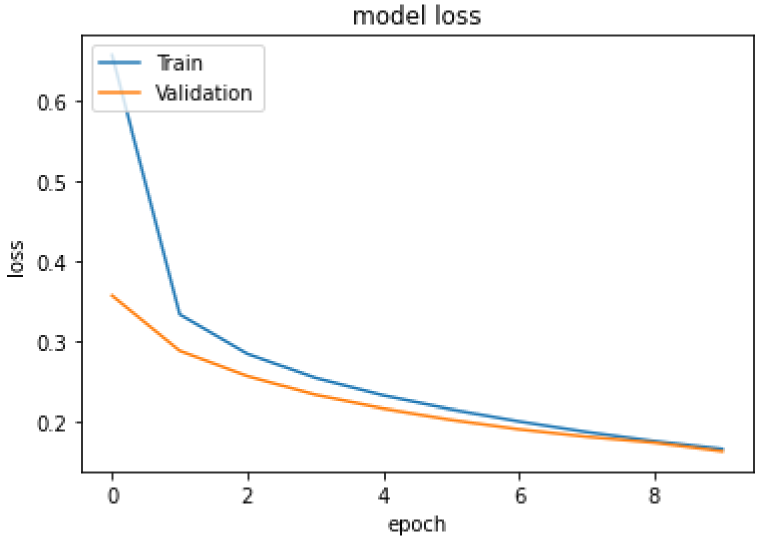
313/313 [==============================] - 1s 2ms/step - loss: 0.1632 - accuracy: 0.9523 Loss=0.163 Accuracy=0.952

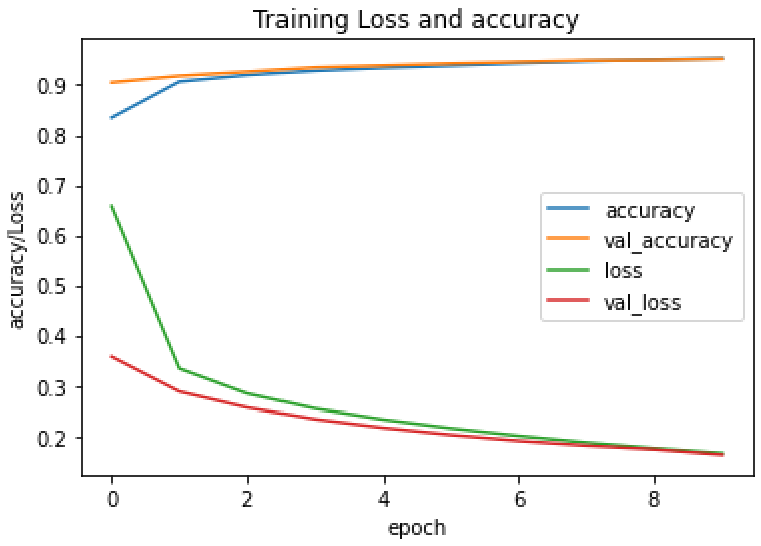


313/313 [==============================] - 1s 2ms/step Handwritten nuber in the image is= 90520

dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])






Training Loss and accuracy
accuracy/Loss
epoch
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
2
4
6
8
accuracy
val_accuracy
loss
val_loss

Assignment No.3

Title : Build the Image classification model

Aim: Build the Image classification model by dividing the model into following 4 stages:

a. Loading and pre-processing the image data

b. Defining the model's architecture

c. Training the model

d. Estimating the model's performance

Theory : 1) What is Image classification problem?
2) Why to use Deep learning for Image classification ? State and compare different Type of Neural Networks used for the Image classification?
3) What is CNN?
4) Explain Convolution operation and Convolution kernel related to Deep learning.
5) Explain how kernel operate on the Input image by taking sample matrix.
6) Explain the types of convolution and convolution layers related to CNN.
7) Explain how the feature extraction is done with convolution layers?
8) Explain

Steps/ Algorithm
1. Choose a dataset of your interest or you can also create your own image dataset (Ref : https://www.kaggle.com/datasets/) Import all necessary files.

(Ref : https://www.analyticsvidhya.com/blog/2021/01/image-classification-using-convolutional-neural-networks-a-step-by-step-guide/)

Libraries and functions required

1. Tensorflow,keras

numpy : NumPy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices. NumPy stands for Numerical Python. To import numpy use

import numpy as np

pandas: pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language. To import pandas use

import pandas as pd

sklearn : Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python. This library, which is largely written in Python, is built upon NumPy, SciPy and Matplotlib. For importing train_test_ split use

2. Prepare Dataset for Training : //Preparing our dataset for training will involve assigning paths and creating categories(labels), resizing our images.

3. Create a Training a Data : // Training is an array that will contain image pixel values and the index at which the image in the CATEGORIES list.

4. Shuffle the Dataset

5. Assigning Labels and Features

6. Normalising X and converting labels to categorical data

7. Split X and Y for use in CNN

8. Define, compile and train the CNN Model

9. Accuracy and Score of model.

Conclusion :

As per the evalution of model write down in line with your output about accuracy and other evaluation parameters.

Input Code:-

```
from google.colab import drive
drive.mount("/content/drive")
```

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


import  matplotlib.image as mping
import matplotlib.pyplot as plt
import seaborn as sns
import cv2



import tensorflow

from keras.preprocessing.image import ImageDataGenerator

%matplotlib inline
```

```
TrainingImagePath="/content/drive/MyDrive/Image /train"
TestImagePath="/content/drive/MyDrive/Image /test"
ValidationImagePath="/content/drive/MyDrive/Image /valid"
```

```
train_datagen = ImageDataGenerator(
   rescale = 1./255,
   shear_range=0.1,
   zoom_range=0.1,
   horizontal_flip=True
)
test_datagen = ImageDataGenerator(rescale=1./255)

training_set = train_datagen.flow_from_directory(
   TrainingImagePath,
   target_size=(128,128),
   batch_size=32,
   class_mode="categorical"
)
```

```
test_set = test_datagen.flow_from_directory(
    TestImagePath,
    target_size = (128,128),
    batch_size=32,
    class_mode="categorical"
)

valid_set = test_datagen.flow_from_directory(
    ValidationImagePath,
    target_size=(128,128),
    batch_size=32,
    class_mode="categorical"
)
```

```
def showImages(class_name):
  random_index = random.choice(list(range(1,49)))
  folder_path = os.path.join(TrainingImagePath, class_name)
  try:
    image_path = os.path.join(folder_path,str(random_index).zfill(3)+".jpg")
    plt.imshow(mping.imread(image_path))
  except:
    image_path = os.path.join(folder_path,str(random_index).zfill(2)+".jpg")
    plt.imshow(mping.imread(image_path))
  plt.title(class_name)
  plt.axis(False)
```

```
plt.figure(figsize = (20,20))
for labels,number in training_set.class_indices.items():
  plt.subplot(6,6,number+1)
  showImages(labels)

test_set.class_indices
```

```
'''################### Creating lookup table for all balls ##############################'''
# class_indices have the numeric tag for each balls
TrainClasses=training_set.class_indices

# Storing the face and the numeric tag for future reference
ResultMap={}
for ballValue,ballName in zip(TrainClasses.values(),TrainClasses.keys()):
    ResultMap[ballValue]=ballName

# Saving the face map for future reference
import pickle
with open(R"E:\Data Sets\Balls Classification\ResultsMap.pkl", 'wb') as f:
    pickle.dump(ResultMap, f, pickle.HIGHEST_PROTOCOL)
```

```
print("Mapping of Face and its ID",ResultMap)

# The number of neurons for the output layer is equal to the number of faces
OutputNeurons=len(ResultMap)
print('\n The Number of output neurons: ', OutputNeurons)
```

```
from keras.models import Sequential
from keras.layers import Convolution2D
from keras.layers import MaxPool2D
from keras.layers import Flatten
from keras.layers import Dense


classifier= Sequential()


classifier.add(Convolution2D(32, kernel_size=(3, 3), strides=(1, 1), input_shape=(128,128,3), activation='relu'))


classifier.add(MaxPool2D(pool_size=(2,2)))

"
classifier.add(Convolution2D(64, kernel_size=(3, 3), strides=(1, 1), activation='relu'))

classifier.add(MaxPool2D(pool_size=(2,2)))


classifier.add(Flatten())

classifier.add(Dense(256, activation='relu'))

classifier.add(Dense(OutputNeurons, activation='softmax'))

classifier.compile(loss='categorical_crossentropy', optimizer = 'rmsprop', metrics=["accuracy"])
```

```
classifier.summary()
```

```
import time
# Measuring the time taken by the model to train
StartTime=time.time()


# Starting the model training
model_history=classifier.fit_generator(
                                    training_set,
                                    steps_per_epoch=len(training_set),
```

```
                    epochs=20,
                     validation_data=valid_set,
                     validation_steps=len(valid_set),
                     verbose=1)

EndTime=time.time()
print("############### Total Time Taken: ", round((EndTime-StartTime)/60), 'Minutes #############')
```

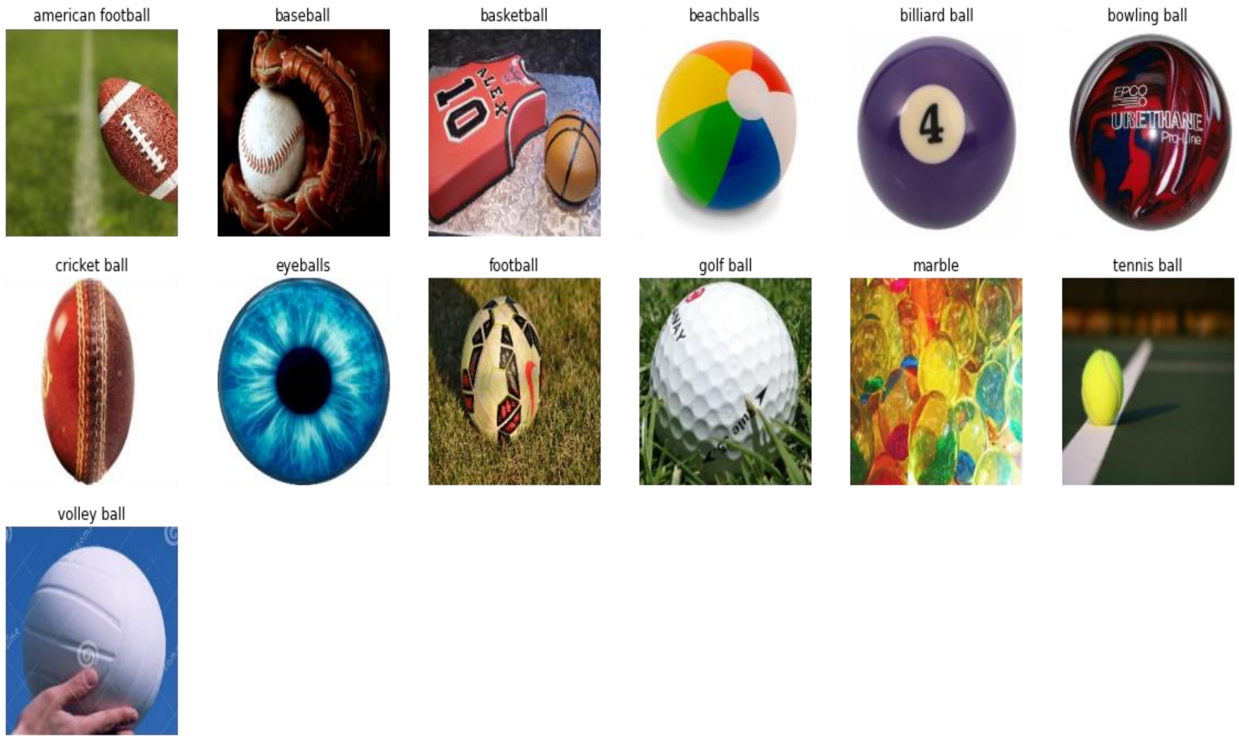
```
accuracy = model_history.history['accuracy']
val_accuracy  = model_history.history['val_accuracy']

loss = model_history.history['loss']
val_loss = model_history.history['val_loss']
```

```
plt.figure(figsize=(15,10))

plt.subplot(2, 2, 1)
plt.plot(accuracy, label = "Training accuracy")
plt.plot(val_accuracy, label="Validation accuracy")
plt.legend()
plt.title("Training vs validation accuracy")


plt.subplot(2,2,2)
plt.plot(loss, label = "Training loss")
plt.plot(val_loss, label="Validation loss")
plt.legend()
plt.title("Training vs validation loss")

plt.show()
```

Output:-

Mounted at /content/drive

Found 650 images belonging to 13 classes.
Found 65 images belonging to 13 classes.
Found 65 images belonging to 13 classes.



{'american football': 0, 'baseball': 1, 'basketball': 2, 'beachballs': 3, 'billiard ball': 4, 'bowling ball': 5, 'cricket ball': 6, 'eyeballs': 7, 'football': 8, 'golf ball': 9, 'marble': 10, 'tennis ball': 11, 'volley ball': 12}

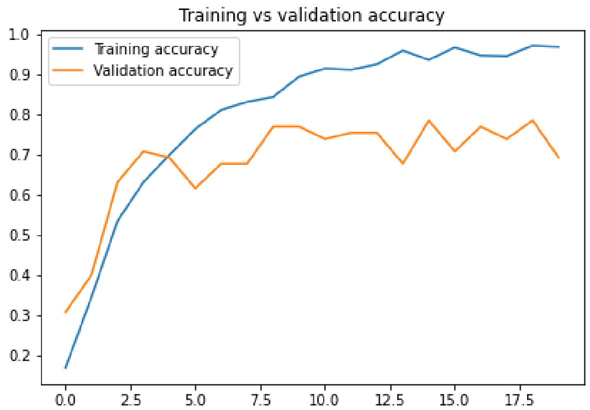
Mapping of Face and its ID {0: 'american football', 1: 'baseball', 2: 'basketball', 3: 'beachballs', 4: 'billiard ball', 5: 'bowling ball', 6: 'cricket ball', 7: 'eyeballs', 8: 'football', 9: 'golf ball', 10: 'marble', 11: 'tennis ball', 12: 'volley ball'}
The Number of output neurons: 13

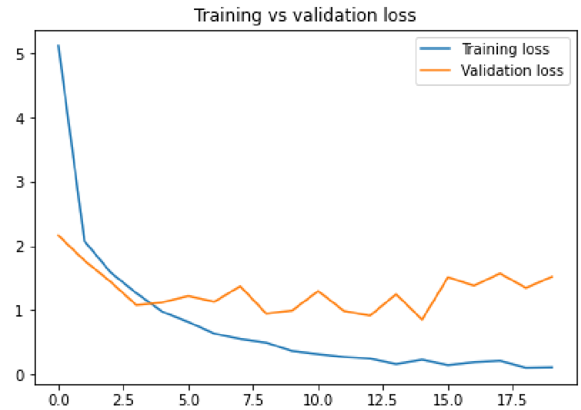
Model: "sequential_2" ____ Layer (type) Output Shape Param # === conv2d_4 (Conv2D) (None, 126, 126, 32) 896 max_pooling2d_4 (MaxPooling (None, 63, 63, 32) 0 2D) conv2d_5 (Conv2D) (None, 61, 61, 64) 18496 max_pooling2d_5 (MaxPooling (None, 30, 30, 64) 0 2D) flatten_2 (Flatten) (None, 57600) 0 dense_3 (Dense) (None, 256) 14745856 dense_4 (Dense) (None, 13) 3341 === Total params: 14,768,589 Trainable params: 14,768,589 Non-trainable params: 0

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generators.
 if sys.path[0] == '':

```
Epoch 1/20
21/21 [==============================] - 21s 973ms/step - loss: 5.1236 - accuracy: 0.1692 - val_loss: 2.1639 - val_accuracy: 0.3077
Epoch 2/20
21/21 [==============================] - 20s 952ms/step - loss: 2.0769 - accuracy: 0.3446 - val_loss: 1.7718 - val_accuracy: 0.4000
Epoch 3/20
21/21 [==============================] - 21s 979ms/step - loss: 1.5903 - accuracy: 0.5338 - val_loss: 1.4442 - val_accuracy: 0.6308
Epoch 4/20
21/21 [==============================] - 20s 952ms/step - loss: 1.2643 - accuracy: 0.6308 - val_loss: 1.0776 - val_accuracy: 0.7077
Epoch 5/20
21/21 [==============================] - 22s 1s/step - loss: 0.9746 - accuracy: 0.6985 - val_loss: 1.1191 - val_accuracy: 0.6923
Epoch 6/20
21/21 [==============================] - 20s 951ms/step - loss: 0.8146 - accuracy: 0.7631 - val_loss: 1.2202 - val_accuracy: 0.6154
Epoch 7/20
21/21 [==============================] - 20s 956ms/step - loss: 0.6327 - accuracy: 0.8108 - val_loss: 1.1284 - val_accuracy: 0.6769
Epoch 8/20
21/21 [==============================] - 20s 960ms/step - loss: 0.5487 - accuracy: 0.8308 - val_loss: 1.3727 - val_accuracy: 0.6769
Epoch 9/20
21/21 [==============================] - 20s 954ms/step - loss: 0.4907 - accuracy: 0.8431 - val_loss: 0.9446 - val_accuracy: 0.7692
Epoch 10/20
21/21 [==============================] - 20s 958ms/step - loss: 0.3610 - accuracy: 0.8938 - val_loss: 0.9923 - val_accuracy: 0.7692
Epoch 11/20
21/21 [==============================] - 22s 1s/step - loss: 0.3118 - accuracy: 0.9138 - val_loss: 1.2967 - val_accuracy: 0.7385
Epoch 12/20
21/21 [==============================] - 20s 960ms/step - loss: 0.2688 - accuracy: 0.9108 - val_loss: 0.9840 - val_accuracy: 0.7538
Epoch 13/20
21/21 [==============================] - 21s 965ms/step - loss: 0.2431 - accuracy: 0.9246 - val_loss: 0.9137 - val_accuracy: 0.7538
Epoch 14/20
21/21 [==============================] - 20s 957ms/step - loss: 0.1579 - accuracy: 0.9585 - val_loss: 1.2483 - val_accuracy: 0.6769
Epoch 15/20
21/21 [==============================] - 20s 962ms/step - loss: 0.2256 - accuracy: 0.9354 - val_loss: 0.8512 - val_accuracy: 0.7846
Epoch 16/20
21/21 [==============================] - 20s 958ms/step - loss: 0.1422 - accuracy: 0.9662 - val_loss: 1.5121 - val_accuracy: 0.7077
Epoch 17/20
21/21 [==============================] - 20s 964ms/step - loss: 0.1866 - accuracy: 0.9462 - val_loss: 1.3806 - val_accuracy: 0.7692
```

Epoch 18/20
21/21 [==============================] - 22s 1s/step - loss: 0.2086 - accuracy: 0.9446 - val_loss: 1.5715 - val_accuracy: 0.7385
Epoch 19/20
21/21 [==============================] - 20s 956ms/step - loss: 0.1014 - accuracy: 0.9708 - val_loss: 1.3456 - val_accuracy: 0.7846
Epoch 20/20
21/21 [==============================] - 20s 961ms/step - loss: 0.1087 - accuracy: 0.9677 - val_loss: 1.5201 - val_accuracy: 0.6923
############### Total Time Taken: 9 Minutes #############





Assignment No.4

Title : ECG Anomaly detection using Autoencoders

Aim: Use Autoencoder to implement anomaly detection. Build the model by using:
a. Import required libraries
b. Upload / access the dataset
c. Encoder converts it into latent representation
d. Decoder networks convert it back to the original input
e. Compile the models with Optimizer, Loss, and Evaluation Metrics

Theory : 1) What is Anomaly Detectection ?
2) What are Autoencoders in Deep learning ?
3) Enlist different applications with Autoencoders in DL.
4) Enlist different types of anomaly detection Algorithms.
5)What is difference between Anomaly detection and Novelty Detection.
6) Explain different blocks and working of Autoencoders.
7) What is reconstruction and Reconstruction errors .
8) What is Minmaxscaler from sklearn.
8) Explain . train_test_split from sklearn.
9) What is anomaly scores.
10) Explain tensorfloe dataset.
11) Describe the ECG Dataset.
12) Explain keras Optimizers
13) Explain keras layers dense and dropouts
14) Explain keras losses and meansquarelogarthmicerror
15) Explain Relu activation function

Steps/ Algorithm
1. Dataset link and libraries :
Dataset : http://storage.googleapis.com/download.tensorflow.org/data/ecg.csv
Libraries required :

Pandas and Numpy for data manipulation

Tensorflow/Keras for Neural Networks

Scikit-learn library for splitting the data into train-test samples, and for some basic model evaluation

For Model building and evaluation following libraries:

sklearn.metrics import accuracy_score

tensorflow.keras.optimizers import Adam

sklearn.preprocessing import MinMaxScaler

tensorflow.keras import Model, Sequential

tensorflow.keras.layers import Dense, Dropout

tensorflow.keras.losses import MeanSquaredLogarithmicError

Ref:https://www.analyticsvidhya.com/blog/2021/05/anomaly-detection-using-autoencoders-a-walk-through-in-python/

a) Import following libraries from SKlearn : i) MinMaxscaler (sklearn.preprocessing) ii) Accuracy(sklearn.metrics) . iii) train_test_split (model_selection)

b) Import Following libraries from tensorflow.keras : models , layers,optimizers,datasets , and set to respective values.

c) Grab to ECG.csv required dataset

d) Find shape of dataset

e) Use train_test_split from sklearn to build model (e.g. train_test_split(

features, target, test_size=0.2, stratify=target)

f) Take usecase Novelty detection hence select training data set as Target class is 1 i.e. Normal class

g) Scale the data using MinMaxScaler.

h) Create Autoencoder Subclass by extending model class from keras.

i) Select parameters as i)Encoder : 4 layers ii) Decoder : 4 layers iii) Activation Function : Relu iv) Model : sequential.

j) Configure model with following parametrs : epoch = 20 , batch size =512 and compile with Mean Squared Logarithmic loss and Adam optimizer.

```
e.g. model = AutoEncoder(output_units=x_train_scaled.shape[1])

# configurations of model

model.compile(loss='msle', metrics=['mse'], optimizer='adam')


history = model.fit(

    x_train_scaled,

    x_train_scaled,

    epochs=20,

    batch_size=512,

    validation_data=(x_test_scaled, x_test_scaled)
```

k) Plot loss,Val_loss, Epochs and msle loss

l) Find threshold for anomaly and do predictions :

```
e.g. : find_threshold(model, x_train_scaled):

  reconstructions = model.predict(x_train_scaled)

  # provides losses of individual instances
```

```
reconstruction_errors = tf.keras.losses.msle(reconstructions, x_train_scaled)
# threshold for anomaly scores
threshold = np.mean(reconstruction_errors.numpy()) \
    + np.std(reconstruction_errors.numpy())
return threshold
```

m) Get accuracy score

Conclusion : Should be based on Evaluation model parameters and plots.

Input Code:

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import tensorflow as tf

from sklearn.metrics import accuracy_score, precision_score, recall_score
from sklearn.model_selection import train_test_split
from tensorflow.keras import layers, losses
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.models import Model
```

```
(x_train, _), (x_test, _) = fashion_mnist.load_data()

x_train = x_train/255.
x_test = x_test/255.

print(x_train.shape)
print(x_test.shape)
```

```
latent_dim = 64
class Autoencoder(Model):
  def __init__(self, latent_dim):
    super(Autoencoder, self).__init__()
    self.latent_dim = latent_dim
    self.encoder = tf.keras.Sequential([
      layers.Flatten(),
      layers.Dense(latent_dim, activation='relu'),
    ])
    self.decoder = tf.keras.Sequential([
      layers.Dense(784, activation='sigmoid'),
      layers.Reshape((28, 28))
    ])

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

autoencoder = Autoencoder(latent_dim)
```

```
autoencoder.compile(optimizer='adam', loss=losses.MeanSquaredError())
```

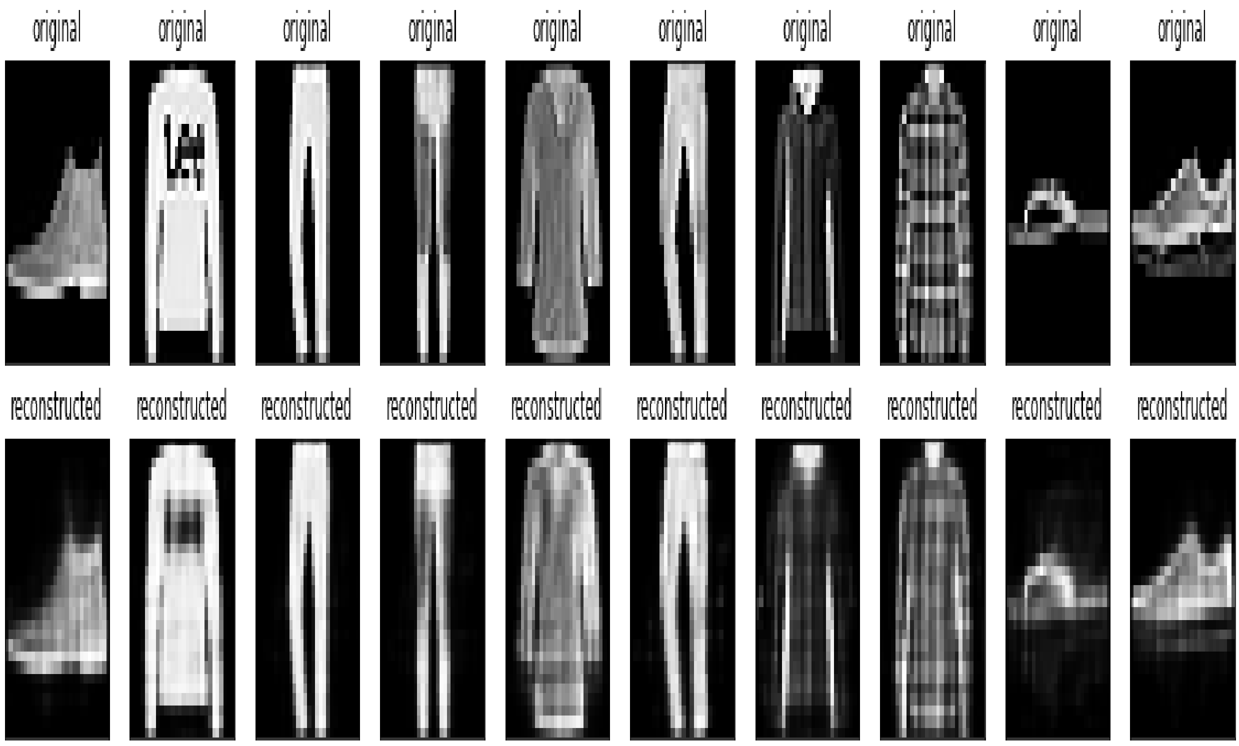
```
autoencoder.fit(x_train, x_train,
                epochs=10,
                shuffle=True,
                validation_data=(x_test, x_test))
```

```
encoded_imgs = autoencoder.encoder(x_test).numpy()
decoded_imgs = autoencoder.decoder(encoded_imgs).numpy()

n = 10
plt.figure(figsize=(20, 4))
for i in range(n):
  # display original
  ax = plt.subplot(2, n, i + 1)
  plt.imshow(x_test[i])
  plt.title("original")
  plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)

  # display reconstruction
  ax = plt.subplot(2, n, i + 1 + n)
  plt.imshow(decoded_imgs[i])
  plt.title("reconstructed")
  plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)
plt.show()
```

Output:-

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

Epoch 1/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0238 - val_loss: 0.0132
Epoch 2/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0116 - val_loss: 0.0106
Epoch 3/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0100 - val_loss: 0.0097
Epoch 4/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0094 - val_loss: 0.0093
Epoch 5/10
1875/1875 [==============================] - 5s 2ms/step - loss: 0.0091 - val_loss: 0.0092
Epoch 6/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0090 - val_loss: 0.0090
Epoch 7/10
1875/1875 [==============================] - 5s 2ms/step - loss: 0.0089 - val_loss: 0.0090
Epoch 8/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0088 - val_loss: 0.0089
Epoch 9/10
1875/1875 [==============================] - 5s 2ms/step - loss: 0.0088 - val_loss: 0.0089
Epoch 10/10
1875/1875 [==============================] - 5s 2ms/step - loss: 0.0087 - val_loss: 0.0088
<keras.callbacks.History at 0x7fe60a33e910>



Assignment No.5

Title : Implement the Continuous Bag of Words (CBOW) Model.

Aim: Implement the Continuous Bag of Words (CBOW) Model. Stages can be:
a. Data preparation
b. Generate training data
c. Train model
d. Output

Theory : 1) What is NLP ?
2) What is Word embedding related to NLP ?
3) Explain Word2Vec techniques.
4) Enlist applications of Word embedding in NLP.
5) Explain CBOW architecture.
6) What will be input to CBOW model and Output to CBW model.
7) What is Tokenizer .
8) Explain window size parameter in detail for CBOW model.
9) Explain Embedding and Lmbda layer from keras
10) What is yield()

Steps/ Algorithm
1. Dataset link and libraries :
Create any English 5 to 10 sententece paragraph as input

Import following data from keras :

keras.models import Sequential

keras.layers import Dense, Embedding, Lambda

keras.utils import np_utils

keras.preprocessing import sequence

keras.preprocessing.text import Tokenizer

Import Gensim for NLP operations : requirements :

Gensim runs on Linux, Windows and Mac OS X, and should run on any other platform that supports Python 3.6+ and NumPy. Gensim depends on the following software: Python, tested with versions 3.6, 3.7 and 3.8. NumPy for number crunching.

Ref: https://analyticsindiamag.com/the-continuous-bag-of-words-cbow-model-in-nlp-hands-on-implementation-with-codes/

a) Import following libraries gemsim and numpy set i.e. text file created . It should be preprocessed.

b) Tokenize the every word from the paragraph . You can call in built tokenizer present in Gensim

c) Fit the data to tokenizer

d) Find total no of words and total no of sentences.

e) Generate the pairs of Context words and target words :

e.g. cbow_model(data, window_size, total_vocab):

```
total_length = window_size*2
for text in data:
    text_len = len(text)
    for idx, word in enumerate(text):
        context_word = []
        target  = []
        begin = idx - window_size
        end = idx + window_size + 1
        context_word.append([text[i] for i in range(begin, end) if 0 <= i < text_len and i 
!= idx])
        target.append(word)
        contextual = sequence.pad_sequences(context_word, total_length=total_length)
        final_target = np_utils.to_categorical(target, total_vocab)
        yield(contextual, final_target)
```

f) Create Neural Network model with following parameters . Model type : sequential

Layers : Dense , Lambda , embedding. Compile Options : (loss='categorical_crossentropy', optimizer='adam')

g) Create vector file of some word for testing

e.g.:dimensions=100

```
vect_file = open('/content/gdrive/My Drive/vectors.txt' ,'w')
vect_file.write('{} {}\n'.format(total_vocab,dimensions)
```

h) Assign weights to your trained model

e.g. weights = model.get_weights()[0]

```
for text, i in vectorize.word_index.items():
    final_vec = ' '.join(map(str, list(weights[i, :])))
    vect_file.write('{} {}\n'.format(text, final_vec)
Close()
```

i) Use the vectors created in Gemsim :

e.g. cbow_output =
gensim.models.KeyedVectors.load_word2vec_format('/content/gdrive/My Drive/vectors.txt', binary=False)

j) choose the word to get similar type of words:

cbow_output.most_similar(positive=['Your word'])

Conclusion: Explain how Neural network is useful for CBOW text analysis.

Input code:-

```
import numpy as np
import keras.backend as K
from keras.models import Sequential
from keras.layers import Dense,Embedding,Lambda
from keras.utils import np_utils
from keras.preprocessing import sequence
from keras.preprocessing.text import Tokenizer
import gensim

data = open("/content/corona.txt","r")
covid_data= [text for text in data if text.count("")>=2]
vectorize=Tokenizer()
vectorize.fit_on_texts(covid_data)
covid_data=vectorize.texts_to_sequences(covid_data)
total_vocab=sum(len(s) for s in covid_data)
word_count=len(vectorize.word_index)+1
window_size=2


def cbow_model(data,windows_size, total_vocab):
  total_length=window_size*2
  for text in data:
    text_len=len(text)
    for idx, word in enumerate(text):
      context_word=[]
      target=[]
      begin=idx-window_size
      end=idx+window_size+1
      context_word.append([text[i] for i in range(begin,end) if 0<- i< text_len and i!=idx])
      target.append(word)
      contextual = sequence.pad_sequences(context_word, total_length=total_length)
      final_target=np_utils.to_categorical(target, total_vocab)
      yield(contextual, final_target)


model=Sequential()
model.add(Embedding(input_dim=total_vocab,output_dim=100,input_length=window_size*2))
model.add(Lambda(lambda x:K.mean(x,axis=1), output_shape=(100,)))
model.add(Dense(total_vocab, activation="softmax"))
model.compile(loss="categorical_crossentropy", optimizer="adam")
for i in range(10):
  cost=0
  for x, y in cbow_model(data,window_size, total_vocab):
    cost+=model.train_on_batch(contextual, final_target)
  print(i, cost)

dimensions = 100
vect_file=open("/content/drive/MyDrive/vector.txt", "w")
vect_file.write('{} {}\n'.format(total_vocab, dimensions))
```

```
weight=model.get_weights()[0]
for text, i in vectorize.word_index.items():
  final_vec="".join(map(str, list(weight[i,:])))
  vect_file.write('{}{}\n'.format(text, final_vec))
vect_file.close()

cbow_output=gensim.models.KeyedVectors.load_word2vec_format("/content/drive/MyDrive/vector.txt", binary=False)
cbow_output.most_similar(positive=["virus"])
```

OUTPUT:-

0 0 1 0 2 0 3 0 4 0 5 0 6 0 7 0 8 0 9 0

8

Assignment No.6

Title : Object detection using Transfer Learning of CNN architectures

Aim: Object detection using Transfer Learning of CNN architectures
a. Load in a pre-trained CNN model trained on a large dataset
b. Freeze parameters (weights) in model's lower convolutional layers
c. Add custom classifier with several layers of trainable parameters to model
d. Train classifier layers on training data available for task
e. Fine-tune hyper parameters and unfreeze more layers as needed

Theory : 1) What is **Transfer learning** ?
2) What are pretrained Neural Network models ?
3) Explain Pytorch library in short.
4) What are advantages of Transfer learning.
5) What are applications of Transfer learning.
6) Explain Caltech 101 images dataset.
7)Explain Imagenet dataset **.**
8) list down basic steps for transfer learning.
9) What is Data augmentation?
10) How and why Data augmentation is done related to transfer learning?
11) Why preprocessing is needed on inputdata in Transfer learning.
12) What is PyTorch Transforms module.Explain following commands w.r.t it :
Compose([
    RandomResizedCrop(size=256, scale=(0.8, 1.0)),
    RandomRotation(degrees=15),
    ColorJitter(),
    RandomHorizontalFlip(),
    CenterCrop(size=224), # Image net standards
  .ToTensor(),
    Normalize
13) Explain the Validation Transforms steps with Pytorch Transforms .
14) Explain VGG-16 model from Pytorch

Steps/ Algorithm

1. Dataset link and libraries :

https://data.caltech.edu/records/mzrjq-6wc02
separate the data into training, validation, and testing sets with a 50%, 25%, 25% split and then structured the directories as follows:
/datadir
/train
/class1
/class2
.
.
/valid
/class1
/class2

.
.
/test
/class1
/class2
.
Libraries required :
PyTorch
torchvision import transforms
torchvision import d

atasets

torch.utils.data import DataLoader

torchvision import models

torch.nn as nn

torch import optim

Ref: https://towardsdatascience.com/transfer-learning-with-convolutional-neural-networks-in-pytorch-dd09190245ce

m) Prepare the dataset in splitting in three directories Train , alidation and test with 50 25 25

n) Do pre-processing on data with transform from Pytorch

Training dataset transformation as follows :

```
transforms.Compose([
    transforms.RandomResizedCrop(size=256, scale=(0.8, 1.0)),
    transforms.RandomRotation(degrees=15),
    transforms.ColorJitter(),
    transforms.RandomHorizontalFlip(),
    transforms.CenterCrop(size=224), # Image net standards
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406],
                         [0.229, 0.224, 0.225]) # Imagenet standards
```

Validation Dataset transform as follows :

```
transforms.Compose([
    transforms.Resize(size=256),
    transforms.CenterCrop(size=224),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
```

o) Create Datasets and Loaders :

```
data = {
  'train':(Our name given to train data set dir created )
  datasets.ImageFolder(root=traindir, transform=image_transforms['train']),
  'valid':
  datasets.ImageFolder(root=validdir, transform=image_transforms['valid']),
}
 dataloaders = {
  'train': DataLoader(data['train'], batch_size=batch_size, shuffle=True),
  'val': DataLoader(data['valid'], batch_size=batch_size, shuffle=True)
}
```

p) Load Pretrain Model : from torchvision import models

model = model.vgg16(pretrained=True)

q) Freez all the Models Weight

```
for param in model.parameters():
  param.requires_grad = False
```

r) Add our own custom classifier with following parameters :

Fully connected with ReLU activation, shape = (n_inputs, 256)

Dropout with 40% chance of dropping

Fully connected with log softmax output, shape = (256, n_classes)

```
import torch.nn as nn
# Add on classifier
model.classifier[6] = nn.Sequential(
                      nn.Linear(n_inputs, 256),
                      nn.ReLU(),
                      nn.Dropout(0.4),
                      nn.Linear(256, n_classes),
                      nn.LogSoftmax(dim=1))
```

s) Only train the sixth layer of classifier keep remaining layers off .

```
Sequential(
 (0): Linear(in_features=25088, out_features=4096, bias=True)
 (1): ReLU(inplace)
 (2): Dropout(p=0.5)
```

```
(3): Linear(in_features=4096, out_features=4096, bias=True)
(4): ReLU(inplace)
(5): Dropout(p=0.5)
(6): Sequential(
  (0): Linear(in_features=4096, out_features=256, bias=True)
  (1): ReLU()
  (2): Dropout(p=0.4)
  (3): Linear(in_features=256, out_features=100, bias=True)
  (4): LogSoftmax()
 )
)
```

t) Initialize the loss and optimizer

```
criteration = nn.NLLLoss()
optimizer = optim.Adam(model.parameters())
```

u) Train the model using Pytorch

```
for epoch in range(n_epochs):
for data, targets in trainloader:
  # Generate predictions
  out = model(data)
  # Calculate loss
  loss = criterion(out, targets)
  # Backpropagation
  loss.backward()
  # Update model parameters
  optimizer.step()
```

v) Perform Early stopping

w) Draw performance curve

x) Calculate Accuracy

```
pred = torch.max(ps, dim=1)
equals = pred == targets
# Calculate accuracy
accuracy = torch.mean(equals)
```

Conclusion: Explains how Transfer training increases the accuracy of Object detection

https://www.google.com/url?q=https://towardsdatascience.com/transfer-learning-with-convolutional-neural-networks-in-pytorch-dd0

Input Code & Output :

Making a pre-trained data model

```
from IPython.core.interactiveshell
import InteractiveShell
import seaborn as sns
import torch
# PyTorch
from torchvision import transforms, datasets, models
from torch import optim, cuda
from torch.utils.data import DataLoader, sampler
import torch.nn as nn
import warningswarnings.filterwarnings('ignore', category=FutureWarning)
# Data science tools
\import numpy as np
import pandas as pd
import os
# Image manipulations
from PIL import Image
# Useful for examining network
from torchsummary import summary
# Timing utility
from timeit import default_timer as timer
# Visualizations
import matplotlib.pyplot as plt
%matplotlib inlineplt.rcParams['font.size'] = 14
# Printing out all outputs
InteractiveShell.ast_node_interactivity = 'all'

# Location of data
datadir = '/home/wjk68/'traindir = datadir + 'train/'validdir = datadir + 'valid/'testdir = datadir + 'test/'
save_file_name = 'vgg16-transfer-4.pt'checkpoint_path = 'vgg16-transfer-4.pth'
# Change to fit hardware
batch_size = 128
# Whether to train on a gpu
train_on_gpu = cuda.is_available()
print(f'Train on gpu: {train_on_gpu}')
# Number of gpus
if train_on_gpu:
    gpu_count = cuda.device_count()
    print(f'{gpu_count} gpus detected.')
    if gpu_count > 1:
        multi_gpu = True
    else:
        multi_gpu = False
```

Ouput:-

Train on gpu: True
2 gpus detected.

Code:-

```
# Empty listscategories = []img_categories = []n_train = []n_valid = []n_test = []hs = []ws = []
# Iterate through each category
for d in os.listdir(traindir):
    categories.append(d)

    # Number of each image
    train_imgs = os.listdir(traindir + d)
    valid_imgs = os.listdir(validdir + d)
    test_imgs = os.listdir(testdir + d)
    n_train.append(len(train_imgs))
    n_valid.append(len(valid_imgs))
    n_test.append(len(test_imgs))

    # Find stats for train images
    for i in train_imgs:
        img_categories.append(d)
        img = Image.open(traindir + d + '/' + i)
        img_array = np.array(img)
        # Shape
        hs.append(img_array.shape[0])
        ws.append(img_array.shape[1])
# Dataframe of categories
cat_df = pd.DataFrame({'category': categories,
                       'n_train': n_train,
                       'n_valid': n_valid, 'n_test': n_test}).\
    sort_values('category')
# Dataframe of training images
image_df = pd.DataFrame({
    'category': img_categories,
    'height': hs,
    'width': ws})
cat_df.sort_values('n_train', ascending=False, inplace=True)
cat_df.head()
cat_df.tail()
```

Output:-

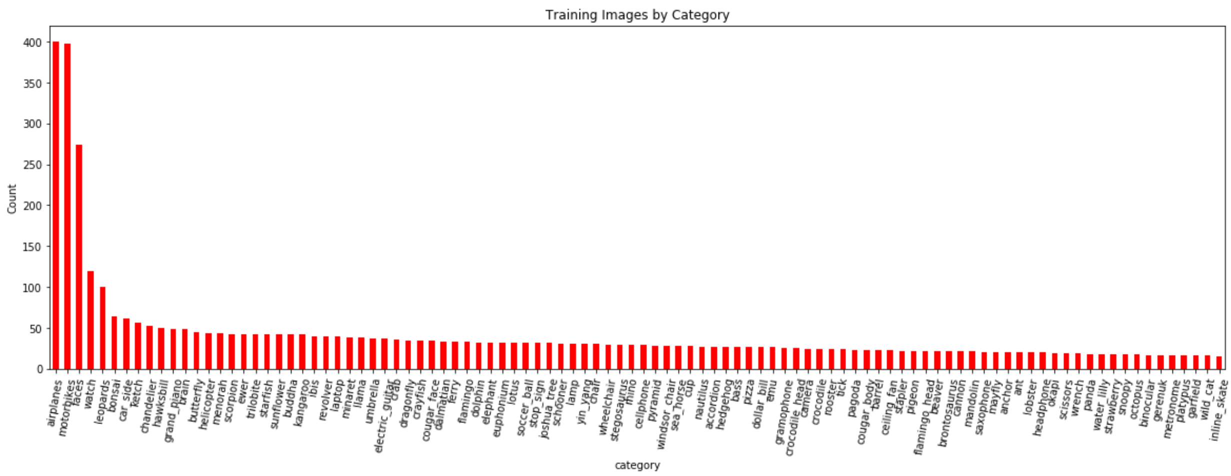
	category	n_train	n_valid	n_test
4	airplanes	400	200	200
2	motorbikes	398	200	200
0	faces	274	138	109
93	watch	119	60	60
1	leopards	100	50	50

Out[3]:

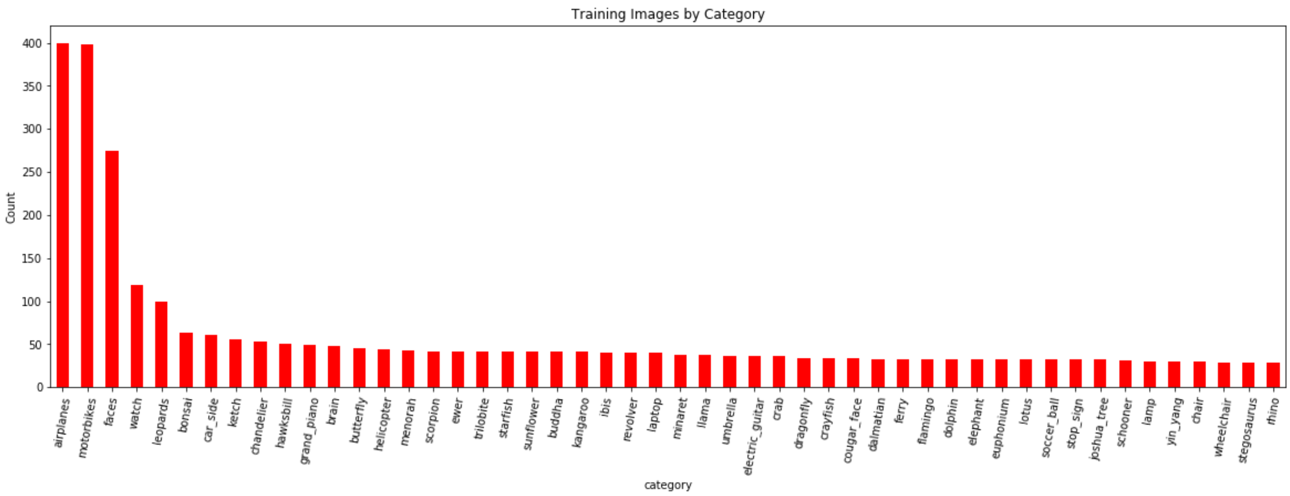
	category	n_train	n_valid	n_test
63	metronome	16	8	8
72	platypus	16	9	9
42	garfield	16	9	9
96	wild_cat	16	9	9
51	inline_skate	15	8	8

Distribution of Image

```
cat_df.set_index('category')['n_train'].plot.bar(color='r', figsize=(20, 6))
plt.xticks(rotation=80)
plt.ylabel('Count')
plt.title('Training Images by Category')
```




```
# Only top 50 categories
cat_df.set_index('category').iloc[:50]['n_train'].plot.bar(color='r', figsize=(20,6))
plt.xticks(rotation=80)
plt.ylabel('Count')
plt.title('Training Images by Category')
```

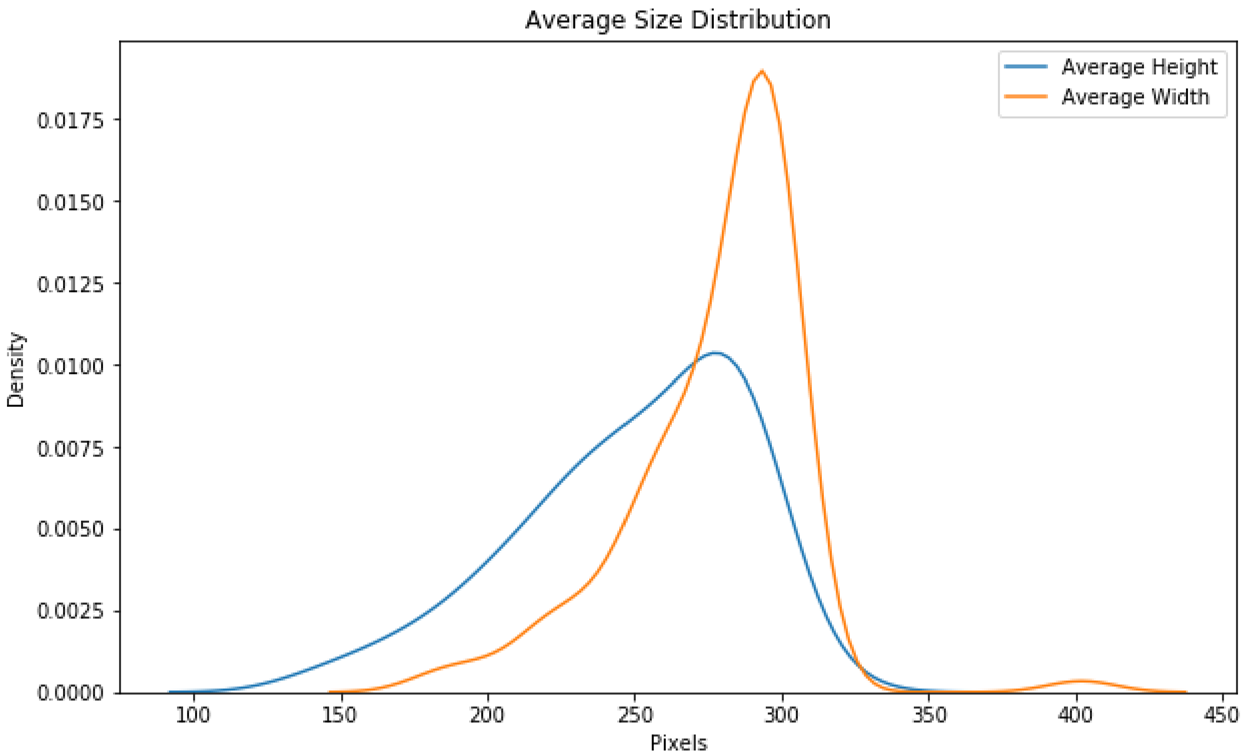


```
img_dsc = image_df.groupby('category').describe()
img_dsc.head()
```

Output:-

heightwidthcountmeanstdmin25%50%75%maxcountmeanstdmin25%50%75%maxcategoryaccordion27.0263.85185235.769243199.0233.00265.0300.00300.027.0280.33333330.849511209.0266.5300.0300.0300.0airplanes400.0158.45500030.847397101.0141.00154.0170.25494.0400.0402.1375008.804965356.0396.0401.0406.0457.0anchor20.0241.00000038.608698170.0219.75236.0264.50300.020.0291.30000022.209766230.0300.0300.0300.0300.0ant20.0211.95000047.137509103.0177.00203.0236.75300.020.0298.6000006.029751273.0300.0300.0300.0300.0barrel23.0284.08695736.455344188.0300.00300.0300.00300.023.0241.86956541.592508168.0205.5235.0283.0300.0

```
plt.figure(figsize=(10, 6))
sns.kdeplot(img_dsc['height']['mean'], label='AverageHeight')
sns.kdeplot(img_dsc['width']['mean'], label='Average Width')
plt.xlabel('Pixels')
plt.ylabel('Density')
plt.title('Average Size Distribution')
```




```
def imshow(image):
    """Display image"""
    plt.figure(figsize=(6, 6))
    plt.imshow(image)
    plt.axis('off')
    plt.show()

# Example
imagex = Image.open(traindir + 'ewer/image_0002.jpg')np.array(x).shapeimshow(x)
```

Pretrained data:-

model_options = pd.read_csv('models.csv')model_options

Output:-
modelparams0AlexNet611008401DenseNet79788562Inception3271612643SqueezeNet12484244alexnet611008405densenet12179788566densenet161286810007densenet169141494808densenet201200139289inception_v32716126410resnet1014454916011resnet1526019280812resnet181168951213resnet342179767214resnet502555703215squeezenet1_0124842416squeezenet1_1123549617vgg1113286333618vgg11_bn13286884019vgg1313304784820vgg13_bn13305373621vgg1613835754422vgg16_bn13836599223vgg1914366724024vgg19_bn143678248

model = models.vgg16(pretrained=**True**)model

```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
```

```
  (18): ReLU(inplace)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): ReLU(inplace)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (27): ReLU(inplace)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace)
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
# Freeze early layersfor param in model.parameters():
    param.requires_grad = False

n_inputs = model.classifier[6].in_features
# Add on classifier
model.classifier[6] = nn.Sequential(nn.Linear(n_inputs, 256), nn.ReLU(),nn.Dropout(0.4),nn.Linear(256, n_classes),
nn.LogSoftmax(dim=1))
model.classifier

Sequential(
  (0): Linear(in_features=25088, out_features=4096, bias=True)
  (1): ReLU(inplace)
  (2): Dropout(p=0.5)
  (3): Linear(in_features=4096, out_features=4096, bias=True)
  (4): ReLU(inplace)
  (5): Dropout(p=0.5)
  (6): Sequential(
    (0): Linear(in_features=4096, out_features=256, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.4)
    (3): Linear(in_features=256, out_features=100, bias=True)
    (4): LogSoftmax()
  )
)
total_params = sum(p.numel() for p in model.parameters())
print(f'{total_params:,} total parameters.')
total_trainable_params = sum(p.numel() for p in model.parameters() if
p.requires_grad)print(f'{total_trainable_params:,} training parameters.')

135,335,076 total parameters.
1,074,532 training parameters.

if train_on_gpu:
    model = model.to('cuda')
if multi_gpu:
    model = nn.DataParallel(model)
```

```
def get_pretrained_model(model_name):
    """Retrieve a pre-trained model from torchvision
    Params    -------        model_name (str): name of the model (currently only accepts vgg16 and resnet50)
    Return    --------        model (PyTorch model): cnn
    """

    if model_name == 'vgg16':
        model = models.vgg16(pretrained=True)

        # Freeze early layers
        for param in model.parameters():
            param.requires_grad = False
        n_inputs = model.classifier[6].in_features

        # Add on classifier
        model.classifier[6] = nn.Sequential(
            nn.Linear(n_inputs, 256), nn.ReLU(), nn.Dropout(0.2),
            nn.Linear(256, n_classes), nn.LogSoftmax(dim=1))

    elif model_name == 'resnet50':
        model = models.resnet50(pretrained=True)

        for param in model.parameters():
            param.requires_grad = False

        n_inputs = model.fc.in_features
        model.fc = nn.Sequential(
            nn.Linear(n_inputs, 256), nn.ReLU(), nn.Dropout(0.2),
            nn.Linear(256, n_classes), nn.LogSoftmax(dim=1))


    # Move to gpu and parallelize
    if train_on_gpu:
        model = model.to('cuda')

    if multi_gpu:
        model = nn.DataParallel(model)

      return model

model = get_pretrained_model('vgg16')
if multi_gpu:
    summary(
        model.module,
        input_size=(3, 224, 224),
        batch_size=batch_size,device='cuda')
else:
    summary(
        model, input_size=(3, 224, 224), batch_size=batch_size, device='cuda')
```

Layer (type)	Output Shape	Param #
Conv2d-1	[128, 64, 224, 224]	1,792
ReLU-2	[128, 64, 224, 224]	0
Conv2d-3	[128, 64, 224, 224]	36,928
ReLU-4	[128, 64, 224, 224]	0
MaxPool2d-5	[128, 64, 112, 112]	0
Conv2d-6	[128, 128, 112, 112]	73,856
ReLU-7	[128, 128, 112, 112]	0
Conv2d-8	[128, 128, 112, 112]	147,584
ReLU-9	[128, 128, 112, 112]	0
MaxPool2d-10	[128, 128, 56, 56]	0
Conv2d-11	[128, 256, 56, 56]	295,168
ReLU-12	[128, 256, 56, 56]	0
Conv2d-13	[128, 256, 56, 56]	590,080
ReLU-14	[128, 256, 56, 56]	0
Conv2d-15	[128, 256, 56, 56]	590,080
ReLU-16	[128, 256, 56, 56]	0
MaxPool2d-17	[128, 256, 28, 28]	0
Conv2d-18	[128, 512, 28, 28]	1,180,160
ReLU-19	[128, 512, 28, 28]	0
Conv2d-20	[128, 512, 28, 28]	2,359,808
ReLU-21	[128, 512, 28, 28]	0
Conv2d-22	[128, 512, 28, 28]	2,359,808
ReLU-23	[128, 512, 28, 28]	0
MaxPool2d-24	[128, 512, 14, 14]	0
Conv2d-25	[128, 512, 14, 14]	2,359,808
ReLU-26	[128, 512, 14, 14]	0
Conv2d-27	[128, 512, 14, 14]	2,359,808
ReLU-28	[128, 512, 14, 14]	0
Conv2d-29	[128, 512, 14, 14]	2,359,808
ReLU-30	[128, 512, 14, 14]	0
MaxPool2d-31	[128, 512, 7, 7]	0
Linear-32	[128, 4096]	102,764,544
ReLU-33	[128, 4096]	0
Dropout-34	[128, 4096]	0
Linear-35	[128, 4096]	16,781,312
ReLU-36	[128, 4096]	0
Dropout-37	[128, 4096]	0
Linear-38	[128, 256]	1,048,832
ReLU-39	[128, 256]	0
Dropout-40	[128, 256]	0
Linear-41	[128, 100]	25,700
LogSoftmax-42	[128, 100]	0

Total params: 135,335,076
Trainable params: 1,074,532
Non-trainable params: 134,260,544

Input size (MB): 73.50
Forward/backward pass size (MB): 27979.45
Params size (MB): 516.26
Estimated Total Size (MB): 28569.21

```
if multi_gpu:
    print(model.module.classifier[6])else:
    print(model.classifier[6])
```

```
Sequential(
  (0): Linear(in_features=4096, out_features=256, bias=True)
  (1): ReLU()
  (2): Dropout(p=0.2)
  (3): Linear(in_features=256, out_features=100, bias=True)
  (4): LogSoftmax()
)
```

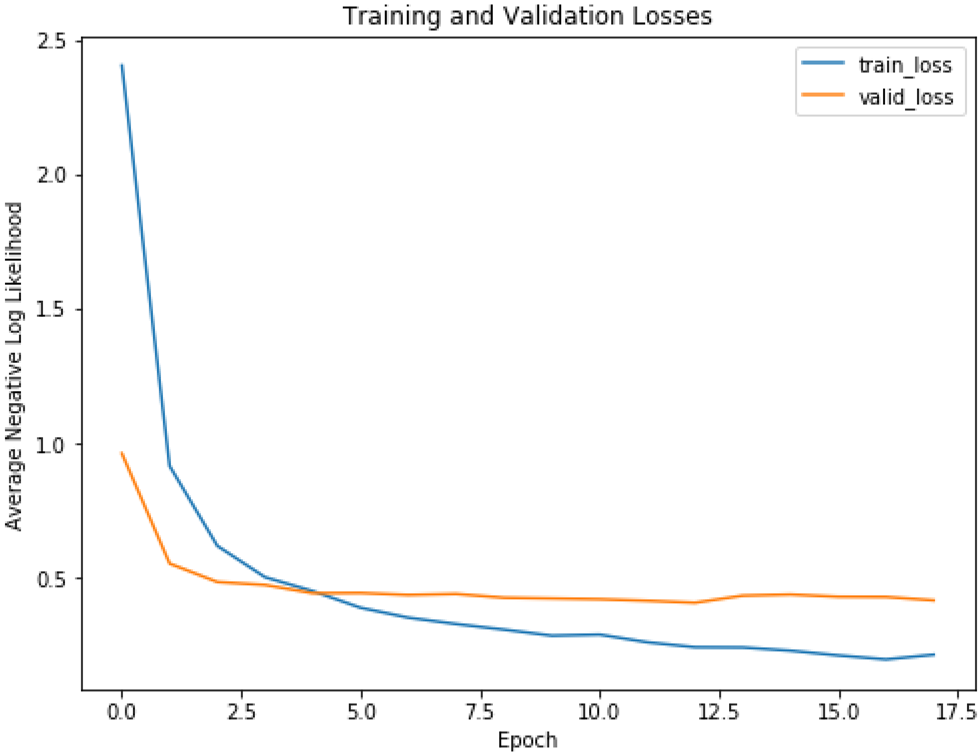
```
model.class_to_idx = data['train'].class_to_idxmodel.idx_to_class = {idx: class_
for class_, idx in model.class_to_idx.items()}
list(model.idx_to_class.items())[:10]
```

```
[(0, 'accordion'),
 (1, 'airplanes'),
 (2, 'anchor'),
 (3, 'ant'),
 (4, 'barrel'),
 (5, 'bass'),
 (6, 'beaver'),
 (7, 'binocular'),
 (8, 'bonsai'),
 (9, 'brain')]
```

```
criterion = nn.NLLLoss()
optimizer = optim.Adam(model.parameters())

for p in optimizer.param_groups[0]['params']:
    if p.requires_grad:
        print(p.shape)
```

```
torch.Size([256, 4096])
torch.Size([256])
torch.Size([100, 256])
torch.Size([100])
```

```
plt.figure(figsize=(8, 6))
for c in ['train_loss', 'valid_loss']:
    plt.plot(history[c], label=c)
    plt.legend()
    plt.xlabel('Epoch')
    plt.ylabel('Average Negative Log Likelihood')
    plt.title('Training and Validation Losses')
```




```
plt.figure(figsize=(8, 6))
for c in ['train_acc', 'valid_acc']:
    plt.plot(100 * history[c],label=c)
    plt.legend()
    plt.xlabel('Epoch')
    plt.ylabel('Average Accuracy')
    plt.title('Training and Validation Accuracy')
```