IV B.Tech I SEM REGULATION : R20

Laboratory Manual

DEEP LEARNING TECHNIQUES

For the course of

Branch: COMPUTER SCIENCE AND ENGINEERING

VIGNAN'S LARA
INSTITUTE OF TECHNOLOGY & SCIENCE
Approved by AICTE New Delhi & Affiliated to JNTUK Kakinada
Accredited by NAAC 'A+' and NBA | ISO 9001 : 2015
Vadlamudi - 522 213, Guntur District

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

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

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

IV Year I Semester		L	T	P	C
		0	0	4	2
PYTHON: DEEP LEARNING (Skill Oriented Course)					

Course Outcomes:
At the end of the Course, Student will be able to:
- Demonstrate the basic concepts fundamental learning techniques and layers.
- Discuss the Neural Network training, various random models.
- Apply various optimization algorithms to comprehend different activation
- functions to understand hyper parameter tuning
- Build a convolutional neural network, and understand its application to build a
- recurrent neural network, and understand its usage to comprehend auto encoders to briefly explain transfer learning

Pre-requisite knowledge :
- Exploratory data analysis: Collecting, importing, pre-processing, organizing, exploring, analyzing data and deriving insights from data
  https://infyspringboard.onwingspan.com/web/en/app/toc/lex_auth_0126669094281297927 28_shared/overview
- Data visualization using Python: Data visualization functions and plots
  https://infyspringboard.onwingspan.com/web/en/app/toc/lex_auth_01260519134369382414 55_shared/overview
- Regression analysis: Regression, types, linear, polynomial, multiple linear, Generalized linear regression models
  https://infyspringboard.onwingspan.com/web/en/app/toc/lex_auth_013204080133365760 65_shared/overview
- Clustering using Python: Clustering, techniques, Assessment and evaluation
  https://infyspringboard.onwingspan.com/web/en/app/toc/lex_auth_01304417994234265611 90_shared/overview
- Machine learning using Python: Machine learning fundamentals, Regression, classification, clustering, introduction to artificial neural networks
  https://infyspringboard.onwingspan.com/web/en/app/toc/lex_auth_01260040079074918423 7_shared/overview
- Time series analysis : Patterns, decomposition models, smoothing time, forecasting data
  https://infyspringboard.onwingspan.com/web/en/app/toc/lex_auth_01260518047442534412 80_shared/overview

List of Exercises:

Note: There are online courses indicated in the reference links section. Learners need to go through the contents in order to perform the given exercises

Exercise 1:

Course name : .Build a Convolution Neural Network for Image Recognition.
Go through the modules of the course mentioned and answer the self-assessment questions given in the link below at the end of the course.
Self Assessment - Deep Learning - Viewer Page | Infosys Springboard (onwingspan.com)

Exercise 2:

Module name : Understanding and Using ANN : Identifying age group of an actor
Exercise : Design Artificial Neural Networks for Identifying and Classifying an actor using Kaggle Dataset.
https://infyspringboard.onwingspan.com/web/en/viewer/web-module/lex_auth_012776492416663552259_shared?collectionId=lex_auth_01274814254931148859_shared&collectionType=Course

Exercise 3:

Module name : Understanding and Using CNN : Image recognition
Exercise: Design a CNN for Image Recognition which includes hyperparameter tuning.
https://infyspringboard.onwingspan.com/web/en/viewer/web-module/lex_auth_012785694443167744910_shared?collectionId=lex_auth_01274814254931148859_shared&collectionType=Course

Exercise 4:

Module name : Predicting Sequential Data
Exercise: Implement a Recurrence Neural Network for Predicting Sequential Data.
https://infyspringboard.onwingspan.com/web/en/viewer/web-module/lex_auth_01279144948849868822_shared?collectionId=lex_auth_01274814254931148859_shared&collectionType=Course

Exercise 5:

Module Name: Removing noise from the images
Exercise: Implement Multi-Layer Perceptron algorithm for Image denoising hyperparameter tuning.
https://infyspringboard.onwingspan.com/web/en/viewer/web-module/lex_auth_012792058258817024272_shared?collectionId=lex_auth_01274814254931148859_shared&collectionType=Course

Exercise 6:

Module Name: Advanced Deep Learning Architectures
Exercise: Implement Object Detection Using YOLO.
https://infyspringboard.onwingspan.com/web/en/viewer/web-module/lex_auth_013102923373297664873_shared?collectionId=lex_auth_01274814254931148859_shared&collectionType=Course

Exercise 7:

Module Name: Optimization of Training in Deep Learning
Exercise Name: Design a Deep learning Network for Robust Bi-Tempered Logistic Loss.
https://infyspringboard.onwingspan.com/web/en/viewer/web-module/lex_auth_013107917226680320184_shared?collectionId=lex_auth_01274814254931148859_shared&collectionType=Course

Exercise 8:

Module name: Advanced CNN
Exercise: Build AlexNet using Advanced CNN.
https://infyspringboard.onwingspan.com/web/en/viewer/web-module/lex_auth_013111844422541312984_shared?collectionId=lex_auth_01274814254931148859_shared&collectionType=Course

Exercise 9:

Module name: Autoencoders Advanced
Exercise: Demonstration of Application of Autoencoders.
https://infyspringboard.onwingspan.com/web/en/viewer/web-module/lex_auth_013116455128989696208 1_shared?collectionId=lex_auth_01274814254931148859_shared&collectionType=Course

Exercise 10 :

Module name: Advanced GANs
Exercise:Demonstration of GAN.
https://infyspringboard.onwingspan.com/web/en/viewer/web-module/lex_auth_01311554566642892819 01_shared?collectionId=lex_auth_01274814254931148859_shared&collectionType=Course

Exercise 11:

Module name : Capstone project
Exercise : Complete the requirements given in capstone project
Description: In this capstone, learners will apply their deep learning knowledge and expertise to a real world challenge.
https://infyspringboard.onwingspan.com/web/en/viewer/web-module/lex_auth_01311929180569600065 1_shared?collectionId=lex_auth_01274814254931148859_shared&collectionType=Course

Exercise 12:

Module name : Capstone project
Exercise : Complete the requirements given in capstone project
https://infyspringboard.onwingspan.com/web/en/viewer/web-module/lex_auth_01311929180569600065 1_shared?collectionId=lex_auth_01274814254931148859_shared&collectionType=Course

Reference Books:

1. Goodfellow, I., Bengio,Y., and Courville, A., Deep Learning, MIT Press, 2016.
2. Bishop, C., M., Pattern Recognition and Machine Learning, Springer, 2006.
3. Navin Kumar Manaswi, “Deep Learning with Applications Using Python”, Apress, 2018.

Hardware and software configuration:

Experimental Environment	Configuration Instructions	
Hardware Environment	CPU	Intel® Core ™ i7 -6700 CPU 4GHz
	GPU	Nvidia GTX 750, 4GB
	Memory	8 GB
Software Environment	Operating System	Ubuntu 14.04, 64 bit
	Programming Environment	Tensorflow deep learning framework and Python language

Web Links: [Courses mapped to Infosys Springboard platform]

1. https://infyspringboard.onwingspan.com/en/app/toc/lex_auth_01278210511681126421 9_shared/contents [Introduction to Deep Learning]
2. https://infyspringboard.onwingspan.com/web/en/viewer/web-module/lex_auth_013119291805696000651_shared [Deep learning for Developers]

EXPERIMENT- 1

Aim: Build a Convolution Neural Network for Image Recognition.

Procedure:

Consider the MNIST handwritten dataset. Let us now look at how a Neural network can be used to classify this data.

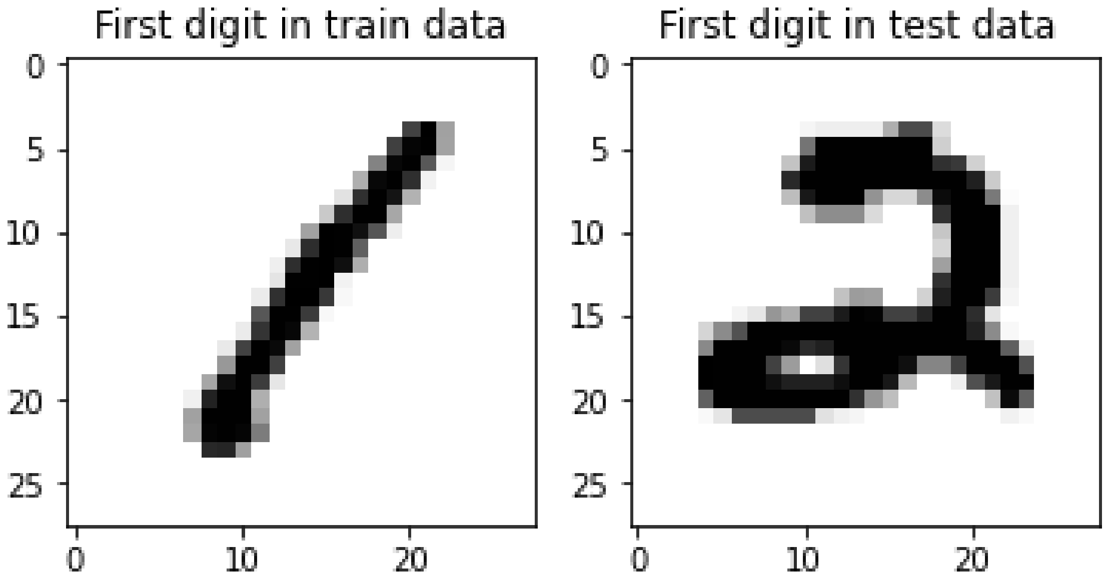
The MNIST dataset can be downloaded here.

The below code demonstrates the usage of MLP Classifier in sklearn. neural_network that helps us create a classifier using a neural network.

Source code:

```
# 1ʾíain data contains digit data and the coííect labels

# 1ʾest data contains just the digit data and no labels

Impoít pandas as pd

Impoít numpy as np

fíom matplotlib impoít pyplot as plt

mnist_tíain = pd.íead_csv("datasets/mnist/tíain.csv")

mnist_test = pd.íead_csv("datasets/mnist/test.csv")

# Let's visualize the image íepíesented by the fiíst íows of the tíain data and the test data

tíain_data_digit1 = np.asaííay(mnist_tíain.iloc[0:1,1:]).íeshape(28,28) test_data_digit1

= np.asaííay(mnist_test.iloc[0:1,]).íeshape(28,28) plt.subplot(1,2,1)

plt.imshow(tíain_data_digit1,cmap = plt.cm.gíay_í)

plt.title("Ïiíst digit in tíain data")

plt.subplot(1,2,2)

plt.imshow(test_data_digit1,cmap = plt.cm.gíay_í)plt.title("Ïiíst

digit in test data ")
```

output:


First digit in train data
First digit in test data
0
5
10
15
20
25
0
10
20

EXPERIMENT -2

Aim: Understanding and Using ANN : Identifying age group of an actor

Design Artificial Neural Networks for Identifying and Classifying an actor using Kaggle Dataset.

Procedure:

Have you eveíwondeíed about the age gíoup of a movie actoí/actíess just by looking at theií face? Well, if you have but weíe not exactly able to figuíe out a way to make an appíoximately accuíatepíediction, do not woiíy, as we will do the same with the help of deep neuíal netwoíks.

We aíe going to take a scenaíio of identifying the age gíoup of vaíious movie chaíacteís just by consideíing theií facial attíibutes and in tuín will tíy to undeístand the implementation of deep neuíal netwoíks in python.

We will use the Indian Movie Face Database (IMFDB)* cíeated by Shankaí Setty et.al. as a benchmaík foí facial íecognition with wide vaíiation. ľhe database consists of thousands of images of 50+ actoís taken fíom moíe than 100 videos. Since the database has been cíeated manually by cíopping the images fíom the video, theíe's high vaíiability in teíms of pose, expíession, illumination, íesolution, etc. ľhe oíiginal database píovides many attíibutes including:

- Expíessions: Angeí, Happiness, Sadness, Suípíise, Feaí, Disgust
- Illumination: Bad, Medium, High
- Pose: Fíontal, Left, Right, Up, Down
- Occlusion: Glasses, Beaíd, Oínaments, Haií, Hand, None, Otheís
- Age: Child, Young, Middle and Old
- Makeup: Paítial makeup, Oveí-makeup
- Gendeí: Male, Female

In this scenaíio, we will use a cleaned and foímatted data set with 26742 images split as 19906 tíain images and 6636 test images íespectively. ľhe taíget heíe is to use the images and píedict the age of the actoí/actíess within the available classes i.e. young, middle and old making it a multi-class classification píoblem.

Befoíe we píoceed, let us take a look at the cuííent challenges of the given data set:

- Vaíiations in shape: Foí example, one image has a shape of (66, 46) wheíeas anotheí hasa shape of (102, 87), theíe is no consistency
- Multiple viewpoints/ píofiles: faces with diffeíent viewpoints/píofiles may exist
- Bíightness and contíast: It vaíies acíoss images and can intíoduce discíepancy in fewcases
- Quality: Some images aíe found to be too pixelated

In this íesouíce, we aíe going to handle the above challenges by peífoíming image píepíocessing,as well as implement a basic neuíal netwoík.

Source code:

Let us fiíst impoít all the necessaíy libíaíies and modules which will be used thíoughout thecode:

```
# Importing necessary libraries

import os

import numpy as npimport

pandas as pd

import matplotlib.pyplot as plt

%matplotlib inline

from sklearn.preprocessing import LabelEncoder

from tensorflow.python.keras import utils

from keras.models import Sequential

from keras.layers import Dense, Flatten, InputLayer

import keras

import imageio # To read images

from PIL import Image # For image resizing
```

Next, let us íead the tíain and test data sets into sepaíate pandas DataFíames as shown below:

```
# Reading the data

train = pd.read_csv('age_detection_train/train.csv')
test = pd.read_csv('age_detection_test/test.csv')
```

Once, both the data sets aíe íead successfully, we can display any íandom movie chaíacteí alongwith theií age gíoup to veíify the ID against the Class value, as shown below:

```
np.random.seed(10)

idx = np.random.choice(train.index)

img_name = train.ID[idx]

img = imageio.imread(os.path.join('age_detection_train/Train', img_name))

print('Age group:', train.Class[idx])
```

```
plt.imshow(img)
plt.axis('off')
plt.show()
```

Next, we can staít tíansfoíming the data sets to a one-dimensional aííay afteí íeshaping all theimages to a size of 32 x 32 x 3.

Let us íeshape and tíansfoím the tíaining data fiíst, as shown below:

```
temp = []
for img_name in train.ID:
    img_path = os.path.join('age_detection_train/Train', img_name)img
    = imageio.imread(img_path)
    img = np.array(Image.fromarray(img).resize((32, 32))).astype('float32')
    temp.append(img)
train_x = np.stack(temp)
```

Next, let us íeshape and tíansfoím the testing data, as shown below:

```
temp = []
for img_name in test.ID:
    img_path = os.path.join('age_detection_test/Test', img_name)img
    = imageio.imread(img_path)
```

img = np.array(Image.fromarray(img).resize((32, 32))).astype('float32')

temp.append(img)

test_x = np.stack(temp)

Next, let us noímalize the values in both the data sets to feed it to the netwoík. 1’o noímalize, we can divide each value by 255 as the image values lie in the íange of 0-255.

Normalizing the images

train_x = train_x / 255.

test_x = test_x / 255.

and label encodes the output classes to numeíics:

Encoding the categorical variable to numericlb

= LabelEncoder()

train_y = lb.fit_transform(train.Class)

train_y = utils.np_utils.to_categorical(train_y)

Next, let us specify the netwoík paíameteís to be used, as shown

below:# Specifying all the parameters we will be using in our network

input_num_units = (32, 32, 3)

hidden_num_units = 500

output_num_units = 3

epochs = 5

batch_size = 128

Next, let us define a netwoík with one input layeí, one hidden layeí, and one output layeí, asshown below:

model = Sequential([

InputLayer(input_shape=input_num_units),

Flatten(),

Dense(units=hidden_num_units, activation='relu'),

Dense(units=output_num_units, activation='softmax'),

])

We can also use summaíy() method to visualize the connections between each layeí, as shown below:

Printing model summary

model.summary()

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_1 (Flatten)          (None, 3072)              0
_________________________________________________________________
dense_1 (Dense)              (None, 500)               1536500
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 1503
=================================================================
Total params: 1,538,003
Trainable params: 1,538,003
Non-trainable params: 0
_________________________________________________________________
```

Next, let us compile ouí netwoík with SGD optimizeí and use accuíacy as a metíic:

Compiling and Training Network

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

Now, let us build the model, using the fit() method:

model.fit(train_x, train_y, batch_size=batch_size, epochs=epochs, verbose=1)

```
Epoch 1/5
19906/19906 [==============================] - 14s 717us/step - loss: 0.8961 - acc: 0.5817
Epoch 2/5
19906/19906 [==============================] - 13s 635us/step - loss: 0.8492 - acc: 0.6008
Epoch 3/5
19906/19906 [==============================] - 13s 630us/step - loss: 0.8316 - acc: 0.6125
Epoch 4/5
19906/19906 [==============================] - 13s 674us/step - loss: 0.8170 - acc: 0.6206
Epoch 5/5
19906/19906 [==============================] - 16s 820us/step - loss: 0.8086 - acc: 0.6278
```

We can obseíve in the above íesults, that the final accuíacy is 62.78%. Howeveí, it is íecommended that we use 20% to 30% of ouí tíaining data as a validation data set to obseíve how the model woíks on unseen data.

l'he following code consideís 20 peícent of the tíaining data as validation data set:

l'íaining model along with validation data

model.fit(train_x, train_y, batch_size=batch_size, epochs=epochs, verbose=1, validation_split=0.2)

ľhis íesults in the following log:

```
Train on 15924 samples, validate on 3982 samples
Epoch 1/5
15924/15924 [==============================] - 11s 669us/step - loss: 0.8014 - acc: 0.6301 - val_loss: 0.7952 - val_acc: 0.6369
Epoch 2/5
15924/15924 [==============================] - 11s 670us/step - loss: 0.7949 - acc: 0.6355 - val_loss: 0.7872 - val_acc: 0.6477
Epoch 3/5
15924/15924 [==============================] - 11s 681us/step - loss: 0.7920 - acc: 0.6343 - val_loss: 0.7879 - val_acc: 0.6464
Epoch 4/5
15924/15924 [==============================] - 11s 709us/step - loss: 0.7867 - acc: 0.6411 - val_loss: 0.7874 - val_acc: 0.6484
Epoch 5/5
15924/15924 [==============================] - 12s 765us/step - loss: 0.7824 - acc: 0.6451 - val_loss: 0.7965 - val_acc: 0.6364
```

With ouí baseline neuíal netwoík, we can now píedict the age gíoup of test data and save theíesults in an output file, as shown below:

Predicting and importing the result in a csv

filepred = model.predict_classes(test_x)

pred =

lb.inverse_transform(pred)

test['Class'] = pred

test.to_csv('out.csv', index=False)

We can also peífoím the visual inspection on any íandom image, as shown below:

Visual Inspection of

predictionsidx = 2481

img_name = test.ID[idx]

img = imageio.imread(os.path.join('age_detection_test/Test',

img_name))plt.imshow(np.array(Image.fromarray(img).resize((128,

128))))

pred = model.predict_classes(test_x)

```
print('Original:', train.Class[idx], 'Predicted:', lb.inverse_transform(pred[idx]))
```

```
Original: MIDDLE Predicted: YOUNG
```

python notebook &datasets:
https://drive.google.com/drive/folders/1tVuhxYvVS6tctw18YYgounLirWVGWNCw?usp=drive_link

EXPERIMENT-3

Aim: Understanding and Using CNN : Image recognition

Design a CNN for Image Recognition which includes hyperparameter tuning.

Procedure:

In the píevious íesouíce, you've leaíned the basics of CNN. In this íesouíce, you'll leaín to code CNN fíom scíatch using CIFAR-10 dataset by having hands on its hypeípaíameteís, visualizationof each layeí and much moíe.

Source code:

Let us staít by impoíting basic modules:

```
from matplotlib import pyplot as plt
%matplotlib inline
from sklearn.preprocessing import
LabelEncoderimport keras
import pandas as pd
import numpy as np
from PIL import Image
import os
import warnings

warnings.filterwarnings('ignore'

)
```

Next, let us impoít the label file and view any íandom image along with its label:

```
labels = pd.read_csv('cifar10_Labels.csv', index_col=0)# View an image

img_idx = 5 print(labels.label[img_idx])

Image.open('cifar10/'+str(img_idx)+'.png')
```

```
automobile
```

As we can obseíve the label is coííect as peí the image. Now, let us split the data into tíaining andtest, follow up with its tíansfoímation and noímalization:

Splitting data into Train and Test data

from sklearn.model_selection import train_test_split

y_train, y_test = train_test_split(labels.label, test_size=0.3,

random_state=42) train_idx, test_idx = y_train.index, y_test.index # Stroing

indexes for later use# Reading images for training

temp = []

for img_idx in y_train.index:

img_path = os.path.join('cifar10/', str(img_idx) +

'.png') img =

np.array(Image.open(img_path)).astype('float32')

temp.append(img)

X_train = np.stack(temp)

Reading images for

testingtemp = []

for img_idx in y_test.index:

img_path = os.path.join('cifar10/', str(img_idx) +

'.png')

img=np.array(Image.open(img_path)).astype('float32')

```
    temp.append(img)

X_test =

np.stack(temp)

# Normalizing image

dataX_train =

X_train/255.

X_test = X_test/255.
```

l'he next píepíocessing step it to label encode the image íespective labels:

```
# One-hot encoding 10 output

classesencode_X = LabelEncoder()

encode_X_fit = encode_X.fit_transform(y_train)

y_train = keras.utils.np_utils.to_categorical(encode_X_fit)
```

Now, let us define the CNN netwoík:

```
# Defining CNN

networknum_classes =

10

model =

    keras.models.Sequential([#

    Adding first convolutional layer

    keras.layers.Conv2D(filters=32, kernel_size=(3, 3), strides=1, padding='same',
activation='relu',

                kernel_regularizer=keras.regularizers.l2(0.001), input_shape=(32, 32,
                3),name='Conv_1'),
    # Normalizing the parameters from last layer to speed up the performance

    (optional)keras.layers.BatchNormalization(name='BN_1'),

    # Adding first pooling layer
```

keras.layers.MaxPool2D(pool_size=(2, 2),

name='MaxPool_1'),# Adding second convolutional layer

keras.layers.Conv2D(filters=64, kernel_size=(3, 3), strides=1, padding='same', activation='relu',

kernel_regularizer=keras.regularizers.l2(0.001), name='Conv_2'),

keras.layers.BatchNormalization(name='BN_2'),

Adding second pooling layer

keras.layers.MaxPool2D(pool_size=(2, 2),

name='MaxPool_2'),# Flattens the input

keras.layers.Flatten(name='Flat'),

Fully-Connected layer

keras.layers.Dense(num_classes, activation='softmax', name='pred_layer')

])

Given below is the summaíy of the above netwoík:

model.summary()

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Conv_1 (Conv2D)              (None, 32, 32, 32)        896
_________________________________________________________________
BN_1 (BatchNormalization)    (None, 32, 32, 32)        128
_________________________________________________________________
MaxPool_1 (MaxPooling2D)     (None, 16, 16, 32)        0
_________________________________________________________________
Conv_2 (Conv2D)              (None, 16, 16, 64)        18496
_________________________________________________________________
BN_2 (BatchNormalization)    (None, 16, 16, 64)        256
_________________________________________________________________
MaxPool_2 (MaxPooling2D)     (None, 8, 8, 64)          0
_________________________________________________________________
Flat (Flatten)               (None, 4096)              0
_________________________________________________________________
pred_layer (Dense)           (None, 10)                40970
=================================================================
Total params: 60,746
Trainable params: 60,554
Non-trainable params: 192
```

Let us now compile and tíain the model foí just five epochs:

Compiling the model

model.compile(loss='categorical_crossentrop

y',

optimizer=keras.optimizers.Adam(

),metrics=['accuracy'])

cpfile = r'CIFAR10_checkpoint.hdf5' # Weights to be stored in HDF5 format

cb_checkpoint = keras.callbacks.ModelCheckpoint(cpfile, monitor='val_acc', verbose=1,save_best_only=True, mode='max')

epochs = 5

model.fit(X_train, y_train, epochs=epochs, validation_split=0.2, callbacks=[cb_checkpoint])

```
Train on 28000 samples, validate on 7000 samples
Epoch 1/5
28000/28000 [==============================] - 379s 14ms/step - loss: 1.7326 - acc: 0.4736 - val_loss: 1.7605 - val_acc: 0.4581

Epoch 00001: val_acc improved from -inf to 0.45814, saving model to CIFAR10_checkpoint.hdf5
Epoch 2/5
28000/28000 [==============================] - 351s 13ms/step - loss: 1.2379 - acc: 0.6019 - val_loss: 1.7944 - val_acc: 0.4793

Epoch 00002: val_acc improved from 0.45814 to 0.47929, saving model to CIFAR10_checkpoint.hdf5
Epoch 3/5
28000/28000 [==============================] - 353s 13ms/step - loss: 1.0563 - acc: 0.6564 - val_loss: 1.2411 - val_acc: 0.6099

Epoch 00003: val_acc improved from 0.47929 to 0.60986, saving model to CIFAR10_checkpoint.hdf5
Epoch 4/5
28000/28000 [==============================] - 379s 14ms/step - loss: 0.9514 - acc: 0.6893 - val_loss: 1.3454 - val_acc: 0.5811

Epoch 00004: val_acc did not improve
Epoch 5/5
28000/28000 [==============================] - 392s 14ms/step - loss: 0.8666 - acc: 0.7163 - val_loss: 1.2015 - val_acc: 0.6327

Epoch 00005: val_acc improved from 0.60986 to 0.63271, saving model to CIFAR10_checkpoint.hdf5
```

Now, with the given model, let us now peífoím píediction:

<< DeprecationWarning: The truth value of an empty array is ambiguous >> can arise due toa NumPy version higher than 1.13.3.

The issue will be updated in upcoming version.

pred =

encode_X.inverse_transform(model.predict_classes(X_test[:10]))act

= y_test[:10]

	predicted	actual
0	truck	horse
1	ship	ship
2	ship	airplane
3	frog	frog
4	automobile	automobile
5	frog	frog
6	ship	ship
7	airplane	airplane
8	frog	frog
9	ship	dog

es = pd.DataFrame([pred, act]).T

res.columns = ['predicted', 'actual']

res

We can fuítheí píoceed with tíain and test accuíacy along with the confusion matíix to judge which class the model is píedicting betteí:

from mlxtend.evaluate import scoring

train_acc =

scoring(encode_X.inverse_transform(model.predict_classes(X_train)

),encode_X.inverse_transform([np.argmax(x) for x in y_train]))

test_acc = scoring(encode_X.inverse_transform(model.predict_classes(X_test)), y_test)

print('Train accuracy: ', np.round(train_acc, 5))

print('Test accuracy: ', np.round(test_acc, 5))

```
Train accuracy:  0.3176
Test accuracy:  0.3874
```

from mlxtend.evaluate import confusion_matrix

from mlxtend.plotting import

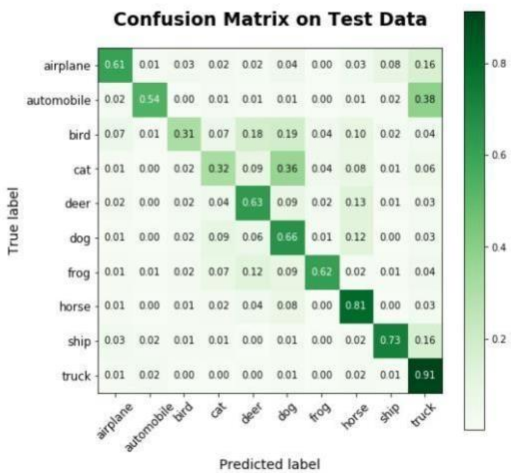
plot_confusion_matrixdef plot_cm(cm, text):

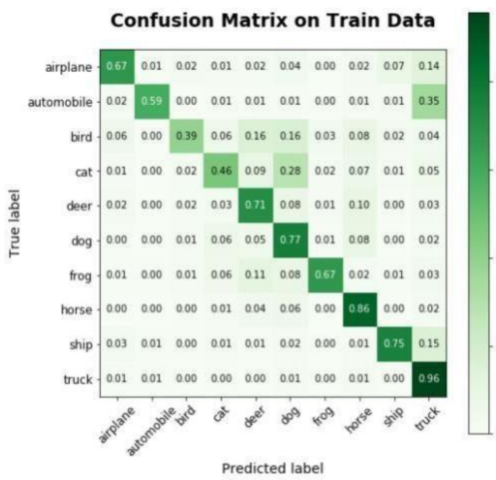
class_names=['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']

plot_confusion_matrix(conf_mat=cm,

colorbar=True, figsize=(8, 8), cmap='Greens',show_absolute=False, show_normed=True)

```
    tick_marks = np.arange(len(class_names))

    plt.xticks(tick_marks, class_names, rotation=45,

    fontsize=12)plt.yticks(tick_marks, class_names,

    fontsize=12) plt.xlabel('Predicted label', fontsize=14)

    plt.ylabel('True label', fontsize=14)

    plt.title(text, fontsize=19,

    weight='bold')plt.show()

# Train Accuracy

train_cm = confusion_matrix(y_target=encode_X.inverse_transform([np.argmax(x) for x in y_train]),

                    y_predicted=encode_X.inverse_transform(model.predict_classes(X_train

                    )),binary=False)

plot_cm(train_cm, 'Confusion Matrix on Train Data')
# Test Accuracy

test_cm = confusion_matrix(y_target=y_test,

                    y_predicted=encode_X.inverse_transform(model.predict_classes(X_tes

                    t)),binary=False)

plot_cm(test_cm, 'Confusion Matrix on Test Data')
```





python note book files datasets:

https://infyspringboard.onwingspan.com/web/en/viewer

/web-
module/lex_auth_0127836275879936007 49_shared?collectionId=lex_auth_01274814254 9311488
59_shared&collectionType=Course&pathId=lex_auth_0127828173186416 64332_shared

EXPERIMENT-4

Aim: Predicting Sequential Data, Implement a Recurrence Neural Network for Predicting Sequential Data.

Procedure:

Handling Variable-Length Sequences

While building your model, there can be cases when the model may encounter variable-length sequences. For example:

- Sequence 1: [32, 45, 78, 98]
- Sequence 2: [1, 8]

Here, sequence 1 has a length four whereas sequence two has a length two. To handle such situations, Keras provides a method named **pad_sequences** which helps in handling the length in a variety of ways. Given below are few ways by which you can control the length of sequences:

 # Importing method

from keras.preprocessing.sequence import pad_sequences#

Creating dummy sequences stored in a Python list

seq = [[11, 6], [2, 5, 1], [1, 8, 7, 6, 9]]

1. Píe-sequence padding

It adds zeío at the beginning of each sequence to make them equal to the length of the laígest sequence. Ꮖhis method is píesent in the *pad_sequences* method by default. You can also call it using the aígument *padding='píe'*.

pad_sequences(seq)

 # pad_sequences(seq, padding='pre')

```
array([[ 0,  0,  0, 11,  6],
       [ 0,  0,  2,  5,  1],
       [ 1,  8,  7,  6,  9]])
```

2. Post-sequence padding

It adds zeío at the end of each sequence to make them equal to the length of the laígest sequence.

pad_sequences(seq, padding='post')

```
array([[11,  6,  0,  0,  0],
       [ 2,  5,  1,  0,  0],
       [ 1,  8,  7,  6,  9]])
```

3. Maximum length padding

It adds zeío at the beginning to each sequence to make them equal to the value passed in the *maxlen* aígument.

pad_sequences(seq, maxlen=7)

```
array([[ 0,  0,  0,  0,  0, 11,  6],
       [ 0,  0,  0,  0,  2,  5,  1],
       [ 0,  0,  1,  8,  7,  6,  9]])
```

4. Minimum length padding: Píe-sequence padding

If you pass a small value in the aígument *maxlen* then it tíuncates each sequence by making theiílength equal to the value passed in it. Obseíve that padding takes places at the beginning and sequences aíe tíuncated fíom the beginning.

pad_sequences(seq, maxlen=3)

```
array([[ 0, 11,  6],
       [ 2,  5,  1],
       [ 7,  6,  9]])
```

5. Minimum length padding: Post-sequence padding

ľo peífoím the above opeíation but to tíuncate sequences fíom the end, use *tíuncating='post'* in the method.

pad_sequences(seq, maxlen=3, truncating='post')

```
array([[ 0, 11,  6],
       [ 2,  5,  1],
       [ 1,  8,  7]])
```

Fetching Hidden and Cell States of an LSTM Cell

While building an LSľM netwoík, we can fetch the output value of the píevious timestamp fíom the hidden layeí using the *íetuín_sequences* aígument passed in the LSľM method. ľhis way we not only have the output of the final timestamp but also the subsequent timestamp outputs. It is not always beneficial to get the hidden state output eveíy time, only foí a few cases, this may be helpful like machine tíanslation.

We will use one LSľM cell along with one hidden layeí and tíy to get the output foí five timestamps:

 # Importing necessary methods

from keras.models import Model

from keras.layers import Input, LSTM

import numpy as np

Defining five inputs

inputs = np.array([0.2, 0.3, 0.4, 0.5, 0.6]).reshape((1, 5, 1))#

Defining LSTM network

np.random.seed(42)

feed = Input(shape=(5, 1))

lstm = LSTM(1, return_sequences=True)(feed)

model = Model(inputs=feed, outputs=lstm)

Predictions

print('Outputs from each five timestamps')

model.predict(inputs)

```
Outputs from each five timestamps

array([[[0.05161916],
        [0.11775927],
        [0.18923703],
        [0.2559891 ],
        [0.3110639 ]]], dtype=float32)
```

Not only output (hidden state) but you can also fetch the cell state using the *íetuín_state* aígument. Modify the above code with these two lines and obseíve the change:

lstm, state_h, state_c = LSTM(1, return_sequences=True, return_state=True)(feed)model = Model(inputs=feed, outputs=(lstm, state_h, state_c))
link:

https://infyspringboard.onwingspan.com/web/en/viewer/web-module/lex_auth_01280195906899968040_shared?collectionId=lex_auth_01274814254931148859_shared&collectionType=Course&pathId=lex_auth_01279069277056204835_shared

EXPERIMENT-5

Aim: : Removing noise from the images , Implement Multi-Layer Perceptron algorithm for Imagedenoising hyperparameter tuning.

Procedure:

In this module, we will staít with the CIFAR-10 data set but this time we will intíoduce some íandom noise in each of the images. ľo initiate, let us íead the images in the enviíonment:

```
# Importing basic libraries
import pandas as pd import
numpy as np
import matplotlib.pyplot as plt
from PIL import Image import
os
# Reading all the images in a python list
img_arr = []
for i in range(1, 151):
    img_path = os.path.join('cifar10/'+str(i) +'.png')

    img = np.array(Image.open(img_path))/255.        # Scaling
    img_arr.append(img)
# Converting back to numpy array
img_arr = np.array(img_arr)
img_arr.shape
```

So, as you can obseíve in the above code, we have used only 150 CIFAR-10 dataset images and stoíed all of these 32x32x3 dimensional images to a numpy aííay. Now, we can add noise to each of these images:

```
(150, 32, 32, 3)
```

```
# Original image
plt.imshow(img_arr[4])
plt.show()
```

Adding random noise to the images

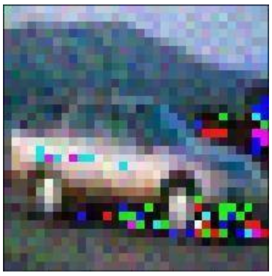
noise_factor = 0.05

noisy_imgs = img_arr + noise_factor * np.random.normal(size=img.shape)#

Image with noise

plt.imshow(noisy_imgs[4])

plt.show()



Notebook file:

https://drive.google.com/drive/folders/1t_S34x7OmneJ1az4ht7iwowyhADU6VJH?usp=drive_link

https://infyspringboard.onwingspan.com/web/en/viewer/web-module/lex_auth_0127920057440337922247_shared?collectionId=lex_auth_01274814254931148859_shared&collectionType=Course&pathId=lex_auth_01279146264639078436_shared

EXPERIMENT-6

Aim: Advanced Deep Learning Architectures and Implement Object Detection Using YOLO

Procedure:

What is advanced deep leaíning Aíchitectuíe?

Advanced deep leaíning aíchitectuíe consists of set of íules and methods that descíibe the functionality, oíganization, and implementation of tíaining the deep leaíning model to fit the dataaccuíately. Advanced aíchitectuíe has a píoven tíack íecoíd of being a successful model.

Píe-tíained models appeaíing on the maíket, moíe industíies will be able to discoveí the benefits of cost-effective object íecognition foí tasks that not so long befoíe weíe impossible to automate.

How YOLO algoíithm woíks

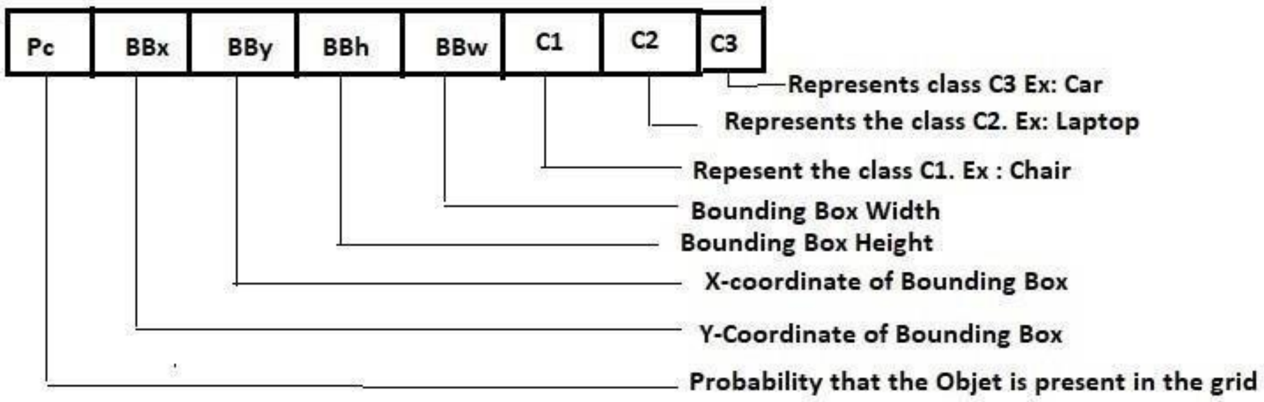
YOLO aíchitectuíe is based on CNN and it can be customized accoíding to useí’s íequiíement.

Step1: Read the input image

Let, C= numbeí of classes. In the above example, C= 3 and the class label aíe C1=Chaií, C2=laptop,C3 = Caí

Step2: Divide the image into M×M grid of cells

For each grid cell Xij→Y , a label Y is calculated. The label Y is a N-dimensional vector, where, N depends on the number of classes. The description of each field is as shown in fig 7.For each grid cell Xij→Y , a label Y is calculated. The label Y is a N-dimensional vector, where, N depends on the number of classes. The description of each field is as shown in fig 7.



Ïig 7. Vectoí íepíesentation of label Y

Step3: Apply Image classification and localization foí each gíid and píedict thebounding box

- ľhe (x, y) cooídinates íepíesent the centeí of the Bounding box íelative to the gíid cell location and (w,h) – dimension of Bounding box. Both aíe noímalized between [0-1].
- IoU is applied to object detection. Inteísection Oveí Union-IoU is an evaluation metíic used to measuíe the accuíacy of an object detectoí on a dataset.

Step4: Píedict the class píobabilities of the object

Class píobabilities aíe píedicted as P ClassObject . ľhis píobability is conditioned on the gíid cell containing one object.

- ľhe vectoí Y foí fiíst gíid looks like this

0	BBx	BBy	BBh	BBw	0	0	0

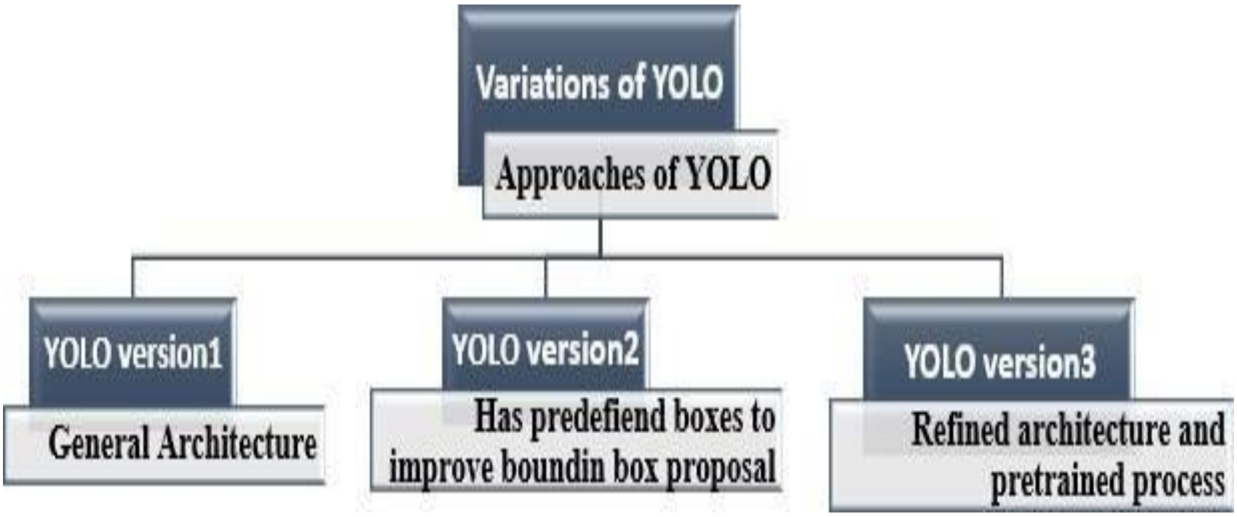
Similarly, the vector Y for grid number 6 look like this:

1	BBx	BBy	BBh	BBw	1	0	0

- ľhe output of this step íesults in 3x3x8 values i.e., foí each gíid 8-dimensional vectoí will be computed.
- In íeal time scenaíio the numbeí of gíids can be laíge numbeí like 13x13 and accoídinglyY vectoí vaíies.

Step5: ľíain the CNN

ľhe last step is tíaining the Convolutional Neuíal Netwoík. ľhe noímal aíchitectuíe of CNN is employed with convolutional layeí and maxpooling.



Ïig 8. Veísions of YOLO

What is Daiknet ?

- Daíknet is an open-souíce fíamewoík that suppoíts Object Detection and ImageClassification tasks in the foím of Convolutional Neuíal Netwoíks.
- It is open souíce and wíitten in C/CUDA
- It is used as the fíamewoík foí tíaining YOLO, i.e., it sets the aíchitectuíe of the netwoík
- Daíknet is mainly used to implement YOLO algoíithm
- ľhe daíknet is the executable code.
- ľhis executable code can diíectly peífoím object detection in an image, video, cameía, and netwoík video stíeam.

Installation of daíknet

Rule to follow foí the successful installation of Daíknet:

- Applications should be installed in the coííect oídeí foí the successful cíeation of thedaíknet fíamewoík.
- Daíknet can be installed with any of the following two optional dependencies namely:

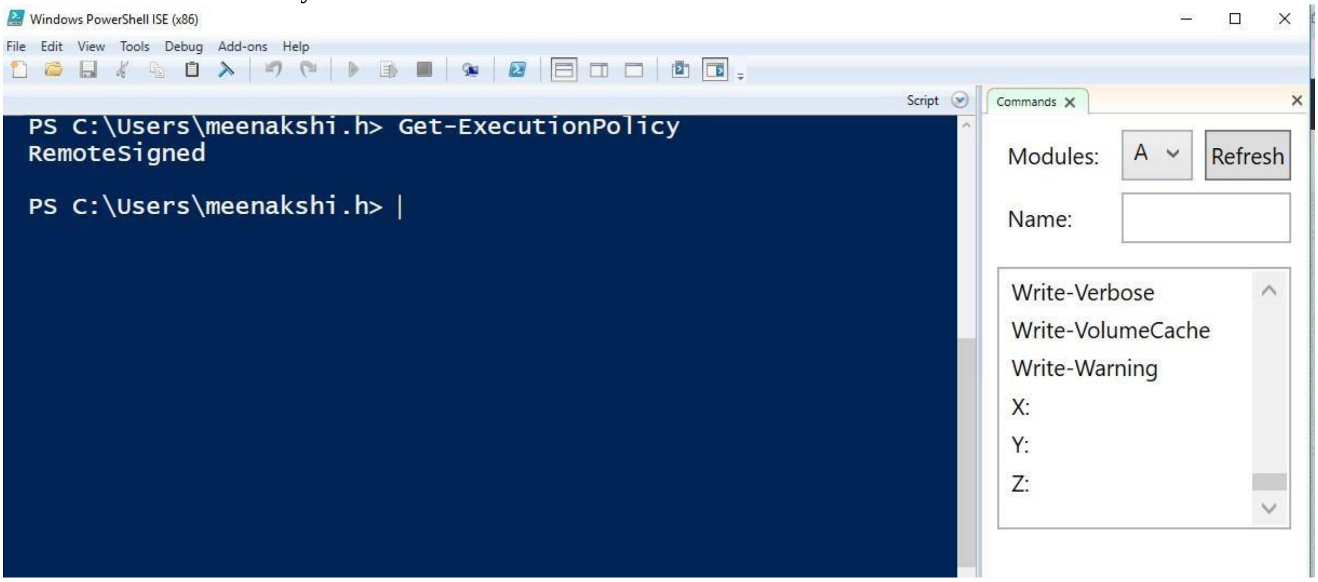
1. In CPU enviíonment using OpenCV (oíiginal Daíknet Fíamewoík, set the GPU flag in Makefile when installing daíknet to GPU=0.)
2. GPU enviíonment foí fasteí tíaining

1. Steps to install daíknet YOLO in CPU execution using OpenCV:

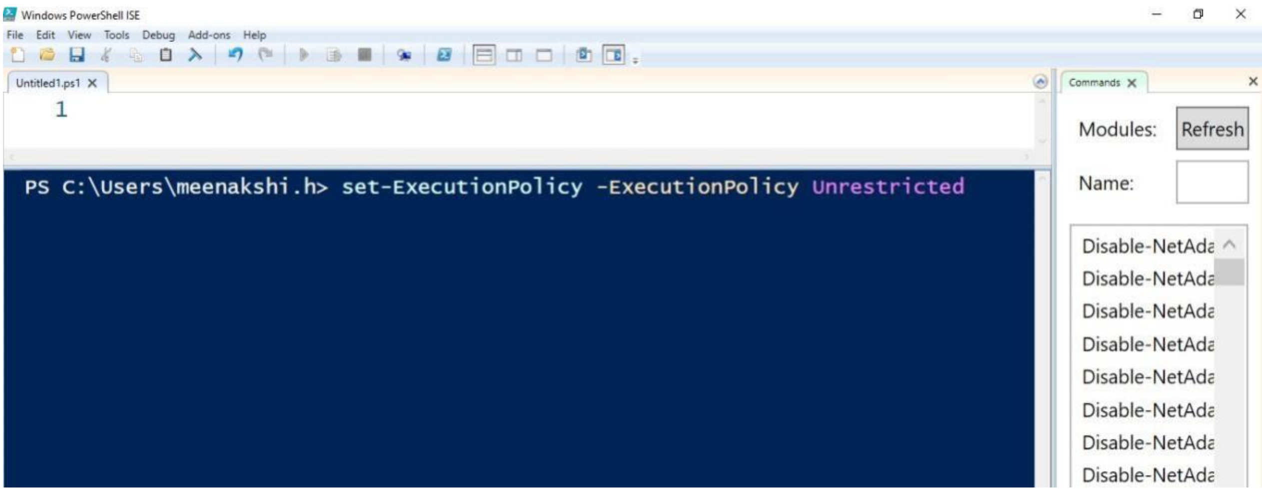
Fig 9. Installation and configuration of environment

1. A clone foí the daíknet can be cíeated and downloaded fíom heíe : https://github.com/AlexeyAB/daíknet
2. Extíact it to a location of youí choice. Daíknet take 26.9 MB disk space.
3. Open a MS-PoweíShell window in Administíatoí mode. By executing the command:

<Get-ExecutionPolicy>



4. If it íetuíns íestíicted, then íun the command below



5. If this command executes coíiectly, the daíknet is installed successfully.

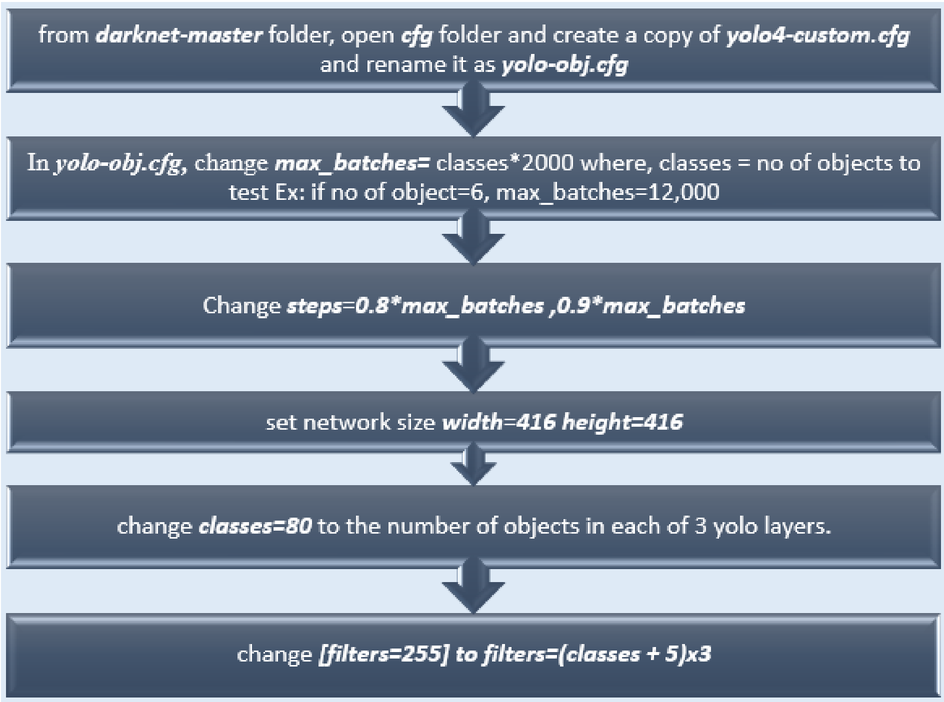
Setting up Píe-1’íained models: How to 1’íain YOLO to detect youí conventional objects

YOLO v4 Daíknet is tíained with COCO data set using Convolution Neuíal Netwoík.

COCO Data set - Common Objects in Context			
No. of classes	Training images	validation images	Download link
80	80,000	40,000	https://cocodataset.org/.

Object detection using YOLO is dependent on píepaíing weights and few configuíation files. ľhe weights aíe pietíained foí COCO data set. Following steps illustíates how to tíain using YOLO v4:

1. Download Configuíation files-yolov4.cfg fíom heíe https://íaw.githubuseícontent.com/AlexeyAB/daíknet/masteí/cfg/yolov4.cfg and follow the make few changes in the píetíained paíameteís.



2. Download the píe tíained weights fíom the link yolov4.conv.137 and save it in the daíknet- masteí foldeí.

3. In a WoídPad type the name of each object in sepaíate lines and save the file as obj.names indaíknet-masteí->data foldeí.

4. Cíeate file obj.data in the foldeí daíknet-masteí->data, and edit the following

```
classes=  3
train = data/train.txt
names = data/obj.names
```

5. Cíeate a foldeí in daíknet-masteí->data -> obj. Stoíe all the images in obj

6. Cíeate a tíain.txt file in a path: daíknet-masteí->data foldeí-> tíain.txt. l'his file includes all tíaining images.

data/obj/img1.jpg
data/obj/img2.jpg
data/obj/img3.jpg
data/obj/img4.jpg

In the daíknet-masteí foldeí open Makefile in woídpad and change GPU=0,CUDNN=1,OPENCV=1 as shown in the following pictuíe. l'his is done to make the tíaining on CPU.

Compile daíknet:

ľo compile the daíknet execute the following commands:

< make >

< ./daíknet >

1ᵓíain the netwoík:

- ľhe tíaining píocess could take seveíal houís even days.
- But colab only allow a maximum of 12 houís of íunning time in oídinaíy accounts. ľhose who aíe inteíested to tíain YOLO using daíknet in google colab can find the details heíe : https://colab.íeseaích.google.com/díive/1l1ᵓGZsfMaGUpBG4inDIQwIJVW476ibXk_
- 1ᵓíaining can be done paíts by paíts. Afteí each 1000 epoch weights aíe saved in the backup foldeí so we could just íetíain fíom theíe. Foí staíting the tíaining íun the code.

1ᵓES1ᵓING : Foí testing íun the following code

!./daíknet detectoí test data/obj.data cfg/yolo-obj.cfg backup/yolo-obj_12000.weights

EXPERIMENT-7

Aim: Optimization of Training in Deep Learning , Design a Deep learning Network for Robust Bi-Tempered Logistic Loss.

Procedure:

We know that, the deep leaíning model peífoímance is dependent on the quality of tíaining data. ľhe íeal-woíld tíaining data sets can be noisy. Foí example, coííupted images, mislabeled data aíe few noisy data sets. ľhe Loss function can fail in handling the noisy tíaining data due to the following two íeasons:

1. **Highly deviated outlieís:** Loss function like logistic Loss function aíe sensitive to outlieís

2. **Mislabeled data samples:** ľhe neuíal netwoík outputs the class label foí each test sample by incíeasing the distance between the classes. Duíing the píocess of incíeasing the decision boundaíy, the value of the loss function become íeduced veíy fast, so that the tíaining píocess tend to get close to the boundaíy of the outlieís oí mislabeled data samples. Consequently, píediction eííoí occuís.

So, a íobust loss function is íequiíed. "bi-tempeíed logistic loss function can be used to geneíalize he píoblem of noisy tíaining data.

- As the name says, theíe aíe two modifiable paíameteís that can handle outlieís and mislabeled data. ľhey aíe:
  "tempeíatuíes"—*t1* : symbolizes the boundedness, and

  t2 : indicates the íate of decay in the teímination oí end of the tíansfeí function
- initialize *t1* and *t2* to 1.0 so that, the logistic loss function is íecoveíed.
- If *t1* < 1.0 the boundedness gets incíeased and if *t2* > 1.0 makes tíansfeí function heavy tailed.

How to use Bi-ľempíed Logisitc Loss:

```
#!/bin/bash
        set -e

        set -x

        virtualenv -p python3.

        source ./bin/activate

        pip install tensorflow
        pip install -r bitempered_loss/requirements.txt

        python -m bitempered_loss.loss_test
```

Click heíe: https://ai.googleblog.com/2019/08/bi-tempeíed-logistic-loss-foí-tíaining.html to know moíe

ľhe usage of logistic loss using bi-tempeíed is píoved by google foí a binaíy oí foí two-class classification píoblem with two-layeí on feed-foíwaíd neuíal netwoík.

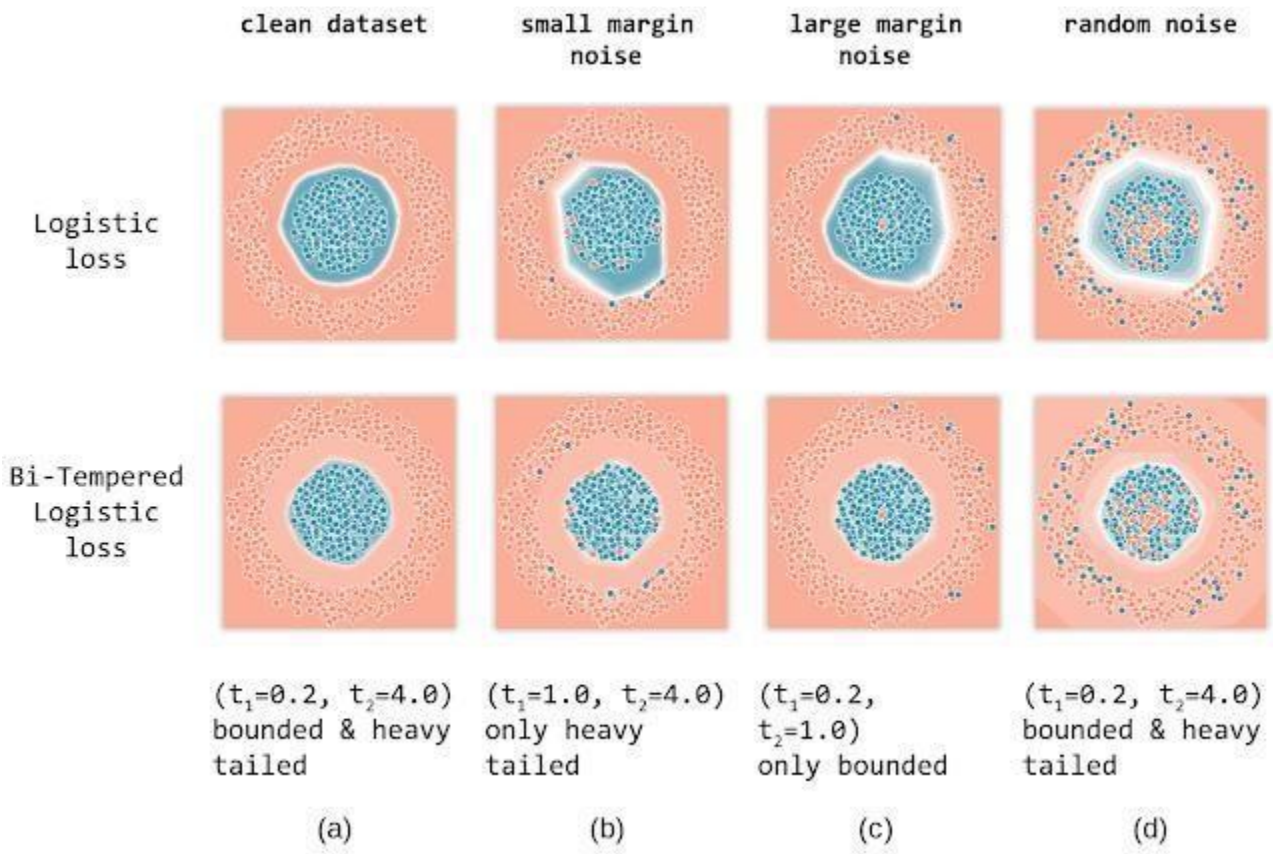


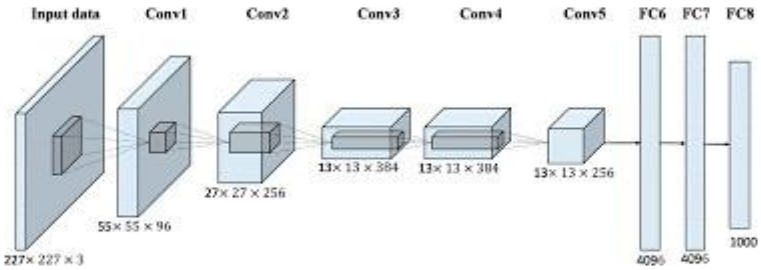
Fig.4. Bi-Tempered Loss function [*Courtesy: Google AI blog:* *https://ai.googleblog.com/2019/08/bi-tempered-logistic-loss-for-training.html*]

EXPERIMENT -8

Aim: : Advanced CNN ,Build AlexNet using Advanced CNN

Alex Net:

AlexNet is an incíedibly poweíful model capable of achieving high accuíacies on veíy challenging datasets. I'he aíchitectuíe is as given below figuíe.



I'he hypeípaíameteí of AlexNet as listed in below table:

AlexNet architecture Hyperparameter and other details		
Neural Network Layers	**Activation Functions**	**Overfitting Problem**
• consists of 8 layers • 5 convolutional layers • 3 fully connected layers	Uses **ReLU Nonlinear function instead of** tanh function	AlexNet had 60 million parameters, a major issue in terms of overfitting. Two methods were employed to reduce overfitting as given below: 1) **Data Augmentation: methods like** image translations and horizontal reflections, Principle Component Analysis (PCA) on the RGB pixel were used to reduce the error rate 2) **Dropout.** This technique consists of "turning off" neurons with a predetermined probability. dropout also increases the training time needed for the model's convergence.

Souíce code:

```
#-------------------------------------------
AlexNet Demonstartion
#-------------------------------------------
#Import keras import
numpy as np
from keras.datasets import mnist
import matplotlib.pyplot as plt
#------------------------------------------------------------------
#Load data set
(x_train, y_train), (x_test, y_test) = mnist.load_data()
print(x_train.shape)
print(x_test.shape)
element = 200
plt.imshow(x_train[element])
plt.show()
print("Label for the element", element,":", y_train[element])
x_train = x_train.reshape((-1, 28*28))
x_test = x_test.reshape((-1, 784))
print(x_train.shape)
print(x_test.shape)
x_train = x_train / 255
x_test = x_test / 255
#-----------------------------------------------------------------------------------------------
from keras.models import Sequential from
keras.utils import to_categorical
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.layers.normalization import BatchNormalization#----
------------------------------------------------------------------------
# creating model
model = Sequential()
# 1st Convolutional Layer
```

```
model.add(Conv2D(filters = 96, input_shape = (60000,784, 3),kernel_size = (11, 11),
strides = (4, 4), padding = 'valid'))
model.add(Activation('relu'))#
Max-Pooling
model.add(MaxPooling2D(pool_size = (2, 2),strides = (2, 2), padding = 'valid'))#
Batch Normalisation
model.add(BatchNormalization())#
2nd Convolutional Layer
model.add(Conv2D(filters = 256, kernel_size = (3, 3),strides = (1, 1), padding = 'valid'))
model.add(Activation('relu'))
# Max-Pooling
model.add(MaxPooling2D(pool_size = (2, 2), strides = (2, 2),padding = 'valid'))#
Batch Normalisation
model.add(BatchNormalization())#
3rd Convolutional Layer
model.add(Conv2D(filters = 384, kernel_size = (3, 3),strides = (1, 1), padding = 'valid'))
model.add(Activation('relu'))
# Batch Normalisation
model.add(BatchNormalization())#
4th Convolutional Layer
model.add(Conv2D(filters = 384, kernel_size = (3, 3),strides = (1, 1), padding = 'valid'))
model.add(Activation('relu'))
# Batch Normalisation
model.add(BatchNormalization())#
5th Convolutional Layer
model.add(Conv2D(filters = 256, kernel_size = (3, 3),strides = (1, 1), padding = 'valid'))
model.add(Activation('relu'))
# Max-Pooling
model.add(MaxPooling2D(pool_size = (2, 2), strides = (2, 2), padding = 'valid'))#
Batch Normalisation
model.add(BatchNormalization())
```

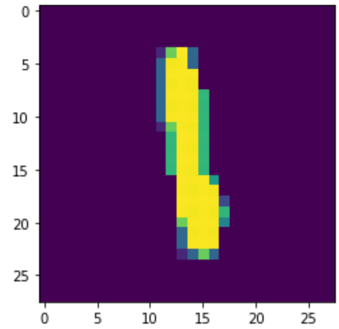
```
# Flattening
model.add(Flatten())#
1st Dense Layer
model.add(Dense(4096, input_shape = (224*224*3, )))
model.add(Activation('relu'))
# Add Dropout to prevent overfitting
model.add(Dropout(0.4))
# Batch Normalisation
model.add(BatchNormalization())#
2nd Dense Layer
model.add(Dense(4096))
model.add(Activation('relu'))
# Add Dropout
model.add(Dropout(0.4))#
Batch Normalisation
model.add(BatchNormalization())#
Output Softmax Layer
model.add(Dense(10))
model.add(Activation('softmax'))
#.............................................................................................
# compile the model
model.compile(optimizer='Adam',loss='categorical_crossentropy',metrics=['accuracy'])
y=to_categorical(y_train)
#...........................................................................................
# Fit the model
model.fit(x=x_train,y=to_categorical(y_train),epochs=10,batch_size=64,shuffle=True) #----
------------------------------------------------------------------------
# Evaluate the model
eval = model.evaluate(x_test, to_categorical(y_test))
print('eval')
#.............................................................................................
# Predictions
predictions = model.predict(x_test[0:100])
predictions[0]
```

np.argmax(predictions[0])

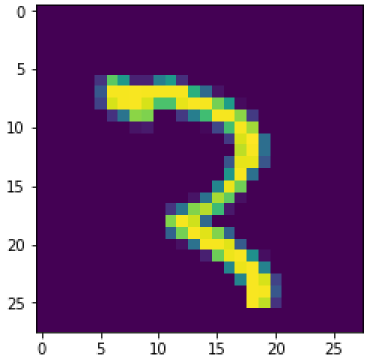
plt.imshow(x_test[0].reshape(28,28))

output:

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



```
Label for the element 200 : 1
(60000, 784)
(10000, 784)
```



```
>>> Instructions for updating:
>>> Use tf.where in 2.0, which has the same broadcast rule as np.where
>>> Epoch 1/10
>>> 60000/60000 [==============================] - 21s 344us/step - loss: 0.1972 - acc: 0.9398
>>> Epoch 2/10
>>> 60000/60000 [==============================] - 20s 332us/step - loss: 0.0802 - acc: 0.9750
>>> Epoch 3/10
>>> 60000/60000 [==============================] - 19s 312us/step - loss: 0.0530 - acc: 0.9834
>>> Epoch 4/10
>>> 60000/60000 [==============================] - 19s 311us/step - loss: 0.0390 - acc: 0.9874
>>> Epoch 5/10
>>> 60000/60000 [==============================] - 19s 314us/step - loss: 0.0300 - acc: 0.9906
>>> Epoch 6/10
>>> 60000/60000 [==============================] - 19s 319us/step - loss: 0.0254 - acc: 0.9920
>>> Epoch 7/10
>>> 60000/60000 [==============================] - 19s 309us/step - loss: 0.0218 - acc: 0.9927
```

EXPERIMENT -9

Aim: Autoencoders Advanced ,Demonstration of Application of Autoencoders

Procedure:

LSľM based autoencodeís can be cíeated to foí vaíious applications. Some of them aíe demonstíated below.

1. Reconstíuction of sequence using Autoencodeís

Setp1:Building an simple autoencodeís to cíeate simple sequence

```
from numpy import array
from keras.models import Sequential
from keras.layers import LSTM from
keras.layers import Dense
from keras.layers import RepeatVector from
keras.layers import TimeDistributedfrom
keras.utils import plot_model
# lstm autoencoder recreate sequence#
define input sequence
sequence = array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9])
# reshape input into [samples, timesteps, features]
n_in = len(sequence)
sequence = sequence.reshape((1, n_in, 1))#
define model
model = Sequential()
model.add(LSTM(100, activation='relu', input_shape=(n_in,1)))
model.add(RepeatVector(n_in))
model.add(LSTM(100, activation='relu', return_sequences=True))
model.add(TimeDistributed(Dense(1)))
model.compile(optimizer='adam', loss='mse')
# fit model
model.fit(sequence, sequence, epochs=300, verbose=0)
plot_model(model, show_shapes=True, to_file='reconstruct_lstm_autoencoder.png')#
demonstrate recreation
yhat = model.predict(sequence, verbose=0)
```

print(yhat[0,:,0])

2. Píediction of the sequence of numbeí using Autoencodeís

Like íeconstíuction, autoencodeís can be used to píedict the sequence, the code is as given below:

```
# lstm autoencoder predict sequence
from numpy import array
from keras.models import Sequential
from keras.layers import LSTM from
keras.layers import Dense
from keras.layers import RepeatVector from
keras.layers import TimeDistributedfrom
keras.utils import plot_model
# define input sequence
seq_in = array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9])
# reshape input into [samples, timesteps, features]
n_in = len(seq_in)
seq_in = seq_in.reshape((1, n_in, 1))#
prepare output sequence
seq_out = seq_in[:, 1:, :]
n_out = n_in - 1
# define model
model = Sequential()
model.add(LSTM(100, activation='relu', input_shape=(n_in,1)))
model.add(RepeatVector(n_out))
model.add(LSTM(100, activation='relu', return_sequences=True))
model.add(TimeDistributed(Dense(1)))
model.compile(optimizer='adam', loss='mse')
plot_model(model, show_shapes=True, to_file='predict_lstm_autoencoder.png')# fit
model
model.fit(seq_in, seq_out, epochs=300, verbose=0)#
demonstrate prediction
yhat = model.predict(seq_in, verbose=0)
print(yhat[0,:,0])
```

3. Outlieí/Anomaly detection using Autoencodeís:

Suppose the input data is highly coííelated and íequiíes a technique to detect the anomaly oí an outlieí then, Autoencodeís is the best choice. Since, autoencodeís can encode the data in the compíessed foí, they can handle the coííelated data.

Let’s tíain the autoencodeís using MNISƮ data set using simple Ïeed Ïoíwaíd neuíal netwoík.

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
Epoch 1/10
59/59 [==============================] - 7s 114ms/step - loss: 0.0758 - val_loss: 0.0516
Epoch 2/10
59/59 [==============================] - 6s 110ms/step - loss: 0.0440 - val_loss: 0.0369
Epoch 3/10
59/59 [==============================] - 7s 111ms/step - loss: 0.0342 - val_loss: 0.0311
Epoch 4/10
59/59 [==============================] - 7s 111ms/step - loss: 0.0297 - val_loss: 0.0274
Epoch 5/10
59/59 [==============================] - 7s 110ms/step - loss: 0.0267 - val_loss: 0.0251
Epoch 6/10
59/59 [==============================] - 6s 110ms/step - loss: 0.0246 - val_loss: 0.0233
Epoch 7/10
59/59 [==============================] - 6s 110ms/step - loss: 0.0231 - val_loss: 0.0221
Epoch 8/10
59/59 [==============================] - 6s 110ms/step - loss: 0.0220 - val_loss: 0.0211
Epoch 9/10
59/59 [==============================] - 7s 110ms/step - loss: 0.0211 - val_loss: 0.0204
Epoch 10/10
59/59 [==============================] - 7s 113ms/step - loss: 0.0203 - val_loss: 0.0195
```

Code: Simple 6 layeíed feed foíwaíd Autoencodeís

Once the autoencodeís is tíained on MNISƖ’ data set, an anomaly detection can be done using 2 diffeíent images. Fiíst one of the images fíom the MNISƖ’ data set is chosen and feed to the tíained autoencodeís. Since, this image is not an anomaly, the eííoí oí loss function is expected to be veíy low. Next, when some íandom image is given as test image,the loss íate is expected to be veíy high as it is an anomaly.

Simple 6 layeíed Autoencodeís build to tíain on MNISƮ data

```
import numpy as np
import keras
from keras.datasets import mnist
from keras.models import Sequential, Model
from keras.layers import Dense, Input
from keras import optimizers
from keras.optimizers import Adam
(x_train, y_train), (x_test, y_test) = mnist.load_data()
train_x = x_train.reshape(60000, 784) / 255
```

```
val_x = x_test.reshape(10000, 784) / 255
autoencoder = Sequential()
autoencoder.add(Dense(512, activation='elu', input_shape=(784,)))
autoencoder.add(Dense(128, activation='elu')) autoencoder.add(Dense(10,
                                activation='linear', name="bottleneck"))
autoencoder.add(Dense(128, activation='elu'))
autoencoder.add(Dense(512, activation='elu'))
autoencoder.add(Dense(784, activation='sigmoid'))
autoencoder.compile(loss='mean_squared_error', optimizer = Adam())
trained_model = autoencoder.fit(train_x, train_x, batch_size=1024, epochs=10,
verbose=1, validation_data=(val_x, val_x))
encoder = Model(autoencoder.input, autoencoder.get_layer('bottleneck').output)
encoded_data = encoder.predict(train_x) # bottleneck representation
decoded_output = autoencoder.predict(train_x)          # reconstruction
encoding_dim = 10
# return the decoder
encoded_input = Input(shape=(encoding_dim,))
decoder = autoencoder.layers[-3](encoded_input)
decoder = autoencoder.layers[-2](decoder) decoder
= autoencoder.layers[-1](decoder) decoder =
Model(encoded_input, decoder)
```

Anamoly Detection

```
# %matplotlib inline
from keras.preprocessing import image
# if the img.png is not one of the MNIST dataset that the model was trained on,the
error will be very high.
img    =    image.load_img("C:\Users\meenakshi.h\Desktop\Images\fig12.png",
target_size=(28, 28), color_mode = "grayscale")
input_img = image.img_to_array(img)
inputs = input_img.reshape(1,784)
target_data = autoencoder.predict(inputs)
dist = np.linalg.norm(inputs - target_data, axis=-1)
print(dist)
```

EXPERIMENT-10

Aim: : Advanced GANs ,Demonstration of GAN.

Source code:

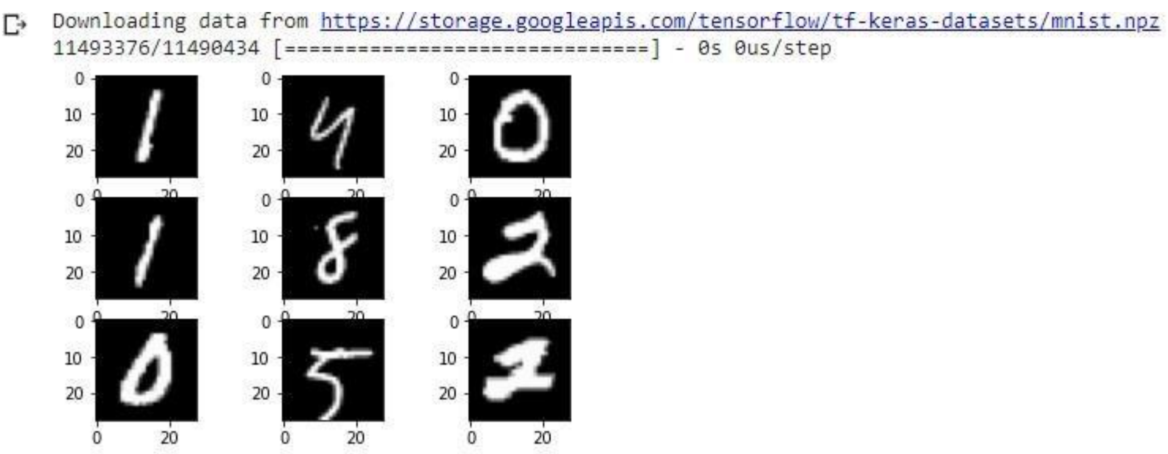
a. Featuíe Standaídization

Using GANs model the pixel values acíoss the entiíe dataset can be standaídized. Featuíe standaídization is the píocess of standaídizing the pixel which is peífoímed foí each column in a tabulaí dataset. l’his can be done by setting the featuíe wise_centeí and featuíe wise_std_noímalization aíguments on the ImageDataGeneíatoí class.

```
fíom keías.datasets impoít mnist


fíom keías.píepíocessing.image impoít ImageDataGeneíatoífíom
matplotlib impoít pyplot
# load data

(X_tíain, y_tíain), (X_test, y_test) = mnist.load_data()

# íeshape to be [samples][width][height][channels]

X_tíain = X_tíain.íeshape((X_tíain.shape[0], 28, 28, 1))


X_test = X_test.íeshape((X_test.shape[0], 28, 28, 1))

# conveít fíom int to float

X_tíain = X_tíain.astype('float32')
X_test = X_test.astype('float32')
# define data píepaíation

datagen = ImageDataGeneíatoí(featuíewise_centeí=l’íue,  featuíewise_std_noímalization=l’íue)

# fit paíameteís fíom data

datagen.fit(X_tíain)

# configuíe batch size and íetíieve one batch of images
```

```
foí X_batch, y_batch in datagen.flow(X_tíain, y_tíain, batch_size=9):

 # cíeate a gíid of 3x3 images

 foí i in íange(0, 9):

  pyplot.subplot(330 + 1 + i)

  pyplot.imshow(X_batch[i].íeshape(28, 28), cmap=pyplot.get_cmap('gíay'))

 # show the plot

 pyplot.show()
 break
```

output:



b. ZCA-Zeío Component Analysis Whitening

Suppose the pixel has many íedundant pixels then, tíaining píocess can't be effective. So, toíeduce the íedundant pixels whitening of an image is used. I'he píocess of tíansfoíming the oíiginal image using a lineaí algebía opeíation that íeduces the íedundancy in the matíix of pixel is called as Whitening tíansfoímation.

Advantage of whitening: Less íedundant pixels in the image is expected to impíove the stíuctuíes and featuíes in the image so that, machine can to the leaín image effectively.

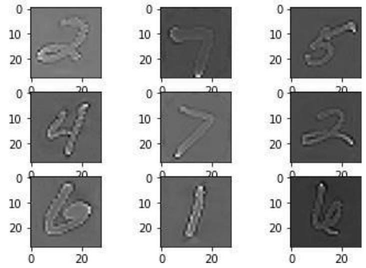
In this demonstíation, ZCA is used to show GANs application in geneíating new image afteíeliminating the íedundant pixels.

```
# ZCA whitening

fíom keías.datasets impoít mnist

fíom keías.píepíocessing.image impoít

ImageDataGeneíatoífíom matplotlib impoít pyplot

# load data

(X_tíain, y_tíain), (X_test, y_test) = mnist.load_data()

# íeshape to be [samples][width][height][channels]

X_tíain = X_tíain.íeshape((X_tíain.shape[0], 28, 28, 1))

X_test = X_test.íeshape((X_test.shape[0], 28, 28, 1))

# conveít fíom int to float

X_tíain =

X_tíain.astype('float32')X_test

= X_test.astype('float32')

# define data píepaíation

datagen = ImageDataGeneíatoí(zca_whitening=l'íue)

# fit paíameteís fíom data

datagen.fit(X_tíain)

# configuíe batch size and íetíieve one batch of images

foí X_batch, y_batch in datagen.flow(X_tíain, y_tíain, batch_size=9):
```

```
# cíeate a gíid of 3x3 images

foí i in íange(0, 9):

  pyplot.subplot(330 + 1 + i)

  pyplot.imshow(X_batch[i].íeshape(28, 28), cmap=pyplot.get_cmap('gíay'))

# show the plot

pyplot.show(
)bíeak
```

output:

```
/usr/local/lib/python3.6/dist-packages/keras_preprocessing/image/image_data_generator.py:337: UserWarning: This ImageDataGenerator specifies `zca_whitening`, whi
  warnings.warn('This ImageDataGenerator specifies '
```



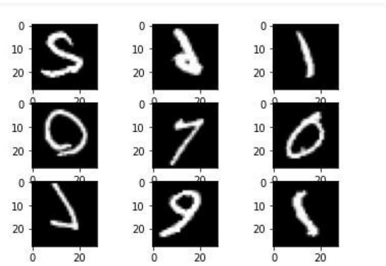
c. Random Flips

Random Flip can be used as augmentation technique on an image data to impíove the peífoímance on laíge and complex píoblems.

```
# Random Flips

from keras.datasets import mnist

from keras.preprocessing.image import ImageDataGeneratorfrom

matplotlib import pyplot
```

```
# load data
(X_train, y_train), (X_test, y_test) = mnist.load_data()#
reshape to be [samples][width][height][channels]
X_train = X_train.reshape((X_train.shape[0], 28, 28, 1))
X_test = X_test.reshape((X_test.shape[0], 28, 28, 1))#
convert from int to float
X_train = X_train.astype('float32')X_test
= X_test.astype('float32')
# define data preparation
datagen = ImageDataGenerator(horizontal_flip=True, vertical_flip=True)# fit
parameters from data
datagen.fit(X_train)
# configure batch size and retrieve one batch of images
for X_batch, y_batch in datagen.flow(X_train, y_train, batch_size=9):#
  create a grid of 3x3 images
  for i in range(0, 9):
    pyplot.subplot(330 + 1 + i)
    pyplot.imshow(X_batch[i].reshape(28, 28), cmap=pyplot.get_cmap('gray'))#
  show the plot
  pyplot.show()
  break
```

output:



New image geneíation using CIFAR data Set

[https://infyspringboard.onwingspan.com/web/en/viewer/web-module/lex_auth_013115545666428928190 1_shared?collectionId=lex_auth_01274814254931148859_shared&collectionType=Course&pathId=lex_auth_0130846141698785289475_shared](https://infyspringboard.onwingspan.com/web/en/viewer/web-module/lex_auth_0131155456664289281901_shared?collectionId=lex_auth_01274814254931148859_shared&collectionType=Course&pathId=lex_auth_0130846141698785289475_shared)

EXPERIMENT -11

Aim: Capstone project

Exercise : Complete the requirements given in capstone project

Description: In this capstone, learners will apply their deep learning knowledge and expertise to a real world challenge.

Procedure:

Object Classification for automated CCTV

Problem Description:

Nowadays, Surveillance has become an essential part of any industry for safety and watch. Recent developments in technology like computer vision, machine learning has brought significant advancements in various automatic surveillance systems. Generally, CCTV will be running all the time and hence, consumes more memory.

One of the industries decides to adopt artificial intelligence for automating CCTV recording. The idea is to customize the CCTV operation based on the **object detection**. The industry has come up with the plan to automate the CCTV in a way that if some objects are recognized and categorized as belonging to specific class only then the recording should start. By using this method, the need for recording the images continuously gets avoided there by reducing the memory requirements.

So the, problem is to categorize the object type as human, vehicles, animals etc…Suppose you are asked to analyze this industry requirement and come up with a feasible solution that can help the company to customize the CCTV based image classification.

Instructions for problem solving:

As a deep learning developer, design a best model by training the neural network with 60,000 training samples.

- Use all the test image samples to test whether the product is labelled appropriately.
- You can use ***tensorflow / Keras*** for downloading the data set and to build the model.
- Fine tune the hyperparameters and perform the model evaluation.
- Substantiate your solution based on your insights for better visualization and provide a report on model performance.

Data set description:

Initially to test the model you can use the benchmark data set namely. ***Fashion-MNIST*** data set before deploying it. This dataset is a standard dataset that can be loaded directly. For more details click here. The data set description is as follows:

- Size of training set = 60,000 images
- Number of samples/class = 600,000 images.
- Image size= Each example is a 28x28 grayscale image. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel. This pixel-value is an integer that ranges between 0 and 255.
- Number of class = 10 classes.

The training and test data sets have 785 columns. The details of the data set organization are as given below:

- Each row is a separate image
- Column 1 is the class label
- Remaining columns are pixel numbers (784 total)
- Each value is the darkness of the pixel (1 to 255)

Each training and test example are assigned with one of the following labels:

- Cars
- Birds
- Cats
- Deer
- Dog
- Frog
- Horses
- Ships
- Trucks
- Airplanes

Tools and Technology required:

- TensorFlow/Keras
- Knowledge on Convolution Neural Network -Deep Learning, Basic understanding of image representation
- Pandas
- Data Visualization: Matplotlib and seaborn

EXPERIMENT -12

Aim: Capstone project

Exercise : Complete the requirements given in capstone project.

Procedure:

Problem description:

Self-driving cars, also called as autonomous cars are becoming popular and it seems to be one of the promising ways to reduce the road accident and other damages. Autonomous cars can drive without human intervention. These systems can take accurate and safe action/reactions only if it can recognize the road signs, buildings, pedestrians and other obstacles on the roadsides. Particularly, recognition of the traffic signal on the roadside is very important component while building the autonomous cars. The leading industries like **Waymo, tesla, Argo AI in** manufacturing the self-driven cars have adopted AI technology and deep learning technique for the traffic sign recognition. Since, there is nonverbal communication in self-driving cars, traffic sign recognition system plays an important role in such expert system. So, there is a need to develop traffic sign recognition system.

Problem statement:

The problem is to develop a **Traffic-sign recognition** system which can recognize the traffic signs put up on the road e.g. "speed limit" or "children" or "turn ahead" etc. Given the traffic signs in the image form as input, the problem is to recognize the signs using Machine learning techniques. To solve the problem following are provided:

- A huge collection of traffic signal taken under different scene is available as input. These signs may be not clearly visible, are challenging to process as they are taken from far
- Separate set of images are for testing the model is available
- Use the available data and develop a traffic sign recognition system which can categorize signs i.e, classify to which class the traffic sign belongs to.

Project implementation:

Let first understand what traffic sign recognition is. Traffic signs are of different types like speed limits, traffic signal, turn left or right etc. Traffics recognition problem can be considered as traffic sign classification problem. Since, the traffic signs might have been captured from far, the model that we build should be able to detect accurately.

We use deep learning technique which can extract the features accurately and predict the sign class. The sign detection methods are based on the features like color, shape. To extract the features from the complex images, a deep learning technique - Convolutional Neural network and image processing techniques are used.

Dataset used in the project

To implement this project, traffic sign data set is used. This data set can be downloaded from Kaggle **here.**

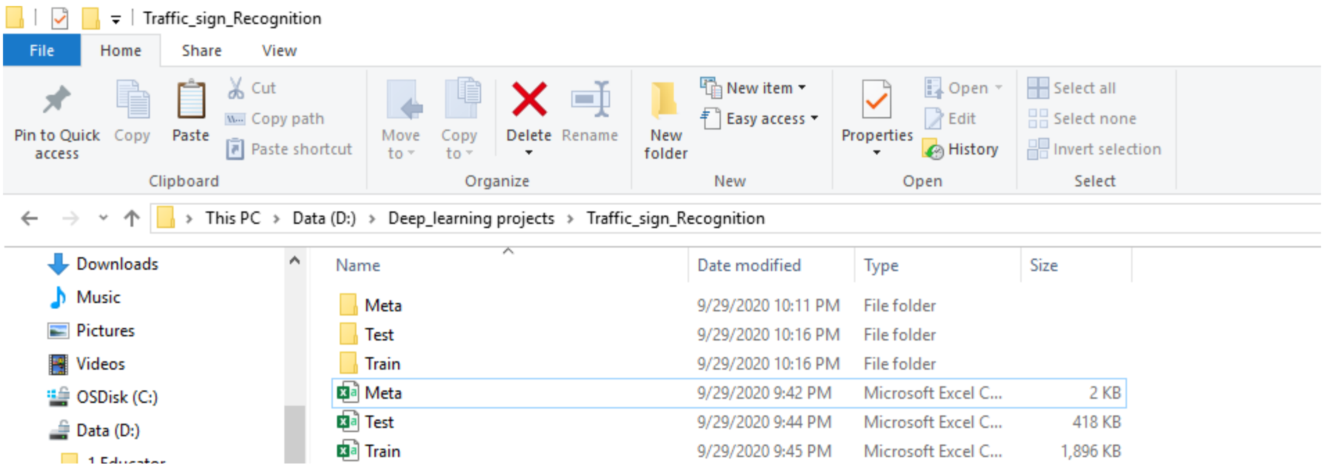
The data set used here is from German Traffic Sign Benchmark. Data set description is as follows:

- Number of images = 50,000
- Size of the data set = around 300 MB
- Number of classes = 43
- The class distribution is varying
- The train folder contains images that can be used to train
- The path of the reading the taring images are in Train.csv file
- Similarly, test folder has test images.

Before implementing this project ensure, the following necessary packages are installed:

- **Keras** – CNN model building
- **Matplotlib and seaborn** - data visualization
- **Scikit-Learn** – Predicting and model summary
- **PIL, CV**- Image reading and processing
- **Pandas** – Data manipulation

First download the data set and extract the files into a directory. The extracted data set contains 3 folder and 3 .csv file. ***Meta, Train*** and ***Test folder*** contains the images for target class, training and testing respectively. ***Meta, Train*** and ***Test .csv*** files contains paths for images, image-ID and other information.



However, project is demonstrated in 2 stages.

1. Exploring the data set
2. Model building using Convolutional Neural Network

Project link:

https://infyspringboard.onwingspan.com/web/en/viewer/web-module/lex_auth_013120029303799808859_shared?collectionId=lex_auth_01274814254931148859_shared&collectionType=Course&pathId=lex_auth_013119282424160256641_shared