GENERATIVE AI

[AIML303]

PRACTICAL LAB FILE

In partial fulfilment of the requirements for the award of the degree of

Bachelor of Technology

In

Computer Science & Technology

Department of Computer Science and Engineering

Amity University, Uttar Pradesh

Submitted by: Vindhya Sree

Enrolment No.: A2305221413

Class: 6CSE4Y

Submitted to: Dr. Rinki Gupta

ASET

INDEX

S.No	Name of Experiment	Date of Experiment	Remarks
1.	Build an Artificial Neural Network to implement a Binary Classification task using Backpropagation algorithm and test the algorithm with inputs.	09/01/24	
2.	Build an Artificial Neural Network to implement a Multiclass Classification task using Backpropagation algorithm and test the algorithm with inputs.	16/01/24	
3.	Design a CNN architecture to implement the image classification task over an image dataset. Perform the Hyper-parameter tuning and record the results.	23/01/24	
4.	Implement an image classification task using pretrained models like VGGNet, InceptionNet and ResNet and compare the results.	06/02/24	
5.	Implement an autoencoder architecture for denoising images.	20/02/24	
6.	Implement GAN architecture on MNIST dataset to recognize images of handwritten digits.	12/03/24	

EXPERIMENT 1: Build an Artificial Neural Network to implement Binary Classification task using the Back-propagation algorithm and test the same using appropriate data sets.

DESCRIPTION:

The data used here is: 'Pima Indians Diabetes Dataset'. It is downloaded from: https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indiansdiabetes.csv. It is a binary (2-class) classification problem. There are 768 observations with 8 input variables and 1 output variable. The variable names are as follows:

1. Number of times pregnant.
2. Plasma glucose concentration a 2 hours in an oral glucose tolerance test.
3. Diastolic blood pressure (mm Hg).
4. Triceps skinfold thickness (mm).
5. 2-Hour serum insulin (mu U/ml).
6. Body mass index (weight in kg/(height in m)^2).
7. Diabetes pedigree function.
8. Age (years).
9. Class variable (0 or 1).

Binary classification is a supervised learning algorithm that categorizes new observations into one of two classes. The Backpropagation algorithm is a supervised learning method for multilayer feed-forward networks from the field of Artificial Neural Networks. Feed-forward neural networks are inspired by the information processing of one or more neural cells, called a neuron. The principle of the backpropagation approach is to model a given function by modifying internal weightings of input signals to produce an expected output signal. Technically, the backpropagation algorithm is a method for training the weights in a multilayer feed-forward neural network. As such, it requires a network structure to be defined of one or more layers where one layer is fully connected to the next layer. A standard network structure is one input layer, one hidden layer, and one output layer. Backpropagation can be used for both classification and regression problems.

CODE:

Data Import and Processing

```
import numpy as np import
pandas as pd import
matplotlib.pyplot as plt import
sklearn # load data
url='https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-
indiansdiabetes.csv'
```

data_pd = pd.read_csv(url,header = None)
print(data_pd.info()) print(data_pd.head())
Output: <class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767 Data columns
(total 9 columns):

```
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
0   0       768 non-null    int64
1   1       768 non-null    int64
2   2       768 non-null    int64
3   3       768 non-null    int64
4   4       768 non-null    int64
5   5       768 non-null    float64
6   6       768 non-null    float64
7   7       768 non-null    int64    8   8       768 non-null    int64
dtypes: float64(2), int64(7) memory
usage: 54.1 KB
None
    0   1    2   3    4     5      6     7  8
0   6  148  72  35   0    33.6   0.627  50  1
1   1   85  66  29   0    26.6   0.351  31  0
2   8  183  64   0   0    23.3   0.672  32  1
3   1   89  66  23   94   28.1   0.167  21  0 4   0  137  40  35  168  43.1   2.288  33
    1
```

#Scaling Numerical columns
from sklearn.preprocessing import StandardScaler std
= StandardScaler()
scaled = std.fit_transform(data_pd.iloc[:,0:8])
scaled = pd.DataFrame(scaled) scaled.head()
Output:

	0	1	2	3	4	5	6	7
0	0.639947	0.848324	0.149641	0.907270	-0.692891	0.204013	0.468492	1.425995
1	-0.844885	-1.123396	-0.160546	0.530902	-0.692891	-0.684422	-0.365061	-0.190672
2	1.233880	1.943724	-0.263941	-1.288212	-0.692891	-1.103255	0.604397	-0.105584
3	-0.844885	-0.998208	-0.160546	0.154533	0.123302	-0.494043	-0.920763	-1.041549
4	-1.141852	0.504055	-1.504687	0.907270	0.765836	1.409746	5.484909	-0.020496

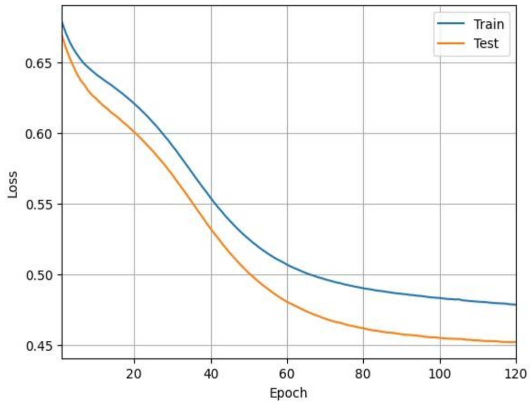
X_data =scaled.to_numpy()
print('X_data:',np.shape(X_data)) Y_data
= data_pd.iloc[:,8]
print('Y_data:',np.shape(Y_data))
Output:
X_data: (768, 8)
Y_data: (768,)

Split data into X_train, X_test, y_train, y_test from
sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_data, Y_data,
test_size=0.25, random_state= 0)
# Check the dimension of the sets
print('X_train:',np.shape(X_train))
print('y_train:',np.shape(y_train))
print('X_test:',np.shape(X_test))
print('y_test:',np.shape(y_test)) **Output:**
X_train: (576, 8)
y_train: (576,) X_test:
(192, 8)
y_test: (192,)

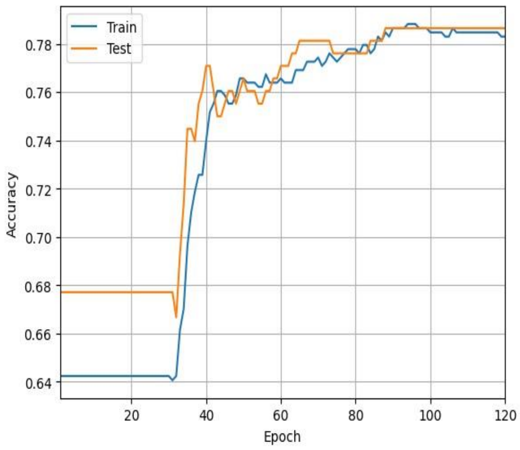
Design the Model import
keras
from keras.models import Sequential # importing Sequential model from
keras.layers import Dense # importing Dense layers
declaring model basic_model =
Sequential() # Adding layers to the
model (DIY)
First layers: 8 neurons/perceptrons that takes the input and uses 'sigmoid'
activation function.
basic_model.add(Dense(8, input_dim=8, activation='sigmoid')) #
Second layers: 4 neurons/perceptrons, 'sigmoid' activation function.
basic_model.add(Dense(4, activation='sigmoid'))
# Final layer: 1 neuron/perceptron to do binary classification
basic_model.add(Dense(1, activation='sigmoid'))
# compiling the model (DIY)
basic_model.compile(loss='binary_crossentropy',optimizer='adam',
metrics=['accuracy'])

Train the Model epochs=120
history = basic_model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=epochs) **Output:**
Epoch 1/120
18/18 [==============================] - 2s 46ms/step - loss: 0.6790 - accuracy: 0.6424 - val_loss: 0.6697 - val_accuracy: 0.6771
Epoch 2/120
18/18 [==============================] - 0s 4ms/step - loss: 0.6713 - accuracy: 0.6424 - val_loss: 0.6610 - val_accuracy: 0.6771
Epoch 3/120
18/18 [==============================] - 0s 4ms/step - loss: 0.6650 - accuracy: 0.6424 - val_loss: 0.6538 - val_accuracy: 0.6771
Epoch 4/120
18/18 [==============================] - 0s 4ms/step - loss: 0.6598 - accuracy: 0.6424 - val_loss: 0.6476 - val_accuracy: 0.6771
Epoch 5/120
18/18 [==============================] - 0s 5ms/step - loss: 0.6557 - accuracy: 0.6424 - val_loss: 0.6418 - val_accuracy: 0.6771
Epoch 6/120
18/18 [==============================] - 0s 6ms/step - loss: 0.6519 - accuracy: 0.6424 - val_loss: 0.6371 - val_accuracy: 0.6771
Epoch 7/120
18/18 [==============================] - 0s 19ms/step - loss: 0.6487 - accuracy: 0.6424 - val_loss: 0.6338 - val_accuracy: 0.6771
Epoch 8/120
18/18 [==============================] - 0s 16ms/step - loss: 0.6462 - accuracy: 0.6424 - val_loss: 0.6298 - val_accuracy: 0.6771
Epoch 9/120
18/18 [==============================] - 0s 7ms/step - loss: 0.6438 - accuracy: 0.6424 - val_loss: 0.6266 - val_accuracy: 0.6771
Epoch 10/120
18/18 [==============================] - 0s 10ms/step - loss: 0.6415 -
.
.
Epoch 120/120
18/18 [==============================] - 0s 4ms/step - loss: 0.4785 - accuracy: 0.7830 - val_loss: 0.4521 - val_accuracy: 0.7865

Evaluate the Model # plot loss vs epochs
epochRange = range(1,epochs+1);
plt.plot(epochRange,history.history['loss'])
plt.plot(epochRange,history.history['val_loss'])
plt.xlabel('Epoch') plt.ylabel('Loss') plt.grid()
plt.xlim((1,epochs)) plt.legend(['Train','Test'])
plt.show() **Output:**



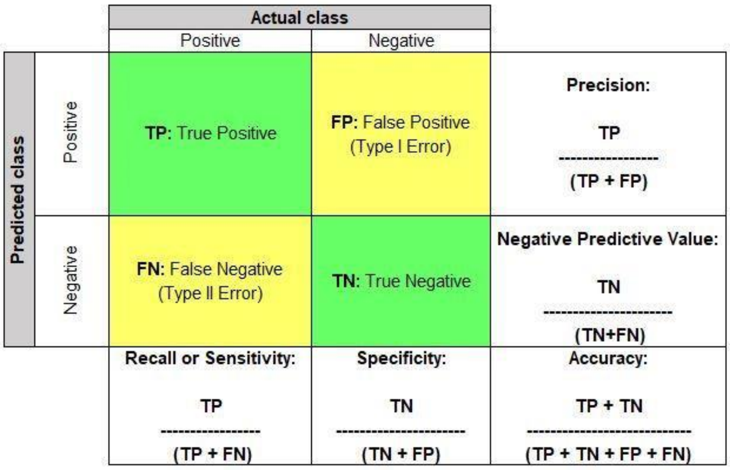
# Plot accuracy vs epochs (DIY)
plt.plot(epochRange, history.history['accuracy'])
plt.plot(epochRange, history.history['val_accuracy'])
plt.xlabel('Epoch') plt.ylabel('Accuracy') plt.grid()
plt.xlim((1,epochs)) plt.legend(['Train', 'Test'])
plt.show() **Output:**



# Test, Loss and accuracy
loss_and_metrics = basic_model.evaluate(X_test, y_test)
print('Loss = ',loss_and_metrics[0]) print('Accuracy = ',loss_and_metrics[1]) **Output:**
6/6 [==============================] - 0s 3ms/step - loss: 0.4521 - accuracy: 0.7865
Loss = 0.452068954706192

Accuracy = 0.7864583134651184

Classification Model Performance measures



y_pred = basic_model.predict(X_test) print(y_test[:5]) print(y_pred[:5]) **Output:**

6/6 [==============================] - 0s 3ms/step

661 1
122 0
113 0
14 1
529 0
Name: 8, dtype: int64
[[0.77858526]
[0.12162784]
[0.09794389]
[0.7211923]
[0.13421191]]

y_pred =[1 if y_pred[aa]>=0.5 else 0 for aa in range(len(y_pred))]
print(y_pred[:5]) [1, 0, 0, 1, 0]
print(sklearn.metrics.classification_report(y_test, y_pred))

Output:

precision recall f1-score support

0 0.83 0.86 0.85 130
1 0.68 0.63 0.66 62 accuracy
0.79 192 macro avg 0.76 0.75
0.75 192 weighted avg 0.78 0.79
0.78 192

CONCLUSION: An artificial neural network has been successfully built to implement binary classification task using Backpropagation algorithm with the Pima Indians Diabetes dataset with an accuracy of 78.64%.

EXPERIMENT 2: Build an Artificial Neural Network to implement Multi-Class Classification task using the Back-propagation algorithm and test the same using appropriate data sets.

DESCRIPTION:

The data that will be incorporated is the MNIST database (Modified National Institute of Standards and Technology database) which contains 60,000 images for training and 10,000 test images. The dataset consists of small square 28×28 pixel grayscale images of handwritten single digits between 0 and 9. The MNIST dataset is conveniently bundled within Keras, and we can easily analyze some of its features in Python.

Multiclass classification is the process of assigning entities with more than two classes. Each entity is assigned to one class without any overlap. An example of multiclass classification, using images of vegetables, where each image is either a carrot, tomato, or zucchini. Each image is placed in one of the three classes. For example, one image cannot be both a carrot and a zucchini. The backpropagation algorithm is a method for training the weights in a multilayer feed-forward neural network. Backpropagation can be used for both classification and regression problems.

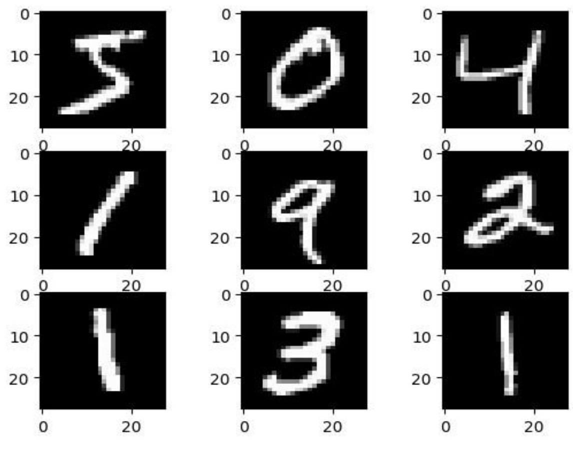
CODE:

pip install matplotlib **Output:**

Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/distpackages (3.7.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.2.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/distpackages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (4.47.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.4.5)
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.10/distpackages (from matplotlib) (1.23.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (23.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/distpackages (from matplotlib) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in

/usr/local/lib/python3.10/dist-packages (from matplotlib) (2.8.2)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.10/distpackages (from python-dateutil>=2.7->matplotlib)
(1.16.0) from tensorflow import keras
from keras.datasets import mnist   # MNIST dataset is included in Keras
(X_train, y_train), (X_test, y_test) = mnist.load_data()
print("X_train shape", X_train.shape) print("y_train
shape", y_train.shape) print("X_test shape",
X_test.shape) print("y_test shape", y_test.shape)
Output:
Downloading data from https://storage.googleapis.com/tensorflow/tf-
kerasdatasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
X_train shape (60000, 28, 28)
y_train shape (60000,) X_test
shape (10000, 28, 28)
y_test shape (10000,)

```
# Plot first few images import
matplotlib.pyplot as plt for i
in range(9):
        # define subplot
        plt.subplot(3,3,i+1) # 3 rows, 3 col, pos
        # plot raw pixel data
        plt.imshow(X_train[i], cmap='gray')
# show the figure
plt.show()
```
Output:



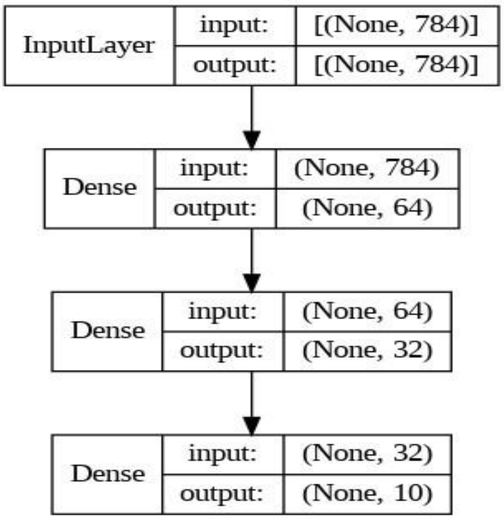
X_train[i].shape **Output:**
(28, 28)

```
# Each pixel is an 8-bit integer from 0-255 (0 is full black, 255 is full white)
# single-channel pixel or monochrome image
X_train[i][10:20,10:20] Output:
array([[  0,   0,  20, 254, 254, 108,   0,   0,   0,   0],
[  0,   0,  16, 239, 254, 143,   0,   0,   0,   0],
       [  0,   0,   0, 178, 254, 143,   0,   0,   0,   0],
       [  0,   0,   0, 178, 254, 143,   0,   0,   0,   0],
       [  0,   0,   0, 178, 254, 162,   0,   0,   0,   0],
       [  0,   0,   0, 178, 254, 240,   0,   0,   0,   0],
[  0,   0,   0, 113, 254, 240,   0,   0,   0,   0],
       [  0,   0,   0,  83, 254, 245,  31,   0,   0,   0],
       [  0,   0,   0,  79, 254, 246,  38,   0,   0,   0],
       [  0,   0,   0,   0, 214, 254, 150,   0,   0,   0]], dtype=uint8)

# reshape 28 x 28 matrices into 784-length vectors
X_train = X_train.reshape(60000, 784)
X_test = X_test.reshape(10000, 784)
# normalize each value for each pixel for the entire vector for each input
# change integers to 32-bit floating point numbers
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
# normalize by dividing by largest pixel value
X_train /= 255 X_test
/= 255
print("Training matrix shape", X_train.shape)
print("Testing matrix shape", X_test.shape) Output:
Training matrix shape (60000, 784)
Testing matrix shape (10000, 784)

# Sequential keras model with Dense layes (DIY) from keras.models
import Sequential  # Model type to be used from keras.layers import
Dense # Types of layers to be used in our model mdl = Sequential()
# Input layer with 64 units and relu activation
mdl.add(Dense(64, input_dim=784, activation='relu')) #
Hidden layer with 32 units and relu activation
mdl.add(Dense(32, activation='relu'))
# Output layer with 10 units and softmax activation mdl.add(Dense(10,
activation='softmax'))
```

Compile model
mdl.compile(optimizer='adam',loss='categorical_crossentropy', metrics=['accuracy']) # Visualize the model from keras.utils import plot_model
plot_model(mdl, show_shapes=True, show_layer_names=False) **Output:**



Display model summary mdl.summary()
Output: Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 64)	50240
dense_1 (Dense)	(None, 32)	2080
dense_2 (Dense)	(None, 10)	330

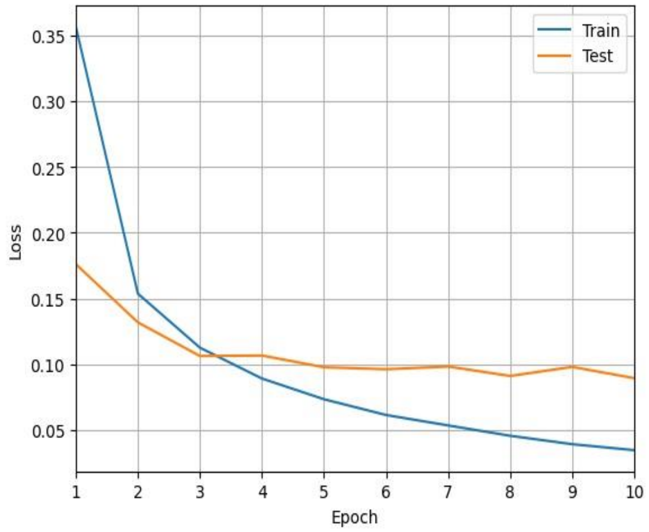
Total params: 52650 (205.66 KB)
Trainable params: 52650 (205.66 KB)
Non-trainable params: 0 (0.00 Byte)
#understand model summary
784*64 + 64
Output: 50240
64*32 + 32
Output: 2080
32*10+10 **Output:**
330

from tensorflow.keras.utils import to_categorical
y_train1 = to_categorical(y_train) y_test1 =
to_categorical(y_test) print(y_test[6])
print(y_test1[6,:]) **Output:**
4
[0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]

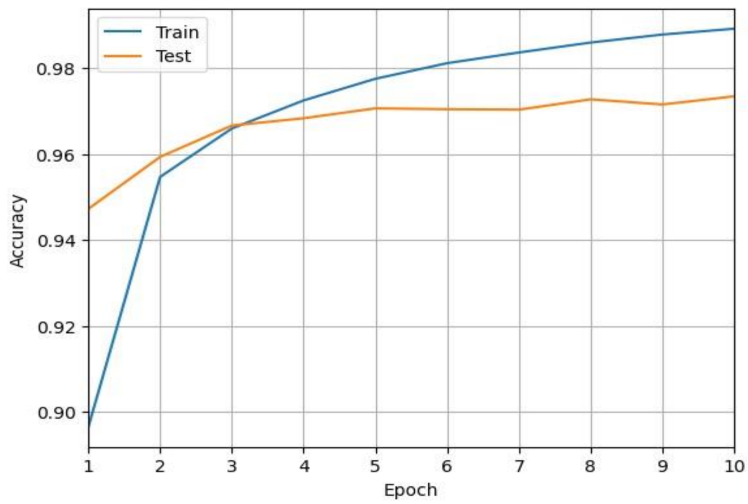
# Train the model epochs=10
batch = 64
history = mdl.fit(X_train, y_train1,epochs=epochs, batch_size=batch,verbose=1,
validation_data=(X_test, y_test1)) **Output:**
Epoch 1/10
938/938 [==============================] - 3s 2ms/step - loss: 0.3567
- accuracy: 0.8964 - val_loss: 0.1761 - val_accuracy: 0.9472
Epoch 2/10
938/938 [==============================] - 2s 2ms/step - loss: 0.1536
- accuracy: 0.9546 - val_loss: 0.1318 - val_accuracy: 0.9593
Epoch 3/10
938/938 [==============================] - 2s 2ms/step - loss: 0.1126
- accuracy: 0.9659 - val_loss: 0.1062 - val_accuracy: 0.9666
Epoch 4/10
938/938 [==============================] - 2s 2ms/step - loss: 0.0891
- accuracy: 0.9725 - val_loss: 0.1066 - val_accuracy: 0.9683
Epoch 5/10
938/938 [==============================] - 2s 2ms/step - loss: 0.0733
- accuracy: 0.9775 - val_loss: 0.0977 - val_accuracy: 0.9706
Epoch 6/10
938/938 [==============================] - 2s 2ms/step - loss: 0.0614
- accuracy: 0.9811 - val_loss: 0.0962 - val_accuracy: 0.9704
Epoch 7/10
938/938 [==============================] - 2s 2ms/step - loss: 0.0534
- accuracy: 0.9836 - val_loss: 0.0982 - val_accuracy: 0.9703 Epoch 8/10
938/938 [==============================] - 2s 2ms/step - loss: 0.0456
- accuracy: 0.9859 - val_loss: 0.0911 - val_accuracy: 0.9727
Epoch 9/10
938/938 [==============================] - 2s 2ms/step - loss: 0.0392
- accuracy: 0.9878 - val_loss: 0.0980 - val_accuracy: 0.9715
Epoch 10/10
938/938 [==============================] - 2s 2ms/step - loss: 0.0346

- accuracy: 0.9891 - val_loss: 0.0894 - val_accuracy: 0.9734

```
epochRange = range(1,epochs+1);
plt.plot(epochRange,history.history['loss'])
plt.plot(epochRange,history.history['val_loss'])
plt.xlabel('Epoch') plt.ylabel('Loss') plt.grid()
plt.xlim((1,epochs)) plt.legend(['Train','Test'])
plt.show()
```
Output:




```
plt.plot(epochRange,history.history['accuracy'])
plt.plot(epochRange,history.history['val_accuracy'])
plt.xlabel('Epoch') plt.ylabel('Accuracy') plt.grid()
plt.xlim((1,epochs)) plt.legend(['Train','Test'])
plt.show()
```
Output:




```
import numpy as np
yhat_test_mdl_prob = mdl.predict(X_test);
yhat_test_mdl = np.argmax(yhat_test_mdl_prob,axis=-1)
```

print(yhat_test_mdl_prob[0]) print(yhat_test_mdl[0:10])
print(y_test[0:10]) **Output:**

```
313/313 [==============================] - 1s 1ms/step
[1.34299620e-08 1.53619965e-07 1.55493431e-08 1.55332032e-06
 1.30692375e-11 5.23531607e-10 7.89350443e-12 9.99989092e-01
 3.76912048e-08 9.19328068e-06]
[7 2 1 0 4 1 4 9 5 9]
[7 2 1 0 4 1 4 9 5 9]
```

from sklearn.metrics import accuracy_score print('Accuracy:')
print(float(accuracy_score(y_test, yhat_test_mdl))*100,'%')
Output: Accuracy: 97.34 %

from sklearn.metrics import confusion_matrix
print('Confusion Matrix:')
print(confusion_matrix(y_test, yhat_test_mdl)) **Output:**

```
Confusion Matrix:
[[ 966    0    0    1    1    2    1    3    4    2]
 [    0 1121    4    3    0    2    1    1    3    0]
 [    5    3 1002    2    4    0    2    6    8    0]
 [    2    0    7  985    1    3    0    3    7    2]
 [    1    1    3    0  960    0    3    3    1   10]
 [    2    0    1   17    1  857    4    0    5    5]
 [    6    3    3    1   10    2  932    0    1    0]
 [    0    2    8    1    3    0    0 1008    1    5]
 [    8    0    5    9    6    9    2    7  922    6]
 [    2    4    0    6    7    2    0    7    0  981]]
```

CONCLUSION:

An artificial neural network to implement multiclass classification task using the backpropagation algorithm with the MNIST database has been built with accuracy 97.34%.

EXPERIMENT 3: Design a CNN architecture to implement the image classification task over an image dataset. Perform the Hyper-parameter tuning and record the results.

DESCRIPTION:

Dataset description: The data that will be incorporated is the MNIST database which contains 60,000 images for training and 10,000 test images. The dataset consists of small square 28×28 pixel grayscale images of handwritten single digits between 0 and 9. The MNIST dataset is conveniently bundled within Keras, and we can easily analyze some of its features in Python.

CNN architecture:

Image classification:

Hyper parameter tuning:

CODE OF MODEL 1:

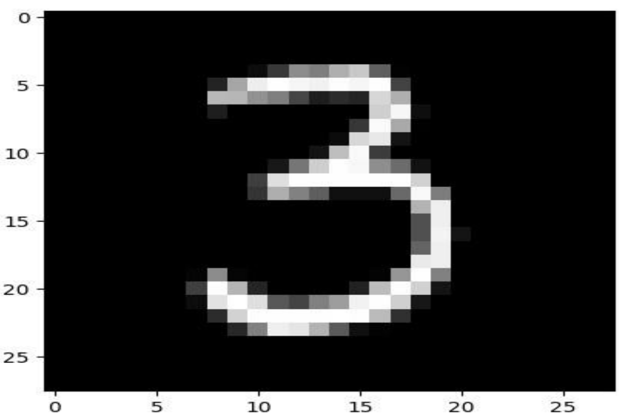
```
from tensorflow import keras from keras.datasets import mnist     #
MNIST dataset is included in Keras
(X_train, y_train), (X_test, y_test) = mnist.load_data()
print("X_train shape", X_train.shape) print("y_train
shape", y_train.shape) print("X_test shape",
X_test.shape) print("y_test shape", y_test.shape)
```

Output: Downloading data from https://storage.googleapis.com/tensorflow/tfkeras-datasets/mnist.npz

```
11490434/11490434 [==============================] - 0s 0us/step
X_train shape (60000, 28, 28)
y_train shape (60000,) X_test
shape (10000, 28, 28)
y_test shape (10000,)
```

```
# Visualize any random image
import matplotlib.pyplot as plt i=50;
plt.imshow(X_train[i], cmap='gray');
```

Output:



FORMATTING THE INPUT
# Single-channel input data (grey-scale)
# First apply convolutions then flatten
X_train = X_train.reshape(60000, 28, 28, 1) # single-channel input
X_test = X_test.reshape(10000, 28, 28, 1)
X_train = X_train.astype('float32')   # change integers to 32-bit floating point numbers
X_test = X_test.astype('float32')
X_train /= 255   # min-max normalization X_test /= 255
print("Training matrix shape", X_train.shape)
print("Testing matrix shape", X_test.shape) Output:
Training matrix shape (60000, 28, 28, 1)
Testing matrix shape (10000, 28, 28, 1)

CONVOLUTIONAL NEURAL NETWORK
from keras import backend as K from
keras import __version__
print('Using Keras version:', __version__, 'backend:', K.backend()) Output:
Using Keras version: 2.15.0 backend: tensorflow

# import cnn layers
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Dense import tensorflow as tf model.add(Conv2D(8, kernel_size=(3, 3), activation='relu', input_shape=(28, 28,
1), padding='valid', strides=1)) model.add(MaxPooling2D(pool_size=(2,
2), strides=2))
# Convolution Layer 2
model.add(Conv2D(16, kernel_size=(3, 3), activation='relu', padding='valid', strides=1))
model.add(MaxPooling2D(pool_size=(2, 2), strides=2))
# Flatten final feature matrix into a 1D array
model.add(Flatten()) # Fully Connected
Layer
model.add(Dense(64, activation='relu'))
# Dropout layer
model.add(Dropout(0.2)) #
Final output dense layer

model.add(Dense(10, activation='softmax'))
# Compile the model with sparse_categorical_crossentropy loss
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
# Display model summary
model.summary() Output:
Model: "sequential"

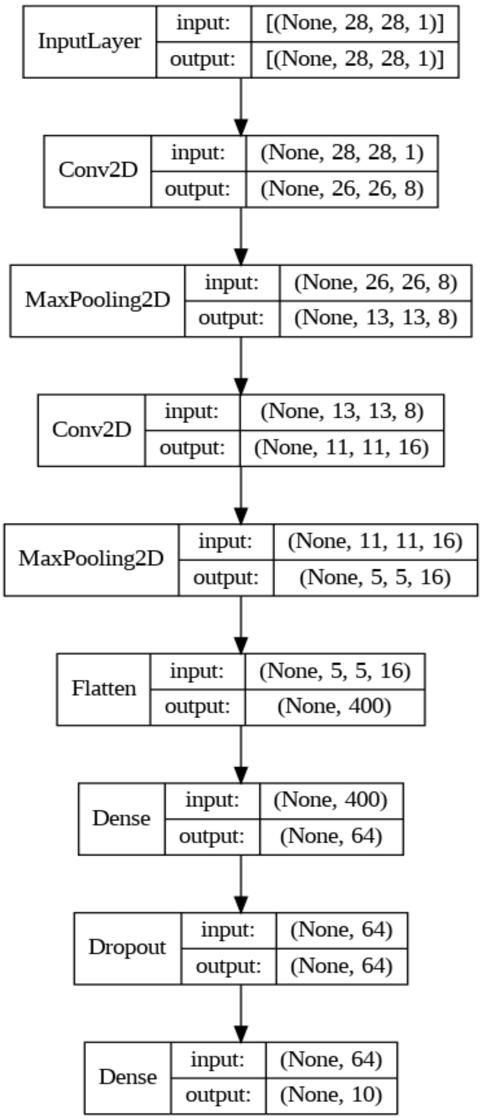
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 8)	80
max_pooling2d (MaxPooling2D)	(None, 13, 13, 8)	0
conv2d_1 (Conv2D)	(None, 11, 11, 16)	1168
max_pooling2d_1(MaxPooling2D)	(None, 5, 5, 16)	0
flatten (Flatten)	(None, 400)	0
dense (Dense)	(None, 64)	25664
dropout (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 10)	650

Total params: 27562 (107.66 KB)
Trainable params: 27562 (107.66 KB) Non-trainable params: 0 (0.00 Byte)

# Conv1: 3x3 kernels, one for each the single channel, 8 such filters and 8 biases
print('Conv1: ',3*3*1*8 + 8)
# Conv2: 3x3 kernels, one for each of the 8 channels, 16 such filters and 16 biases
print('Conv2: ',3*3*8*16 + 16) # input to dense layer print('Flatten:', 5*5*16)
# 400 inputs, 1 bias connected to each of 64 units in dense layer print('Dense1: ',400*64+64)
# 64 inputs, 1 bias connected to each of 10 units in output layer
print('Dense2: ',64*10+10) Output:

Conv1: 80
Conv2: 1168
Flatten: 400
Dense1: 25664
Dense2: 650

Visualize the model from
keras.utils import plot_model
plot_model(model, show_shapes=True, show_layer_names=False) <u>Output:</u>



TRAIN THE MODEL
Validation data=0.2*60000=12000, batch size=128
Number of batches during training are (60000-12000)/128=375

batch_size=128 epochs=10
hist=model.fit(X_train,y_train,epochs=epochs,batch_size=batch_size,verbose=1 ,validation_split=0.2) Output:
Epoch 1/10
375/375 [==============================] - 16s 41ms/step - loss: 0.5030 - accuracy: 0.8499 - val_loss: 0.1272 - val_accuracy: 0.9629 Epoch 2/10
375/375 [==============================] - 14s 37ms/step - loss: 0.1367 - accuracy: 0.9579 - val_loss: 0.0799 - val_accuracy: 0.9762 Epoch 3/10
375/375 [==============================] - 14s 36ms/step - loss: 0.0950 - accuracy: 0.9710 - val_loss: 0.0666 - val_accuracy: 0.9802 Epoch 4/10
375/375 [==============================] - 13s 36ms/step - loss: 0.0793 - accuracy: 0.9755 - val_loss: 0.0582 - val_accuracy: 0.9828 Epoch 5/10
375/375 [==============================] - 17s 47ms/step - loss: 0.0684 - accuracy: 0.9788 - val_loss: 0.0591 - val_accuracy: 0.9812 Epoch 6/10
375/375 [==============================] - 15s 40ms/step - loss: 0.0608 - accuracy: 0.9810 - val_loss: 0.0516 - val_accuracy: 0.9842 Epoch 7/10
375/375 [==============================] - 14s 37ms/step - loss: 0.0542 - accuracy: 0.9828 - val_loss: 0.0486 - val_accuracy: 0.9858 Epoch 8/10
375/375 [==============================] - 14s 36ms/step - loss: 0.0467 - accuracy: 0.9856 - val_loss: 0.0517 - val_accuracy: 0.9862 Epoch 9/10
375/375 [==============================] - 13s 35ms/step - loss: 0.0432 - accuracy: 0.9862 - val_loss: 0.0453 - val_accuracy: 0.9866
Epoch 10/10
375/375 [==============================] - 14s 37ms/step - loss: 0.0417 - accuracy: 0.9868 - val_loss: 0.0459 - val_accuracy: 0.9870

EVALUATE THE MODEL

score = model.evaluate(X_test, y_test, verbose = 0)
print('Test loss:', score[0]) print('Test accuracy:', score[1])
Output: Test loss: 0.034943293780088425

Test accuracy: 0.9876999855041504
make one prediction print('Actual class:',y_test[0]) print('Class Probabilities:')
model.predict(X_test[0].reshape(1,28,28,1)) Output:

```
Actual class: 7
Class Probabilities:
1/1 [==============================] - 0s 115ms/step
array([[1.1689102e-10, 6.6642991e-07, 1.0182861e-05, 3.3980712e-06,
        2.7866585e-09, 7.0861939e-10, 2.2234354e-12, 9.9998355e-01,
        5.1310349e-07, 1.6389805e-06]], dtype=float32)
```

```
import numpy as np
yhat_test = np.argmax(model.predict(X_test),axis=-1); print(yhat_test[0:10]);
print(y_test[0:10]); Output:
313/313 [==============================] - 1s 5ms/step
[7 2 1 0 4 1 4 9 5 9]
[7 2 1 0 4 1 4 9 5 9]
```

```
from sklearn.metrics import accuracy_score print('Accuracy:')
print(float(accuracy_score(y_test, yhat_test))*100,'%') Output:
Accuracy: 98.77
%
```

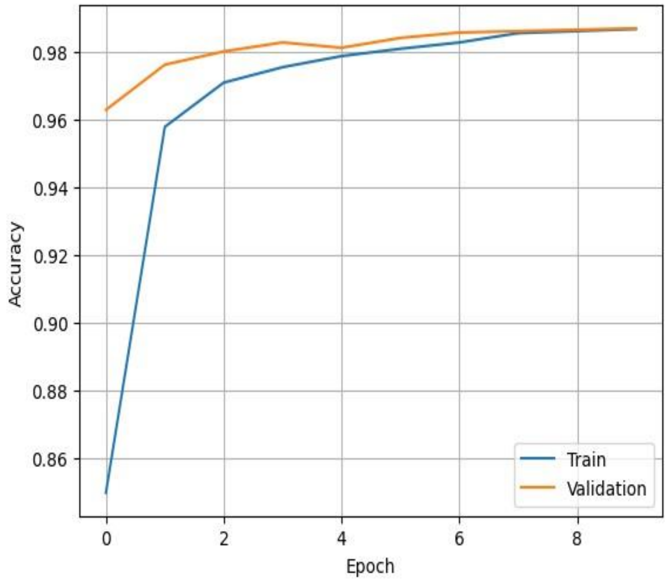
```
from sklearn.metrics import confusion_matrix
print('Confusion Matrix:')
print(confusion_matrix(y_test, yhat_test)) Output:
Confusion Matrix:
[[ 974    1    0    0    0    1    3    1    0    0]
 [    0 1130    2    0    0    0    3    0    0    0]
 [    1    2 1023    0    1    0    0    4    1    0]
 [    0    0    4  993    0    8    0    3    2    0]
 [    1    0    0    0  977    0    2    0    0    2]
 [    2    0    0    3    0  884    2    0    1    0]
 [    5    2    0    0    1    5  945    0    0    0]
 [    1    4    8    1    0    0    0 1013    0    1]
 [    2    0    4    3    1    3    2    2  951    6]
 [    3    4    1    1    8    2    0    3    0  987]]
```

PLOT LEARNING CURVES

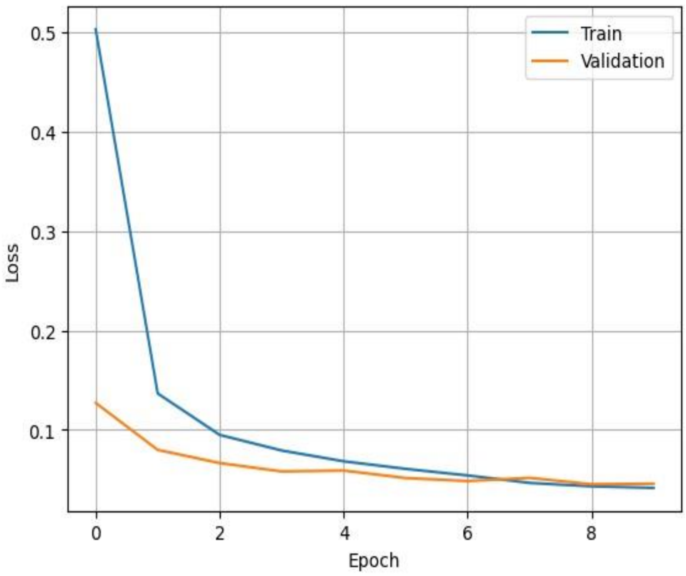
hist.history.keys() Output:

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

# Plot Accuracy vs epochs (DIY) plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy']) plt.xlabel('Epoch') plt.ylabel('Accuracy')
plt.legend(['Train', 'Validation']) plt.show() Output:



# Plot Loss vs epochs (DIY)
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.xlabel('Epoch') plt.ylabel('Loss')
plt.legend(['Train', 'Validation'])
plt.show() Output:



CODE OF MODEL 2:

Changes made: The number of filters in Convolutional layer 1 has been changed from 8 to 16 and that in convolutional layer 2 has been changed from 16 to 32.

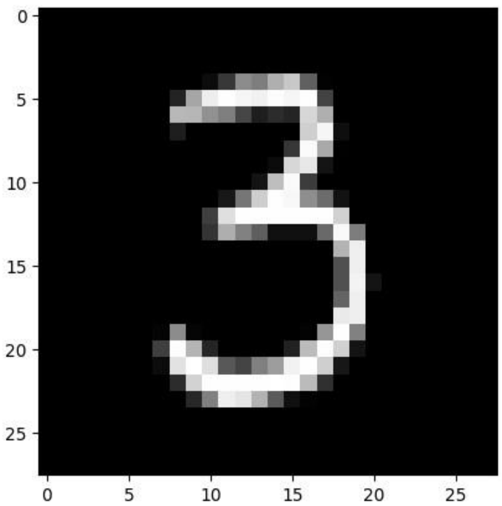
```
from tensorflow import keras
from keras.datasets import mnist     # MNIST dataset is included in Keras
(X_train, y_train), (X_test, y_test) = mnist.load_data()
print("X_train shape", X_train.shape) print("y_train
shape", y_train.shape) print("X_test shape",
X_test.shape) print("y_test shape", y_test.shape)
```

Output:

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-
kerasdatasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
X_train shape (60000, 28, 28)
y_train shape (60000,) X_test
shape (10000, 28, 28)
y_test shape (10000,)
```

```
# Visualize any random image
import matplotlib.pyplot as plt i=50;
plt.imshow(X_train[i], cmap='gray');
```

Output:



FORMATTING THE INPUT

```
# Single-channel input data (grey-scale)
# First apply convolutions then flatten
X_train = X_train.reshape(60000, 28, 28, 1) # single-channel input
X_test = X_test.reshape(10000, 28, 28, 1)
```

X_train = X_train.astype('float32')   # change integers to 32-bit floating point numbers
X_test = X_test.astype('float32')
X_train /= 255   # min-max normalization X_test /= 255
print("Training matrix shape", X_train.shape) print("Testing matrix shape", X_test.shape) Output:
Training matrix shape (60000, 28, 28, 1)
Testing matrix shape (10000, 28, 28, 1)

CONVOLUTIONAL NEURAL NETWORK

from keras import backend as K from keras import __version__
print('Using Keras version:', __version__, 'backend:', K.backend()) Output:
Using Keras version: 2.15.0 backend: tensorflow

# import cnn layers
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Dense
import tensorflow as tf
model = Sequential()   # Linear stacking of layers
# Convolution Layer 1: 16 filters, kernel size 3x3, relu activation, valid padding, stride 1 model.add(Conv2D(16, kernel_size=(3, 3), activation='relu', input_shape=(28, 28, 1), padding='valid', strides=1)) model.add(MaxPooling2D(pool_size=(2, 2), strides=2))
# Convolution Layer 2: 32 filters, kernel size 3x3, relu activation, valid padding, stride 1
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', padding='valid', strides=1))
model.add(MaxPooling2D(pool_size=(2, 2), strides=2))
# Flatten final feature matrix into a 1D array
model.add(Flatten()) # Fully Connected Layer
model.add(Dense(64, activation='relu'))
# Dropout layer
model.add(Dropout(0.2)) # Final output dense layer
model.add(Dense(10, activation='softmax'))

# Compile the model with sparse_categorical_crossentropy loss
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.002), loss='sparse_categorical_crossentropy', metr ics=['accuracy'])
model.summary() Output:
Model: "sequential"

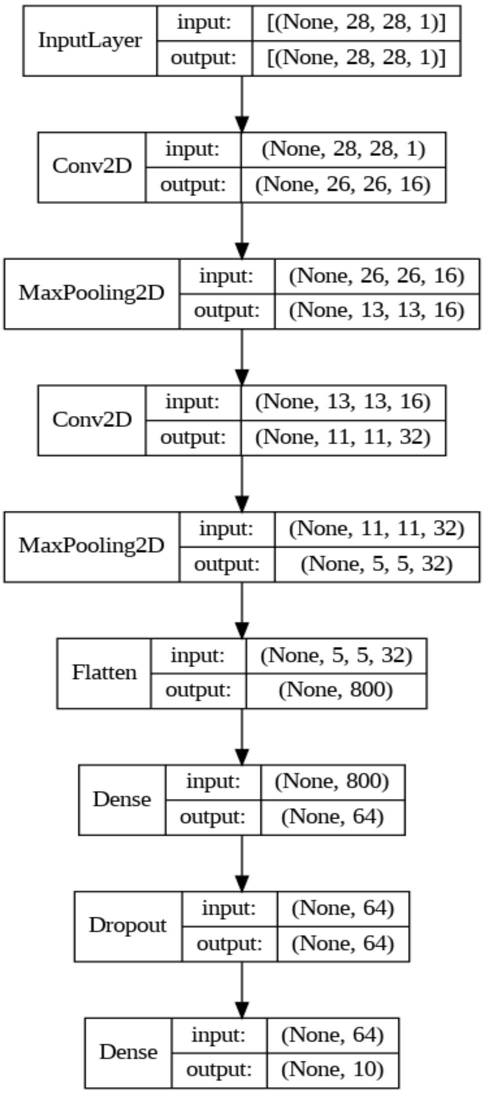
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 16)	160
max_pooling2d (MaxPooling2D)	(None, 13, 13, 16)	0
conv2d_1 (Conv2D)	(None, 11, 11, 32)	4640
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 32)	0
flatten (Flatten)	(None, 800)	0
dense (Dense)	(None, 64)	51264
dropout (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 10)	650

Total params: 56714 (221.54 KB)
Trainable params: 56714 (221.54 KB)
Non-trainable params: 0 (0.00 Byte)

# Conv1: 3x3 kernels, one for each the single channel, 8 such filters and 8 biases
print('Conv1: ',3*3*1*16 + 16)
# Conv2: 3x3 kernels, one for each of the 8 channels, 16 such filters and 16 biases print('Conv2: ',3*3*16*32 + 32)
# input to dense layer print('Flatten:',
5*5*32)
400 inputs, 1 bias connected to each of 64 units in dense layer print('Dense1: ',400*64+64)
64 inputs, 1 bias connected to each of 10 units in output layer
print('Dense2: ',64*10+10) Output:
Conv1: 160
Conv2: 4640

Flatten: 800
Dense1: 25664
Dense2: 650

Visualize the model from
keras.utils import plot_model
plot_model(model, show_shapes=True, show_layer_names=False) Output:



TRAIN THE MODEL

batch_size=128 epochs=10
hist=model.fit(X_train,y_train,epochs=epochs,batch_size=batch_size,verbose=1,validation_split=0.2) Output:

Epoch 1/10
375/375 [==============================] - 43s 101ms/step - loss: 0.2848 - accuracy: 0.9146 - val_loss: 0.0863 - val_accuracy: 0.9743 Epoch 2/10
375/375 [==============================] - 20s 54ms/step - loss:

0.0897 - accuracy: 0.9729 - val_loss: 0.0676 - val_accuracy: 0.9787 Epoch 3/10
375/375 [==============================] - 22s 59ms/step - loss: 0.0625 - accuracy: 0.9808 - val_loss: 0.0458 - val_accuracy: 0.9866 Epoch 4/10
375/375 [==============================] - 21s 57ms/step - loss: 0.0497 - accuracy: 0.9841 - val_loss: 0.0427 - val_accuracy: 0.9881
Epoch 5/10
375/375 [==============================] - 22s 60ms/step - loss: 0.0416 - accuracy: 0.9869 - val_loss: 0.0398 - val_accuracy: 0.9882 Epoch 6/10
375/375 [==============================] - 22s 58ms/step - loss: 0.0361 - accuracy: 0.9881 - val_loss: 0.0394 - val_accuracy: 0.9880 Epoch 7/10
375/375 [==============================] - 23s 61ms/step - loss: 0.0317 - accuracy: 0.9901 - val_loss: 0.0428 - val_accuracy: 0.9880 Epoch 8/10
375/375 [==============================] - 20s 54ms/step - loss: 0.0277 - accuracy: 0.9906 - val_loss: 0.0420 - val_accuracy: 0.9895 Epoch 9/10
375/375 [==============================] - 23s 60ms/step - loss: 0.0251 - accuracy: 0.9918 - val_loss: 0.0407 - val_accuracy: 0.9887
Epoch 10/10
375/375 [==============================] - 22s 59ms/step - loss: 0.0228 - accuracy: 0.9924 - val_loss: 0.0415 - val_accuracy: 0.9889

EVALUATE MODEL

score = model.evaluate(X_test, y_test, verbose = 0)
print('Test loss:', score[0]) print('Test accuracy:', score[1]) <u>Output:</u>
Test loss: 0.038076359778642654
Test accuracy: 0.9897000193595886

# make one prediction print('Actual class:',y_test[0]) print('Class Probabilities:')
model.predict(X_test[0].reshape(1,28,28,1))
<u>Output:</u> Actual class: 7 Class Probabilities:

```
1/1 [==============================] - 0s 126ms/step
array([[2.5885712e-13, 8.7241402e-11, 1.1904942e-10, 6.3510042e-09,
        2.3912567e-14, 7.2296216e-13, 2.7762177e-19, 1.0000000e+00,
        2.3823884e-14, 2.0128242e-12]], dtype=float32)
```

```
import numpy as np
yhat_test = np.argmax(model.predict(X_test),axis=-1);
print(yhat_test[0:10]); print(y_test[0:10]);
```
Output:
```
313/313 [==============================] - 2s 6ms/step
[7 2 1 0 4 1 4 9 5 9]
[7 2 1 0 4 1 4 9 5 9]
```

```
from sklearn.metrics import accuracy_score print('Accuracy:')
print(float(accuracy_score(y_test, yhat_test))*100,'%')
```
Output:
```
Accuracy: 98.97
%
```

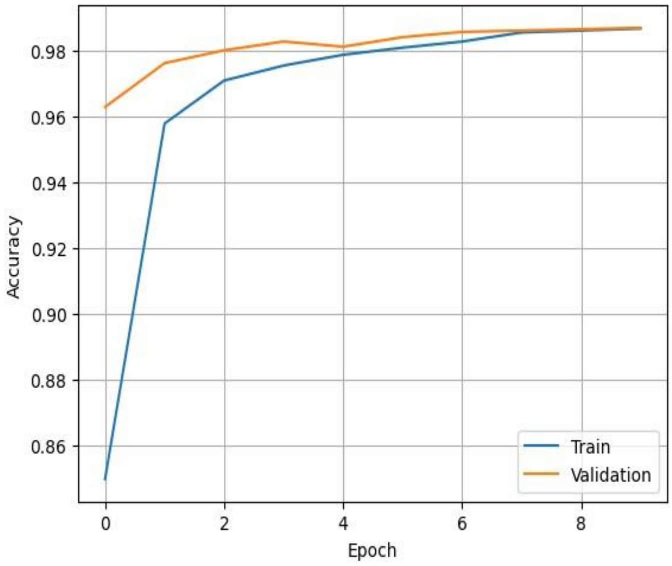
```
from sklearn.metrics import confusion_matrix
print('Confusion Matrix:')
print(confusion_matrix(y_test, yhat_test))
```
Output:
```
Confusion Matrix:
[[ 975    0    0    0    2    0    2    1    0    0]
 [   0 1126    1    5    0    1    0    0    2    0]
 [   1    0 1023    0    0    0    0    7    1    0]
 [   0    0    3 1003    0    1    0    3    0    0]
 [   0    0    0    0  980    0    0    0    0    2]
 [   1    0    0    8    0  877    1    1    1    3]
 [   6    2    0    0    4    4  939    0    3    0]
 [   0    2    5    0    0    0    0 1020    0    1]
 [   2    0    1    7    0    1    0    3  959    1]
 [   0    0    1    2    3    1    0    6    1  995]]
```

PLOT LEARNING CURVES

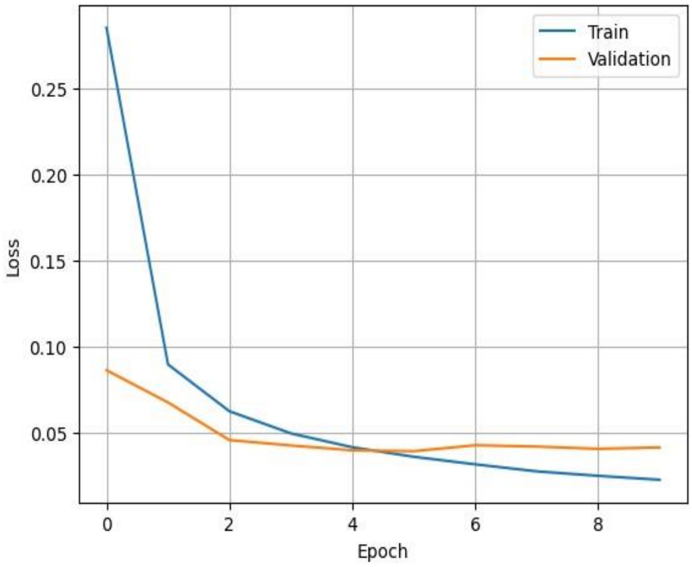
```
hist.history.keys()
```
Output:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

```
# Plot Accuracy vs epochs (DIY)
plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
```

plt.xlabel('Epoch')
plt.ylabel('Accuracy') plt.grid()
plt.legend(['Train', 'Validation'])
plt.show() Output:



# Plot Loss vs epochs (DIY)
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.xlabel('Epoch')
plt.ylabel('Loss') plt.grid()
plt.legend(['Train', 'Validation'])
plt.show() Output:

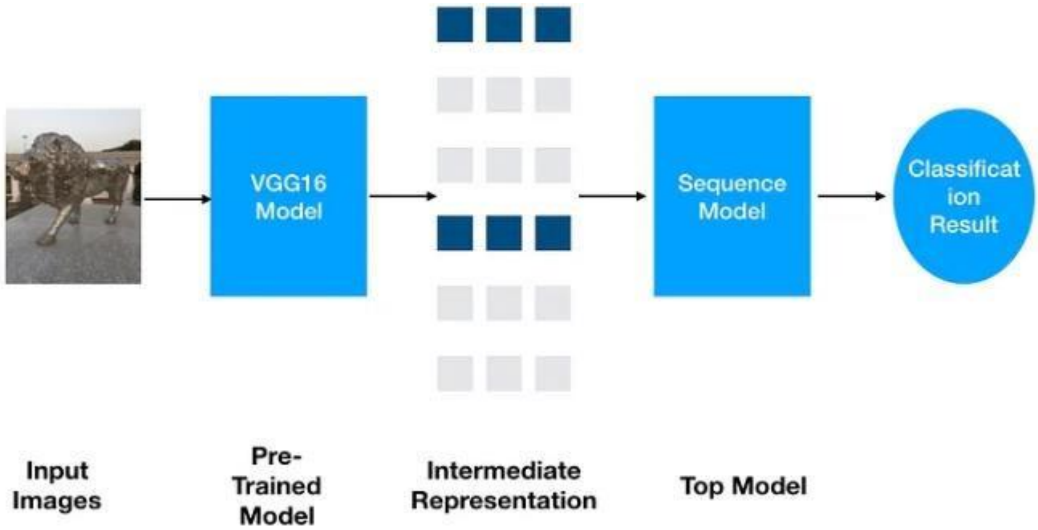


EXPERIMENT 4: Implement an image classification task using pre-trained models like VGGNet, InceptionNet and ResNet and compare the results.

DESCRIPTION:

Dataset description: 39 different classes of plant leaf and background images are available. The dataset contains 61,486 images. We resize the input image to 128x128. Training set: 70%, validation set: 20%, test set: 10%

Transfer learning is a technique that works in image classification and natural language processing tasks. It is about leveraging feature representations from a pre-trained model so we don't have to train a new model from scratch. The weights obtained from the pre-trained models can be reused in making predictions on new tasks or integrated into the process of training a new model. It leads to lower training time and lower generalization error. It is very useful when we have a small training dataset.



ImageNet is a project aimed at labelling and categorizing images into almost 22,000 separate object categories for the purpose of computer vision research. Here, we are referring to ImageNet Large Scale Visual Recognition Challenge or ILSVRC for short. The goal of the challenge is to train a model that can correctly classify an input image into 1000 separate object categories. Models are trained on 1.2 million training images with another 50,000 images for validation and 100,000 images for testing.

VGGNet has a total of 16 layers that has some weights. Only convolution and pooling layers are used. Always used as 3x3 kernel for convolution. It consists of 138 million parameters and has an accuracy of 92.7%. Another version VGG 19 has a total of 19 layers with weights. It is a very good deep learning architecture for benchmarking on any particular task.

InceptionNet is a convolutional neural network (CNN) architecture that Google developed to improve upon the performance of previous CNNs on the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) benchmark. It uses "inception modules" that apply a combination of 1x1, 3x3, and 5x5 convolutions on the input data and utilizes auxiliary classifiers to improve performance. InceptionNet won the 2014 ILSVRC competition and has been used in various applications, including image classification, object detection, and image segmentation.

ResNets learn residual functions with reference to the layer inputs, instead of learning unreferenced functions. Instead of hoping each few stacked layers directly fit a desired underlying mapping, residual nets let these layers fit a residual mapping. They stack residual blocks on top of each other to form network: e.g. a ResNet-50 has fifty layers using these blocks.

CODE:

```
import keras import numpy
as np from keras import
Input from keras import
models from keras import
layers from keras import
optimizers from
keras.models import Model
from keras import
applications # from keras
import backend as k import
matplotlib.pyplot as plt from
keras.optimizers import
SGD, Adam
from keras.layers import  Dense, Flatten, Conv2D, MaxPooling2D
# from keras.preprocessing import image from
keras.models import Sequential, Model
from keras.preprocessing.image import ImageDataGenerator
from keras.layers import Dropout, Flatten, Dense, GlobalAveragePooling2D
#from keras.callbacks import ModelCheckpoint, LearningRateScheduler,
TensorBoard, EarlyStopping
```

Output:

2024-02-06 04:03:03.586262: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations. To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.

Loading the Training and Testing Data and Defining the Basic Parameters

About data: https://data.mendeley.com/datasets/tywbtsjrjv/1

We are resizing the input image to 128x128

In the dataset : Training Set : 70% Validation Set : 20% Test Set : 10%

!unzip "plant_villa ge.zip" -d "plant_village"

<u>Output:</u> Archive: plant_village.zip

```
creating:          plant_village/plant_village/
creating: plant_village/plant_village/test/
   creating: plant_village/plant_village/test/Apple_Frogeye_Spot/
  inflating: plant_village/plant_village/test/Apple_Frogeye_Spot/4231e86c-5f4c-439c-be0f-1fa0274581c6___JR_FrgE.S 3067(1).JPG
  inflating: plant_village/plant_village/test/Apple_Frogeye_Spot/4231e86c-5f4c-439c-be0f-1fa0274581c6___JR_FrgE.S 3067.JPG
.
.
.
inflating: plant_village/plant_village/val/Apple___healthy/feb2c7d8-0cfd-4eb1-86d0-78a893dd8943___RS_HL 7866.JPG
```

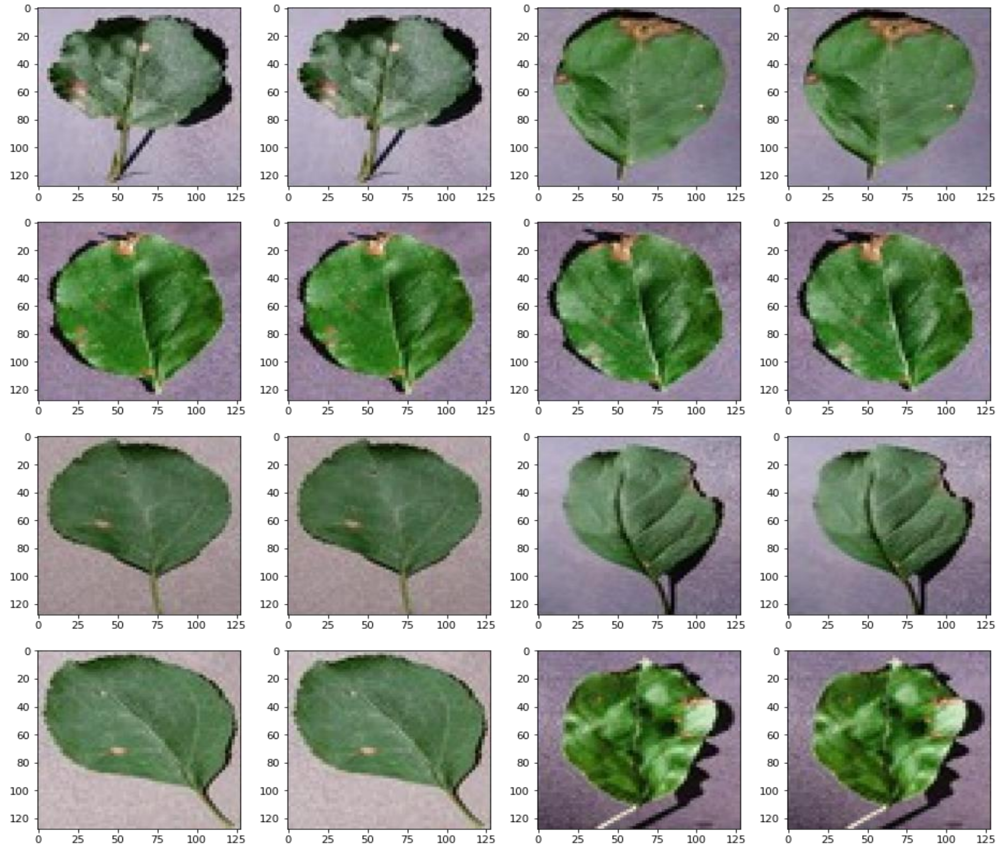
```
# Normalize training and validation data in the range of 0 to 1
train_datagen = ImageDataGenerator(rescale=1/255) # vertical_flip=True,
# horizontal_flip=True, # height_shift_range=0.1,# width_shift_range=0.1
validation_datagen = ImageDataGenerator(rescale=1/255) test_datagen =
ImageDataGenerator(rescale=1/255) # Read the training sample and set
the batch size train_generator = train_datagen.flow_from_directory(
'plant_village/plant_village/train/',      target_size=(128,   128),
         batch_size=16, class_mode='categorical')
# Read Validation data from directory and define target size with batch size
validation_generator = validation_datagen.flow_from_directory(
       'plant_village/plant_village/val/', target_size=(128, 128), batch_size=16,
class_mode='categorical',          shuffle=False)          test_generator          =
test_datagen.flow_from_directory(
       'plant_village/plant_village/test/', target_size=(128, 128), batch_size=1,
class_mode='categorical', shuffle=False)
```

<u>Output:</u>

Found 3033 images belonging to 4 classes.
Found 635 images belonging to 4 classes. Found
566 images belonging to 4 classes.

```
plt.figure(figsize=(16, 16)) for i in
range(1, 17):   plt.subplot(4, 4, i)
img, label = test_generator.next()
```

# print(img.shape) # print(label)
plt.imshow(img[0]) plt.show()
Output:



img, label = test generator.next()
Img[0].shape
Output: (128, 128, 3)

ImageNet

from tensorflow.keras.applications.vgg16 import VGG16
Loading VGG16 model base_model = VGG16(weights="imagenet", include_top=False, input_shape= (128, 128, 3)) # Include_top = False means excluding the model fully connected layers
base_model.trainable = False ## Not trainable weights
base_model.summary() Output:
2024-01-30 05:23:38.723565: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1639] Created device /job:localhost/replica:0/task:0/device:GPU:0 with 678 MB memory: -> device: 0, name: NVIDIA A100-SXM4-40GB MIG 1g.5gb, pci bus id: 0000:87:00.0, compute capability: 8.0

Downloading data from https://storage.googleapis.com/tensorflow/kerasapplications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 [==============================] - 7s 0us/step
Model: "vgg16"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 128, 128, 3)]	0
block1_conv1 (Conv2D)	(None, 128, 128, 64)	1792
block1_conv2 (Conv2D)	(None, 128, 128, 64)	36928
block1_pool (MaxPooling2D)	(None, 64, 64, 64)	0
block2_conv1 (Conv2D)	(None, 64, 64, 128)	73856
block2_conv2 (Conv2D)	(None, 64, 64, 128)	147584
block2_pool (MaxPooling2D)	(None, 32, 32, 128)	0
block3_conv1 (Conv2D)	(None, 32, 32, 256)	295168
block3_conv2 (Conv2D)	(None, 32, 32, 256)	590080
block3_conv3 (Conv2D)	(None, 32, 32, 256)	590080
block3_pool (MaxPooling2D)	(None, 16, 16, 256)	0
block4_conv1 (Conv2D)	(None, 16, 16, 512)	1180160
block4_conv2 (Conv2D)	(None, 16, 16, 512)	2359808
block4_conv3 (Conv2D)	(None, 16, 16, 512)	2359808
block4_pool (MaxPooling2D)	(None, 8, 8, 512)	0
block5_conv1 (Conv2D)	(None, 8, 8, 512)	2359808

block5_conv2 (Conv2D)   (None, 8, 8, 512)   2359808

block5_conv3 (Conv2D)   (None, 8, 8, 512)   2359808

block5_pool (MaxPooling2D) (None, 4, 4, 512)   0
===
Total params: 14714688 (56.13 MB)
Trainable params: 0 (0.00 Byte)
Non-trainable params: 14714688 (56.13 MB)

Adding top layers according to number of classes in our data [ ]:
flatten_layer = layers.GlobalAveragePooling2D() prediction_layer
= layers.Dense(4, activation='softmax')
model = models.Sequential([base_model, flatten_layer, prediction_layer])
model.summary() Output:
Model: "sequential"

____
Layer (type)   Output Shape   Param #
===
vgg16 (Functional)   (None, 4, 4, 512)   14714688

global_average_pooling2d (None, 512)   0
(GlobalAveragePooling2D)

dense (Dense)   (None, 4)   2052
===
Total params: 14716740 (56.14 MB)
Trainable params: 2052 (8.02 KB)
Non-trainable params: 14714688 (56.13 MB)

____
Training
model.compile(optimizer=Adam(learning_rate=0.001),
loss='categorical_crossentropy', metrics=['acc'])
# Train the model
history = model.fit(train_generator,
    steps_per_epoch=train_generator.samples/train_generator.batch_size,
epochs=30,validation_data=validation_generator,

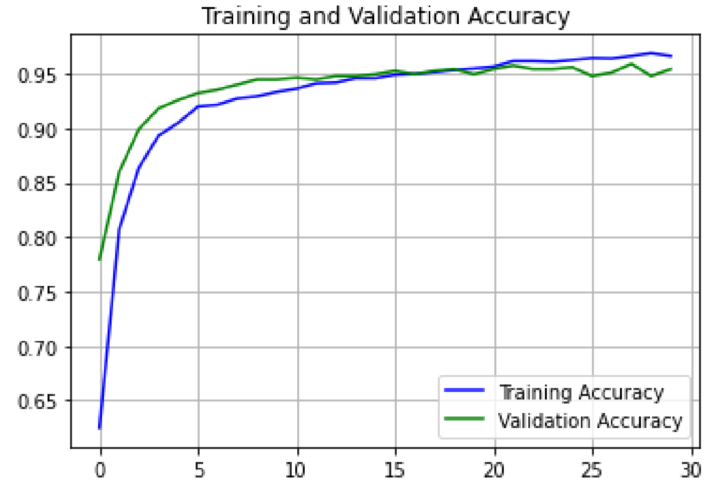
validation_steps=validation_generator.samples/validation_generator.batch_size, verbose=1) Output:
Epoch 1/30
189/189 [==============================] - 34s 174ms/step - loss: 0.9795 - acc: 0.6248 - val_loss: 0.7192 - val_acc: 0.7795
Epoch 2/30
189/189 [==============================] - 33s 173ms/step - loss: 0.6447 - acc: 0.8071 - val_loss: 0.5273 - val_acc: 0.8598
Epoch 3/30
189/189 [==============================] - 33s 173ms/step - loss: 0.4981 - acc: 0.8638 - val_loss: 0.4285 - val_acc: 0.8992
Epoch 4/30
189/189 [==============================] - 33s 173ms/step - loss: 0.4157 - acc: 0.8932 - val_loss: 0.3640 - val_acc: 0.9181 Epoch 5/30
189/189 [==============================] - 32s 171ms/step - loss: 0.3620 - acc: 0.9050 - val_loss: 0.3240 - val_acc: 0.9260
.
.
Epoch 30/30
189/189 [==============================] - 33s 175ms/step - loss: 0.1206 - acc: 0.9664 - val_loss: 0.1378 - val_acc: 0.9543

Saving the model
model.save("VGG16_plant_deseas.h5")
print("Saved model to disk") Output:
Saved model to disk

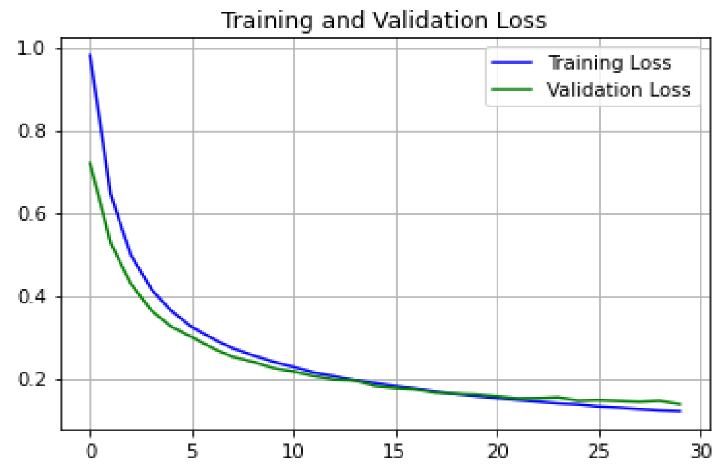
Loading the model
model = models.load_model('VGG16_plant_deseas.h5')
print("Model is loaded") Output: Model is loaded

Visualization of training over epoch
train_acc = history.history['acc'] val_acc = history.history['val_acc'] train_loss = history.history['loss'] val_loss = history.history['val_loss'] epochs = range(len(train_acc))

plt.plot(epochs, train_acc, 'b', label='Training Accuracy')
plt.plot(epochs, val_acc, 'g', label='Validation Accuracy')
plt.title('Training and Validation Accuracy') plt.grid()
plt.legend() plt.figure() plt.show()
plt.plot(epochs, train_loss, 'b', label='Training Loss')
plt.plot(epochs, val_loss, 'g', label='Validation Loss')
plt.title('Training and Validation Loss') plt.grid()
plt.legend() plt.show() Output:



<Figure size 432x288 with 0 Axes>



Performance measure

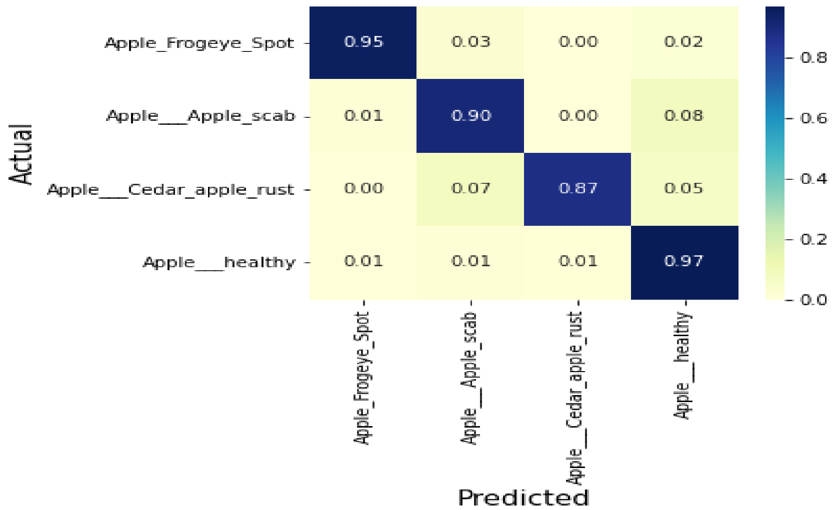
# Get the filenames from the generator
fnames = test_generator.filenames #
Get the ground truth from generator
ground_truth = test_generator.classes
# Get the label to class mapping from the generator label2index
= test_generator.class_indices
Getting the mapping from class index to class label idx2label =
dict((v,k) for k,v in label2index.items()) # Get the predictions from the

model using the generator
predictions=model.predict_generator(test_generator,
steps=test_generator.samples/test_generator.batch_size,verbose=1)
predicted_classes = np.argmax(predictions,axis=1) errors =
np.where(predicted_classes != ground_truth)[0] print("No of errors =
{}/{}".format(len(errors),test_generator.samples)) Output:
566/566 [==============================] - 18s 32ms/step No
of errors = 34/566

accuracy = ((test_generator.samples-len(errors))/test_generator.samples) * 100
accuracy
Output: 93.99293286219081

!pip install seaborn Output:
Looking in indexes: https://pypi.org/simple, https://pypi.ngc.nvidia.com
Collecting seaborn
  Downloading seaborn-0.13.2-py3-none-any.whl (294 kB)
    |████████████████████████████████| 294 kB 2.1 MB/s
eta 0:00:01
Requirement already satisfied: pandas>=1.2 in
/opt/conda/lib/python3.8/sitepackages (from seaborn) (1.3.5) Installing collected
packages: seaborn Successfully installed seaborn-0.13.2

from sklearn.metrics import confusion_matrix
import seaborn as sns import numpy as np
from matplotlib import pyplot as plt
cm = confusion_matrix(y_true=ground_truth, y_pred=predicted_classes)
cm = np.array(cm) # Normalise
cmn = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis] fig,
ax = plt.subplots(figsize=(5,4))
sns.heatmap(cmn, annot=True, fmt='.2f', xticklabels=label2index,
yticklabels=label2index, cmap="YlGnBu") plt.ylabel('Actual', fontsize=15)
plt.xlabel('Predicted', fontsize=15) plt.show(block=False) Output:



from sklearn.metrics import classification_report
print(classification_report(ground_truth,predicted_classes,
target_names=label2index)) <u>Output:</u>

	precision	recall	f1-score	support
Apple_Frogeye_Spot	0.96	0.95	0.96	103
Apple___Apple_scab	0.92	0.90	0.91	134
Apple___Cedar_apple_rust	0.94	0.87	0.91	55
Apple___healthy	0.94	0.97	0.95	274
accuracy			0.94	566
macro avg	0.94	0.92	0.93	566
weighted avg	0.94	0.94	0.94	566

<u>InceptionNet</u>

from keras import applications
Loading InceptionV3 model
base_model = applications.InceptionV3(weights="imagenet", include_top=False, input_shape= (128, 128, 3)) base_model.trainable = False ## Not trainable weights base_model.summary() <u>Output:</u>

Downloading data from https://storage.googleapis.com/tensorflow/kerasapplications/inception_v3 /inception_v3_weights_tf_dim_ordering_tf_kernels_n otop.h5
87910968/87910968 [==============================] - 10s 0us/step
Model: "inception_v3"

Layer (type)	Output Shape	Param #	Connected to
input_2 (InputLayer)	[(None, 128, 128, 3)]	0	[]

```
 conv2d (Conv2D)            (None, 63, 63, 32)          864       ['input_2[0][0]']

 batch_normalization (Batch  (None, 63, 63, 32)          96       ['conv2d[0][0]']
 Normalization)

 activation (Activation)  (None, 63, 63, 32)            0
batch_normalization[0][0]']

 conv2d_1 (Conv2D)          (None, 61, 61, 32)          9216      ['activation[0][0]']
.
.
 mixed9_1 (Concatenate)    (None, 2, 2, 768)       0        ['activation_87[0][0]',
                                                         'activation_88[0][0]']
concatenate_1 (Concatenate)  (None, 2, 2, 768)   0        ['activation_91[0][0]',
                                                            'activation_92[0][0]']

 activation_93 (Activation) (None, 2, 2, 192)   0
['batch_normalization_93[0][0]']

 mixed10 (Concatenate)  (None, 2, 2, 2048)    0        ['activation_85[0][0]',
                                                     'mixed9_1[0][0]',
                                                     'concatenate_1[0][0]',
                                                     'activation_93[0][0]']
==================================================================
Total params: 21802784 (83.17 MB)
Trainable params: 0 (0.00 Byte)
Non-trainable params: 21802784 (83.17 MB)
__________________________________________________________________
```

```
# include GlobalAveragePooling2D
flatten_layer = layers.GlobalAveragePooling2D()
# include final Dense layer
prediction_layer = layers.Dense(4, activation='softmax')
model = models.Sequential([base_model, flatten_layer, prediction_layer])
model.summary()
```

<u>Output:</u>

```
Model: "sequential_1"
__________________________________________________________________
 Layer (type)                    Output Shape           Param #
==================================================================
```

```
 inception_v3 (Functional)    (None, 2, 2, 2048)       21802784

 global_average_pooling2d_1  (None, 2048)           0
  (GlobalAveragePooling2D)

 dense_1 (Dense)               (None, 4)             8196
=================================================================
Total params: 21810980 (83.20 MB)
Trainable params: 8196 (32.02 KB)
Non-trainable params: 21802784 (83.17 MB)
_________________________________________________________________
```

```
model.compile(optimizer = Adam(learning_rate = 0.001),
loss='categorical_crossentropy', metrics=['acc']) history =
model.fit(train_generator,
     steps_per_epoch=train_generator.samples/train_generator.batch_size,
epochs=30, validation_data=validation_generator,
 validation_steps=validation_generator.samples/validation_generator.batch_size,
verbose=1)
```

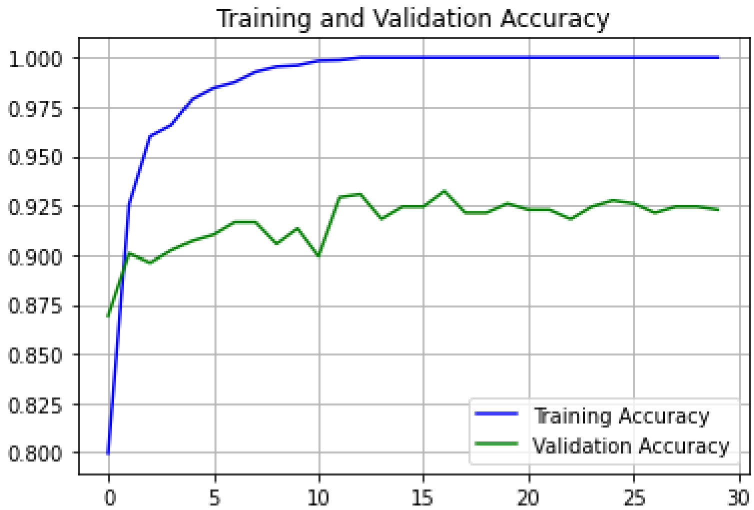
Output:

```
Epoch 1/30
189/189 [==============================] - 21s 100ms/step - loss:
0.5765 - acc: 0.7995 - val_loss: 0.3564 - val_acc: 0.8693 Epoch
2/30
189/189 [==============================] - 17s 89ms/step - loss:
0.2158 - acc: 0.9255 - val_loss: 0.2712 - val_acc: 0.9008 Epoch
3/30
189/189 [==============================] - 17s 92ms/step - loss:
0.1368 - acc: 0.9601 - val_loss: 0.2954 - val_acc: 0.8961
.
.
Epoch 29/30
189/189 [==============================] - 18s 95ms/step - loss:
0.0022 - acc: 1.0000 - val_loss: 0.2422 - val_acc: 0.9244
Epoch 30/30
189/189 [==============================] - 18s 97ms/step - loss:
0.0019 - acc: 1.0000 - val_loss: 0.2410 - val_acc: 0.9228
```

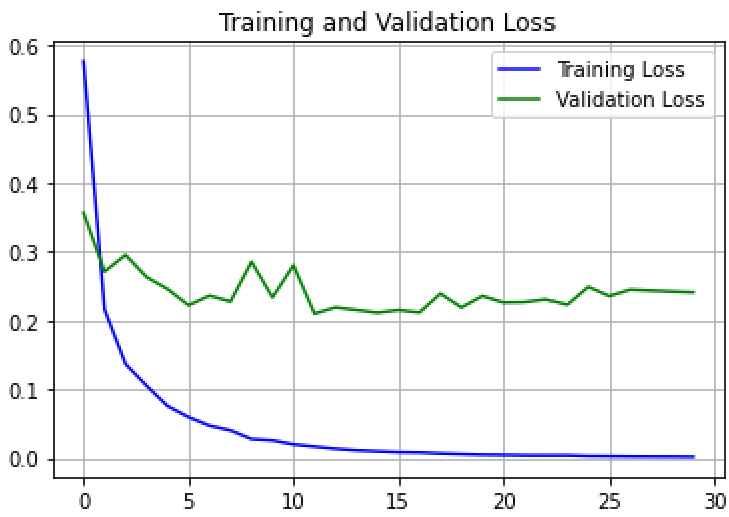
```
model.save("InceptionV3_plant_disease.h5") print("Saved
model to disk")
```

model = models.load_model('InceptionV3_plant_disease.h5')
print("Model is loaded") Output: Saved model to disk
Model is loaded

train_acc = history.history['acc']
val_acc = history.history['val_acc']
train_loss = history.history['loss']
val_loss = history.history['val_loss'] #
Display loss/accuracies vs epochs
epochs = range(len(train_acc))
plt.plot(epochs, train_acc, 'b', label='Training Accuracy') plt.plot(epochs, val_acc, 'g', label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.grid() plt.legend() plt.figure() plt.show()
plt.plot(epochs, train_loss, 'b', label='Training Loss')
plt.plot(epochs, val_loss, 'g', label='Validation Loss')
plt.title('Training and Validation Loss') plt.grid()
plt.legend() plt.show() Output:

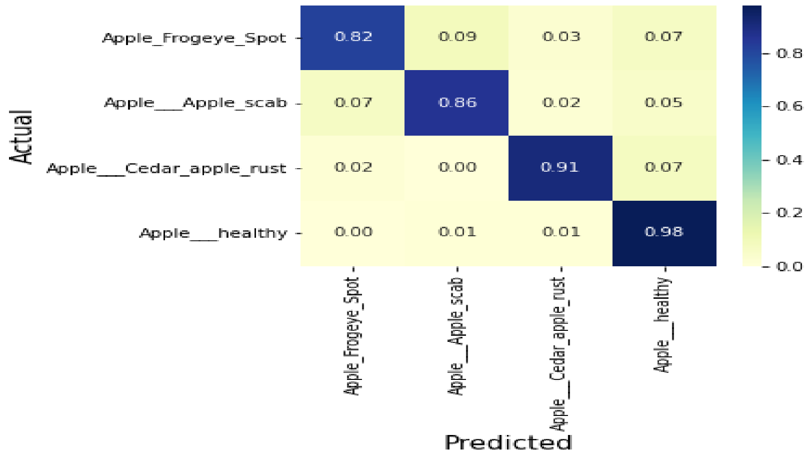


<Figure size 432x288 with 0 Axes>



Get the filenames from the generator
#fnames = test_generator.filenames #
Get the ground truth from generator
ground_truth = test_generator.classes
# Get the label to class mapping from the generator label2index
= test_generator.class_indices
# Get the predictions from the model using the generator
predictions = model.predict_generator(test_generator, steps=test_generator.samples/test_generator.batch_size,verbose=1)
predicted_classes = np.argmax(predictions,axis=1) errors =
np.where(predicted_classes != ground_truth)[0]
print("No of errors = {}/{}".format(len(errors),test_generator.samples)) Output:
566/566 [==============================] - 9s 14ms/step No
of errors = 49/566
accuracy = ((test_generator.samples-len(errors))/test_generator.samples) * 100
accuracy
Output: 91.34275618374559

from sklearn.metrics import confusion_matrix
import seaborn as sns import numpy as np
from matplotlib import pyplot as plt
cm = confusion_matrix(y_true=ground_truth, y_pred=predicted_classes)
cm = np.array(cm) # Normalise
cmn = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis] fig,
ax = plt.subplots(figsize=(5,4))
sns.heatmap(cmn, annot=True, fmt='.2f', xticklabels=label2index,
yticklabels=label2index, cmap="YlGnBu") plt.ylabel('Actual', fontsize=15)
plt.xlabel('Predicted', fontsize=15) plt.show(block=False) Output:



from sklearn.metrics import classification_report
print(classification_report(ground_truth,predicted_classes, target_names=label2index))

<u>Output:</u>

	precision	recall	f1-score	support
Apple_Frogeye_Spot	0.89	0.82	0.85	103
Apple___Apple_scab	0.91	0.86	0.88	134
Apple___Cedar_apple_rust	0.85	0.91	0.88	55
Apple___healthy	0.94	0.98	0.96	274
accuracy			0.91	566
macro avg	0.90	0.89	0.89	566
weighted avg	0.91	0.91	0.91	566

<u>ResNet</u>

from keras import applications ##
Loading ResNet50 model
base_model =
applications.ResNet50(weights="
imagenet", include_top=False,
input_shape= (128, 128, 3))
base_model.trainable = False ## Not trainable weights
base_model.summary() <u>Output:</u>

Downloading data from https://storage.googleapis.com/tensorflow/kerasapplications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94765736/94765736 [==============================] - 10s 0us/step

```
Model: "resnet50"
__________________________________________________________________________________________________
 Layer (type)                Output Shape                 Param #   Connected to
==================================================================================================
 input_3 (InputLayer)            [(None, 128, 128, 3)]       0        []

 conv1_pad (ZeroPadding2D)   (None, 134, 134, 3)         0        ['input_3[0][0]']

 conv1_conv (Conv2D)                (None, 64, 64, 64)      9472
['conv1_pad[0][0]']

 conv1_bn(BatchNormalization) (None, 64, 64, 64)   256
['conv1_conv[0][0]']

 conv1_relu (Activation)        (None, 64, 64, 64)          0        ['conv1_bn[0][0]']

 pool1_pad (ZeroPadding2D)    (None, 66, 66, 64)          0
['conv1_relu[0][0]']

 pool1_pool (MaxPooling2D)   (None, 32, 32, 64)          0
['pool1_pad[0][0]']

 conv2_block1_1_conv(Conv2D) (None, 32, 32, 64) 4160    ['pool1_pool[0][0]']
.
.
conv5_block3_add (Add)      (None, 4, 4, 2048)     0
['conv5_block2_out[0][0]',
                                                          'conv5_block3_3_bn[0][0]']

 conv5_block3_out (Activation) (None, 4, 4, 2048) 0
['conv5_block3_add[0][0]']
==================================================================================================
Total params: 23587712 (89.98 MB)
Trainable params: 0 (0.00 Byte)
Non-trainable params: 23587712 (89.98 MB)
__________________________________________________________________________________________________
# include GlobalAveragePooling2D
flatten_layer = layers.GlobalAveragePooling2D()
# include final Dense layer
```

prediction_layer = layers.Dense(4, activation='softmax')
model = models.Sequential([base_model, flatten_layer, prediction_layer])
model.summary() Output:
Model: "sequential_2"

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50 (Functional)       (None, 4, 4, 2048)        23587712

 global_average_pooling2d_2  (None, 2048)              0
 (GlobalAveragePooling2D)

 dense_2 (Dense)             (None, 4)                 8196
=================================================================
Total params: 23595908 (90.01 MB)
Trainable params: 8196 (32.02 KB)
Non-trainable params: 23587712 (89.98 MB)
_________________________________________________________________
```

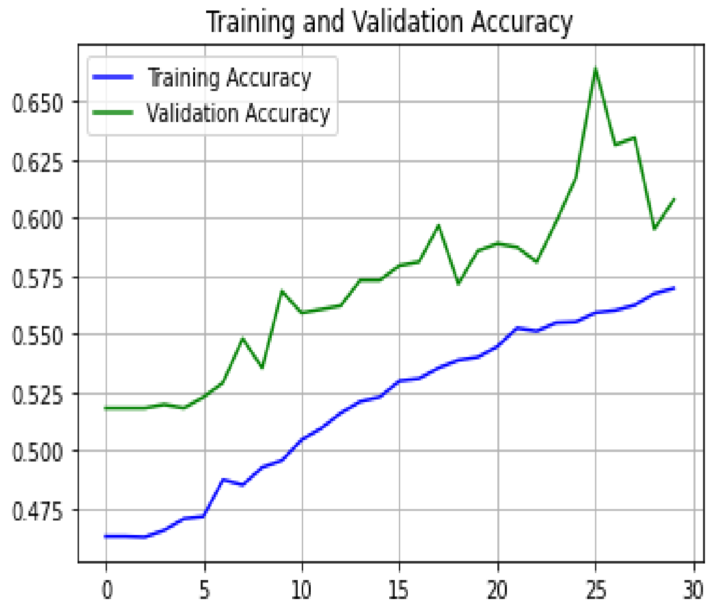
model.compile(optimizer = Adam(learning_rate = 0.001), loss='categorical_crossentropy', metrics=['acc']) history
= model.fit(train_generator,
steps_per_epoch=train_generator.samples/train_generator.batch_size, epochs=30, validation_data=validation_generator,
validation_steps=validation_generator.samples/validation_generator.batch_size, verbose=1) Output:

```
Epoch 1/30
189/189 [==============================] - 33s 163ms/step - loss: 1.2569 - acc: 0.4629 - val_loss: 1.1813 - val_acc: 0.5181
Epoch 2/30
189/189 [==============================] - 31s 162ms/step - loss: 1.2296 - acc: 0.4629 - val_loss: 1.1550 - val_acc: 0.5181
Epoch 3/30
189/189 [==============================] - 30s 159ms/step - loss: 1.2090 - acc: 0.4626 - val_loss: 1.1642 - val_acc: 0.5181
.
.
Epoch 29/30
189/189 [==============================] - 30s 159ms/step - loss: 0.9811 - acc: 0.5674 - val_loss: 0.9265 - val_acc: 0.5953
```

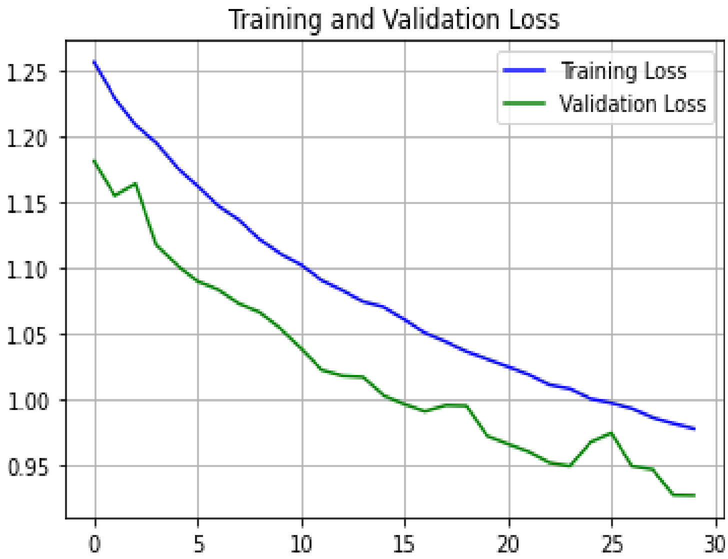
Epoch 30/30
189/189 [==============================] - 30s 160ms/step - loss: 0.9772 - acc: 0.5697 - val_loss: 0.9262 - val_acc: 0.6079

model.save("ResNet50_plant_disease.h5")
print("Saved model to disk")
model = models.load_model('ResNet50_plant_disease.h5') print("Model is loaded")
<u>Output:</u>
Saved model to disk
Model is loaded

train_acc = history.history['acc']
val_acc = history.history['val_acc']
train_loss = history.history['loss']
val_loss = history.history['val_loss'] #
Display loss/accuracies vs epochs
epochs = range(len(train_acc))
plt.plot(epochs, train_acc, 'b', label='Training Accuracy')
plt.plot(epochs, val_acc, 'g', label='Validation Accuracy')
plt.title('Training and Validation Accuracy') plt.grid()
plt.legend() plt.figure() plt.show()
plt.plot(epochs, train_loss, 'b', label='Training Loss')
plt.plot(epochs, val_loss, 'g', label='Validation Loss')
plt.title('Training and Validation Loss') plt.grid()
plt.legend() plt.show() <u>Output:</u>



<Figure size 432x288 with 0 Axes>

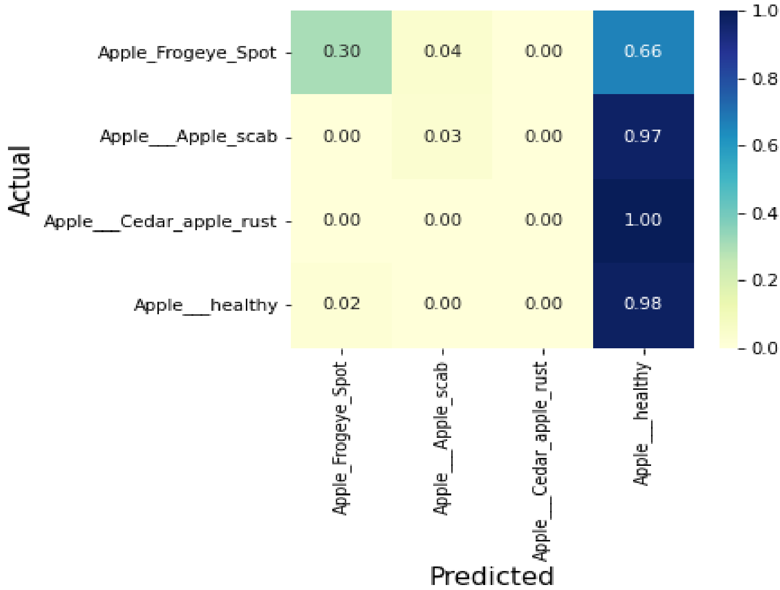


Get the filenames from the generator
#fnames = test_generator.filenames #
Get the ground truth from generator
ground_truth = test_generator.classes
# Get the label to class mapping from the generator label2index
= test_generator.class_indices
Getting the mapping from class index to class label
#idx2label = dict((v,k) for k,v in label2index.items()) # Get the
predictions from the model using the generator
predictions=model.predict_generator(test_generator,
steps=test_generator.samples/test_generator.batch_size,verbose=1)
predicted_classes = np.argmax(predictions,axis=1) errors =
np.where(predicted_classes != ground_truth)[0]
print("No of errors = {}/{}".format(len(errors),test_generator.samples)) Output:
566/566 [==============================] - 25s 44ms/step No
of errors = 263/566

accuracy = ((test_generator.samples-len(errors))/test_generator.samples) * 100
accuracy
Output: 53.53356890459364

from sklearn.metrics import confusion_matrix
import seaborn as sns import numpy as np
from matplotlib import pyplot as plt
cm = confusion_matrix(y_true=ground_truth, y_pred=predicted_classes)
cm = np.array(cm) # Normalise

cmn = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis] fig,
ax = plt.subplots(figsize=(5,4))
sns.heatmap(cmn, annot=True, fmt='.2f', xticklabels=label2index,
yticklabels=label2index, cmap="YlGnBu") plt.ylabel('Actual', fontsize=15)
plt.xlabel('Predicted', fontsize=15) plt.show(block=False) Output:



from sklearn.metrics import classification_report
print(classification_report(ground_truth,predicted_classes,
target_names=label2index))

Output:

	precision	recall	f1-score	support
Apple_Frogeye_Spot	0.86	0.30	0.45	103
Apple___Apple_scab	0.44	0.03	0.06	134
Apple___Cedar_apple_rust	0.00	0.00	0.00	55
Apple___healthy	0.51	0.98	0.67	274
accuracy			0.54	566
macro avg	0.45	0.33	0.29	566
weighted avg	0.51	0.54	0.42	566

CONCLUSION: An image classification task using pre-trained models like VGGNet, InceptionNet and ResNet has been implemented and the results were compared.

EXPERIMENT 5: Implement an autoencoder architecture for denoising images.

DESCRIPTION:

Dataset description: The data that will be incorporated is the MNIST database which contains 60,000 images for training and 10,000 test images. The dataset consists of small square 28×28 pixel grayscale images of handwritten single digits between 0 and 9. The MNIST dataset is conveniently bundled within Keras, and we can easily analyze some of its features in Python.

CODE:

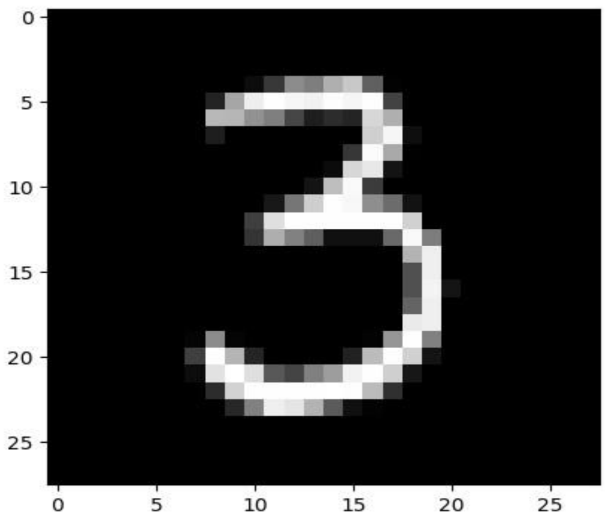
```
import tensorflow
from tensorflow.keras.datasets import mnist # MNIST dataset included in Keras
(X_train, y_train), (X_test, y_test) = mnist.load_data()
print("X_train shape", X_train.shape) print("y_train
shape", y_train.shape) print("X_test shape",
X_test.shape) print("y_test shape", y_test.shape)
```

Output:

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-
kerasdatasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
X_train shape (60000, 28, 28)
y_train shape (60000,) X_test
shape (10000, 28, 28)
y_test shape (10000,)
```

```
# Visualize any random image
import matplotlib.pyplot as plt i=50;
plt.imshow(X_train[i], cmap='gray');
```

Output:

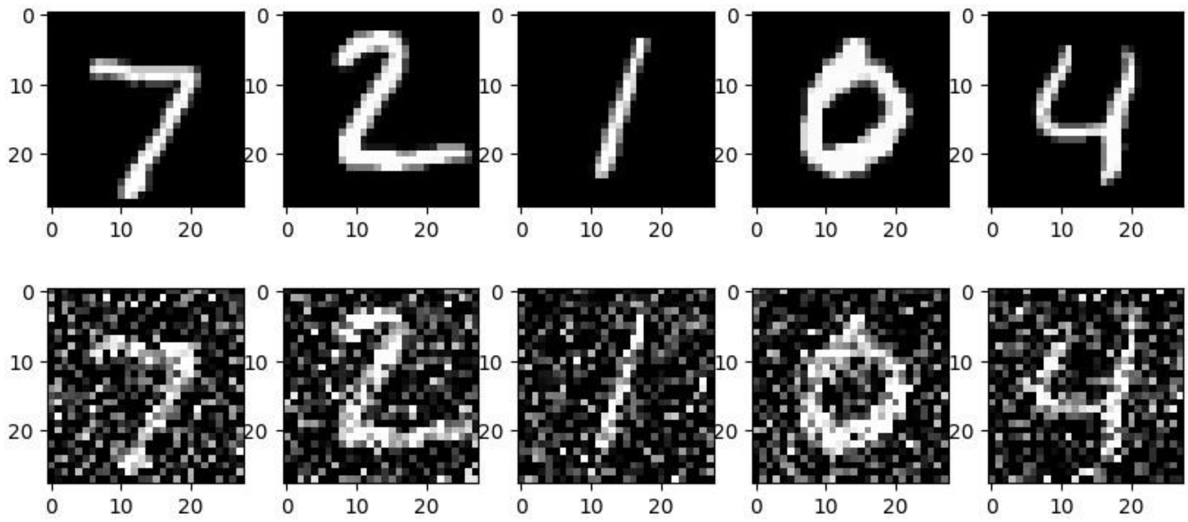


Formatting input

```
# reshape 28 x 28 matrices into 784-length vectors
X_train = X_train.reshape(60000, 784)
X_test = X_test.reshape(10000, 784)
# normalize each value for each pixel for the entire vector for each input
# change integers to 32-bit floating point numbers
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
# normalize by dividing by largest pixel value
X_train /= 255 X_test
/= 255
print("Training matrix shape", X_train.shape)
print("Testing matrix shape", X_test.shape)
```
Output:

Training matrix shape (60000, 784)
Testing matrix shape (10000, 784)

```
from tensorflow.keras.models import Model, Sequential from
tensorflow.keras.layers import Dense, Input
```
Create noisy data #
Add noise to input
```
import numpy as np
noise_factor = 0.4
X_train_noisy = X_train + noise_factor *
np.random.normal(size=X_train.shape)
X_test_noisy  = X_test  + noise_factor * np.random.normal(size=X_test.shape)
X_train_noisy= np.clip(X_train_noisy, 0.0, 1.0)
X_test_noisy = np.clip(X_test_noisy , 0.0, 1.0) n
= 5
plt.figure(figsize=(10, 4.5)) for i in
range(n):    # plot original image    ax =
plt.subplot(2, n, i + 1)
plt.imshow(X_test[i].reshape(28, 28))
plt.gray()    if i == n/2:
      ax.set_title('Original Images')
  # plot noisy image    ax = plt.subplot(2, n, i
+ 1 + n)
plt.imshow(X_test_noisy[i].reshape(28, 28))
plt.gray()    if i == n/2:
      ax.set_title('Noisy Input')
```
Output:



Design and train Fully-connected DAE

input_size = 784 hidden_size = 128
code_size = 32 # Input layer
model=Sequential()
# Hidden layer 1 in Encoder with 128 units, relu activation
model.add(Dense(128, activation='relu'))
# Hidden layer 2 (Code) in Encoder with 32 units, relu activation
model.add(Dense(32, activation='relu'))
# Hidden layer 1 in Decoder with 128 units, relu activation
model.add(Dense(128, activation='relu'))
# Hidden layer 2 in Encoder with 784 units, sigmoid activation
model.add(Dense(784, activation='sigmoid'))
# Compile the model, adam optimizer MeanSquaredError loss function
model.compile(loss='mean_squared_error', optimizer='adam')
# Display model Summary
model.fit(X_train_noisy, X_train, validation_data=(X_test_noisy, X_test), epochs=5, batch_size=200) model.summary() <u>Output:</u>

Epoch 1/5
300/300 [==============================] - 6s 15ms/step - loss: 0.0576 - val_loss: 0.0361 Epoch 2/5
300/300 [==============================] - 3s 10ms/step - loss: 0.0322 - val_loss: 0.0288 Epoch 3/5
300/300 [==============================] - 3s 11ms/step - loss: 0.0270 - val_loss: 0.0251

Epoch 4/5
300/300 [==============================] - 3s 10ms/step - loss: 0.0247 - val_loss: 0.0235 Epoch
5/5
300/300 [==============================] - 4s 14ms/step - loss: 0.0233 - val_loss: 0.0224
Model: "sequential_6"

Layer (type)	Output Shape	Param #
dense_24 (Dense)	(200, 128)	100480
dense_25 (Dense)	(200, 32)	4128
dense_26 (Dense)	(200, 128)	4224
dense_27 (Dense)	(200, 784)	101136

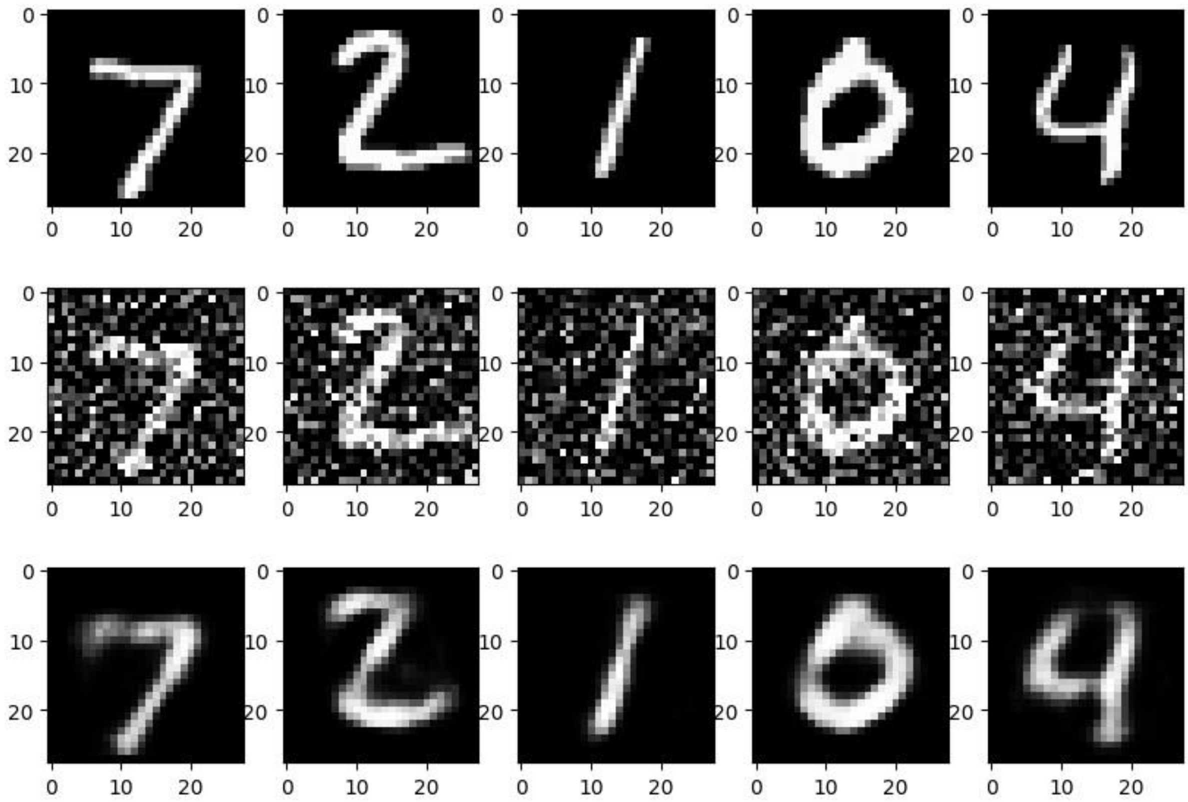
Total params: 209968 (820.19 KB)
Trainable params: 209968 (820.19 KB)
Non-trainable params: 0 (0.00 Byte)

```
# Reconstruct Images from Noisy X_test images
X_test_noisy_recons = model.predict(X_test_noisy)
```

<u>Output:</u>
313/313 [==============================] - 1s 2ms/step

```
n = 5
plt.figure(figsize=(10, 7)) for i in
range(n):    # plot original image    ax =
plt.subplot(3, n, i + 1)
plt.imshow(X_test[i].reshape(28, 28))
plt.gray()    if i == n/2:
      ax.set_title('Original Images')
  # plot noisy image    ax = plt.subplot(3, n, i
+ 1 + n)
plt.imshow(X_test_noisy[i].reshape(28, 28))
plt.gray()    if i == n/2:
```

```
        ax.set_title('Noisy Input')
# plot noisy image
    ax = plt.subplot(3, n, i + 1 + 2*n)
    plt.imshow(X_test_noisy_recons[i].reshape(28, 28))
    plt.gray()
if i == n/2:
        ax.set_title('Autoencoder Output')
```
Output:



CONCLUSION: An autoencoder architecture for denoising images has been implemented.

EXPERIMENT 6: Implement GAN architecture on MNIST dataset.

DESCRIPTION:

The aim is to build a generator to generate a set of input images of handwritten digits and build and train an effective discriminator that discerns generated model examples from real examples. By training the discriminator to be effective, we can stack generator and discriminator to form a GAN, freeze the weights in the adversarial part of the network and train the generative network weights to push random noisy inputs towards the real example class output of the adversarial half.

INPUT CODE:

```
import numpy as np import
matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Input, Dense, Reshape, Flatten, Conv2D,
Conv2DTranspose, LeakyReLU from tensorflow.keras.optimizers import Adam
from tensorflow.keras.datasets import mnist
# Load MNIST data
(X_train,y_train), (X_test, y_test) = mnist.load_data()
print(X_train.shape) Output: (60000, 28, 28)

# Normalize data
X_train = (X_train.astype(np.float32) - 127.5) / 127.5
X_train = np.expand_dims(X_train, axis=3)
print(X_train.shape) Output:
(60000, 28, 28, 1)

# Define discriminator def
build_discriminator():
model = Sequential()
   model.add(Conv2D(64, (4, 4), strides=2, padding='same', input_shape=(28,
28, 1))) # Added input_shape
model.add(LeakyReLU(0.2))
   model.add(Conv2D(128, (4, 4), strides=2, padding='same'))
   model.add(LeakyReLU(0.2))
model.add(Flatten())
   model.add(Dense(1, activation='sigmoid')) # Corrected activation to sigmoid
return model # Compile discriminator discriminator = build_discriminator()
discriminator.compile(loss='binary_crossentropy', optimizer=Adam(lr=0.0002,
beta_1=0.5), metrics=['accuracy'])
discriminator.summary() Output:
```

Model: "sequential_3"
Layer (type)   Output Shape   Param #
==

conv2d_5 (Conv2D)   (None, 14, 14, 64)   1088

leaky_re_lu_5 (LeakyReLU) (None, 14, 14, 64)   0

conv2d_6 (Conv2D)   (None, 7, 7, 128)   131200

leaky_re_lu_6 (LeakyReLU) (None, 7, 7, 128)   0
flatten
(Flatten)   (None, 6272)   0
dense (Dense)
(None, 1)   6273
==
Total params: 138561 (541.25 KB)
Trainable params: 138561 (541.25 KB)
Non-trainable params: 0 (0.00 Byte)

Define generator def
build_generator():
model = Sequential()
model.add(Dense(7*7*128, input_dim=100))
model.add(LeakyReLU(0.2))
model.add(Reshape((7, 7, 128))) # Corrected Reshape parameters
model.add(Conv2DTranspose(64, kernel_size=4, strides=2, padding='same'))
model.add(LeakyReLU(0.2))
model.add(Conv2DTranspose(1, kernel_size=4, strides=2, padding='same',
activation='tanh'))   return model
# Combine generator and discriminator into a single model
generator = build_generator() generator.summary()
Output:
Model: "sequential_4"
Layer (type)   Output Shape   Param #
==

dense_1 (Dense)   (None, 6272)   633472

leaky_re_lu_7 (LeakyReLU)   (None, 6272)   0

```
 reshape (Reshape)            (None, 7, 7, 128)      0

 conv2d_transpose (Conv2DTr  (None, 14, 14, 64)     131136
anspose)

 leaky_re_lu_8 (LeakyReLU)   (None, 14, 14, 64)     0

 conv2d_transpose_1 (Conv2D  (None, 28, 28, 1)      1025
 Transpose)
=================================================================
Total params: 765633 (2.92 MB)
Trainable params: 765633 (2.92 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
# Input Latent Variable z =
Input(shape=(100,)) # Output
of generator img =
generator(z) # Freeze
Discriminator
discriminator.trainable = False
# Give output of Generator to the Discriminator
validity = discriminator(img) # Build DCGAN
gan = Model(z, validity)
gan.compile(loss='binarycrossentropy',optimizer=Adam(lr=0.0002,beta_1=0.5))
gan.summary()
```

Output:

```
Model: "model"
Layer (type)            Output Shape           Param #
=================================================================
input_1 (InputLayer)      [(None, 100)]          0

 sequential_4 (Sequential)   (None, 28, 28, 1)      765633

 sequential_3 (Sequential)   (None, 1)              138561
=================================================================
Total params: 904194 (3.45 MB)
Trainable params: 765633 (2.92 MB) Non-trainable
params: 138561 (541.25 KB)
```

Train DCGAN epochs
= 5000 batch_size = 64
d_loss_all, g_loss_all = [], [] for
epoch in range(epochs):   # Select a
random batch of images   idx =
np.random.randint(0,
X_train.shape[0], batch_size)
real_images = X_train[idx]   #
Generate fake images
   noise = np.random.normal(0, 1, (batch_size, 100))
fake_images = generator.predict(noise)

   # Train discriminator   d_loss_real =
discriminator.train_on_batch(real_images, np.ones(batch_size))
   d_loss_fake = discriminator.train_on_batch(fake_images,
np.zeros(batch_size))
   d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)
d_loss_all.append(d_loss[0])

   # Train generator   noise = np.random.normal(0, 1,
(batch_size, 100))   g_loss = gan.train_on_batch(noise,
np.ones(batch_size))   g_loss_all.append(g_loss)

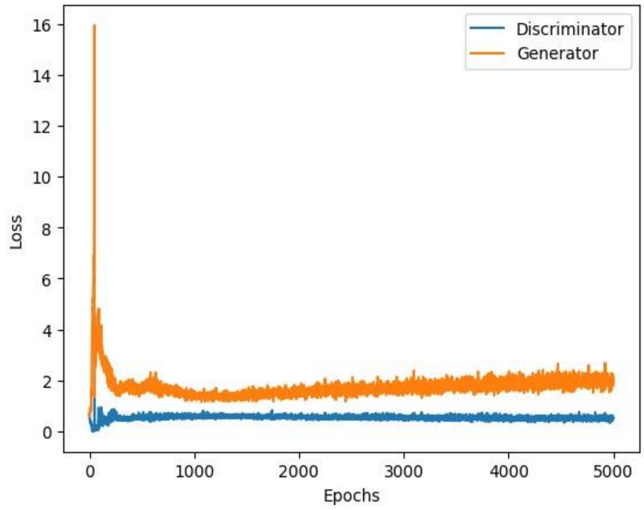
   # Print progress   if
epoch % 100 == 0:
      print(f"Epoch: {epoch} \t Discriminator Loss: {d_loss[0]} \t Generator
Loss: {g_loss}") Output:

Streaming output truncated to the last 5000 lines.
2/2 [==============================] - 0s 88ms/step
2/2 [==============================] - 0s 60ms/step
…
Epoch: 4500 Discriminator Loss: 0.5025077760219574 Generator
Loss: 1.8742070198059082
2/2 [==============================] - 0s 30ms/step
2/2 [==============================] - 0s 35ms/step
Epoch: 4600 Discriminator Loss: 0.5813893675804138 Generator
Loss: 1.757987380027771
2/2 [==============================] - 0s 36ms/step
2/2 [==============================] - 0s 37ms/step

Epoch: 4700 Discriminator Loss: 0.4938610792160034 Generator Loss: 1.955761432647705
2/2 [==============================] - 0s 33ms/step
2/2 [==============================] - 0s 38ms/step
Epoch: 4800 Discriminator Loss: 0.39186854660511017 Generator Loss: 1.9223999977111816
2/2 [==============================] - 0s 34ms/step
2/2 [==============================] - 0s 56ms/step
Epoch: 4900 Discriminator Loss: 0.4820183664560318 Generator Loss: 2.0517730712890625
2/2 [==============================] - 0s 28ms/step
2/2 [==============================] - 0s 31ms/step
..

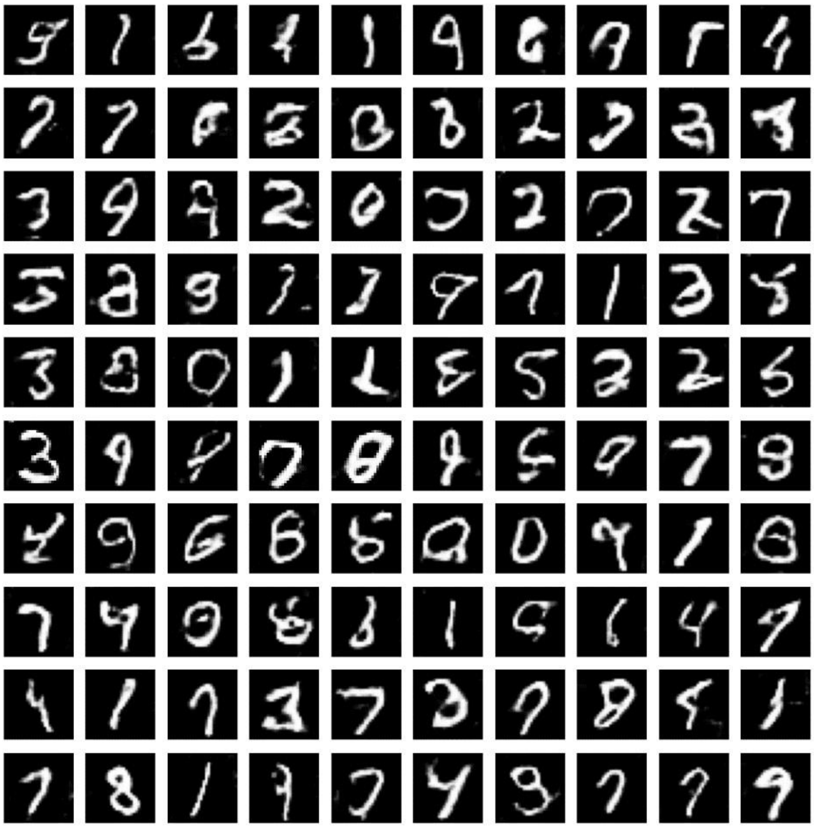
```
plt.plot(d_loss_all) plt.plot(g_loss_all)
plt.legend(('Discriminator','Generator'))
plt.xlabel('Epochs') plt.ylabel('Loss')
plt.show()
```

Output:




```
# Generate images
noise = np.random.normal(0, 1, (100, 100)) generated_images
= generator.predict(noise)
# Display generated images
plt.figure(figsize=(10, 10)) for
i in range(100):
   plt.subplot(10, 10, i+1)
   plt.imshow(generated_images[i, :, :, 0], cmap='gray')
plt.axis('off') plt.tight_layout() plt.show()
```

Output:



CONCLUSION: A GAN architecture has been implemented on MNIST dataset to recognize images of handwritten digits.