Task 4:

Implement a simple machine/deep learning algorithm using TensorFlow or PyTorch to classify a set of images. The algorithm should be trained on a labeled dataset and evaluated on a test dataset to measure its accuracy. Dataset: https://www.kaggle.com/datasets/alxmamaev/flowers-recognition
Hint: You can also use tools to train your model.

Flower Recognition CNN Keras

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

```
Mounted at /content/drive
```

```
import os
print(os.listdir('/content/drive/MyDrive/archive/flowers'))
```

```
['daisy', 'sunflower', 'dandelion', 'rose', 'tulip']
```

+ Code + Text

CONTENTS ::

Double-click (or enter) to edit

1) Importing Various Modules.

```
# Ignore  the warnings
import warnings
warnings.filterwarnings('always')
warnings.filterwarnings('ignore')

# data visualisation and manipulation
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import style
import seaborn as sns

#configure
# sets matplotlib to inline and displays graphs below the corressponding cell.
%matplotlib inline
style.use('fivethirtyeight')
sns.set(style='whitegrid',color_codes=True)

#model selection
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score,precision_score,recall_score,confusion_mat
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import LabelEncoder

#preprocess.
from keras.preprocessing.image import ImageDataGenerator

#dl libraraies
from keras import backend as K
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam,SGD,Adagrad,Adadelta,RMSprop
from keras.utils import to_categorical

# specifically for cnn
from keras.layers import Dropout, Flatten,Activation
from keras.layers import Conv2D, MaxPooling2D, BatchNormalization

import tensorflow as tf
import random as rn

# specifically for manipulating zipped images and getting numpy arrays of pixel value
import cv2
import numpy as np
from tqdm import tqdm
import os
from random import shuffle
from zipfile import ZipFile
from PIL import Image
```

2) Preparing the Data

2.1) Making the functions to get the training and validation set from the Images

```
X=[]
Z=[]
IMG_SIZE=150
FLOWER_DAISY_DIR='/content/drive/MyDrive/archive/flowers/daisy'
FLOWER_SUNFLOWER_DIR='/content/drive/MyDrive/archive/flowers/sunflower'
FLOWER_TULIP_DIR='/content/drive/MyDrive/archive/flowers/rose'
FLOWER_DANDI_DIR='/content/drive/MyDrive/archive/flowers/dandelion'
FLOWER_ROSE_DIR='/content/drive/MyDrive/archive/flowers/tulip'
```

```
def assign_label(img,flower_type):
    return flower_type
```

```
def make_train_data(flower_type,DIR):
    for img in tqdm(os.listdir(DIR)):
        label=assign_label(img,flower_type)
        path = os.path.join(DIR,img)
        img = cv2.imread(path,cv2.IMREAD_COLOR)
        img = cv2.resize(img, (IMG_SIZE,IMG_SIZE))

        X.append(np.array(img))
        Z.append(str(label))
```

```
make_train_data('Daisy',FLOWER_DAISY_DIR)
print(len(X))
```

```
100%|██████████| 764/764 [00:10<00:00, 73.23it/s] 764
```

```
make_train_data('Sunflower',FLOWER_SUNFLOWER_DIR)
print(len(X))
```

```
100%|██████████| 733/733 [00:09<00:00, 75.52it/s] 1497
```

```
make_train_data('Tulip',FLOWER_TULIP_DIR)
print(len(X))
```

```
100%|██████████| 784/784 [00:11<00:00, 70.98it/s] 2281
```

```
make_train_data('Dandelion',FLOWER_DANDI_DIR)
print(len(X))
```

```
100%|██████████| 1052/1052 [00:13<00:00, 75.93it/s] 3333
```

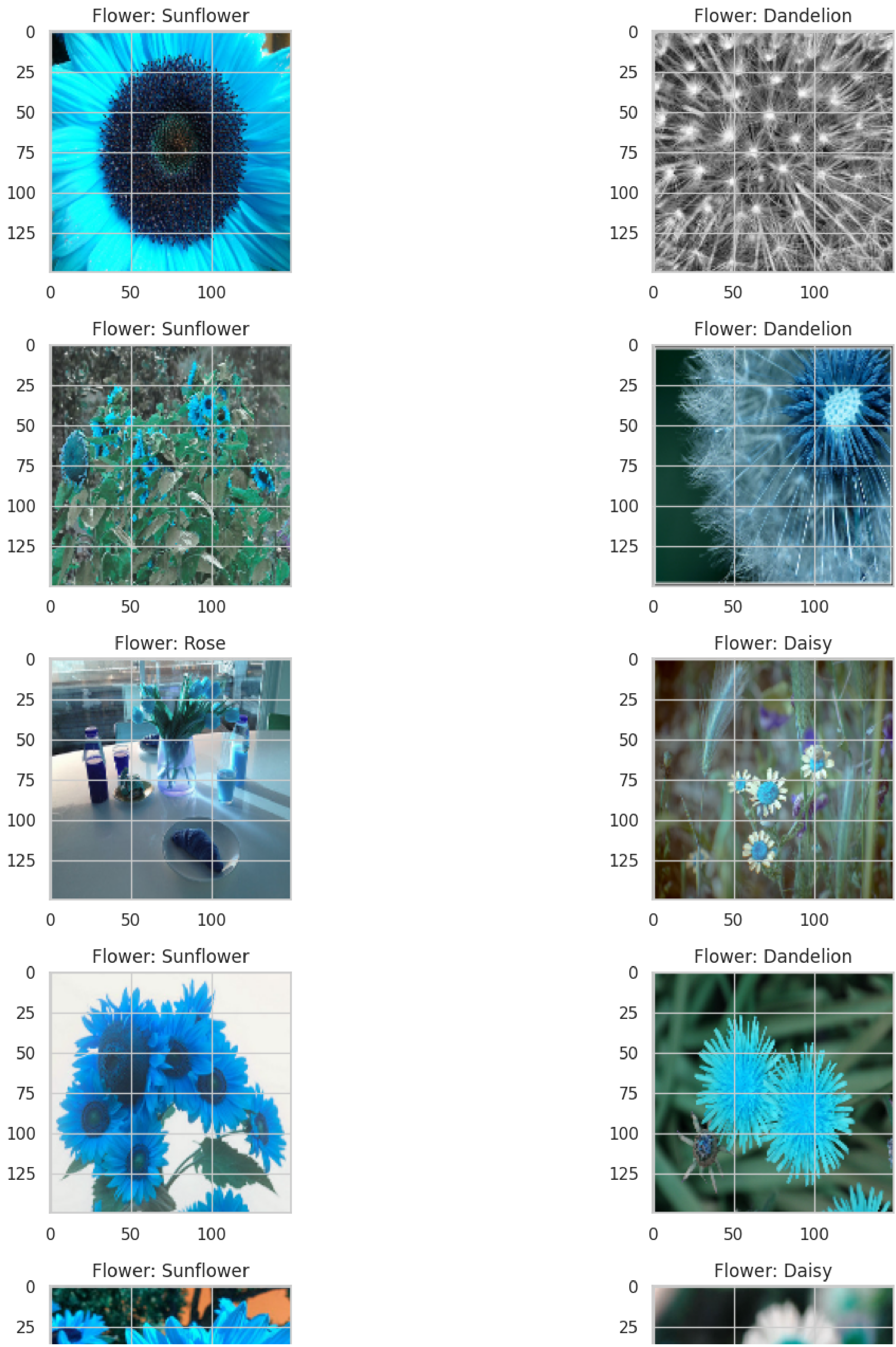
```
make_train_data('Rose',FLOWER_ROSE_DIR)
print(len(X))
```

```
100%|██████████| 984/984 [00:13<00:00, 75.34it/s] 4317
```

2.2) Visualizing some Random Images

```
fig,ax=plt.subplots(5,2)
fig.set_size_inches(15,15)
for i in range(5):
    for j in range (2):
        l=rn.randint(0,len(Z))
        ax[i,j].imshow(X[l])
        ax[i,j].set_title('Flower: '+Z[l])

plt.tight_layout()
```


Flower: Sunflower
Flower: Dandelion
Flower: Sunflower
Flower: Dandelion
Flower: Rose
Flower: Daisy
Flower: Sunflower
Flower: Dandelion
Flower: Sunflower
Flower: Daisy

2.3) Label Encoding the Y array (i.e. Daisy->0, Rose->1 etc...) & then One Hot Encoding

```
le=LabelEncoder()
Y=le.fit_transform(Z)
Y=to_categorical(Y,5)
X=np.array(X)
X=X/255
```

2.4) Splitting into Training and Validation Sets

```
x_train,x_test,y_train,y_test=train_test_split(X,Y,test_size=0.25,random_state=42)
```

2.5) Setting the Random Seeds

```
np.random.seed(42)
rn.seed(42)
tf.random.set_seed(42)
```

3) Modelling

3.1) Building the ConvNet Model

```
# # modelling starts using a CNN.
```

```
model = Sequential()
model.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same',activation ='relu
model.add(MaxPooling2D(pool_size=(2,2)))


model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',activation ='relu
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))


model.add(Conv2D(filters =96, kernel_size = (3,3),padding = 'Same',activation ='relu'
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))

model.add(Conv2D(filters = 96, kernel_size = (3,3),padding = 'Same',activation ='relu
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))

model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dense(5, activation = "softmax"))
```

3.2) Using a LR Annealer

```
batch_size=128
epochs=50

from keras.callbacks import ReduceLROnPlateau
red_lr= ReduceLROnPlateau(monitor='val_acc',patience=3,verbose=1,factor=0.1)
```

3.3) Data Augmentation to prevent Overfitting

```
datagen = ImageDataGenerator(
        featurewise_center=False,  # set input mean to 0 over the dataset
        samplewise_center=False,  # set each sample mean to 0
        featurewise_std_normalization=False,  # divide inputs by std of the dataset
        samplewise_std_normalization=False,  # divide each input by its std
        zca_whitening=False,  # apply ZCA whitening
        rotation_range=10,  # randomly rotate images in the range (degrees, 0 to 180)
        zoom_range = 0.1, # Randomly zoom image
        width_shift_range=0.2,  # randomly shift images horizontally (fraction of tot
        height_shift_range=0.2,  # randomly shift images vertically (fraction of tota
        horizontal_flip=True,  # randomly flip images
        vertical_flip=False)  # randomly flip images


datagen.fit(x_train)
```

3.4) Compiling the Keras Model & Summary

```
model.compile(optimizer=Adam(lr=0.001),loss='categorical_crossentropy',metrics=['accu
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 150, 150, 32)      2432

 max_pooling2d (MaxPooling2D  (None, 75, 75, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 75, 75, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 37, 37, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 37, 37, 96)        55392

 max_pooling2d_2 (MaxPooling  (None, 18, 18, 96)       0
 2D)

 conv2d_3 (Conv2D)           (None, 18, 18, 96)        83040

 max_pooling2d_3 (MaxPooling  (None, 9, 9, 96)         0
 2D)

 flatten (Flatten)           (None, 7776)              0

 dense (Dense)               (None, 512)               3981824

 activation (Activation)     (None, 512)               0

 dense_1 (Dense)             (None, 5)                 2565

=================================================================
Total params: 4,143,749
Trainable params: 4,143,749
Non-trainable params: 0
_________________________________________________________________
```

3.5) Fitting on the Training set and making predcitons on the Validation set

```
History = model.fit_generator(datagen.flow(x_train,y_train, batch_size=batch_size),
                              epochs = epochs, validation_data = (x_test,y_test),
                              verbose = 1, steps_per_epoch=x_train.shape[0] // batch_
# model.fit(x_train,y_train,epochs=epochs,batch_size=batch_size,validation_data = (x_
```

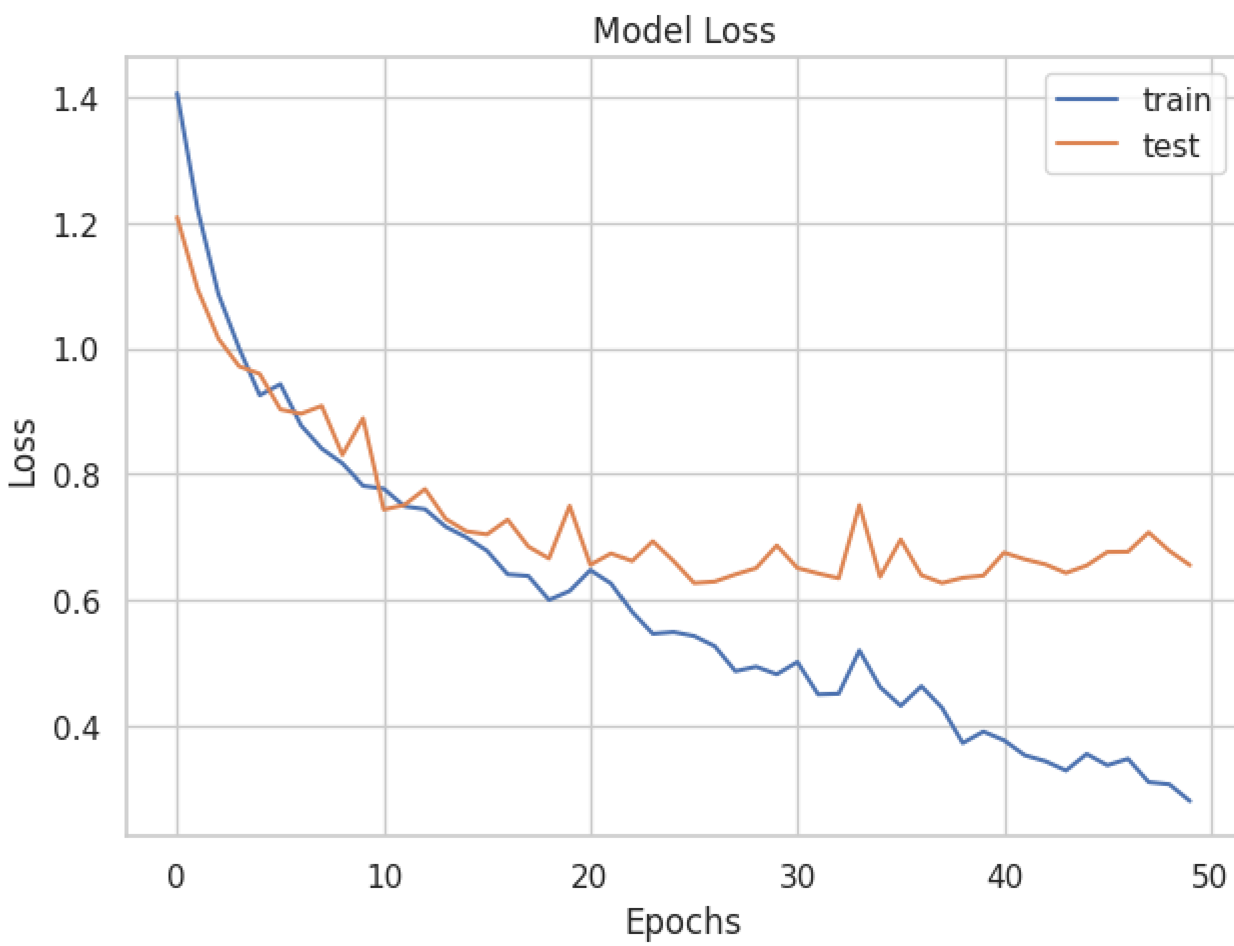
```
Epoch 1/50
25/25 [==============================] - 34s 886ms/step - loss: 1.5004 - accur
Epoch 2/50
25/25 [==============================] - 19s 750ms/step - loss: 1.2555 - accur
Epoch 3/50
25/25 [==============================] - 19s 748ms/step - loss: 1.1067 - accur
Epoch 4/50
25/25 [==============================] - 20s 798ms/step - loss: 1.0266 - accur
Epoch 5/50
25/25 [==============================] - 18s 746ms/step - loss: 0.9843 - accur
Epoch 6/50
25/25 [==============================] - 20s 796ms/step - loss: 0.9282 - accur
Epoch 7/50
25/25 [==============================] - 18s 713ms/step - loss: 0.8874 - accur
Epoch 8/50
25/25 [==============================] - 18s 727ms/step - loss: 0.8335 - accur
Epoch 9/50
25/25 [==============================] - 18s 720ms/step - loss: 0.8050 - accur
Epoch 10/50
25/25 [==============================] - 22s 888ms/step - loss: 0.8105 - accur
Epoch 11/50
25/25 [==============================] - 18s 715ms/step - loss: 0.7980 - accur
Epoch 12/50
25/25 [==============================] - 18s 713ms/step - loss: 0.7490 - accur
Epoch 13/50
25/25 [==============================] - 19s 765ms/step - loss: 0.7591 - accur
Epoch 14/50
25/25 [==============================] - 17s 698ms/step - loss: 0.6997 - accur
Epoch 15/50
25/25 [==============================] - 18s 736ms/step - loss: 0.7166 - accur
Epoch 16/50
25/25 [==============================] - 21s 826ms/step - loss: 0.7049 - accur
Epoch 17/50
25/25 [==============================] - 19s 749ms/step - loss: 0.6768 - accur
Epoch 18/50
25/25 [==============================] - 19s 734ms/step - loss: 0.6604 - accur
Epoch 19/50
25/25 [==============================] - 18s 717ms/step - loss: 0.6501 - accur
Epoch 20/50
25/25 [==============================] - 19s 775ms/step - loss: 0.6228 - accur
Epoch 21/50
25/25 [==============================] - 18s 710ms/step - loss: 0.6318 - accur
Epoch 22/50
25/25 [==============================] - 20s 786ms/step - loss: 0.5704 - accur
Epoch 23/50
25/25 [==============================] - 20s 814ms/step - loss: 0.5755 - accur
Epoch 24/50
25/25 [==============================] - 21s 864ms/step - loss: 0.5750 - accur
Epoch 25/50
25/25 [==============================] - 22s 894ms/step - loss: 0.5704 - accur
Epoch 26/50
```

```
25/25 [==============================] - 18s 724ms/step - loss: 0.5395 - accur
Epoch 27/50
25/25 [==============================] - 18s 751ms/step - loss: 0.5393 - accur
Epoch 28/50
25/25 [==============================] - 18s 728ms/step - loss: 0.5048 - accur
Epoch 29/50
```

```
model.save("model.keras")
```

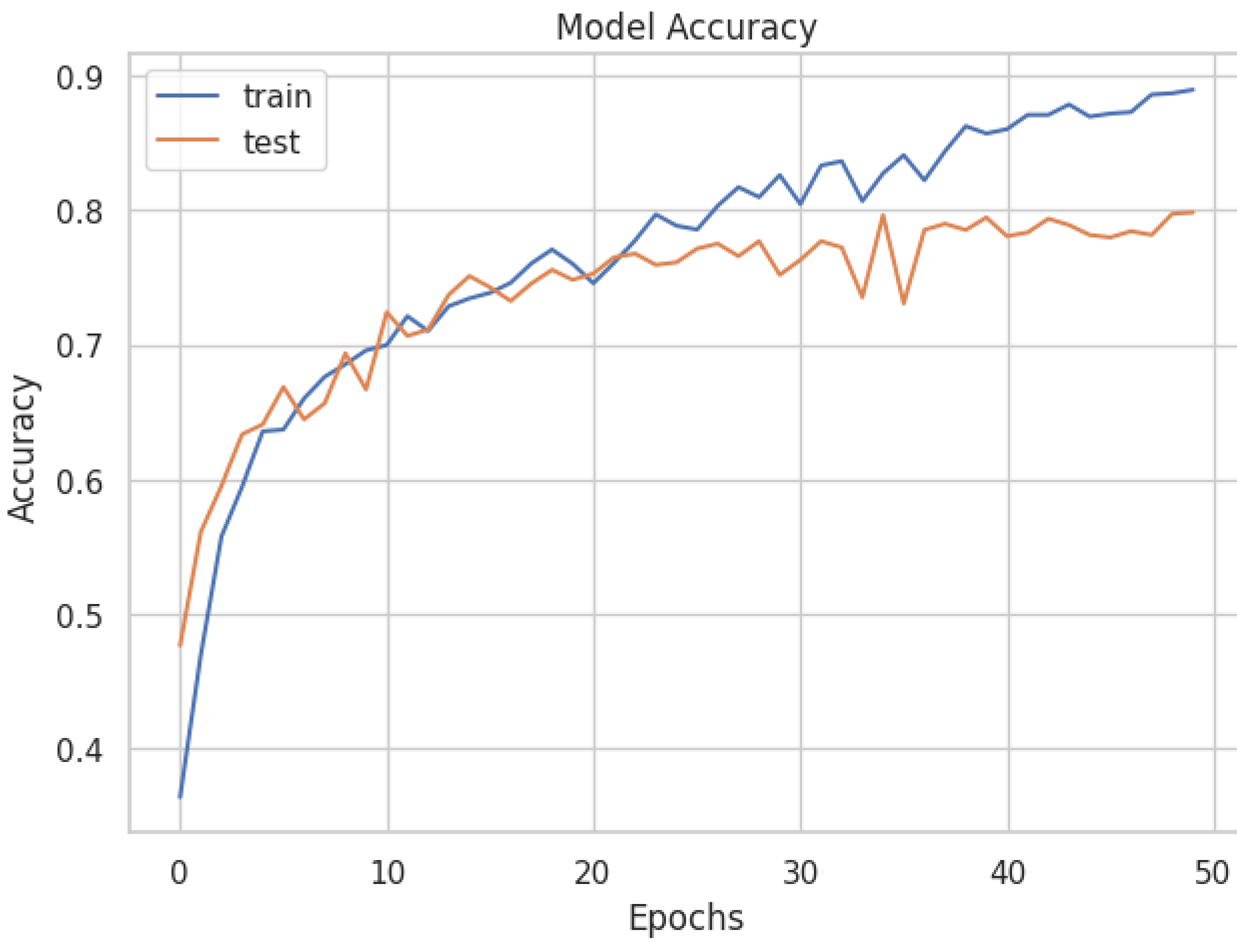
4) Evaluating the Model Performance

```
plt.plot(History.history['loss'])
plt.plot(History.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epochs')
plt.legend(['train', 'test'])
plt.show()
```




```
plt.plot(History.history['accuracy'])
plt.plot(History.history['val_accuracy'])
```

```
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epochs')
plt.legend(['train', 'test'])
plt.show()
```



THE END.

Task 5:

Create a Flask API for Task 4 to test the model. The application should allow users to interact with a trained model by submitting input data and receiving predictions.

1. Install Flask: Flask is a popular web framework for Python. You can install it using pip:

```
pip install flask
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-whe
Requirement already satisfied: flask in /usr/local/lib/python3.10/dist-packages
```

```
Requirement already satisfied: Werkzeug>=2.2.2 in /usr/local/lib/python3.10/dist
Requirement already satisfied: Jinja2>=3.0 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: itsdangerous>=2.0 in /usr/local/lib/python3.10/di
Requirement already satisfied: click>=8.0 in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist
```

```
loaded_model = tf.keras.models.load_model("model.keras")
```

```
import flask
import io
import string
import time
import os
import numpy as np
import tensorflow as tf
from PIL import Image
from flask import Flask, jsonify, request


def prepare_image(img):
    img = Image.open(io.BytesIO(img))
    img = img.resize((224, 224))
    img = np.array(img)
    img = np.expand_dims(img, 0)
    return img



def predict_result(img):
    return 1 if model.predict(img)[0][0] > 0.5 else 0



app = Flask(__name__)


@app.route('/predict', methods=['POST'])
def infer_image():
    # Catch the image file from a POST request
    if 'file' not in request.files:
        return "Please try again. The Image doesn't exist"

    file = request.files.get('file')

    if not file:
        return

    # Read the image
    img_bytes = file.read()

    # Prepare the image
    img = prepare_image(img_bytes)
```

```
    # Return on a JSON format
    return jsonify(prediction=predict_result(img))


@app.route('/', methods=['GET'])
def index():
    return 'Machine Learning Inference'


if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0')
```

```
 * Serving Flask app '__main__'
 * Debug mode: on
INFO:werkzeug:WARNING: This is a development server. Do not use it in a producti
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.28.0.12:5000
INFO:werkzeug:Press CTRL+C to quit
INFO:werkzeug: * Restarting with stat
```

STEPS:

Importing libraries

Load the machine learning model

Build functions to preprocess and to predict the image

Initialize the flask object

Set the route and the function that returns something to the user’s browser

Run and test the API