A STUDY TO DESIGN A SOFTWARE MODULE FOR A FORENSIC SOFTWARE

Veritas Volume: 2, Issue: 2, Pages:46-62

Mr. Varun Gupta
Ms. Sneha R
Mr. Madhukar.K
Mr. Jeasmon Thomas

INTRODUCTION

Technologies are used to perpetrate crimes, and law enforcement in creasingly employs computer systems to fight crime because of the rising science of virtual proof forensics. The information recorded or transferred in digital shape that may be utilised in the courtroom is referred to as virtual evidence. It may be located on a PC (Personal Computer), fixed disc, or a cellular cell phone. Digital crimes, or e-crimes, consisting of child pornography or credit card fraud, are often connected with virtual proof. Virtual proof, however, is being employed within the prosecution of all types of crimes, not solely for e-crimes. Suspects' email or cell telephone documents may encompass important proof about their purpose, whereabouts at the time of the crime, and relationships with different suspects. For instance, in 2005, a floppy disk led the government to the BTK (Bind, Torture, Kill) serial assassin, who had prevented police arrest for the reason that 1974 and killed at least ten people. The medical gathering, evaluation, and renovation of statistics contained in electronic media that can be used as proof in a court of regulation are known as virtual forensics. Now police regulation enforcement groups are incorporating virtual forensics into their infrastructure to combat e-crime and acquire relevant virtual evidence for all offences.

AIM

To design a software module for a forensic software

OBJECTIVES:

1. To develop software capable of finding files on the basis of keywords whether it is hidden or visible to the user.
2. To create a software that can recognize and categorise photos, reduce noise from the dataset regardless of rotation, magnification, or both.
3. To Incorporate the criminal record in a decentralised manner through the use of a consolidated blockchain application.

METHOD

Using FileTree, Python, IPFS (InterPlanetary File System), and blockchain resources, a suitable program code was created to test the efficiency of finding files from basic keywords

RESULTS

```
from itertools import chain
import os
import sys

print("Do you want to search file.")
print ("1. word list in the dictory")
print ("2. On the basis extension")
A = float(input("Enter the choice: "))
Document = ('.docx', '.pdf', 'doc', '.csv', '.xlsx', '.xlsm', '.xlsb', '.xltx', '.xltm', '.xls', '.xlt', '.xls', '.xml', '.xlam')
Image = ('.apng', '.avif', '.gif', '.jpg', '.jpeg', '.jfif', '.pjpeg', '.pjp', '.png', '.svg', '.webp', '.bmp', '.ico', '.cur', '.tif', '.tiff', '.heic')
Video = ('.mp4', '.mov', '.wmv', '.avi', '.avchd', '.flv', '.f4v', '.swf', '.mkv')
Rape = ('Child abuse','Domestic violence','Blackmail','Guilt','Sex','Murder','Victim','Fear','Gang Rape','Rod','Vagina','Condoms')
Murder = ('Arson','Hit and run','Homicide','Poaching','Genocide','Assassination','Felony','Manslaughter','Bloodshed','Guns','Pistols','Rifle','Knife','Hammer','Strangulation')
Terrorism = ("Bombing","Espionage","Dynamite","Conflict","Hijacking","Kidnapping","Cyber terrorism","Explosive","Explosion")
paths = ('C:','D:')

if (A == 1):
    print("Which Case")
    print("1.Murder")
    print("2.Rape")
    print("3.Terrorism")
    Madhu = float(input("Enter You Choice: "))
    Z = open('D:\Outputs\Murder\output.txt','w+')
    sys.stdout = Z
    if (Madhu == 1):
        for root, dirs, files in chain.from_iterable(os.walk(path) for path in paths):
            for file in files:
                if file.startswith(Murder):
                    print(os.path.join(root, file))
```

```
                print(os.path.join(root, file))
    Z.close()
    N = open('D:\Outputs\Rape\output.txt','w+')
    sys.stdout = N
    if (Madhu == 2):
##Print the paths of all files in the specified directories that start with the specified string.
#@param paths - the paths to search for files in.
#@param Rape - the string to search for in the file names
        for root, dirs, files in chain.from_iterable(os.walk(path) for path in paths):
            for file in files:
                if file.startswith(Rape):
                    print(os.path.join(root, file))
    N.close()
    I = open('D:\Outputs\Terrorism\output.txt' , 'w+')
    sys.stdout = I
    if (Madhu == 3):
        for root, dirs, files in chain.from_iterable(os.walk(path) for path in paths):
            for file in files:
                if file.startswith(Terrorism):
                    print(os.path.join(root, file))
    I.close()
if (A==2):
    print("Which File type you want to search")
    print("1.Picture")
    print("2.Document")
    print("3.Video")
    D = float(input("Enter Choice: "))
    L = open('D:\Outputs\Picture\output.txt', 'w+')
    sys.stdout = L
    if (D == 1):
        for root, dirs, files in chain.from_iterable(os.walk(path) for path in paths):
            for file in files:
                if file.endswith(Image):
                    print(os.path.join(root, file))
    L.close()
    M = open('D:\Outputs\Document\output.txt','w+')
    sys.stdout = M
    if (D == 2):
        for root, dirs, files in chain.from_iterable(os.walk(path) for path in paths):
            for file in files:
                if file.endswith(Document):
                    print(os.path.join(root, file))
    M.close()
    Q = open('D:\Outputs\Video\output.txt','w+')
    sys.stdout = Q
    if (D==3):
        for root, dirs, files in chain.from_iterable(os.walk(path) for path in paths):
            for file in files:
                if file.endswith(video):
                    print(os.path.join(root, file))
    Q.close()
```

In [2]:

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

from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
```

In [3]:

```
import pathlib
dataset_url = "https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz"
data_dir = tf.keras.utils.get_file('flower_photos', origin=dataset_url, untar=True)
data_dir = pathlib.Path(data_dir)
```

In [4]:

```
image_count = len(list(data_dir.glob('*/*.jpg')))
print(image_count)
```

```
3670
```

In [5]:

```
roses = list(data_dir.glob('roses/*'))
PIL.Image.open(str(roses[0]))
```

Out[5]:

In [6]:

```
PIL.Image.open(str(roses[1]))
```

Out[6]:

In [7]:

```
tulips = list(data_dir.glob('tulips/*'))
PIL.Image.open(str(tulips[0]))
```

Out[7]:

In [8]:

```
PIL.Image.open(str(tulips[1]))
```

Out[8]:

In [9]:

```
batch_size = 32
img_height = 180
img_width = 180
```

In [10]:

```
train_ds = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="training",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

```
Found 3670 files belonging to 5 classes.
Using 2936 files for training.
```

In [11]:

```
val_ds = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="validation",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

```
Found 3670 files belonging to 5 classes.
Using 734 files for validation.
```

In [12]:

```
class_names = train_ds.class_names
print(class_names)
```

```
['daisy', 'dandelion', 'roses', 'sunflowers', 'tulips']
```

In [13]:

```
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 10))
for images, labels in train_ds.take(1):
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names[labels[i]])
    plt.axis("off")
```

In [14]:

```
for image_batch, labels_batch in train_ds:
  print(image_batch.shape)
  print(labels_batch.shape)
  break
```

```
(32, 180, 180, 3)
(32,)
```

In [15]:

```
AUTOTUNE = tf.data.AUTOTUNE
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

In [16]:

```
normalization_layer = layers.Rescaling(1./255)
```

In [17]:

```
normalized_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
image_batch, labels_batch = next(iter(normalized_ds))
first_image = image_batch[0]
print(np.min(first_image), np.max(first_image))
```

```
0.0 1.0
```

In [18]:

```
num_classes = len(class_names)
model = Sequential([
  layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes)
])
```

In [19]:

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

In [20]:

```
model.summary()
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
rescaling_1 (Rescaling)	(None, 180, 180, 3)	0
conv2d (Conv2D)	(None, 180, 180, 16)	448
max_pooling2d (MaxPooling2D)	(None, 90, 90, 16)	0
conv2d_1 (Conv2D)	(None, 90, 90, 32)	4640
max_pooling2d_1 (MaxPooling2	(None, 45, 45, 32)	0
conv2d_2 (Conv2D)	(None, 45, 45, 64)	18496

```
max_pooling2d_2 (MaxPooling2  (None, 22, 22, 64)        0
flatten (Flatten)             (None, 30976)             0
dense (Dense)                 (None, 128)               3965056
dense_1 (Dense)               (None, 5)                 645
=================================================================
Total params: 3,989,285
Trainable params: 3,989,285
Non-trainable params: 0
```

In [21]:

```
epochs= 20
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
```

```
Epoch 1/20
92/92 [==============================] - 25s 113ms/step - loss: 1.2804 - accuracy: 0.4486
- val_loss: 1.0471 - val_accuracy: 0.5559
Epoch 2/20
92/92 [==============================] - 4s 46ms/step - loss: 0.9594 - accuracy: 0.6178 -
val_loss: 0.9224 - val_accuracy: 0.6267
Epoch 3/20
92/92 [==============================] - 4s 45ms/step - loss: 0.7853 - accuracy: 0.7040 -
val_loss: 0.9125 - val_accuracy: 0.6376
Epoch 4/20
92/92 [==============================] - 4s 44ms/step - loss: 0.6064 - accuracy: 0.7728 -
val_loss: 0.9624 - val_accuracy: 0.6362
Epoch 5/20
92/92 [==============================] - 4s 42ms/step - loss: 0.3887 - accuracy: 0.8607 -
val_loss: 1.0244 - val_accuracy: 0.6362
Epoch 6/20
92/92 [==============================] - 4s 43ms/step - loss: 0.2010 - accuracy: 0.9336 -
val_loss: 1.1821 - val_accuracy: 0.6580
Epoch 7/20
92/92 [==============================] - 4s 44ms/step - loss: 0.1135 - accuracy: 0.9659 -
val_loss: 1.2181 - val_accuracy: 0.6335
Epoch 8/20
92/92 [==============================] - 4s 44ms/step - loss: 0.0595 - accuracy: 0.9881 -
val_loss: 1.5906 - val_accuracy: 0.6621
Epoch 9/20
92/92 [==============================] - 4s 44ms/step - loss: 0.0478 - accuracy: 0.9867 -
val_loss: 1.5029 - val_accuracy: 0.6608
Epoch 10/20
92/92 [==============================] - 4s 43ms/step - loss: 0.0352 - accuracy: 0.9915 -
val_loss: 1.8065 - val_accuracy: 0.6403
Epoch 11/20
92/92 [==============================] - 4s 43ms/step - loss: 0.0139 - accuracy: 0.9969 -
val_loss: 1.9720 - val_accuracy: 0.6390
Epoch 12/20
92/92 [==============================] - 4s 43ms/step - loss: 0.0064 - accuracy: 0.9993 -
val_loss: 2.1985 - val_accuracy: 0.6458
Epoch 13/20
92/92 [==============================] - 4s 44ms/step - loss: 0.0164 - accuracy: 0.9956 -
val_loss: 2.0582 - val_accuracy: 0.6294
Epoch 14/20
```

```
92/92 [==============================] - 4s 44ms/step - loss: 0.0385 - accuracy: 0.9891 -
val_loss: 1.7845 - val_accuracy: 0.6499
Epoch 15/20
92/92 [==============================] - 4s 45ms/step - loss: 0.0236 - accuracy: 0.9942 -
val_loss: 2.0582 - val_accuracy: 0.6444
Epoch 16/20
92/92 [==============================] - 4s 46ms/step - loss: 0.0026 - accuracy: 1.0000 -
val_loss: 2.1042 - val_accuracy: 0.6635
Epoch 17/20
92/92 [==============================] - 4s 45ms/step - loss: 7.3304e-04 - accuracy: 1.00
00 - val_loss: 2.1749 - val_accuracy: 0.6580
Epoch 18/20
92/92 [==============================] - 4s 45ms/step - loss: 3.9710e-04 - accuracy: 1.00
00 - val_loss: 2.2380 - val_accuracy: 0.6608
Epoch 19/20
92/92 [==============================] - 4s 46ms/step - loss: 2.9957e-04 - accuracy: 1.00
00 - val_loss: 2.2946 - val_accuracy: 0.6608
Epoch 20/20
92/92 [==============================] - 4s 44ms/step - loss: 2.3911e-04 - accuracy: 1.00
00 - val_loss: 2.3444 - val_accuracy: 0.6594
```

In [22]:

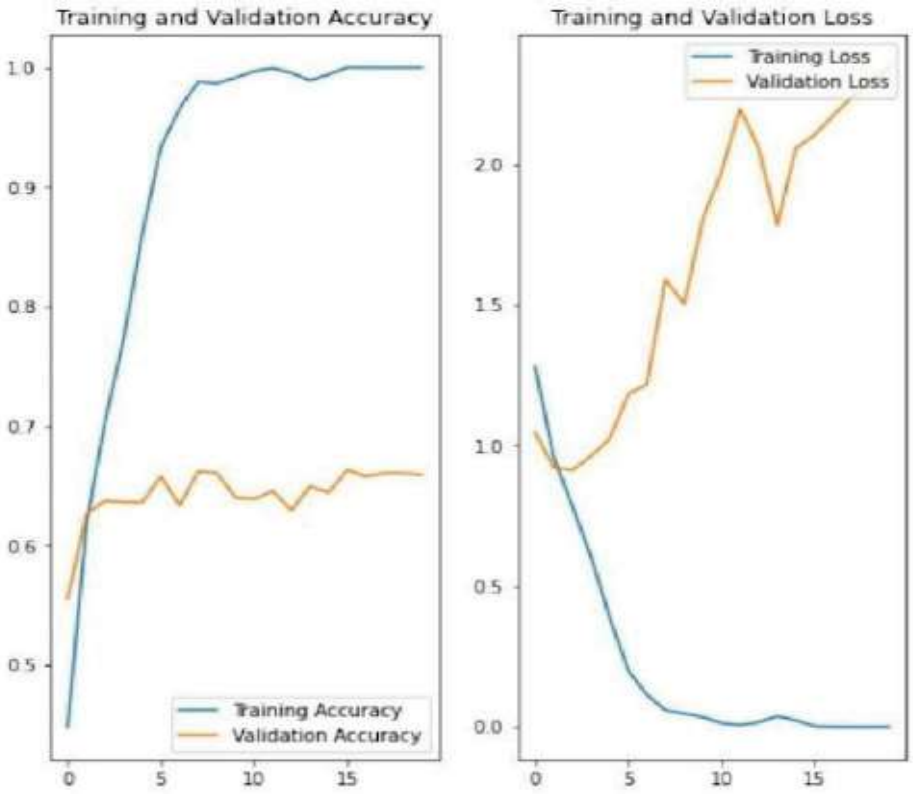
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

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

epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```

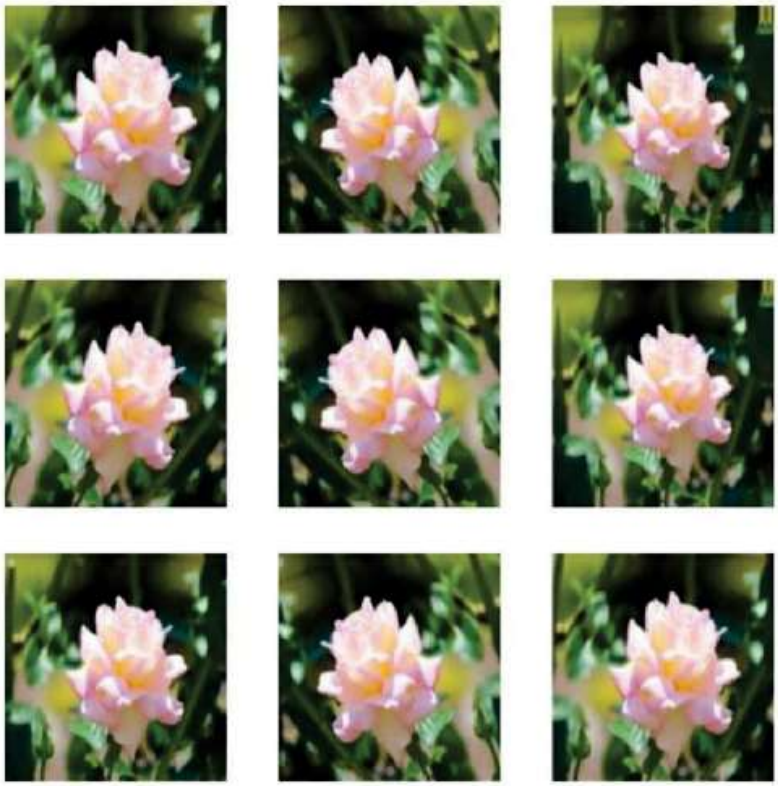


In [23]:

```
data_augmentation = keras.Sequential(
  [
    layers.RandomFlip("horizontal",
                      input_shape=(img_height,
                                   img_width,
                                   3)),
    layers.RandomRotation(0.1),
    layers.RandomZoom(0.1),
  ]
)
```

In [24]:

```
plt.figure(figsize=(10, 10))
for images, _ in train_ds.take(1):
  for i in range(9):
    augmented_images = data_augmentation(images)
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(augmented_images[0].numpy().astype("uint8"))
    plt.axis("off")
```



In [25]:

```
model = Sequential([
  data_augmentation,
  layers.Rescaling(1./255),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Dropout(0.2),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes)
])
```

In [26]:

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

In [27]:

```
model.summary()
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
sequential_1 (Sequential)	(None, 180, 180, 3)	0

rescaling_2 (Rescaling)	(None, 180, 180, 3)	0
conv2d_3 (Conv2D)	(None, 180, 180, 16)	448
max_pooling2d_3 (MaxPooling2	(None, 90, 90, 16)	0
conv2d_4 (Conv2D)	(None, 90, 90, 32)	4640
max_pooling2d_4 (MaxPooling2	(None, 45, 45, 32)	0
conv2d_5 (Conv2D)	(None, 45, 45, 64)	18496
max_pooling2d_5 (MaxPooling2	(None, 22, 22, 64)	0
dropout (Dropout)	(None, 22, 22, 64)	0
flatten_1 (Flatten)	(None, 30976)	0
dense_2 (Dense)	(None, 128)	3965056
dense_3 (Dense)	(None, 5)	645

Total params: 3,989,285
Trainable params: 3,989,285
Non-trainable params: 0

In [28]:

```
epochs = 35
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
```

```
Epoch 1/35
92/92 [==============================] - 9s 60ms/step - loss: 1.4114 - accuracy: 0.3988 -
val_loss: 1.1200 - val_accuracy: 0.5450
Epoch 2/35
92/92 [==============================] - 5s 55ms/step - loss: 1.1092 - accuracy: 0.5548 -
val_loss: 1.0482 - val_accuracy: 0.5790
Epoch 3/35
92/92 [==============================] - 5s 54ms/step - loss: 0.9933 - accuracy: 0.6090 -
val_loss: 0.9491 - val_accuracy: 0.6090
Epoch 4/35
92/92 [==============================] - 5s 56ms/step - loss: 0.9017 - accuracy: 0.6499 -
val_loss: 0.8856 - val_accuracy: 0.6335
Epoch 5/35
92/92 [==============================] - 5s 57ms/step - loss: 0.8647 - accuracy: 0.6621 -
val_loss: 0.8315 - val_accuracy: 0.6826
```

```
Epoch 6/35
92/92 [==============================] - 5s 53ms/step - loss: 0.8052 - accuracy: 0.6836 -
val_loss: 0.8086 - val_accuracy: 0.7044
Epoch 7/35
92/92 [==============================] - 5s 52ms/step - loss: 0.7510 - accuracy: 0.7136 -
val_loss: 0.8714 - val_accuracy: 0.6567
Epoch 8/35
92/92 [==============================] - 5s 55ms/step - loss: 0.7359 - accuracy: 0.7173 -
val_loss: 0.8816 - val_accuracy: 0.6485
Epoch 9/35
92/92 [==============================] - 5s 57ms/step - loss: 0.7102 - accuracy: 0.7340 -
val_loss: 0.7484 - val_accuracy: 0.7030
Epoch 10/35
92/92 [==============================] - 5s 56ms/step - loss: 0.6564 - accuracy: 0.7493 -
val_loss: 0.8033 - val_accuracy: 0.7030
Epoch 11/35
92/92 [==============================] - 5s 56ms/step - loss: 0.6407 - accuracy: 0.7541 -
val_loss: 0.7205 - val_accuracy: 0.7193
Epoch 12/35
92/92 [==============================] - 5s 54ms/step - loss: 0.6252 - accuracy: 0.7687 -
val_loss: 0.7144 - val_accuracy: 0.7166
Epoch 13/35
92/92 [==============================] - 5s 53ms/step - loss: 0.6054 - accuracy: 0.7728 -
val_loss: 0.7206 - val_accuracy: 0.7207
Epoch 14/35
92/92 [==============================] - 5s 54ms/step - loss: 0.5691 - accuracy: 0.7820 -
val_loss: 0.7445 - val_accuracy: 0.7289
Epoch 15/35
92/92 [==============================] - 5s 53ms/step - loss: 0.5535 - accuracy: 0.7905 -
val_loss: 0.7016 - val_accuracy: 0.7302
Epoch 16/35
92/92 [==============================] - 5s 53ms/step - loss: 0.4949 - accuracy: 0.8113 -
val_loss: 0.7093 - val_accuracy: 0.7343
Epoch 17/35
92/92 [==============================] - 5s 55ms/step - loss: 0.5035 - accuracy: 0.8038 -
val_loss: 0.7374 - val_accuracy: 0.7207
Epoch 18/35
92/92 [==============================] - 5s 53ms/step - loss: 0.4715 - accuracy: 0.8215 -
val_loss: 0.6697 - val_accuracy: 0.7452
Epoch 19/35
92/92 [==============================] - 5s 53ms/step - loss: 0.4555 - accuracy: 0.8324 -
val_loss: 0.6963 - val_accuracy: 0.7480
Epoch 20/35
92/92 [==============================] - 5s 52ms/step - loss: 0.4439 - accuracy: 0.8321 -
val_loss: 0.6642 - val_accuracy: 0.7670
```

```
Epoch 21/35
92/92 [==============================] - 5s 53ms/step - loss: 0.4315 - accuracy: 0.8386 -
val_loss: 0.7395 - val_accuracy: 0.7398
Epoch 22/35
92/92 [==============================] - 5s 54ms/step - loss: 0.4229 - accuracy: 0.8430 -
val_loss: 0.6980 - val_accuracy: 0.7520
Epoch 23/35
92/92 [==============================] - 5s 53ms/step - loss: 0.3817 - accuracy: 0.8583 -
val_loss: 0.8149 - val_accuracy: 0.7262
Epoch 24/35
92/92 [==============================] - 5s 56ms/step - loss: 0.3776 - accuracy: 0.8610 -
val_loss: 0.6811 - val_accuracy: 0.7725
Epoch 25/35
92/92 [==============================] - 5s 53ms/step - loss: 0.3409 - accuracy: 0.8702 -
val_loss: 0.7799 - val_accuracy: 0.7520
Epoch 26/35
92/92 [==============================] - 5s 55ms/step - loss: 0.3486 - accuracy: 0.8682 -
val_loss: 0.7301 - val_accuracy: 0.7520
Epoch 27/35
92/92 [==============================] - 5s 53ms/step - loss: 0.3146 - accuracy: 0.8794 -
val_loss: 0.7901 - val_accuracy: 0.7548
Epoch 28/35
92/92 [==============================] - 5s 54ms/step - loss: 0.2991 - accuracy: 0.8968 -
val_loss: 0.7249 - val_accuracy: 0.7670
Epoch 29/35
92/92 [==============================] - 5s 53ms/step - loss: 0.3128 - accuracy: 0.8893 -
val_loss: 0.7471 - val_accuracy: 0.7602
Epoch 30/35
92/92 [==============================] - 5s 53ms/step - loss: 0.2812 - accuracy: 0.8958 -
val_loss: 0.7277 - val_accuracy: 0.7738
Epoch 31/35
92/92 [==============================] - 5s 53ms/step - loss: 0.2688 - accuracy: 0.8999 -
val_loss: 0.7353 - val_accuracy: 0.7520
Epoch 32/35
92/92 [==============================] - 5s 55ms/step - loss: 0.2611 - accuracy: 0.9067 -
val_loss: 0.8474 - val_accuracy: 0.7371
Epoch 33/35
92/92 [==============================] - 5s 54ms/step - loss: 0.2534 - accuracy: 0.9118 -
val_loss: 0.7922 - val_accuracy: 0.7711
Epoch 34/35
92/92 [==============================] - 5s 53ms/step - loss: 0.2297 - accuracy: 0.9138 -
val_loss: 0.8594 - val_accuracy: 0.7493
Epoch 35/35
92/92 [==============================] - 5s 54ms/step - loss: 0.2312 - accuracy: 0.9227 -
val_loss: 0.8389 - val_accuracy: 0.7425

In [29]:
```

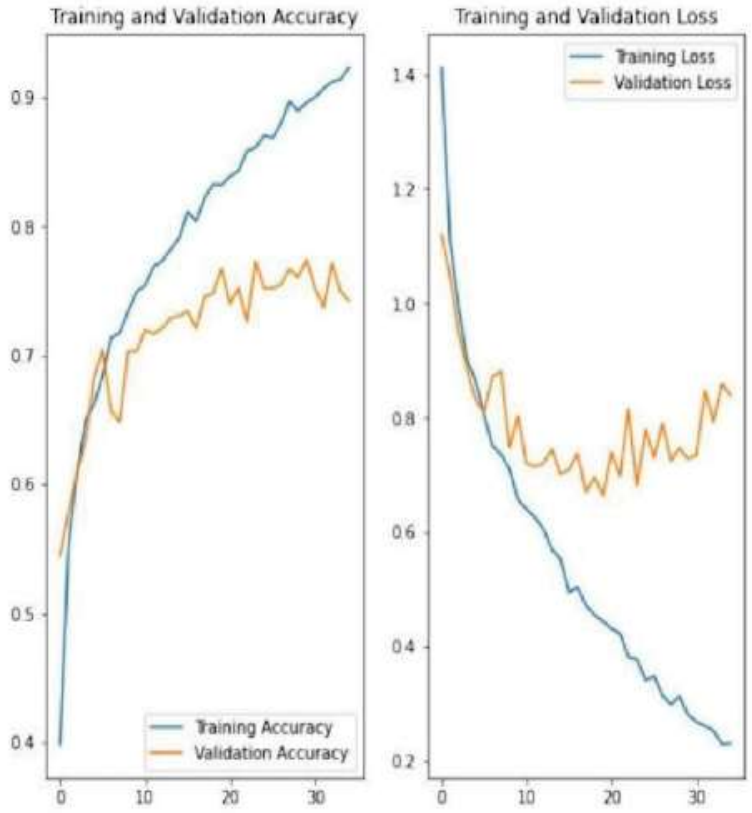
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

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

epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```



In [30]:

```
tulip_url = "https://5.imimg.com/data5/ZR/OO/EA/SELLER-34246236/tulip-flower-500x500.jpg"
tulip_path = tf.keras.utils.get_file('sunflowers', origin=tulip_url)

img = tf.keras.utils.load_img(
    tulip_path, target_size=(img_height, img_width)
)
img_array = tf.keras.utils.img_to_array(img)
img_array = tf.expand_dims(img_array, 0) # Create a batch

predictions = model.predict(img_array)
score = tf.nn.softmax(predictions[0])

print(
    "This image most likely belongs to {} with a {:.2f} percent confidence."
    .format(class_names[np.argmax(score)], 100 * np.max(score))
)
```

```
This image most likely belongs to tulips with a 99.98 percent confidence.
```

CONCLUSION:

We were able to use AI (Artificial Intelligence) and blockchain in virtual forensics by establishing a document finder, image classifier, Filecoin, and IPFS by storing criminal records in a decentralised format, including a blockchain that combines the first-class functions of IPFS and Filecoin, and to analyse and correlate the virtual statistics contained in an investigation's evidence based on their expertise, thereby reducing the number of records. In addition, we developed an application that provides an efficient storing and sharing machine.

REFERENCES

- Sanders, Nathan E., Chris Faesi, and Alyssa A. Goodman. "A New Approach to Developing Interactive Software Modules Through Graduate Education." Journal of Science Education and Technology 23, no. 3 (2014): 431–40. http://www.jstor.org/stable/24019797.
- Z.L. Gu, Research on development model of multi-media teaching software base on CSCW, Journal of Computer Engineering and Applications, issue 9, pp.1628-1630, 2006.

- Y.S.Zhang, X. Li, Software Development Models: a Survey, Journal of Computer Engineering and Applications, issue 3, pp.109-110, 2006.
- Y.M. Du, S.X. Li, Estimation Process Model for RUP Project, Journal of Computer Science, vol. 40, issue 6, pp.21-26, 2013.
- X.Zou, Method of Construction: Modern Software Engineering, Posts & Telecom Press, 2018.
- J.J. Yu, Research on designing and achievement on RUP improvement model of instructional software, Journal of E-education Research, issue 4, pp.76-81, 2012.
- D.Q. Xie, Double Iteration Model of Agile Software Development, Journal of Computer Applications and Software, vol. 29, issue 6, pp.176-178, 2012.
- Y. Wu, J.Y. Qian and Y. Liu, MDA-Based Component Employer Method Research and Implementation, Journal of Control Automation, issue 27, pp.198-200, 2010.
- S.M. Zhu, Software Testing (Second Version). Posts &Telecom Press, 2016.
- R.M. Zhang, D. Yang and J. Li, Design of requirements negotiation tool based on WinWin theory, Journal of Computer Engineering and Applications, issue 1, pp.100-104, 2009.
- J.J. Ma, Three Triple Iterative Model Based on Agile, Journal of Electronic Technology & Software Engineering, issue 6, pp.52-54, 2017.
- S.J. Chen, Inverted A-Model for Stable Software Development, Journal of Software, vol. 37, issue 12, pp.07-11, 2016.
- Y.L. Si and W.H. Liu, Research of a New Software Development Cooperative Model, Journal of Microelectronics Computer, issue 5, pp.73-76, 2012.