Sensitive Data Classification Using Deep Learning

Importing libraries and downloading the dataset

```
# Importing the necessary libraries
import tensorflow as tf
import numpy as np
import os

# Downloading the dataset
git_folder = "/content/SPECIAL-TOPICS-67"
if os.path.exists(git_folder) == False:
  !git clone https://github.com/manjuv03/SPECIAL-TOPICS-67

training_folder·=·git_folder·+·"/dataset/training"
validation_folder·=·git_folder·+·"/dataset/validation"
```

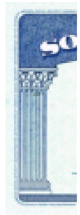
```
Cloning into 'SPECIAL-TOPICS-67'...
remote: Enumerating objects: 711, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 711 (delta 0), reused 6 (delta 0), pack-reused 705
Receiving objects: 100% (711/711), 78.76 MiB | 46.62 MiB/s, done.
Resolving deltas: 100% (2/2), done.
```

Sample Data

```
%matplotlib inline

import matplotlib.pyplot as plt
import matplotlib.image as mpimg

# Parameters for our graph; we'll output images in a 4x4 configuration
nrows = 4
ncols = 4

# Index for iterating over images
pic_index = 0

# Set up matplotlib fig, and size it to fit 4x4 pics
fig = plt.gcf()
fig.set_size_inches(ncols * 4, nrows * 4)

train_sensitive_dir = os.path.join(training_folder+"/sensitive")
train_nonsensitive_dir = os.path.join(training_folder+"/nonsensitive")
train_sensitive_names = os.listdir(train_sensitive_dir)
train_nonsensitive_names = os.listdir(train_nonsensitive_dir)
```

```
pic_index += 8
next_sensitive_pix = [os.path.join(train_sensitive_dir, fname)
                for fname in train_sensitive_names[pic_index-8:pic_index]]
next_nonsensitive_pix = [os.path.join(train_nonsensitive_dir, fname)
                for fname in train_nonsensitive_names[pic_index-8:pic_index]]

for i, img_path in enumerate(next_sensitive_pix+next_nonsensitive_pix):
  # Set up subplot; subplot indices start at 1
  sp = plt.subplot(nrows, ncols, i + 1)
  sp.axis('Off') # Don't show axes (or gridlines)

  img = mpimg.imread(img_path)
  plt.imshow(img)

plt.show()
```

Model

```
# Callbacks to cancel training after reaching a desired accuracy
# This is done to avoid overfitting
DESIRED_ACCURACY = 0.98

class myCallback(tf.keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs={}):
    if logs.get('accuracy') > DESIRED_ACCURACY:
      print("Reached 98% accuracy so cancelling training!")
      self.model.stop_training = True

callbacks = myCallback()

# Sequential - defines a SEQUENCE of layers in the neural network.
model = tf.keras.models.Sequential([
  # 2D Convolution Layer - Filter, Kernel_size, activation fn
  tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(150,150,3)),
  # Max pooling operation for 2D data - Pool size
  tf.keras.layers.MaxPooling2D(2,2),
  tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
  tf.keras.layers.MaxPooling2D(2,2),
  tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
  tf.keras.layers.MaxPooling2D(2,2),
  tf.keras.layers.Conv2D(256, (3,3), activation='relu'),
  tf.keras.layers.MaxPooling2D(2,2),
  # Flattens the input. Does not affect the batch size.
  tf.keras.layers.Flatten(),
  # Regular densely-connected Neural Network layer with ReLU activation function.
  tf.keras.layers.Dense(512, activation='relu'),
  # Regular densely-connected Neural Network layer with sigmoid activation function.
  tf.keras.layers.Dense(1, activation='sigmoid')
])

from tensorflow.keras.optimizers import RMSprop

# model.compile - Configures the model for training.
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
# Adam -  optimization algorithm used instead of the classical stochastic gradient desc
```

```
# Display the summary of the model
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 148, 148, 32)      896

 max_pooling2d (MaxPooling2D  (None, 74, 74, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 72, 72, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 36, 36, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 34, 34, 128)       73856

 max_pooling2d_2 (MaxPooling  (None, 17, 17, 128)      0
 2D)

 conv2d_3 (Conv2D)           (None, 15, 15, 256)       295168

 max_pooling2d_3 (MaxPooling  (None, 7, 7, 256)        0
 2D)

 flatten (Flatten)           (None, 12544)             0

 dense (Dense)               (None, 512)               6423040

 dense_1 (Dense)             (None, 1)                 513

=================================================================
Total params: 6,811,969
Trainable params: 6,811,969
Non-trainable params: 0
```

Preprocessing, Data Augmentation & Training

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Rescaling - 1/255 is to transform every pixel value from range [0,255] -> [0,1]
# Performing image augmentation for training data
train_datagen = ImageDataGenerator(rescale=1/255,
                                  rotation_range=20,
                                  width_shift_range=0.2,
                                  height_shift_range=0.2,
                                  shear_range=0.2,
                                  zoom_range=0.2,
                                  horizontal_flip=True,
                                  vertical_flip=True,
                                  fill_mode='nearest')
validation_datagen = ImageDataGenerator(rescale=1/255)
```

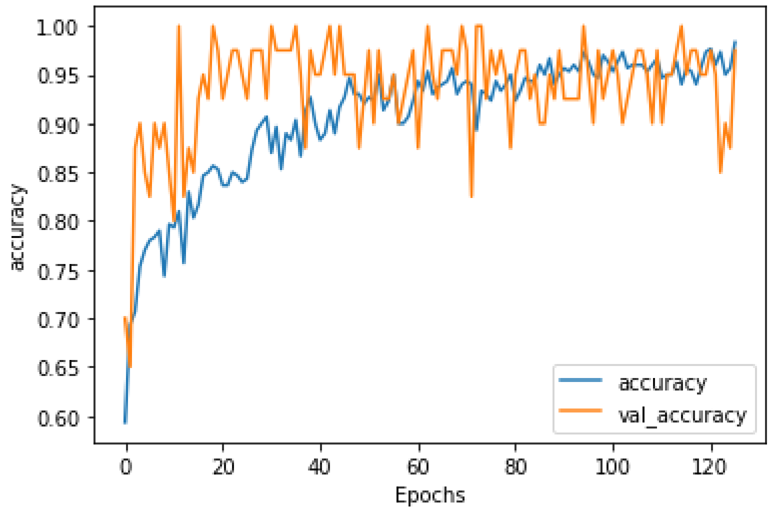
```
# flow_from_directory - Takes the path to a directory & generates batches of data.
train_generator = train_datagen.flow_from_directory(
        training_folder,
        target_size=(150, 150),
        batch_size=30,
        class_mode='binary',
        shuffle=True
       )

validation_generator = validation_datagen.flow_from_directory(
        validation_folder,
        target_size=(150, 150),
        batch_size=5,
        class_mode='binary',
        shuffle=True
        )

num_epochs = 500
# model.fit - Train the model for a fixed number of epochs
history = model.fit(
      train_generator,
      steps_per_epoch=10,
      epochs=num_epochs,
      verbose=1,
      validation_data = validation_generator,
      validation_steps=8,
      callbacks=[callbacks])
```

```
Found 600 images belonging to 2 classes.
Found 100 images belonging to 2 classes.
Epoch 1/500
10/10 [==============================] - 15s 360ms/step - loss: 0.7869 - accura
Epoch 2/500
10/10 [==============================] - 4s 381ms/step - loss: 0.5833 - accurac
Epoch 3/500
10/10 [==============================] - 4s 374ms/step - loss: 0.5806 - accurac
Epoch 4/500
10/10 [==============================] - 4s 378ms/step - loss: 0.5319 - accurac
Epoch 5/500
10/10 [==============================] - 4s 355ms/step - loss: 0.4672 - accurac
Epoch 6/500
10/10 [==============================] - 4s 382ms/step - loss: 0.4858 - accurac
Epoch 7/500
10/10 [==============================] - 5s 475ms/step - loss: 0.4732 - accurac
Epoch 8/500
10/10 [==============================] - 4s 386ms/step - loss: 0.4542 - accurac
Epoch 9/500
10/10 [==============================] - 4s 382ms/step - loss: 0.4721 - accurac
Epoch 10/500
10/10 [==============================] - 4s 387ms/step - loss: 0.4427 - accurac
Epoch 11/500
10/10 [==============================] - 4s 371ms/step - loss: 0.4896 - accurac
Epoch 12/500
10/10 [==============================] - 4s 363ms/step - loss: 0.4561 - accurac
Epoch 13/500
10/10 [==============================] - 4s 367ms/step - loss: 0.5001 - accurac
Epoch 14/500
```

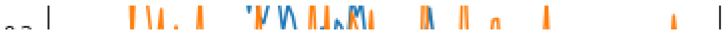
```
10/10 [==============================] - 4s 366ms/step - loss: 0.3883 - accurac
Epoch 15/500
10/10 [==============================] - 4s 374ms/step - loss: 0.4079 - accurac
Epoch 16/500
10/10 [==============================] - 4s 396ms/step - loss: 0.3949 - accurac
Epoch 17/500
10/10 [==============================] - 4s 384ms/step - loss: 0.3838 - accurac
Epoch 18/500
10/10 [==============================] - 4s 389ms/step - loss: 0.3678 - accurac
Epoch 19/500
10/10 [==============================] - 4s 391ms/step - loss: 0.3356 - accurac
Epoch 20/500
10/10 [==============================] - 4s 364ms/step - loss: 0.3431 - accurac
Epoch 21/500
10/10 [==============================] - 4s 377ms/step - loss: 0.3968 - accurac
Epoch 22/500
10/10 [==============================] - 4s 375ms/step - loss: 0.3693 - accurac
Epoch 23/500
10/10 [==============================] - 4s 382ms/step - loss: 0.3633 - accurac
Epoch 24/500
10/10 [==============================] - 4s 413ms/step - loss: 0.3132 - accurac
Epoch 25/500
10/10 [==============================] - 4s 386ms/step - loss: 0.3215 - accurac
Epoch 26/500
10/10 [==============================] - 4s 371ms/step - loss: 0.3534 - accurac
Epoch 27/500
10/10 [==============================] - 4s 374ms/step - loss: 0.2998 - accurac
```

Plotting Accuracy and Loss Functions

```
import matplotlib.pyplot as plt

# Plot the accuracy and loss functions
def plot_graphs(history, string):
  plt.plot(history.history[string])
  plt.plot(history.history['val_'+string])
  plt.xlabel("Epochs")
  plt.ylabel(string)
  plt.legend([string, 'val_'+string])
  plt.show()

plot_graphs(history, "accuracy")
plot_graphs(history, "loss")
```



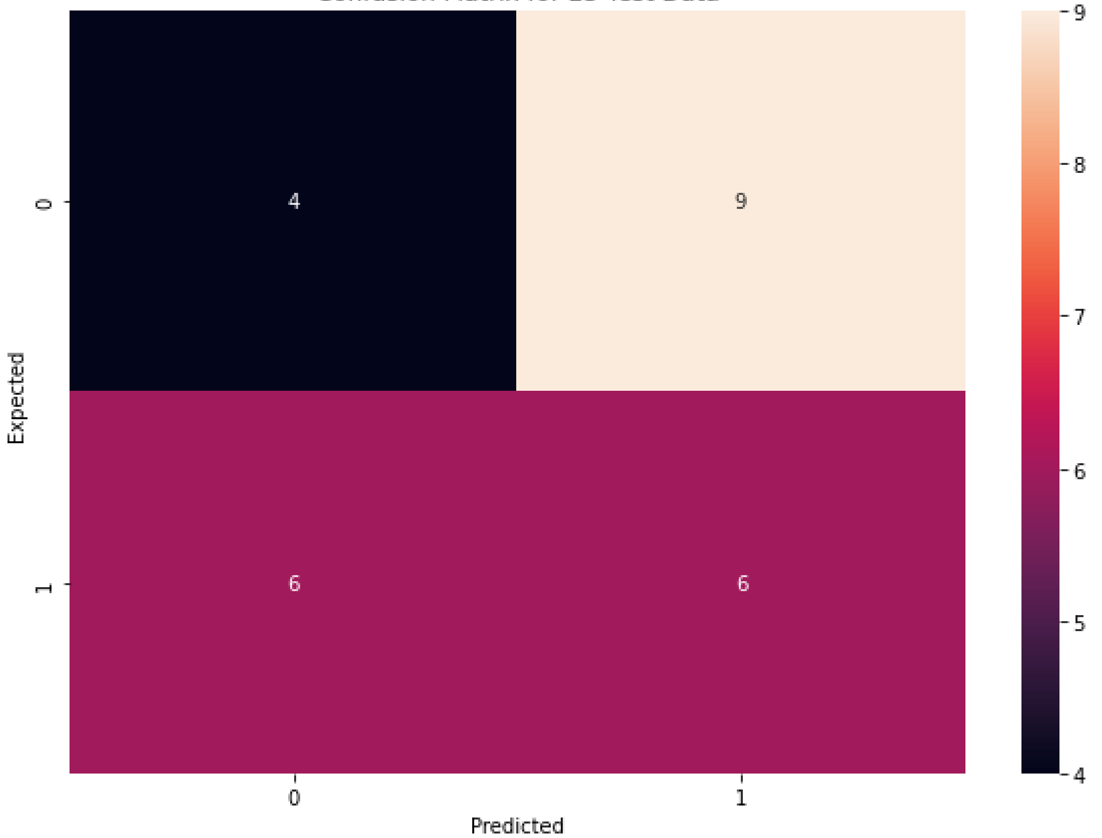
Confusion Matrix



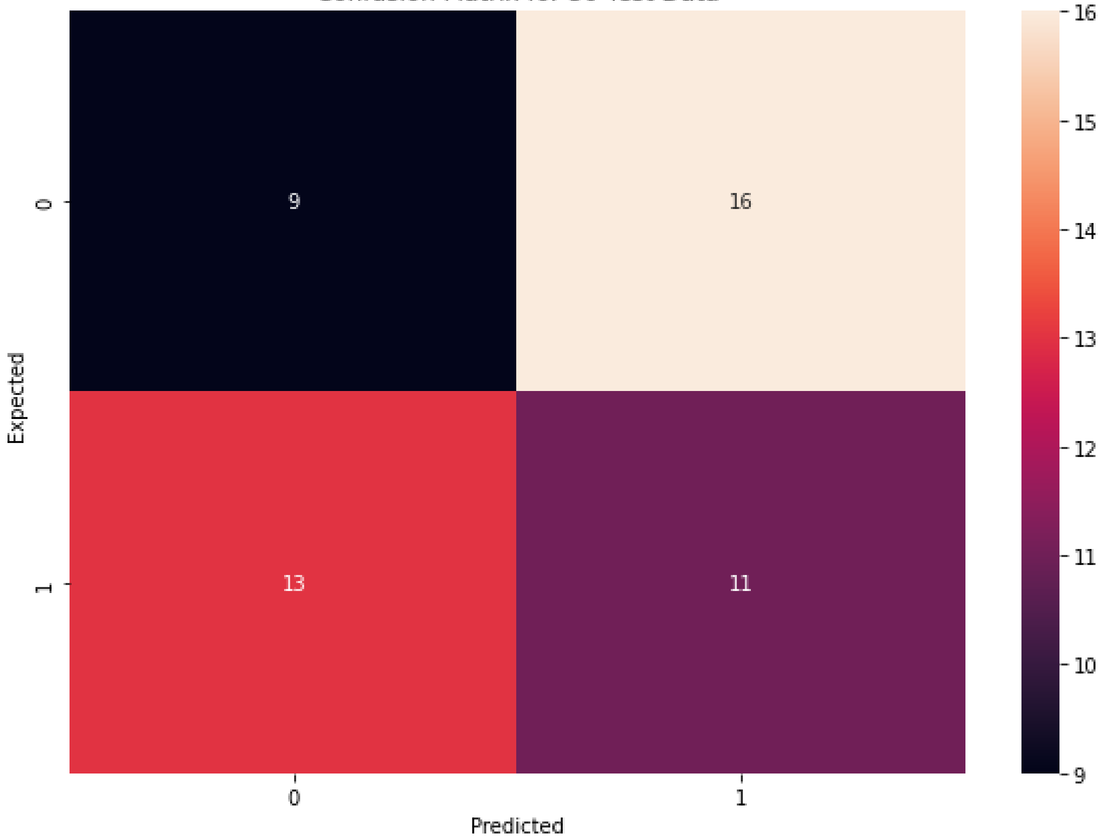
```
import seaborn
y_pred = model.predict(validation_generator, 20)
print('Confusion Matrix')
y_predicted_labels = y_pred > 0.5

size = np.size(y_predicted_labels)
y_predicted_labels = y_predicted_labels.reshape(size, )

for i in range (1, 5):
  total = i * size // 4
  mid = 49
  start = mid-((total+1)//2)+1
  end = mid+((total+1)//2)
  cm = tf.math.confusion_matrix(labels=validation_generator.labels[start:end],predictic

  # Calculate accuracy
  cm_np = cm.numpy()
  conf_acc = (cm_np[0, 0] + cm_np[1, 1])/ np.sum(cm_np) * 100
  print("Accuracy for", str(total), "Test Data = ", conf_acc)

  # Plot the confusion matrix
  plt.figure(figsize = (10,7))
  seaborn.heatmap(cm, annot=True, fmt='d')
  plt.title("Confusion Matrix for " + str(total) + " Test Data")
  plt.xlabel('Predicted')
  plt.ylabel('Expected')
```

```
Confusion Matrix
Accuracy for 25 Test Data =  40.0
Accuracy for 50 Test Data =  40.816326530612244
Accuracy for 75 Test Data =  45.333333333333333
Accuracy for 100 Test Data =  51.515151515151516
```

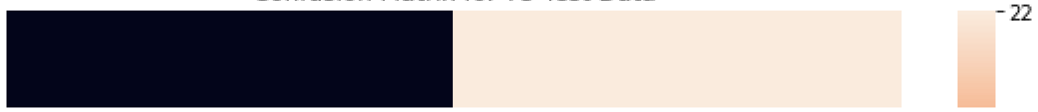
Confusion Matrix for 25 Test Data

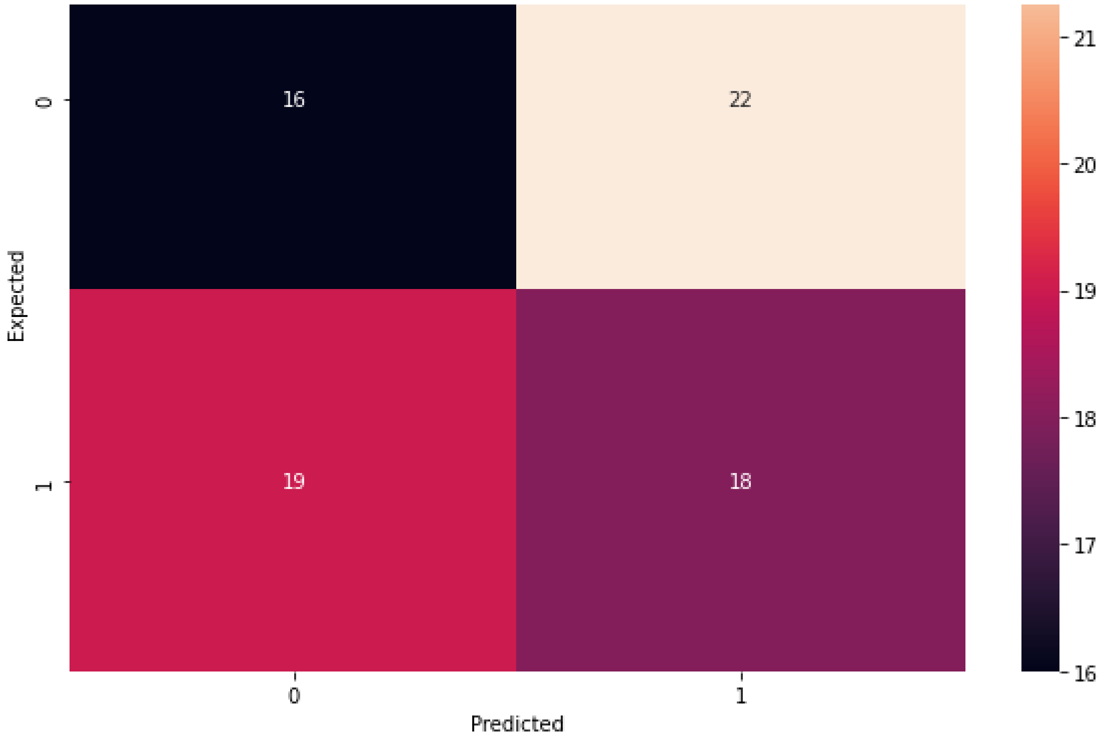


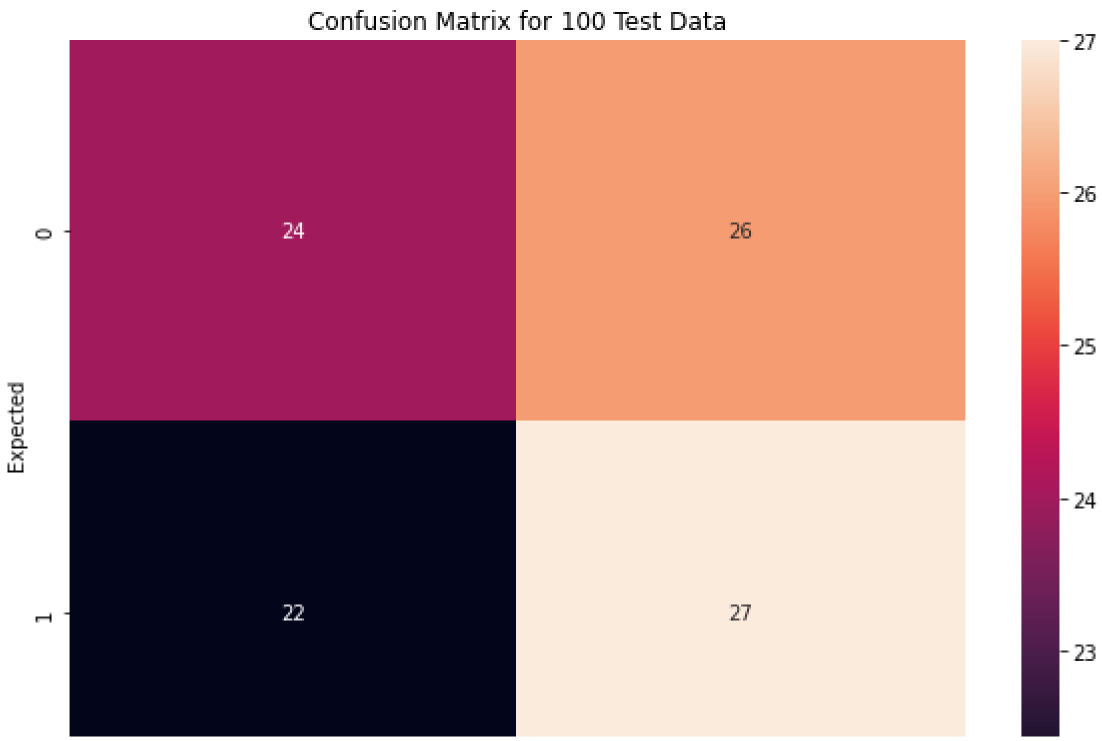
Confusion Matrix for 50 Test Data



Confusion Matrix for 75 Test Data







Sample Example

```
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()
result = dict()

for fn in uploaded.keys():

  # predicting images
  path = '/content/' + fn
  img = image.load_img(path, target_size=(150, 150))
```

```
    x = image.img_to_array(img)
    x = np.expand_dims(x, axis=0)

    images = np.vstack([x])
    classes = model.predict(images, batch_size=10)
    print(classes[0])
    if classes[0]>0.5:
      print(fn + " - Sensitive")
      result[fn] = "Sensitive"
    else:
      print(fn + " - Non-sensitive")
      result[fn] = "Non-sensitive"

  plt.figure(figsize=(20,20))

  for i, fn in enumerate(uploaded.keys()):
    image = plt.imread(fn)
    plt.subplot(5, 5, i+1)
    plt.axis("off")
    plt.imshow(image)
    ans = fn + ": " + result[fn]
    plt.title(ans)
```

Choose Files No file chosen Upload widget is only available when the cell has been executed i session. Please rerun this cell to enable.

```
Saving 1 (1).gif to 1 (1).gif
Saving 1 (1).jpeg to 1 (1) (1).jpeg
Saving 1 (1).jpg to 1 (1).jpg
Saving 1 (1).png to 1 (1).png
Saving 1 (3).jpeg to 1 (3).jpeg
[1.]
1 (1).gif - Sensitive
[1.]
1 (1).jpeg - Sensitive
[0.]
1 (1).jpg - Non-sensitive
[1.]
1 (1).png - Sensitive
[1.]
1 (3).jpeg - Sensitive
```

1 (1).gif: Sensitive

1 (1).jpeg: Sensitive

1 (1).jpg: Non-sensitive

1 (1).png: Sensitive