

```


import os
import time
import shutil
import pathlib
import itertools
from PIL import Image
import random

# import data handling tools
import cv2
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
# import Deep learning Libraries
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam, Adamax
from tensorflow.keras.metrics import categorical_crossentropy
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Activation, Dropout, BatchNormalization
from tensorflow.keras import regularizers

# Ignore Warnings
import warnings
warnings.filterwarnings("ignore")

print ('modules loaded')

```

 modules loaded

```

# Define the directories
train_dir = r'C:\Users\shash\Downloads\archive\train'      # Replace with your train directory
valid_dir = r'C:\Users\shash\Downloads\archive\valid'      # Replace with your validation directory
test_dir = r'C:\Users\shash\Downloads\archive\test'        # Replace with your test directory

# Function to display one random sample from each class
def display_random_sample(dataset_dir):
    classes = os.listdir(dataset_dir) # List of class directories
    plt.figure(figsize=(15, 10)) # Adjust the figure size if needed

    for i, class_name in enumerate(classes):
        class_dir = os.path.join(dataset_dir, class_name)
        image_files = os.listdir(class_dir) # Get all images in the class folder
        random_image = random.choice(image_files) # Select a random image
        img_path = os.path.join(class_dir, random_image)

        # Load the image with its original size
        img = load_img(img_path)

        # Display the image
        plt.subplot(1, len(classes), i + 1) # Display images in a single row
        plt.imshow(img)
        plt.axis('off') # Turn off axis for better visualization
        plt.title(f"{class_name} ({img.size[0]}x{img.size[1]})") # Show class name and image size

    plt.show()

# Display one random sample from train, validation, and test sets
print("Training set samples:")
display_random_sample(train_dir)

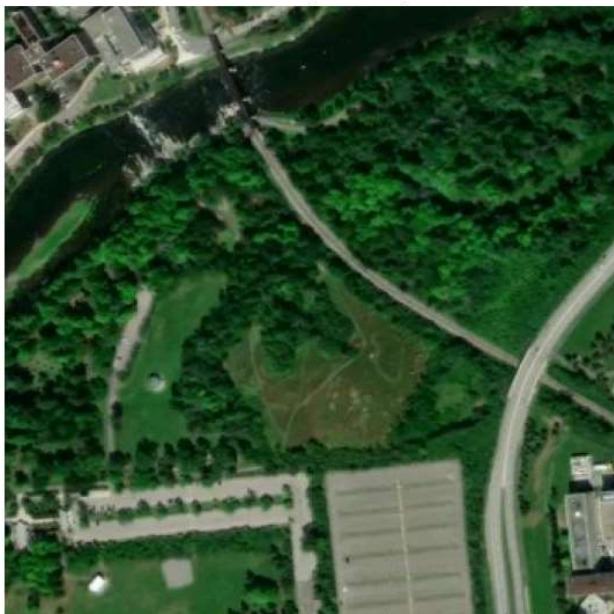
print("Validation set samples:")
display_random_sample(valid_dir)

print("Test set samples:")
display_random_sample(test_dir)

```

↻ Training set samples:

nowildfire (350x350)

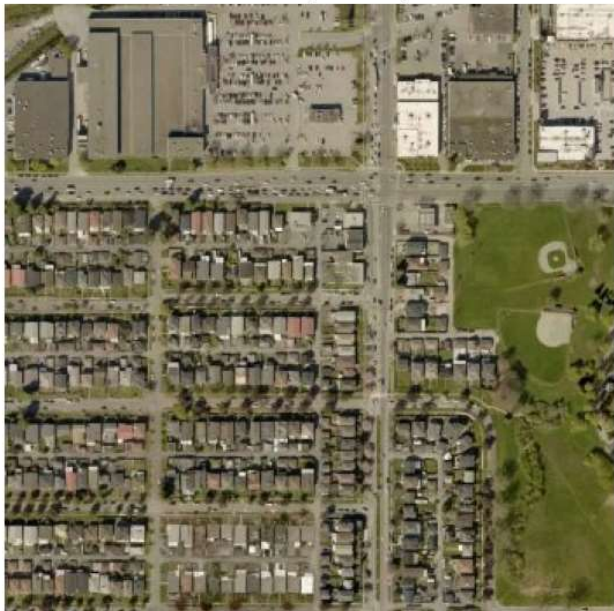


wildfire (350x350)



Validation set samples:

nowildfire (350x350)



wildfire (350x350)



Test set samples:

nowildfire (350x350)



wildfire (350x350)





```
dir = r'C:\Users\shash\Downloads\archive\train'
x_train = []
y_train = []
for direct in os.listdir(dir):
    print("Loading dataset training {}".format(direct))
    for filename in os.listdir(os.path.join(dir,direct)):
        img_path = os.path.join(dir,direct,filename)
        img = cv2.imread(img_path)
        img = cv2.resize(img, (32,32))
        img = np.array(img)
        img = img/255
        x_train.append(img)
        y_train.append(direct)
```

↗ Loading dataset training nowildfire
Loading dataset training wildfire

```
dir_val = r'C:\Users\shash\Downloads\archive\valid'
x_val=[]
y_val=[]
for direct in os.listdir(dir_val):
    print("Loading dataset validation {}".format(direct))
    for filename in os.listdir(os.path.join(dir_val,direct)):
        img_path = os.path.join(dir_val,direct,filename)
        image = cv2.imread(img_path)
        image = cv2.resize(image,(32,32))
        image = np.array(image)
        image = image/255
        x_val.append(image)
        y_val.append(direct)
```

↗ Loading dataset validation nowildfire
Loading dataset validation wildfire

```
dir_test = r'C:\Users\shash\Downloads\archive\test'
x_test=[]
y_test=[]
for direct in os.listdir(dir_test):
    print("Loading dataset test {}".format(direct))
    for filename in os.listdir(os.path.join(dir_test,direct)):
        img_path = os.path.join(dir_test,direct,filename)
        image = cv2.imread(img_path)
        image = cv2.resize(image,(32,32))
        image = np.array(image)
        image = image/255
        x_test.append(image)
        y_test.append(direct)
```

↗ Loading dataset test nowildfire
Loading dataset test wildfire

```
x_train = np.array(x_train)
x_val = np.array(x_val)
x_test = np.array(x_test)
```

```
y_train[30000]
```

```
↗ 'wildfire'
```

```
# Replace "wildfire" with 1 and "nowildfire" with 0
y_train = [1 if label == 'wildfire' else 0 for label in y_train]
y_val = [1 if label == 'wildfire' else 0 for label in y_val]
y_test = [1 if label == 'wildfire' else 0 for label in y_test]
```

```
y_train = np.array(y_train)
y_val = np.array(y_val)
y_test = np.array(y_test)
```

```
len(x_train[4][4])
```

```
↗ 32
```

```
# Now check the shape of your datasets
print("x_train shape:", x_train.shape) # Should be (num_samples, height, width, num_channels)
print("x_valid shape:", x_val.shape)
print("x_test shape:", x_test.shape)
```

```
print("y_train shape:", y_train.shape)
print("y_valid shape:", y_val.shape)
print("y_test shape:", y_test.shape)
```

```
↗ x_train shape: (30250, 32, 32, 3)
x_valid shape: (6300, 32, 32, 3)
x_test shape: (6300, 32, 32, 3)
y_train shape: (30250,)
y_valid shape: (6300,)
y_test shape: (6300,)
```

```
# Step 1: Build the CNN model
```

```
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    Conv2D(64, (3, 3), activation='relu', padding='same'),
    Conv2D(64, (3, 3), activation='relu', padding='same'),
    BatchNormalization(),
    MaxPooling2D((2, 2)),

    Conv2D(128, (3, 3), activation='relu', padding='same'),
    Conv2D(128, (3, 3), activation='relu', padding='same'),
    BatchNormalization(),
    MaxPooling2D((2, 2)),

    Conv2D(256, (3, 3), activation='relu', padding='same'),
    Conv2D(256, (3, 3), activation='relu', padding='same'),
    BatchNormalization(),
    MaxPooling2D((2, 2)),

    Conv2D(128, (3, 3), activation='relu', padding='same'),
    Conv2D(128, (3, 3), activation='relu', padding='same'),

    Flatten(),

    Dense(128, activation='relu'),
    Dense(128, activation='relu'),
    BatchNormalization(),
    Dropout(0.5),


    Dense(64, activation='relu'),
    Dense(32, activation='relu'),
    BatchNormalization(),

    Dense(1, activation='sigmoid') # Output layer with number of classes
])
```

```
# Step 2: Compile the model
```

```
model.compile(optimizer='adam',
              loss='binary_crossentropy', # Loss function for multi-class classification
              metrics=['accuracy'])
```

```
model.summary()
```



 Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 30, 30, 32)	896
conv2d_1 (Conv2D)	(None, 30, 30, 64)	18,496
conv2d_2 (Conv2D)	(None, 30, 30, 64)	36,928
batch_normalization (BatchNormalization)	(None, 30, 30, 64)	256
max_pooling2d (MaxPooling2D)	(None, 15, 15, 64)	0
conv2d_3 (Conv2D)	(None, 15, 15, 128)	73,856
conv2d_4 (Conv2D)	(None, 15, 15, 128)	147,584
batch_normalization_1 (BatchNormalization)	(None, 15, 15, 128)	512
max_pooling2d_1 (MaxPooling2D)	(None, 7, 7, 128)	0
conv2d_5 (Conv2D)	(None, 7, 7, 256)	295,168
conv2d_6 (Conv2D)	(None, 7, 7, 256)	590,080
batch_normalization_2 (BatchNormalization)	(None, 7, 7, 256)	1,024
max_pooling2d_2 (MaxPooling2D)	(None, 3, 3, 256)	0
conv2d_7 (Conv2D)	(None, 3, 3, 128)	295,040
conv2d_8 (Conv2D)	(None, 3, 3, 128)	147,584
flatten (Flatten)	(None, 1152)	0
dense (Dense)	(None, 128)	147,584
dense_1 (Dense)	(None, 128)	16,512
batch_normalization_3 (BatchNormalization)	(None, 128)	512
dropout (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 64)	8,256
dense_3 (Dense)	(None, 32)	2,080
batch_normalization_4 (BatchNormalization)	(None, 32)	128
dense_4 (Dense)	(None, 1)	33

```
from tensorflow.keras.callbacks import EarlyStopping
```

```
# Step 3: Train the model
```

```
history = model.fit(
    x_train, y_train,
    validation_data=(x_val, y_val),
    epochs=20,
    batch_size=64,
    callbacks=[EarlyStopping(monitor='val_loss', patience=4)]
)
```

 Epoch 1/20
473/473 ————— 368s 732ms/step - accuracy: 0.8854 - loss: 0.2866 - val_accuracy: 0.5854 - val_loss: 0.8763
Epoch 2/20
473/473 ————— 325s 687ms/step - accuracy: 0.9231 - loss: 0.1913 - val_accuracy: 0.4663 - val_loss: 1.6018
Epoch 3/20
473/473 ————— 299s 631ms/step - accuracy: 0.9344 - loss: 0.1688 - val_accuracy: 0.8922 - val_loss: 0.3052
Epoch 4/20
473/473 ————— 329s 695ms/step - accuracy: 0.9385 - loss: 0.1536 - val_accuracy: 0.9143 - val_loss: 0.2365
Epoch 5/20
473/473 ————— 314s 664ms/step - accuracy: 0.9434 - loss: 0.1448 - val_accuracy: 0.6963 - val_loss: 0.9643
Epoch 6/20
473/473 ————— 327s 691ms/step - accuracy: 0.9485 - loss: 0.1356 - val_accuracy: 0.7127 - val_loss: 0.7983
Epoch 7/20
473/473 ————— 349s 737ms/step - accuracy: 0.9511 - loss: 0.1283 - val_accuracy: 0.9235 - val_loss: 0.2031

```
Epoch 8/20
473/473 ————— 357s 754ms/step - accuracy: 0.9558 - loss: 0.1171 - val_accuracy: 0.9389 - val_loss: 0.1692
Epoch 9/20
473/473 ————— 286s 605ms/step - accuracy: 0.9578 - loss: 0.1098 - val_accuracy: 0.8790 - val_loss: 0.3389
Epoch 10/20
473/473 ————— 190s 402ms/step - accuracy: 0.9584 - loss: 0.1092 - val_accuracy: 0.7043 - val_loss: 0.8300
Epoch 11/20
473/473 ————— 206s 435ms/step - accuracy: 0.9643 - loss: 0.0981 - val_accuracy: 0.8965 - val_loss: 0.2740
Epoch 12/20
473/473 ————— 180s 381ms/step - accuracy: 0.9674 - loss: 0.0933 - val_accuracy: 0.9152 - val_loss: 0.2866
```

```
# Evaluate the model on test data
```

```
test_loss, test_accuracy = model.evaluate(x_test, y_test)
```

```
print(f'Test Accuracy: {test_accuracy:.4f}')
```

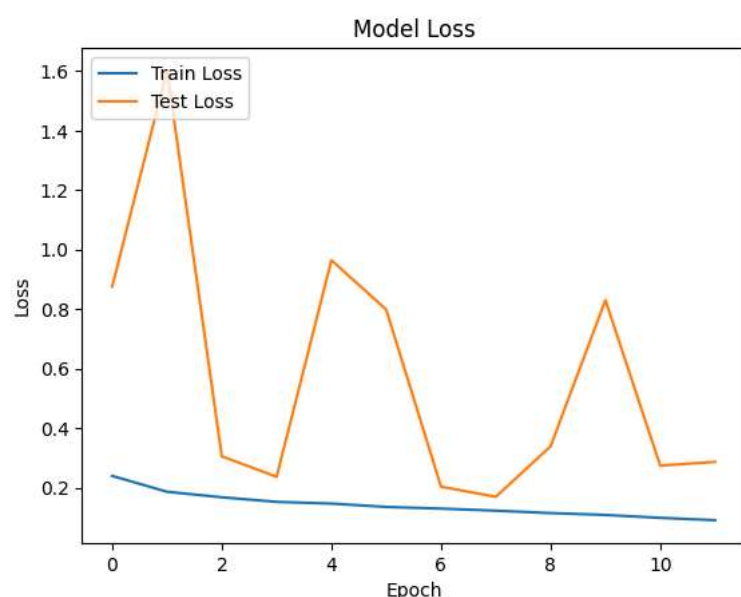
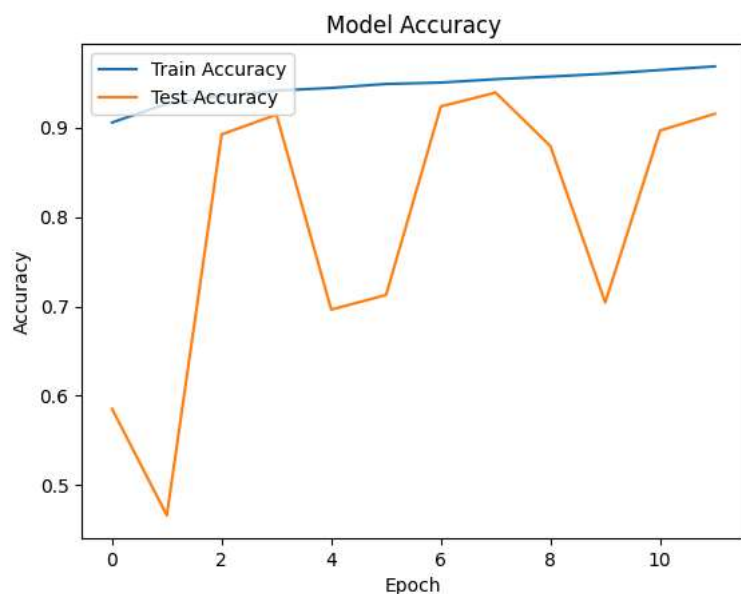
```
→ 197/197 ————— 13s 65ms/step - accuracy: 0.9671 - loss: 0.1152
Test Accuracy: 0.9197
```

```
# Plot training & validation accuracy values
```

```
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Test Accuracy')
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(loc='upper left')
plt.show()
```

```
# Plot training & validation loss values
```

```
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Test Loss')
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(loc='upper left')
plt.show()
```



```
# Predict on the datasets
y_train_pred = model.predict(x_train)
y_val_pred = model.predict(x_val)
y_test_pred = model.predict(x_test)

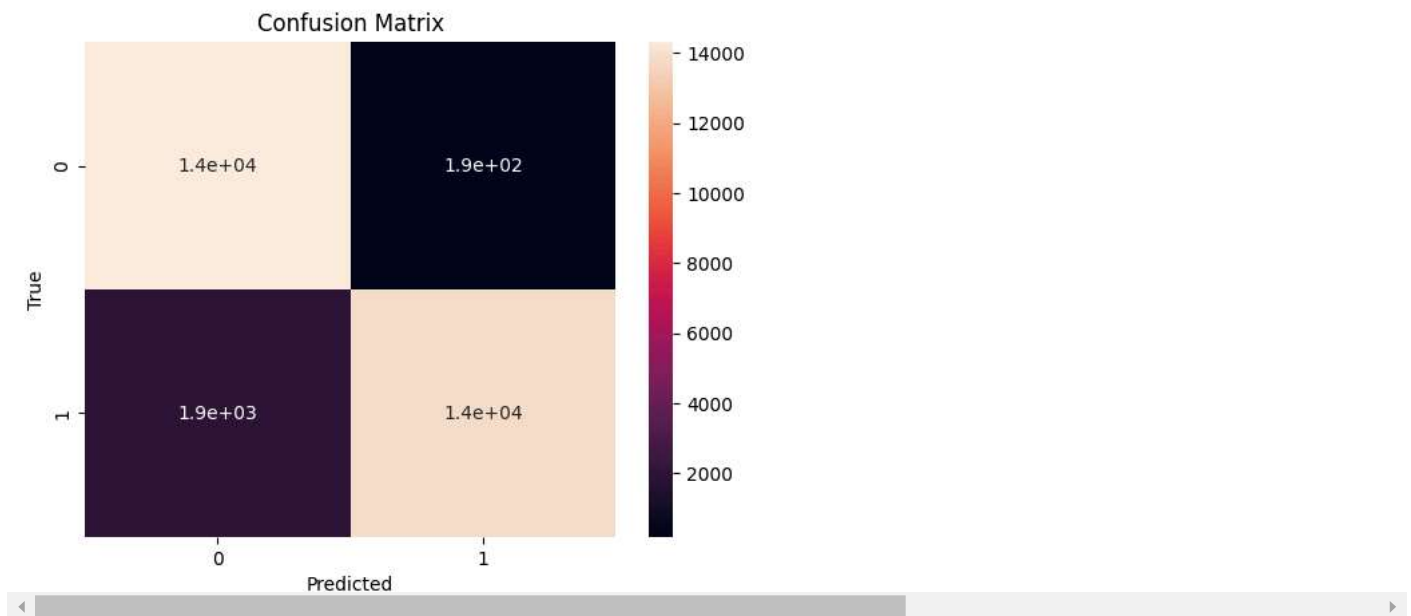
# Convert probabilities to binary classes if using a classification model
y_train_pred_classes = (y_train_pred > 0.5).astype("int32")
y_val_pred_classes = (y_val_pred > 0.5).astype("int32")
y_test_pred_classes = (y_test_pred > 0.5).astype("int32")
```



```
946/946 ————— 106s 111ms/step
197/197 ————— 29s 149ms/step
197/197 ————— 26s 131ms/step
```

```
print("Confusion matrix for train: \n")
cm = confusion_matrix(y_train, y_train_pred_classes)
sns.heatmap(cm, annot=True)
plt.title(f"Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.show()
```

↗ Confusion matrix for train:



```
from sklearn.metrics import classification_report
```

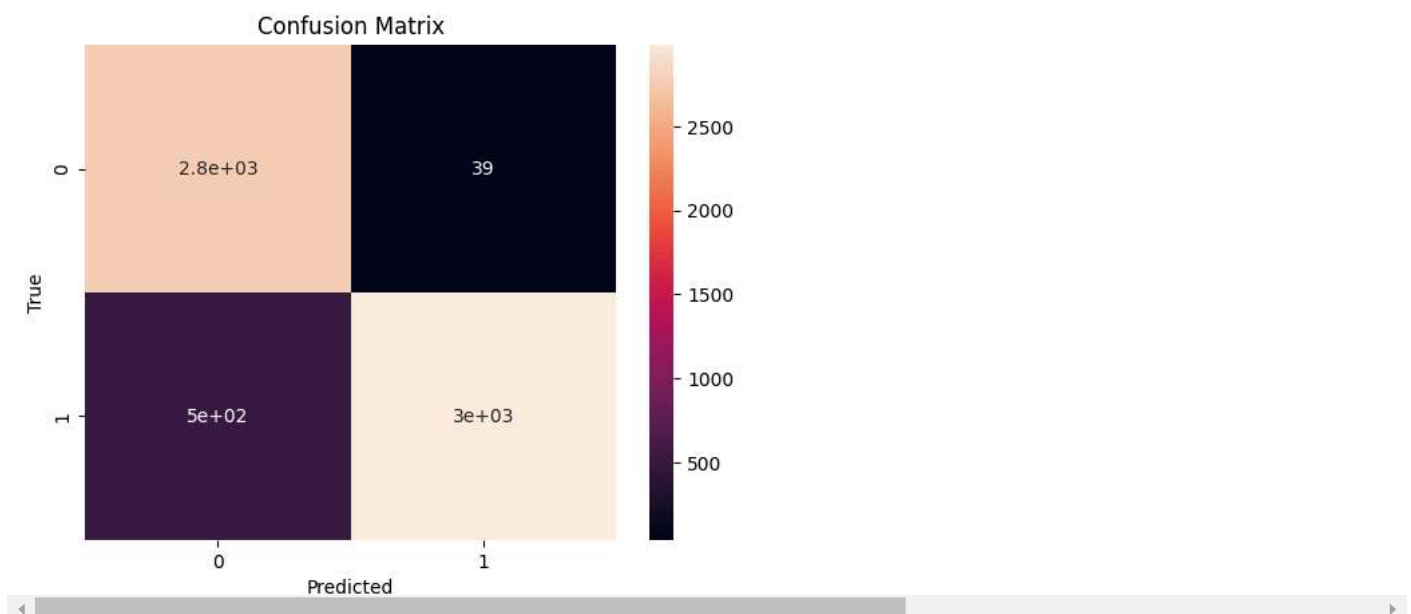
```
print(f"classification report for train : \n{classification_report(y_train, y_train_pred_classes)}")
```

↗ classification report for train :

	precision	recall	f1-score	support
0	0.88	0.99	0.93	14500
1	0.99	0.88	0.93	15750
accuracy			0.93	30250
macro avg	0.93	0.93	0.93	30250
weighted avg	0.94	0.93	0.93	30250

```
print("Confusion matrix for valid: \n")
cm = confusion_matrix(y_val, y_val_pred_classes)
sns.heatmap(cm, annot=True)
plt.title(f"Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.show()
```

↗ Confusion matrix for valid:




```
print(f"classification report for valid : \n{classification_report(y_val, y_val_pred_classes)}")
```

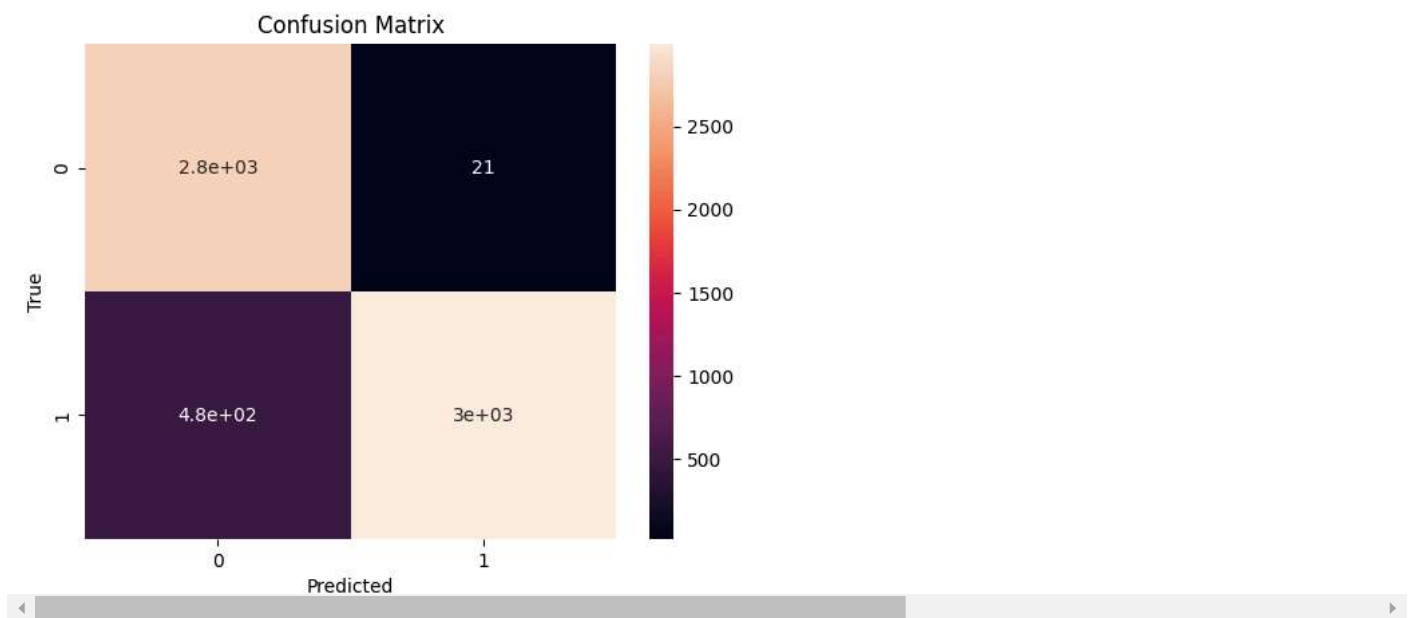
```
↗ classification report for valid :
      precision    recall  f1-score   support

     0       0.85      0.99      0.91      2820
     1       0.99      0.86      0.92      3480

 accuracy      0.92
 macro avg     0.92
 weighted avg  0.93
```

```
print("Confusion matrix for test: \n")
cm = confusion_matrix(y_test, y_test_pred_classes)
sns.heatmap(cm, annot=True)
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.show()
```

```
↗ Confusion matrix for test:
```



```
print(f"classification report for test : \n{classification_report(y_test, y_test_pred_classes)}")
```

```
↗ classification report for test :
      precision    recall  f1-score   support

     0       0.85      0.99      0.92      2820
     1       0.99      0.86      0.92      3480

 accuracy      0.92
 macro avg     0.92
 weighted avg  0.93
```

```
import cv2
img = cv2.imread('-79.4733,47.6094.jpg')
plt.imshow(cv2.cvtColor(img,cv2.COLOR_BGR2RGB))
plt.show()
```



When ever we r going to pass a image to neural network the image has to be 256px high 256 px wide and 3 tunnels

```
resize = tf.image.resize(img, (256,256))
plt.imshow(resize.numpy().astype(int))
plt.show()
```



```
resize.shape
```

```
TensorShape([256, 256, 3])
```

```
np.expand_dims(resize,0).shape
```

```
(1, 256, 256, 3)
```

```
print(model.input_shape)
```

```
(None, 32, 32, 3)
```

Start coding or [generate](#) with AI.