```
#https://www.kaggle.com/code/holdmykaggle/fire-detection-in-images/notebook
from google.colab import drive

# Mount Google Drive
drive.mount('/content/drive')
zip_file_path = '/content/drive/MyDrive/FireImageData.zip'
extract_path = '/content/FireData'

import zipfile

# Create a ZipFile object
with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    # Extract the contents to the specified folder
    zip_ref.extractall(extract_path)
```

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

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import os
import tensorflow as tf
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split

sns.set_style('darkgrid')
```

```
def extract_dataset(zip_file_path, extract_path, fire_subpath, non_fire_subpath):
    with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
        zip_ref.extractall(extract_path)

    df = pd.DataFrame(columns=['path', 'label'])

    # Loop over fire images and label them 1
    for dirname, _, filenames in os.walk(os.path.join(extract_path, fire_subpath)):
        for filename in filenames:
            df = pd.concat([df, pd.DataFrame([[os.path.join(dirname, filename), 'fire']], columns=['path', 'label'])],
                           ignore_index=True)

    # Loop over non-fire images and label them 0
    for dirname, _, filenames in os.walk(os.path.join(extract_path, non_fire_subpath)):
        for filename in filenames:
            df = pd.concat([df, pd.DataFrame([[os.path.join(dirname, filename), 'non_fire']], columns=['path', 'label'])],
                           ignore_index=True)

    # Shuffle the dataset to redistribute the labels
    df = df.sample(frac=1).reset_index(drop=True)
    return df
```

```
zip_file_path_1 = '/content/drive/MyDrive/TrainingDataset.zip'
extract_path_1 = '/content/FireTest'
df_2 = extract_dataset(zip_file_path_1, extract_path_1, 'TrainingDataset/Fire', 'TrainingDataset/NoFire')
df_2.head(10)

drive.mount('/content/drive')
zip_file_path_2 = '/content/drive/MyDrive/FireImageData.zip'
extract_path_2 = '/content/FireData'
df_1 = extract_dataset(zip_file_path_2, extract_path_2, 'fire_dataset/fire_images', 'fire_dataset/non_fire_images')
df_1.head(10)
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mour
```

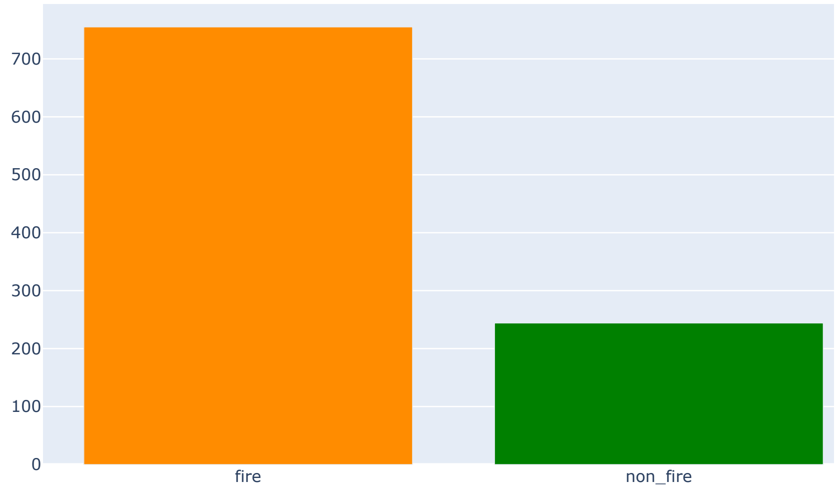
	path	label
0	/content/FireData/fire_dataset/non_fire_images...	non_fire
1	/content/FireData/fire_dataset/fire_images/fir...	fire
2	/content/FireData/fire_dataset/non_fire_images...	non_fire
3	/content/FireData/fire_dataset/non_fire_images...	non_fire
4	/content/FireData/fire_dataset/fire_images/fir...	fire
5	/content/FireData/fire_dataset/fire_images/fir...	fire
6	/content/FireData/fire_dataset/fire_images/fir...	fire
7	/content/FireData/fire_dataset/non_fire_images...	non_fire
8	/content/FireData/fire_dataset/fire_images/fir...	fire
9	/content/FireData/fire_dataset/fire_images/fir...	fire

Next steps: View recommended plots

```
# Check the size of the first dataset
print("Size of the first dataset (df_1):", df_1.shape[0])

# Check the size of the second dataset
print("Size of the second dataset (df_2):", df_2.shape[0])
```

```
Size of the first dataset (df_1): 999
Size of the second dataset (df_2): 2425
```

```
fig = make_subplots(rows=1, cols=2, specs=[[{"type": "xy"}, {"type": "pie"}]])


fig.add_trace(go.Bar(x =df_1['label'].value_counts().index,y=df_1['label'].value_counts().to_numpy(),marker_color=['darkorange','green'],shc

fig.add_trace(go.Pie(
     values=df_1['label'].value_counts().to_numpy(),
     labels=df_1['label'].value_counts().index,
    marker=dict(colors=['darkorange','green'])),
    row=1, col=2)
```



```
def shaper(row):
    shape = image.load_img(row['path']).size
    row['height'] = shape[1]
    row['width'] = shape[0]
    return row
df_1 = df_1.apply(shaper,axis=1)
df_1.head(5)
```

	path	label	height	width
0	/content/FireData/fire_dataset/non_fire_images...	non_fire	1733	2600
1	/content/FireData/fire_dataset/fire_images/fir...	fire	482	800
2	/content/FireData/fire_dataset/non_fire_images...	non_fire	185	560
3	/content/FireData/fire_dataset/non_fire_images...	non_fire	450	767
4	/content/FireData/fire_dataset/fire_images/fir...	fire	430	614

Next steps: View recommended plots

```
train_df, test_df = train_test_split(df_1, test_size=0.2, random_state=42)

# Image Augmentation

generator = ImageDataGenerator(
    rotation_range= 20,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range = 2,
    zoom_range=0.2,
    rescale = 1/255,
    validation_split=0.2,
    #horizontal_flip=True,
)

train_gen = generator.flow_from_dataframe(train_df,x_col='path',y_col='label',images_size=(256,256),class_mode='binary',subset='training')
val_gen = generator.flow_from_dataframe(train_df,x_col='path',y_col='label',images_size=(256,256),class_mode='binary',subset='validation')
test_gen = generator.flow_from_dataframe(test_df, x_col='path', y_col='label', target_size=(256, 256), class_mode='binary', subset=None)
```

```
Found 640 validated image filenames belonging to 2 classes.
Found 159 validated image filenames belonging to 2 classes.
Found 200 validated image filenames belonging to 2 classes.
```

```
class_indices = {}
for key in train_gen.class_indices.keys():
    class_indices[train_gen.class_indices[key]] = key

print(class_indices)
```

```
{0: 'fire', 1: 'non_fire'}
```

```
sns.set_style('dark')
pics = 6 #set the number of pics
fig,ax = plt.subplots(int(pics//2),2,figsize=(15,15))
plt.suptitle('Generated images in training set')
ax = ax.ravel()
for i in range((pics//2)*2):
    ax[i].imshow(train_gen[0][0][i])
    ax[i].axes.xaxis.set_visible(False)
    ax[i].axes.yaxis.set_visible(False)
```

Generated images in training set

```
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten
from tensorflow.keras.layers import Conv2D, AveragePooling2D, MaxPool2D

model = Sequential()
model.add(Conv2D(filters=16, kernel_size=(3, 3), activation='relu', input_shape=train_gen[0][0][0].shape))
model.add(AveragePooling2D())
model.add(Dropout(0.5))

model.add(Conv2D(filters=32, kernel_size=(3, 3), activation='relu'))
model.add(AveragePooling2D())
model.add(Dropout(0.5))

model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
model.add(AveragePooling2D())
model.add(Dropout(0.5))

model.add(Conv2D(filters=128, kernel_size=(3, 3), activation='relu'))
model.add(AveragePooling2D())
model.add(Dropout(0.5))

model.add(Conv2D(filters=128, kernel_size=(3, 3), activation='relu'))
model.add(AveragePooling2D())
model.add(Dropout(0.5))

model.add(Flatten())
model.add(Dense(units=256, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(units=128, activation='relu'))
model.add(Dense(1,activation = 'sigmoid'))
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 254, 254, 16)      448

 average_pooling2d (Average  (None, 127, 127, 16)      0
 Pooling2D)

 dropout (Dropout)           (None, 127, 127, 16)      0

 conv2d_1 (Conv2D)           (None, 125, 125, 32)      4640

 average_pooling2d_1 (Avera  (None, 62, 62, 32)        0
 gePooling2D)

 dropout_1 (Dropout)         (None, 62, 62, 32)        0

 conv2d_2 (Conv2D)           (None, 60, 60, 64)        18496

 average_pooling2d_2 (Avera  (None, 30, 30, 64)        0
 gePooling2D)

 dropout_2 (Dropout)         (None, 30, 30, 64)        0

 conv2d_3 (Conv2D)           (None, 28, 28, 128)       73856

 average_pooling2d_3 (Avera  (None, 14, 14, 128)       0
 gePooling2D)

 dropout_3 (Dropout)         (None, 14, 14, 128)       0

 conv2d_4 (Conv2D)           (None, 12, 12, 128)       147584

 average_pooling2d_4 (Avera  (None, 6, 6, 128)         0
 gePooling2D)

 dropout_4 (Dropout)         (None, 6, 6, 128)         0

 flatten (Flatten)           (None, 4608)              0

 dense (Dense)               (None, 256)               1179904

 dropout_5 (Dropout)         (None, 256)               0

 dense_1 (Dense)             (None, 128)               32896
 dense_2 (Dense)             (None, 1)                 129

=================================================================
Total params: 1457953 (5.56 MB)
Trainable params: 1457953 (5.56 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
#Custom implementation of focal loss function
def focal_loss(y_true, y_pred, gamma=6.0, alpha=0.25):
    # Binary crossentropy
    bce = tf.keras.backend.binary_crossentropy(y_true, y_pred)

    # Calculate the modulating factor
    p_t = tf.math.exp(-bce)
    focal_loss = alpha * (1 - p_t)**gamma * bce

    return focal_loss
```

```
from tensorflow.keras.metrics import Recall,AUC
import time

model.compile(optimizer='adam',loss=focal_loss,metrics=['accuracy',Recall(),AUC()])
# model.compile(loss='sparse_categorical_crossentropy', optimizer= 'adam',metrics=['accuracy',Recall(),AUC()])

start_time = time.time()
```

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

early_stoppping = EarlyStopping(monitor='val_loss',patience=5,restore_best_weights=True)
reduce_lr_on_plateau = ReduceLROnPlateau(monitor='val_loss',factor=0.1,patience=5)
model.fit(x=train_gen,batch_size=32,epochs=15,validation_data=val_gen,callbacks=[early_stoppping,reduce_lr_on_plateau])

end_time = time.time()

training_time = end_time - start_time

print(f"Average training time per epoch: {training_time / 15} seconds")
```

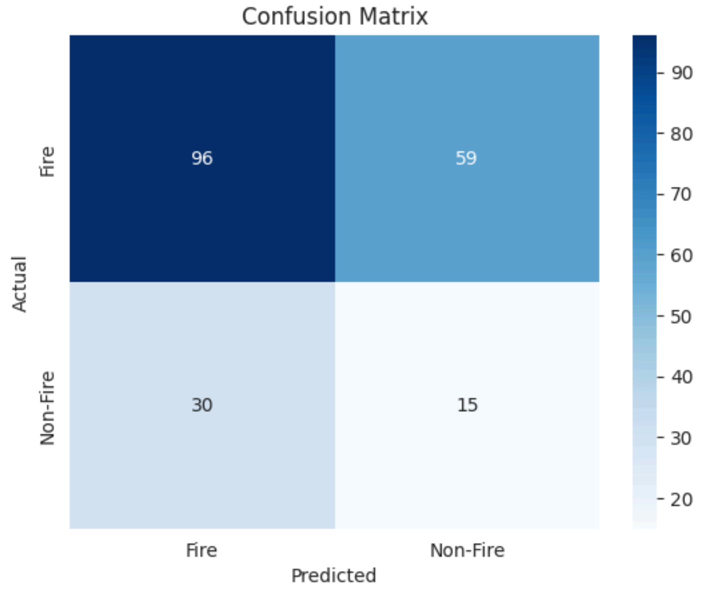
```
Epoch 1/15
20/20 [==============================] - 40s 1s/step - loss: 0.0029 - accuracy: 0.7172 - recall: 0.0723 - auc: 0.5674 - val_loss: 0.0023
Epoch 2/15
20/20 [==============================] - 25s 1s/step - loss: 0.0019 - accuracy: 0.7406 - recall: 0.0000e+00 - auc: 0.8459 - val_loss: 0.
Epoch 3/15
20/20 [==============================] - 26s 1s/step - loss: 0.0015 - accuracy: 0.8469 - recall: 0.6325 - auc: 0.9106 - val_loss: 0.0012
Epoch 4/15
20/20 [==============================] - 26s 1s/step - loss: 0.0019 - accuracy: 0.7953 - recall: 0.8193 - auc: 0.8678 - val_loss: 0.0029
Epoch 5/15
20/20 [==============================] - 26s 1s/step - loss: 0.0023 - accuracy: 0.7297 - recall: 0.0783 - auc: 0.6154 - val_loss: 0.0019
Epoch 6/15
20/20 [==============================] - 26s 1s/step - loss: 0.0016 - accuracy: 0.7922 - recall: 0.2711 - auc: 0.9002 - val_loss: 0.0018
Epoch 7/15
20/20 [==============================] - 26s 1s/step - loss: 0.0014 - accuracy: 0.8359 - recall: 0.8554 - auc: 0.9036 - val_loss: 0.0017
Epoch 8/15
20/20 [==============================] - 30s 2s/step - loss: 0.0017 - accuracy: 0.8141 - recall: 0.9277 - auc: 0.8871 - val_loss: 0.0019
Average training time per epoch: 17.126077953974406 seconds
```

```
eval_list = model.evaluate(test_gen,return_dict=True)
for metric in eval_list.keys():
    print(metric+f": {eval_list[metric]:.2f}")
```

```
7/7 [==============================] - 8s 965ms/step - loss: 0.0013 - accuracy: 0.8500 - recall: 0.9778 - auc: 0.9256
loss: 0.00
accuracy: 0.85
recall: 0.98
auc: 0.93
```

```
def plot_confusion_matrix(conf_matrix, class_names):
    fig, ax = plt.subplots()
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=class_names, yticklabels=class_names)
    plt.ylabel('Actual')
    plt.xlabel('Predicted')
    plt.title('Confusion Matrix')
    plt.show()


#To Generate A Confusion Matrix
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix, classification_report
import numpy as np

# Encode true labels to integers
label_encoder = LabelEncoder()
test_df['label'] = label_encoder.fit_transform(test_df['label'])
y_true = test_df['label'].values
y_pred_prob = model.predict(test_gen)

y_pred = (y_pred_prob > 0.5).astype(int)

conf_matrix = confusion_matrix(y_true, y_pred)

class_indices = {v: k for k, v in train_gen.class_indices.items()}

class_names = ['Fire', 'Non-Fire']

# Plot the confusion matrix
plot_confusion_matrix(conf_matrix, class_names)

print("Class Labels:")
for i in range(len(class_indices)):
    print(f"{class_indices[i]}: {i}")

# Print a classification report for more detailed metrics
class_report = classification_report(y_true, y_pred)
print("Classification Report:")
print(class_report)
```

```
7/7 [==============================] - 6s 867ms/step
```



```
Class Labels:
fire: 0
non_fire: 1
Classification Report:
              precision    recall  f1-score   support

           0       0.76      0.62      0.68       155
           1       0.20      0.33      0.25        45

    accuracy                           0.56       200
   macro avg       0.48      0.48      0.47       200
weighted avg       0.64      0.56      0.59       200
```

```
!curl https://static01.nyt.com/images/2021/02/19/world/19storm-briefing-texas-fire/19storm-briefing-texas-fire-articleLarge.jpg --output pre
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 50241  100 50241    0     0   114k      0 --:--:-- --:--:-- --:--:--  114k
```

```
from tensorflow.keras.preprocessing import image
#loading the image
img = image.load_img('predict.jpg')
img
```

```
img = image.img_to_array(img)/255
img = tf.image.resize(img,(256,256))
img = tf.expand_dims(img,axis=0)

print("Image Shape",img.shape)
```

```
Image Shape (1, 256, 256, 3)
```

```
prediction = int(tf.round(model.predict(x=img)).numpy()[0][0])
print("The predicted value is: ",prediction,"and the predicted label is:",class_indices[prediction])
```

```
1/1 [==============================] - 0s 382ms/step
The predicted value is:  0 and the predicted label is: fire
```

```
def shaper(row):
    shape = image.load_img(row['path']).size
    row['height'] = shape[1]
    row['width'] = shape[0]
    return row
df_2 = df_1.apply(shaper,axis=1)
df_2.head(5)
```

	path	label	height	width
0	/content/FireData/fire_dataset/non_fire_images...	non_fire	1733	2600
1	/content/FireData/fire_dataset/fire_images/fir...	fire	482	800
2	/content/FireData/fire_dataset/non_fire_images...	non_fire	185	560
3	/content/FireData/fire_dataset/non_fire_images...	non_fire	450	767
4	/content/FireData/fire_dataset/fire_images/fir...	fire	430	614

Next steps: View recommended plots

```
generator = ImageDataGenerator(
    rotation_range= 20,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range = 2,
    zoom_range=0.2,
    rescale = 1/255,
    validation_split=0.2,
    #horizontal_flip=True,
)

test_df = generator.flow_from_dataframe(df_2, x_col='path', y_col='label', target_size=(256, 256), class_mode='binary', subset=None)
```

```
Found 999 validated image filenames belonging to 2 classes.
```

```
eval_list = model.evaluate(test_gen,return_dict=True)
for metric in eval_list.keys():
    print(metric+f": {eval_list[metric]:.2f}")
```

```
7/7 [==============================] - 8s 1s/step - loss: 0.0013 - accuracy: 0.8500 - recall: 0.9778 - auc: 0.9383
loss: 0.00
accuracy: 0.85
recall: 0.98
auc: 0.94
```

```
# Encode true labels to integers
label_encoder = LabelEncoder()
df_2['label'] = label_encoder.fit_transform(df_2['label'])
y_true = df_2['label'].values
y_pred_prob = model.predict(test_df)

y_pred = (y_pred_prob > 0.5).astype(int)

conf_matrix = confusion_matrix(y_true, y_pred)

class_indices = {v: k for k, v in train_gen.class_indices.items()}
```