

suspicious-activity-detection-2

June 1, 2024

```
[1]: !pip install kaggle
```

```
Requirement already satisfied: kaggle in /usr/local/lib/python3.10/dist-packages (1.6.12)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.10/dist-packages (from kaggle) (1.16.0)
Requirement already satisfied: certifi>=2023.7.22 in /usr/local/lib/python3.10/dist-packages (from kaggle) (2024.2.2)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.8.2)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.31.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from kaggle) (4.66.4)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.10/dist-packages (from kaggle) (8.0.4)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.0.7)
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-packages (from kaggle) (6.1.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-packages (from bleach->kaggle) (0.5.1)
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.10/dist-packages (from python-slugify->kaggle) (1.3)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->kaggle) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->kaggle) (3.7)
```

```
[2]: from google.colab import files
      files.upload()
```

```
<IPython.core.display.HTML object>
```

```
Saving kaggle (1).json to kaggle (1).json
```

```
[2]: {'kaggle (1).json':
      b'{"username":"prabhakarsharma17","key":"6be0f49dcdd6c99de7540a382277dd40"}'}
```

```
[3]: !mkdir -p ~/.kaggle
      !cp kaggle.json ~/.kaggle/
      !chmod 600 ~/.kaggle/kaggle.json
```

```
cp: cannot stat 'kaggle.json': No such file or directory
chmod: cannot access '/root/.kaggle/kaggle.json': No such file or directory
```

```
[3]:
```

```
[4]: !kaggle datasets download -d odins0n/ucf-crime-dataset
      !unzip ucf-crime-dataset.zip
```

Streaming output truncated to the last 5000 lines.

```
inflating: Train/Vandalism/Vandalism035_x264_230.png
inflating: Train/Vandalism/Vandalism035_x264_240.png
inflating: Train/Vandalism/Vandalism035_x264_250.png
inflating: Train/Vandalism/Vandalism035_x264_260.png
inflating: Train/Vandalism/Vandalism035_x264_270.png
inflating: Train/Vandalism/Vandalism035_x264_280.png
inflating: Train/Vandalism/Vandalism035_x264_290.png
inflating: Train/Vandalism/Vandalism035_x264_30.png
inflating: Train/Vandalism/Vandalism035_x264_300.png
inflating: Train/Vandalism/Vandalism035_x264_310.png
inflating: Train/Vandalism/Vandalism035_x264_320.png
inflating: Train/Vandalism/Vandalism035_x264_330.png
inflating: Train/Vandalism/Vandalism035_x264_340.png
inflating: Train/Vandalism/Vandalism035_x264_350.png
inflating: Train/Vandalism/Vandalism035_x264_360.png
inflating: Train/Vandalism/Vandalism035_x264_370.png
inflating: Train/Vandalism/Vandalism035_x264_380.png
inflating: Train/Vandalism/Vandalism035_x264_390.png
inflating: Train/Vandalism/Vandalism035_x264_40.png
inflating: Train/Vandalism/Vandalism035_x264_400.png
inflating: Train/Vandalism/Vandalism035_x264_410.png
inflating: Train/Vandalism/Vandalism035_x264_420.png
inflating: Train/Vandalism/Vandalism035_x264_430.png
inflating: Train/Vandalism/Vandalism035_x264_440.png
inflating: Train/Vandalism/Vandalism035_x264_450.png
inflating: Train/Vandalism/Vandalism035_x264_460.png
inflating: Train/Vandalism/Vandalism035_x264_470.png
inflating: Train/Vandalism/Vandalism035_x264_480.png
inflating: Train/Vandalism/Vandalism035_x264_490.png
inflating: Train/Vandalism/Vandalism035_x264_50.png
inflating: Train/Vandalism/Vandalism035_x264_500.png
inflating: Train/Vandalism/Vandalism035_x264_510.png
inflating: Train/Vandalism/Vandalism035_x264_520.png
inflating: Train/Vandalism/Vandalism035_x264_530.png
inflating: Train/Vandalism/Vandalism035_x264_540.png
```


[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]


```

inflating: Train/Vandalism/Vandalism050_x264_710.png
inflating: Train/Vandalism/Vandalism050_x264_720.png
inflating: Train/Vandalism/Vandalism050_x264_730.png
inflating: Train/Vandalism/Vandalism050_x264_740.png
inflating: Train/Vandalism/Vandalism050_x264_750.png
inflating: Train/Vandalism/Vandalism050_x264_760.png
inflating: Train/Vandalism/Vandalism050_x264_770.png
inflating: Train/Vandalism/Vandalism050_x264_780.png
inflating: Train/Vandalism/Vandalism050_x264_790.png
inflating: Train/Vandalism/Vandalism050_x264_80.png
inflating: Train/Vandalism/Vandalism050_x264_800.png
inflating: Train/Vandalism/Vandalism050_x264_810.png
inflating: Train/Vandalism/Vandalism050_x264_820.png
inflating: Train/Vandalism/Vandalism050_x264_830.png
inflating: Train/Vandalism/Vandalism050_x264_840.png
inflating: Train/Vandalism/Vandalism050_x264_850.png
inflating: Train/Vandalism/Vandalism050_x264_860.png
inflating: Train/Vandalism/Vandalism050_x264_870.png
inflating: Train/Vandalism/Vandalism050_x264_880.png
inflating: Train/Vandalism/Vandalism050_x264_890.png
inflating: Train/Vandalism/Vandalism050_x264_90.png

```

```

[5]: !pip install image-classifiers
    !pip install vit-keras

```

Collecting image-classifiers

Downloading image_classifiers-1.0.0-py3-none-any.whl (19 kB)

Collecting keras-applications<=1.0.8,>=1.0.7 (from image-classifiers)

Downloading Keras_Applications-1.0.8-py3-none-any.whl (50 kB)

50.7/50.7 kB

3.3 MB/s eta 0:00:00

Requirement already satisfied: numpy>=1.9.1 in

/usr/local/lib/python3.10/dist-packages (from keras-applications<=1.0.8,>=1.0.7->image-classifiers) (1.25.2)

Requirement already satisfied: h5py in /usr/local/lib/python3.10/dist-packages (from keras-applications<=1.0.8,>=1.0.7->image-classifiers) (3.9.0)

Installing collected packages: keras-applications, image-classifiers

Successfully installed image-classifiers-1.0.0 keras-applications-1.0.8

Collecting vit-keras

Downloading vit_keras-0.1.2-py3-none-any.whl (24 kB)

Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from vit-keras) (1.11.4)

Collecting validators (from vit-keras)

Downloading validators-0.28.1-py3-none-any.whl (39 kB)

Requirement already satisfied: numpy<1.28.0,>=1.21.6 in /usr/local/lib/python3.10/dist-packages (from scipy->vit-keras) (1.25.2)

Installing collected packages: validators, vit-keras

Successfully installed validators-0.28.1 vit-keras-0.1.2

```
[6]: !pip install tensorflow_addons
```

```
Collecting tensorflow_addons
```

```
  Downloading tensorflow_addons-0.23.0-cp310-cp310-manylinux_2_17_x86_64.manylin  
ux2014_x86_64.whl (611 kB)
```

```
611.8/611.8
```

```
kB 7.3 MB/s eta 0:00:00
```

```
Requirement already satisfied: packaging in
```

```
/usr/local/lib/python3.10/dist-packages (from tensorflow_addons) (24.0)
```

```
Collecting typeguard<3.0.0,>=2.7 (from tensorflow_addons)
```

```
  Downloading typeguard-2.13.3-py3-none-any.whl (17 kB)
```

```
Installing collected packages: typeguard, tensorflow_addons
```

```
Successfully installed tensorflow_addons-0.23.0 typeguard-2.13.3
```

0.1 Introduction

We will be working with the UCF-Crime dataset, a large-scale surveillance video dataset containing 128 hours of real-world surveillance footage. The dataset consists of 1,900 long and uncut videos that recorded a variety of anomalous activities, including Abuse, Arrest, Arson, Assault, Traffic Accident, Theft, Explosion, Fight, Robbery, Shooting, Theft, Shoplifting, and Vandalism. These anomalies were chosen because of their significant impact on public safety. This dataset can be used for two main tasks:

- Task 1: General Anomaly Detection As part of this task, we will consider all anomalies in one group and all normal activities in another group. The goal is to develop models capable of detecting any anomaly in surveillance video recordings.
- Task 2: Anomaly Recognition For this task, we aim to recognize each of the specific anomalous activities individually. We will train the models to identify and classify each type of anomaly.

As a metric for the quality of solving the problem, we will use the area under the AUC curve for each class of anomalies.

```
[7]: import os
import glob

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import animation, rc
rc('animation', html='jshtml')
import seaborn as sns
import plotly.express as px
import cv2
from tqdm import tqdm
from vit_keras import vit, utils
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.preprocessing import LabelBinarizer
```

```

from sklearn.metrics import roc_curve, auc, roc_auc_score, accuracy_score
from classification_models.keras import Classifiers

from IPython.display import clear_output
import warnings
warnings.filterwarnings('ignore')

```

```

/usr/local/lib/python3.10/dist-
packages/tensorflow-addons/utils/tfa_eol_msg.py:23: UserWarning:

```

TensorFlow Addons (TFA) has ended development and introduction of new features. TFA has entered a minimal maintenance and release mode until a planned end of life in May 2024.

Please modify downstream libraries to take dependencies from other repositories in our TensorFlow community (e.g. Keras, Keras-CV, and Keras-NLP).

For more information see: <https://github.com/tensorflow/addons/issues/2807>

```
warnings.warn(
```

Experimental constants

0.2 Data Loading and Preprocessing

In this section, we'll look at the data loading and preprocessing steps. This includes reading the video data, extracting the frames, and preparing the dataset for training the models.

```

[8]: data_hyperparams = {
      'img_size': (64, 64),
      'batch_size': 256,
      '#preprocessing_function': tf.keras.applications.densenet.preprocess_input,
    }

```

The dataset presented to us has an imbalance of classes – there are many more examples labeled “Normal” than other classes combined. This property can degrade the quality of neural network training.

To solve this problem, we can use the augmentation of examples that are not of the “Normal” class. Augmentation is the process of enhancing or expanding image data by applying a variety of techniques. These techniques can include resizing, rotating, changing contrast, and other transformations. Data augmentation is often used in machine learning to improve the performance of a model on new data by providing it with a variety of input image options.

```

[9]: # Randomly convert image to black and white color scheme
def random_grayscale(x):
    if np.random.randint(0, 2) == 1:
        return tf.image.rgb_to_grayscale(x)
    return x

```

```

train_datagen = ImageDataGenerator(
    rescale=1./255, # Scaling pixel values to the range [0, 1]
    rotation_range=20, # Rotation angle in degrees
    width_shift_range=0.2, # Width Shift
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest',
    preprocessing_function=random_grayscale
)

test_datagen = ImageDataGenerator(
    rescale = 1./255
)

```

```

[10]: # Define the directories
TRAIN_DIR = './Train/'
TEST_DIR = './Test/'
NUM_CLASSES = 14
SEED = 42
np.random.seed(SEED)
CLASS_LABELS = ['Abuse', 'Arrest', 'Arson', 'Assault',
                'Burglary', 'Explosion', 'Fighting',
                "Normal", 'RoadAccidents', 'Robbery',
                'Shooting', 'Shoplifting', 'Stealing',
                'Vandalism']

```

```

[11]: # Create a list of file paths to create a training dataset
filenames = []
labels = []

for root, subdirs, files in tqdm(os.walk(TRAIN_DIR)):
    for subdir in subdirs:
        for file in os.listdir(TRAIN_DIR+subdir):
            if file.lower().endswith('.png'):
                # Take only half of the frames from the normal class
                if subdir == 'NormalVideos':
                    if np.random.randint(0, 2) == 1:
                        filenames.append(TRAIN_DIR+subdir+'/'+file)
                        labels.append(subdir)
                else:
                    filenames.append(TRAIN_DIR+subdir+'/'+file)
                    labels.append(subdir)

df_train = pd.DataFrame({
    'filename': filenames,

```

```

        'class': labels
    })
    df_train.head()

```

15it [00:11, 1.27it/s]

```

[11]:
      filename      class
0  ./Train/Fighting/Fighting026_x264_730.png  Fighting
1  ./Train/Fighting/Fighting019_x264_3150.png  Fighting
2  ./Train/Fighting/Fighting041_x264_12470.png  Fighting
3  ./Train/Fighting/Fighting029_x264_1140.png  Fighting
4  ./Train/Fighting/Fighting041_x264_24520.png  Fighting

```

```

[12]: # Create a list of file paths to create a test dataset
filenames = []
labels = []

for root, subdirs, files in tqdm(os.walk(TEST_DIR)):
    for subdir in subdirs:
        for file in os.listdir(TEST_DIR+subdir):
            if file.lower().endswith('.png'):
                # Take only half of the frames from the normal class
                if subdir == 'NormalVideos':
                    if np.random.randint(0, 2) == 1:
                        filenames.append(TEST_DIR+subdir+'/'+file)
                        labels.append(subdir)
                else:
                    filenames.append(TEST_DIR+subdir+'/'+file)
                    labels.append(subdir)

df_test = pd.DataFrame({
    'filename': filenames,
    'class': labels
})
df_test.head()

```

15it [00:00, 17.05it/s]

```

[12]:
      filename      class
0  ./Test/Fighting/Fighting018_x264_410.png  Fighting
1  ./Test/Fighting/Fighting003_x264_890.png  Fighting
2  ./Test/Fighting/Fighting047_x264_2150.png  Fighting
3  ./Test/Fighting/Fighting042_x264_970.png  Fighting
4  ./Test/Fighting/Fighting047_x264_2480.png  Fighting

```

```

[13]: train_generator = train_datagen.flow_from_dataframe(
        df_train, directory=None,

```

```

x_col='filename', y_col='class',
weight_col=None,
color_mode='rgb',
class_mode='categorical',
target_size = data_hyperparams['img_size'],
batch_size = data_hyperparams['batch_size'],
shuffle=True, seed=SEED,
save_prefix='', save_format='png',
subset=None, interpolation='nearest',
validate_filenames=True
)
train_generator

```

Found 792295 validated image filenames belonging to 14 classes.

[13]: <keras.src.preprocessing.image.DataFrameIterator at 0x7f261fa9af20>

```

[14]: test_generator = test_datagen.flow_from_dataframe(
    df_test, directory=None,
    x_col='filename', y_col='class',
    weight_col=None,
    color_mode='rgb',
    class_mode='categorical',
    target_size = data_hyperparams['img_size'],
    batch_size = data_hyperparams['batch_size'],
    shuffle=True, seed=SEED,
    save_prefix='', save_format='png',
    subset=None, interpolation='nearest',
    validate_filenames=True
)
test_generator

```

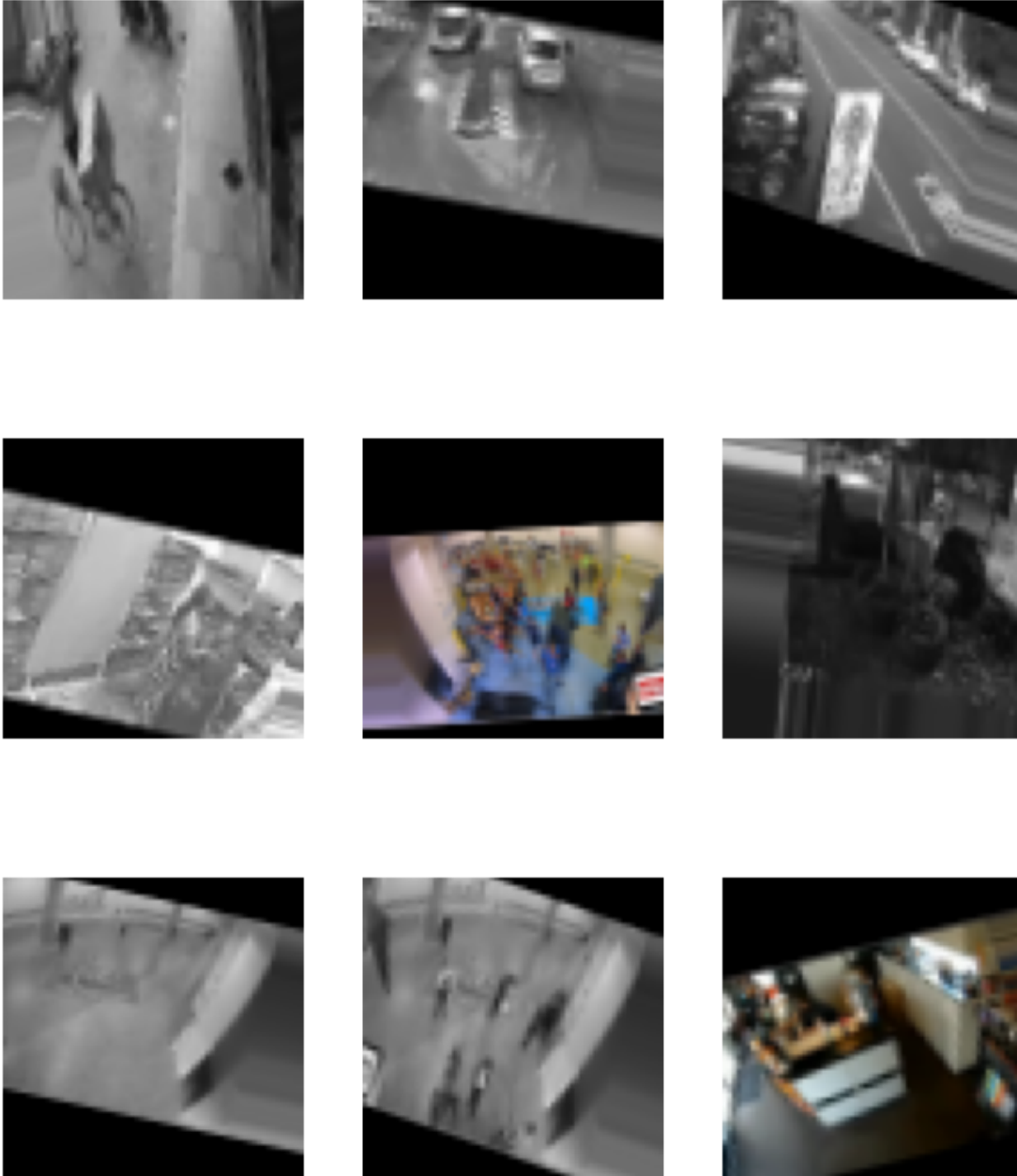
Found 78787 validated image filenames belonging to 14 classes.

[14]: <keras.src.preprocessing.image.DataFrameIterator at 0x7f26b9336aa0>

```

[15]: # Image generation and visualization after augmentation
for i in range(130,133):
    img, label = train_generator[i]
    plt.figure(figsize=(10, 10))
    for j in range(3):
        plt.subplot(1, 4, j+1)
        plt.imshow(img[j])
        plt.axis('off')
    plt.show()

```



0.3 EDA (Exploratory Data Analysis)

Exploratory data analysis (EDA) is an important step in understanding a dataset and its characteristics. In this section, we will be conducting EDA to gain insight into the UCF-Crime dataset. We examine statistics, distributions, and the presence of any patterns in the data

0.3.1 Categorizing examples

It is important to consider the distribution of samples by anomalous activities and normal activities. We visualize class distribution to understand potential class imbalances.

```
[16]: fig = px.bar(x = CLASS_LABELS,
                  y = [list(train_generator.classes).count(i)
                       for i in np.unique(train_generator.classes)] ,
                  color = np.unique(train_generator.classes) ,
                  color_continuous_scale="Viridis")
fig.update_xaxes(title="Classes")
fig.update_yaxes(title = "Number of Images")
fig.update_layout(showlegend = True,
                  title = {
                      'text': 'Train Data Distribution ',
                      'y': 0.95,
                      'x': 0.5,
                      'xanchor': 'center',
                      'yanchor': 'top'})
fig.show()
```

```
[17]: fig = px.bar(x = CLASS_LABELS,
                  y = [list(test_generator.classes).count(i)
                       for i in np.unique(test_generator.classes)] ,
                  color = np.unique(train_generator.classes) ,
                  color_continuous_scale="Sunset")
fig.update_xaxes(title="Classes")
fig.update_yaxes(title = "Number of Images")
fig.update_layout(showlegend = True,
                  title = {
                      'text': 'Test Data Distribution ',
                      'y': 0.95,
                      'x': 0.5,
                      'xanchor': 'center',
                      'yanchor': 'top'})
fig.show()
```

0.3.2 Distribution of Normal Class and Other Class Examples

```
[18]: n_total_train = len(train_generator.classes)
n_normal_train = list(train_generator.classes).count(CLASS_LABELS.
    ↪index('Normal'))

fig = px.bar(x = ['Normal', 'Not Normal'],
            y = [n_normal_train, n_total_train-n_normal_train],
            color = ['Normal', 'Not Normal'],
            color_continuous_scale="Viridis")
fig.update_xaxes(title="Classes")
```

```

fig.update_yaxes(title = "Number of Images")
fig.update_layout(showlegend = False,
    title = {
        'text': 'Train Data Distribution ',
        'y': 0.95,
        'x': 0.5,
        'xanchor': 'center',
        'yanchor': 'top'})
fig.show()

```

```

[19]: n_total_test = len(test_generator.classes)
n_normal_test = list(test_generator.classes).count(CLASS_LABELS.index('Normal'))

fig = px.bar(x = ['Normal', 'Not Normal'],
    y = [n_normal_test, n_total_test-n_normal_test],
    color = ['Normal', 'Not Normal'],
    color_continuous_scale="Viridis")
fig.update_xaxes(title="Classes")
fig.update_yaxes(title = "Number of Images")
fig.update_layout(showlegend = False,
    title = {
        'text': 'Test Data Distribution ',
        'y': 0.95,
        'x': 0.5,
        'xanchor': 'center',
        'yanchor': 'top'})
fig.show()

```

0.3.3 Visualization of an individual video

To better understand the content of the dataset, we will select and visualize frames from an individual video. This will help us understand the complexities and variations in the dataset.

```

[20]: def load_png_line(path):
    t_paths = sorted(
        glob.glob(os.path.join(path, "*")),
        key=lambda x: x,
    )
    images = []
    for i,filename in enumerate(np.array(t_paths)):
        if i%2==0:
            data = cv2.imread(filename)
            if data.max() == 0:
                continue
            images.append(data)
    return images

```

```

def create_animation(ims):
    fig=plt.figure(figsize=(6,6))
    plt.axis('off')
    im=plt.imshow(cv2.cvtColor(ims[0],cv2.COLOR_BGR2RGB))

    def animate_func(i):
        im.set_array(cv2.cvtColor(ims[i],cv2.COLOR_BGR2RGB))
        return [im]

    return animation.FuncAnimation(fig, animate_func, frames=len(ims),
    ↪interval=1000//10)

```

may not work because of directory issue

```

[21]: category = 'Arson' # category of crime to visualize
      images = load_png_line(f"./Test/{category}")
      create_animation(images)

```

WARNING:matplotlib.animation:Animation size has reached 20981157 bytes, exceeding the limit of 20971520.0. If you're sure you want a larger animation embedded, set the animation.embed_limit rc parameter to a larger value (in MB). This and further frames will be dropped.

```

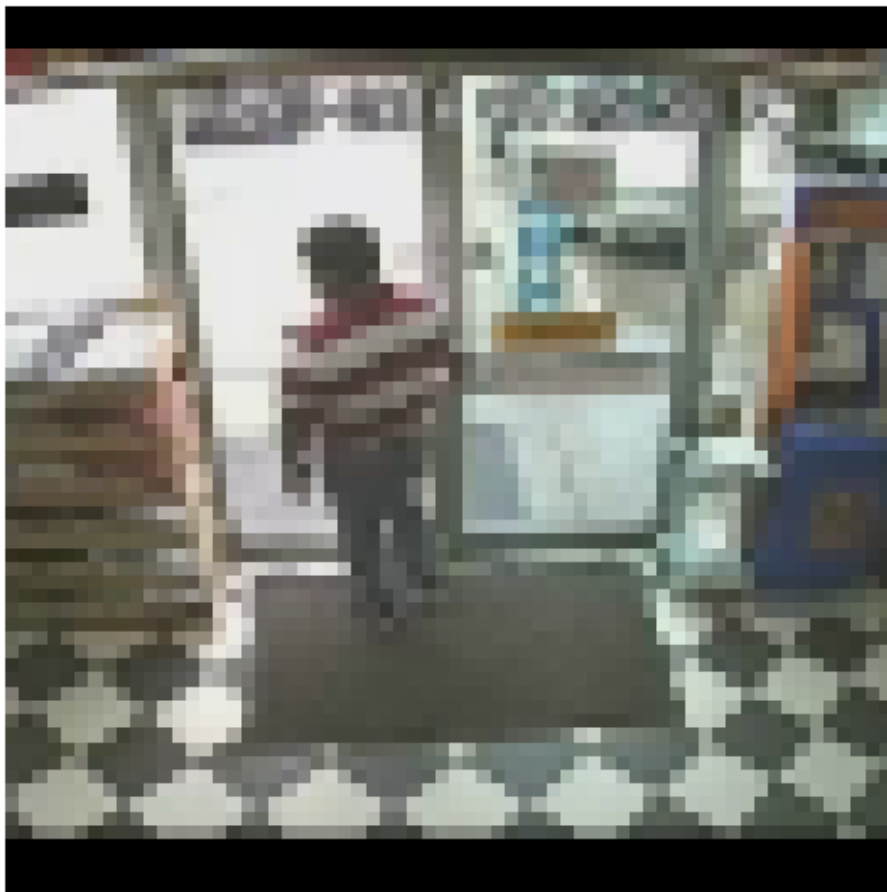
[21]: <matplotlib.animation.FuncAnimation at 0x7f2619e7e830>

```



```
[22]: category = 'Robbery' # category of crime to visualize  
      images = load_png_line(f"./Test/{category}")  
      create_animation(images)
```

```
[22]: <matplotlib.animation.FuncAnimation at 0x7f2619d5a020>
```



0.4 Modeling

In the simulation section, we will develop and evaluate various models to solve the problem. The general architecture of each model is as follows: feature extraction + classifier.

```
[23]: mlp_hyperparams = {
    'pooling': 'average',
    'sizes': (64, 128, 256),
    'activations': ('relu', 'relu', 'relu'),
    'dropouts': (0.3, 0.3, 0.3),
    'num_classes': NUM_CLASSES
}

def mlp_classifier(inputs, hyperparams):
    if hyperparams['pooling'] == 'average':
        x = tf.keras.layers.GlobalAveragePooling2D()(inputs)
    else:
        x = inputs
```

```

# hidden layers
for i in range(len(hyperparams['sizes'])):
    x = tf.keras.layers.Dense(hyperparams['sizes'][i],
                               activation=hyperparams['activations'][i])(x)
    x = tf.keras.layers.Dropout(hyperparams['dropouts'][i])(x)

# output layer
x = tf.keras.layers.Dense(hyperparams['num_classes'],
                           activation="softmax",
                           name="classification")(x)

return x

```

0.4.1 Helper functions for evaluating the quality of the model

```

[24]: def evaluate(model, data_generator, model_name=''):
    y_test = []
    preds = []
    for idx, batch in tqdm(enumerate(data_generator)):
        preds.append(model.predict(batch[0], verbose=False))
        y_test.append(batch[1].argmax(1))
        if idx == len(data_generator):
            break

    y_test = np.concatenate(y_test)
    preds = np.concatenate(preds)

    fig, c_ax = plt.subplots(1,1, figsize = (15,8))

    def multiclass_roc_auc_score(y_test, y_pred, average="macro"):
        lb = LabelBinarizer()
        lb.fit(y_test)
        y_test = lb.transform(y_test)
        for (idx, c_label) in enumerate(CLASS_LABELS):
            fpr, tpr, thresholds = roc_curve(y_test[:,idx].astype(int), y_pred[:,
↪,idx])
            c_ax.plot(fpr, tpr, lw=2, label = '%s (AUC:%0.2f)' % (c_label,
↪auc(fpr, tpr)))
            c_ax.plot(fpr, fpr, 'black', linestyle='dashed', lw=4, label = 'Random
↪Guessing')
        return roc_auc_score(y_test, y_pred, average=average)

    roc_auc_score_value = multiclass_roc_auc_score(y_test , preds , average =
↪"micro")
    print('ROC AUC score:', roc_auc_score_value)
    plt.title(model_name)

```

```

plt.grid(True)
plt.xlabel('FALSE POSITIVE RATE', fontsize=18)
plt.ylabel('TRUE POSITIVE RATE', fontsize=16)
plt.legend(fontsize = 11.5)
plt.show()
return y_test, preds, roc_auc_score_value

```

0.5 ResNet 50

Let's examine the quality of the pre-imagenet architecture model [ResNet50](#) as a feature extractor.

```

[25]: # def resnet50_extractor(inputs, hyperparams):
#     feature_extractor = tf.keras.applications.resnet50.ResNet50(
#         include_top=False,
#         weights='imagenet',
#         input_shape=hyperparams['input_shape'],
#         pooling=hyperparams['pooling'],
#         classes=1000
#     )(inputs)
#     return feature_extractor

[26]: # resnet50_hyperparams = {
#     'input_shape': (64, 64, 3),
#     'weights': "imagenet",
#     'classes': 1000,
#     'pooling': None, # 'avg'      'max'
#     'trainable': False,
#     'lr': 0.001
# }

# inputs = tf.keras.layers.Input(shape=resnet50_hyperparams['input_shape'])
# feature_extractor = resnet50_extractor(inputs, resnet50_hyperparams)
# classification_output = mlp_classifier(feature_extractor, mlp_hyperparams)
# model = tf.keras.Model(inputs=inputs, outputs = classification_output)
# model.layers[1].trainable = resnet50_hyperparams['trainable']
# model.compile(optimizer=tf.keras.optimizers.SGD(resnet50_hyperparams['lr']),
#               loss='categorical_crossentropy',
#               metrics = [tf.keras.metrics.AUC()])

# clear_output()
# model.summary()

[27]: # history = model.fit(x = train_generator,
#                       validation_data=test_generator,
#                       epochs = 1)

[28]: # y_test, preds, roc_auc_score = evaluate(model, test_generator, 'ResNet50')

```

```
[29]: # threshold = 0.3

# accuracy_results = {}
# for i in range(len(CLASS_LABELS)):
#     y_pred_i = (preds.T[i] > threshold)
#     y_test_i = (y_test == i)
#     accuracy_results[CLASS_LABELS[i]] = accuracy_score(y_pred_i, y_test_i)

# accuracy_results
```

ResNet 34

```
[30]: # def resnet34_extractor(inputs, hyperparams):
#     ResNet34, preprocess_input = Classifiers.get('resnet34')
#     feature_extractor = ResNet34(
#         hyperparams['input_shape'],
#         weights=hyperparams['weights'],
#         include_top=False
#     )(inputs)
#     return feature_extractor
```

```
[31]: # resnet34_hyperparams = {
#     'input_shape': (64, 64, 3),
#     'weights': "imagenet",
#     'classes': 1000,
#     'trainable': False,
#     'lr': 0.001
# }

# inputs = tf.keras.layers.Input(shape=resnet34_hyperparams['input_shape'])
# feature_extractor = resnet34_extractor(inputs, resnet34_hyperparams)
# mlp_hyperparams['pooling'] = 'average'
# classification_output = mlp_classifier(feature_extractor, mlp_hyperparams)
# model = tf.keras.Model(inputs=inputs, outputs = classification_output)
# model.layers[1].trainable = resnet34_hyperparams['trainable']
# model.compile(optimizer=tf.keras.optimizers.SGD(resnet34_hyperparams['lr']),
#               loss='categorical_crossentropy',
#               metrics = [tf.keras.metrics.AUC()])

# clear_output()
# model.summary()
```

```
[32]: # history = model.fit(x = train_generator,
#                       validation_data=test_generator,
#                       epochs = 1)
```

Validation


```
[33]: # y_test, preds, roc_auc_score = evaluate(model, test_generator, 'ResNet34')
```

```
[34]: # threshold = 0.3

# accuracy_results = {}
# for i in range(len(CLASS_LABELS)):
#     y_pred_i = (preds.T[i] > threshold)
#     y_test_i = (y_test == i)
#     accuracy_results[CLASS_LABELS[i]] = accuracy_score(y_pred_i, y_test_i)

# accuracy_results
```

```
[35]: # threshold = 0.5

# accuracy_results = {}
# for i in range(len(CLASS_LABELS)):
#     y_pred_i = (preds.T[i] > threshold)
#     y_test_i = (y_test == i)
#     accuracy_results[CLASS_LABELS[i]] = accuracy_score(y_pred_i, y_test_i)

# accuracy_results
```

0.5.1 MobileNet v2

Let's examine the quality of the [MobileNet v2](#) architecture model pretrained on imagenet as a feature extractor.

```
[36]: # def mobilenetv2_extractor(inputs, hyperparams):
#     feature_extractor = tf.keras.applications.MobileNetV2(
#         input_shape=hyperparams['input_shape'],
#         include_top=False,
#         weights=hyperparams["weights"],
#         alpha=hyperparams['alpha'])(inputs)
#     return feature_extractor
```

```
[37]: # mobilenetv2_hyperparams = {
#     'input_shape': (64, 64, 3),
#     'weights': "imagenet",
#     'classes': 1000,
#     'alpha': 1.0,
#     'trainable': False,
#     'lr': 0.001
# }

# inputs = tf.keras.layers.Input(shape=mobilenetv2_hyperparams['input_shape'])
# feature_extractor = mobilenetv2_extractor(inputs, mobilenetv2_hyperparams)
```

```

# classification_output = mlp_classifier(feature_extractor, mlp_hyperparams)
# model = tf.keras.Model(inputs=inputs, outputs = classification_output)
# model.layers[1].trainable = mobilenetv2_hyperparams['trainable']
# model.compile(optimizer=tf.keras.optimizers.
    ↪SGD(mobilenetv2_hyperparams['lr']),
#             loss='categorical_crossentropy',
#             metrics = [tf.keras.metrics.AUC()])

# clear_output()
# model.summary()

```

```

[38]: # history = model.fit(x = train_generator,
#                         validation_data=test_generator,
#                         epochs = 1)

```

Validation

```

[39]: # y_test, preds, roc_auc_score = evaluate(model, test_generator, 'MobileNetV2')

```

0.6 VGG-19

Let's examine the quality of the architecture model **VGG-19** pretrained on imagenet as a feature extractor.

```

[40]: # def vgg19_extractor(inputs, hyperparams):
#     feature_extractor = tf.keras.applications.vgg19.VGG19(
#         input_shape=hyperparams['input_shape'],
#         include_top=False,
#         weights=hyperparams["weights"],
#         pooling=hyperparams["pooling"],
#         classifier_activation='softmax'
#     )(inputs)

#     return feature_extractor

# mlp_hyperparams = {
#     'pooling': None,
#     'sizes': (256, 102, 512),
#     'activations': ('relu', 'relu', 'relu'),
#     'dropouts': (0.3, 0.5, 0.4),
#     'num_classes': 14
# }

# vgg19_hyperparams = {
#     'input_shape': (64, 64, 3),
#     'weights': "imagenet",
#     'pooling': 'avg',

```

```

#     'trainable': True,
#     'lr': 0.001
# }

# inputs = tf.keras.layers.Input(shape=vgg19_hyperparams['input_shape'])
# feature_extractor = vgg19_extractor(inputs, vgg19_hyperparams)
# classification_output = mlp_classifier(feature_extractor, mlp_hyperparams)
# model = tf.keras.Model(inputs=inputs, outputs = classification_output)
# model.layers[1].trainable = vgg19_hyperparams['trainable']

# model.compile(optimizer=tf.keras.optimizers.SGD(vgg19_hyperparams['lr']),
#               loss='categorical_crossentropy',
#               metrics = [tf.keras.metrics.AUC()])

# clear_output()
# model.summary()

# history = model.fit(x = train_generator,
#                     validation_data = test_generator,
#                     epochs = 1)

```

```

[41]: # import os
# import tensorflow as tf
# from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, ␣
# ↪TensorBoard
# from google.colab import drive

# # Mount Google Drive
# drive.mount('/content/drive')

# # Path to the folder where you want to save all the things
# folder_path = '/content/drive/My Drive/suspicious_detection_all_things/'

# # Create the folder if it doesn't exist
# if not os.path.exists(folder_path):
#     os.makedirs(folder_path)

# # Callbacks
# callbacks = [
#     EarlyStopping(monitor='val_loss', patience=3),
#     ModelCheckpoint(folder_path + 'best_model.h5', monitor='val_loss', ␣
# ↪save_best_only=True),
#     TensorBoard(log_dir=folder_path + 'logs/')
# ]

# # Compile the model

```

```

# model.compile(optimizer=tf.keras.optimizers.SGD(vgg19_hyperparams['lr']),
#               loss='categorical_crossentropy',
#               metrics=[tf.keras.metrics.AUC()])

# # Train the model
# history = model.fit(x=train_generator,
#                     validation_data=test_generator,
#                     epochs=1,
#                     callbacks=callbacks)

# # Save the model
# model.save(folder_path + 'vgg19_ucf_crime_model.h5')

# # Load the model from the folder (if needed)
# # loaded_model = tf.keras.models.load_model(folder_path +
# ↪ 'vgg19_ucf_crime_model.h5')

# # Display the loaded model summary
# # loaded_model.summary()

```

0.7 Validation

[41]:

```

[42]: import os
import tensorflow as tf
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint,
↪ TensorBoard
from google.colab import drive
from IPython.display import clear_output

# Mount Google Drive
drive.mount('/content/drive')

# Path to the folder where you want to save all the things
folder_path = '/content/drive/My Drive/suspicious_detection_all_things/'

# Create the folder if it doesn't exist
if not os.path.exists(folder_path):
    os.makedirs(folder_path)

# VGG19 feature extractor function
def vgg19_extractor(inputs, hyperparams):
    feature_extractor = tf.keras.applications.vgg19.VGG19(
        input_shape=hyperparams['input_shape'],
        include_top=False,
        weights=hyperparams["weights"],

```

```

        pooling=hyperparams["pooling"]
    )(inputs)
    return feature_extractor

# MLP classifier function
def mlp_classifier(inputs, hyperparams):
    x = inputs
    for size, activation, dropout in zip(hyperparams['sizes'],
    ↪hyperparams['activations'], hyperparams['dropouts']):
        x = tf.keras.layers.Dense(size, activation=activation)(x)
        if dropout is not None:
            x = tf.keras.layers.Dropout(dropout)(x)
        outputs = tf.keras.layers.Dense(hyperparams['num_classes'],
    ↪activation='softmax')(x)
    return outputs

# Hyperparameters
mlp_hyperparams = {
    'pooling': None,
    'sizes': (256, 102, 512),
    'activations': ('relu', 'relu', 'relu'),
    'dropouts': (0.3, 0.5, 0.4),
    'num_classes': 14
}

vgg19_hyperparams = {
    'input_shape': (64, 64, 3),
    'weights': "imagenet",
    'pooling': 'avg',
    'trainable': True,
    'lr': 0.001
}

# Model construction
inputs = tf.keras.layers.Input(shape=vgg19_hyperparams['input_shape'])
feature_extractor = vgg19_extractor(inputs, vgg19_hyperparams)
classification_output = mlp_classifier(feature_extractor, mlp_hyperparams)
model = tf.keras.Model(inputs=inputs, outputs=classification_output)
model.layers[1].trainable = vgg19_hyperparams['trainable']

# Callbacks
callbacks = [
    EarlyStopping(monitor='val_loss', patience=3),
    ModelCheckpoint(folder_path + 'best_model.h5', monitor='val_loss',
    ↪save_best_only=True),
    TensorBoard(log_dir=folder_path + 'logs/')
]

```

```

# Compile the model
model.compile(optimizer=tf.keras.optimizers.SGD(vgg19_hyperparams['lr']),
              loss='categorical_crossentropy',
              metrics=[tf.keras.metrics.AUC()])

# Display the model summary
clear_output()
model.summary()

# Train the model
history = model.fit(x=train_generator,
                   validation_data=test_generator,
                   epochs=1,
                   callbacks=callbacks)

# Save the model
model.save(folder_path + 'vgg19_ucf_crime_model.h5')

# Load the model from the folder (if needed)
# loaded_model = tf.keras.models.load_model(folder_path +
# ↪ 'vgg19_ucf_crime_model.h5')

# Display the loaded model summary
# loaded_model.summary()

```

Model: "model"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 64, 64, 3)]	0
vgg19 (Functional)	(None, 512)	20024384
dense (Dense)	(None, 256)	131328
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 102)	26214
dropout_1 (Dropout)	(None, 102)	0
dense_2 (Dense)	(None, 512)	52736
dropout_2 (Dropout)	(None, 512)	0
dense_3 (Dense)	(None, 14)	7182

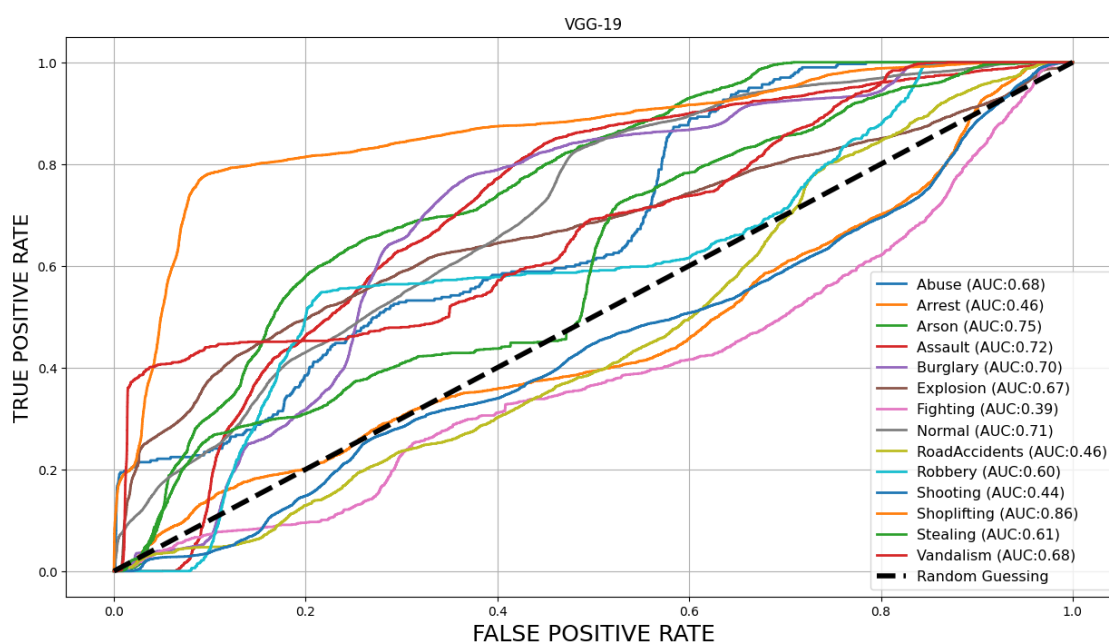
```
=====
Total params: 20241844 (77.22 MB)
Trainable params: 20241844 (77.22 MB)
Non-trainable params: 0 (0.00 Byte)
```

```
-----
3095/3095 [=====] - 4223s 1s/step - loss: 1.4983 - auc:
0.8824 - val_loss: 2.2501 - val_auc: 0.7510
```

```
[43]: y_test, preds, roc_auc_score = evaluate(model, test_generator, 'VGG-19')
```

```
308it [02:01, 2.53it/s]
```

```
ROC AUC score: 0.7514537129770813
```



```
[43]:
```

```
[43]:
```