Ethan Trott

Professor Roy Turner

COS 470

Semester Project

Home Security Using Machine Learning

Abstract

This project explored the potential for the creation of a home security system that primarily utilizes a machine learning model to recognize significant dangers to a household, such as house fires. This was done using a combination of internet-provided datasets and real word examples. The security system itself was comprised of multiple camera modules, streaming their contents to a central server equipped with a CUDA-enabled GPU to perform the analysis.

Introduction

This is a physical implementation project in which a security system has been created that is able to monitor and protect homes from physical dangers. The uses a CNN to detect these dangers. The system is composed of multiple monitoring units, to monitor both indoors and outdoors. The units send live video and audio back to the central server machine, which analyzes the feeds to detect the dangers.

The purpose of this project is to be able to monitor a house, but also have the monitoring system able to detect any dangers autonomously. The benefits of using this system rather than enlisting a third-party company are two-fold. Primarily, this solution will be much less costly than purchasing or subscribing to a third-party product, provided you already have the materials necessary. Secondly, there will be fewer security concerns, as this system will only talk to itself; no internet connectivity required. The only networking involved will be on a local network, so concerns of outsiders hijacking the cameras is minimal. The source code will be released publicly, along with the trained model, so that others, who are looking for a cost-effective and secure home security option, can freely download and use the system for no cost.

Background

AI-powered home security is still a new and emerging technology. Over the past few years there have been quite a few attempts at creating a home security system that required no external monitoring, but most of these just detect basic things like motion and loud noises. While these basic variables will alert you to the security events that users are concerned with, they also alert to many false alarms, such as a squirrel moving in your yard, or the sound of the vacuum cleaner running.

Nest is an example of a product-line that uses basic variables like motion and sound to detect dangers. However, they do offer "human detection" on some of their newer products, but this is quite limited and only offers the ability to detect if a face is present in the frame.

ADT is another highly recognized name in the space of home security. They have been around for years and are known for their 24/7 monitoring capabilities. Under the hood, this service uses the same basic technologies mentioned earlier: motion detection and audio volume monitoring. Where ADT takes it a step beyond is their support team, they have employees that are constantly monitoring when one of these alarms go off to confirm if there is a security event taking place that would result in the need to alert somebody. Still, there is little, if any, AI being utilized here. If there is any, they do not advertise it publicly.

Of course, this approach is much different than both of these, taking the home security space into an area that it has not been fully exposed to already.

Model Generation

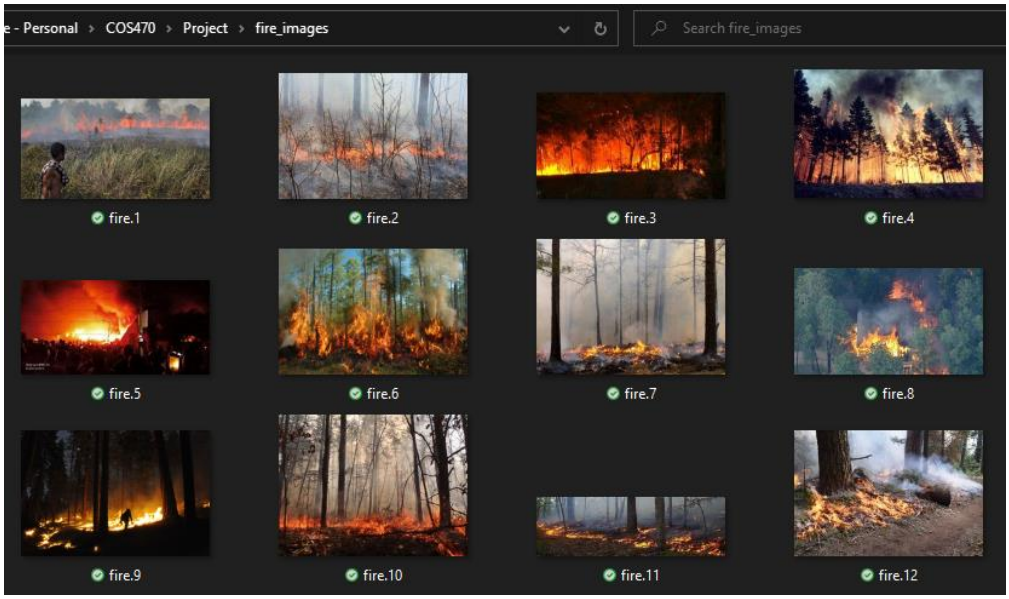
Python, TensorFlow, and Keras were used for the machine learning in this project.

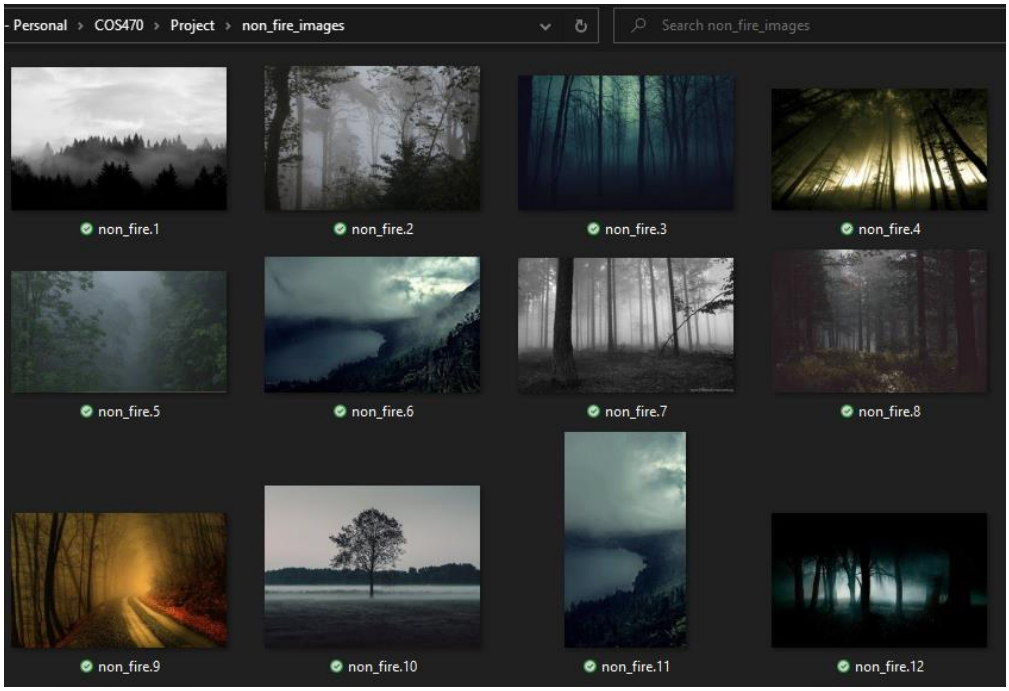
For this first iteration of the project, the model was specifically target toward fire recognition. The specifics of the model structure used for the system can be seen below.

```
model = tf.keras.Sequential([
    tf.keras.layers.SeparableConv2D(16, (7, 7), padding="same", input_shape=(128,128,3)),
    tf.keras.layers.Activation("relu"),
    tf.keras.layers.BatchNormalization(axis=-1),
    tf.keras.layers.SeparableConv2D(64, (3, 3), padding="same"),
    tf.keras.layers.Activation("relu"),
    tf.keras.layers.BatchNormalization(axis=-1),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128),
    tf.keras.layers.Activation("relu"),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.5),

    tf.keras.layers.Dense(2),
    tf.keras.layers.Activation("softmax")
])

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

As you can see, quite a few layers are in use here. These were selected and combined specifically to maximize detection of larger color blobs, which are very common in fires.

As the model is classifying into two categories, “fires” and “non-fires”, we require two datasets to train on, each representing one of these two categories. These datasets were sourced from Kaggle and GitHub (links available on the final page). In total, 3,692 images were used in training the model.





The model was compiled using a binary cross-entropy loss function and an SGD optimizer. The binary cross-entropy loss function is specifically tailored for binary classification. In this case, it is used to classify if something is fire or not. The SGD optimizer specified used a momentum factor of 0.9. This allows for the use of gradient descent while also increasing the learning rate.

```
74   print("Compiling model...")
75   opt = tf.keras.optimizers.SGD(lr=0.01, momentum=0.9,
76     decay=0.01 / NUM_EPOCHS)
77   model.compile(loss="binary_crossentropy", optimizer=opt,
78     metrics=["accuracy"])
79
80   print("Fitting model...")
81   H = model.fit(
82     x=trainX,
83     y=trainY,
84     batch_size=64,
85     validation_data=(testX, testY),
86     epochs=NUM_EPOCHS,
87     verbose=1)
```

The model was then trained on a shuffle of 90% of the fire and non-fire images, and subsequently tested on the remaining 10%.

```
2021-04-29 20:58:57.838305: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1406] Created TensorFlow device (/job:loc
alhost/replica:0/task:0/device:GPU:0 with 10491 MB memory) -> physical GPU (device: 0, name: GeForce RTX 3060, pci bus i
d: 0000:01:00.0, compute capability: 8.6)
2021-04-29 20:58:57.840165: I tensorflow/compiler/jit/xla_gpu_device.cc:99] Not creating XLA devices, tf_xla_enable_xla_
devices not set
Loading and resizing images...
Loading and resizing images...
Compiling model...
Fitting model...

Epoch 1/10
52/52 [==============================] - 22s 264ms/step - loss: 0.7597 - accuracy: 0.7440 - val_loss: 0.2065 - val_accur
acy: 0.9595
Epoch 2/10
52/52 [==============================] - 13s 255ms/step - loss: 0.1916 - accuracy: 0.9597 - val_loss: 0.1634 - val_accur
acy: 0.9595
Epoch 3/10
52/52 [==============================] - 13s 250ms/step - loss: 0.0918 - accuracy: 0.9804 - val_loss: 0.1683 - val_accur
acy: 0.9595
Epoch 4/10
52/52 [==============================] - 13s 252ms/step - loss: 0.0934 - accuracy: 0.9741 - val_loss: 0.2116 - val_accur
acy: 0.9595
Epoch 5/10
52/52 [==============================] - 14s 261ms/step - loss: 0.0669 - accuracy: 0.9852 - val_loss: 0.1899 - val_accur
acy: 0.9595
Epoch 6/10
52/52 [==============================] - 13s 245ms/step - loss: 0.0578 - accuracy: 0.9858 - val_loss: 0.2215 - val_accur
acy: 0.9622
Epoch 7/10
52/52 [==============================] - 13s 258ms/step - loss: 0.0470 - accuracy: 0.9903 - val_loss: 0.1755 - val_accur
acy: 0.9622
Epoch 8/10
52/52 [==============================] - 13s 255ms/step - loss: 0.0456 - accuracy: 0.9880 - val_loss: 0.1774 - val_accur
acy: 0.9622
Epoch 9/10
52/52 [==============================] - 13s 257ms/step - loss: 0.0478 - accuracy: 0.9872 - val_loss: 0.1299 - val_accur
acy: 0.9703
Epoch 10/10
52/52 [==============================] - 14s 257ms/step - loss: 0.0585 - accuracy: 0.9862 - val_loss: 0.1245 - val_accur
acy: 0.9676
Testing accuracy...
              precision    recall  f1-score   support

    non-fire       1.00      0.20      0.33        15
        fire       0.97      1.00      0.98       355

    accuracy                           0.97       370
   macro avg       0.98      0.60      0.66       370
weighted avg       0.97      0.97      0.96       370

Saving model...
```

Results

Once the model was trained and saved, it was time to test some real-world data on it.

The first test of the model used a video that I had taken myself. The video used was a clip consisting of 850 frames, in which each frame contained fire. The clip was recorded on my iPhone, with the resolution scaled down.

The results for this test were 100% accurate, showing the datasets provided a good baseline for recognizing fires.

```
Frame 833 => [0. 1.] || 1 (expected 1)
Frame 834 => [0. 1.] || 1 (expected 1)
Frame 835 => [0. 1.] || 1 (expected 1)
Frame 836 => [0. 1.] || 1 (expected 1)
Frame 837 => [0. 1.] || 1 (expected 1)
Frame 838 => [0. 1.] || 1 (expected 1)
Frame 839 => [0. 1.] || 1 (expected 1)
Frame 840 => [0. 1.] || 1 (expected 1)
Frame 841 => [0. 1.] || 1 (expected 1)
Frame 842 => [0. 1.] || 1 (expected 1)
Frame 843 => [0. 1.] || 1 (expected 1)
Frame 844 => [0. 1.] || 1 (expected 1)
Frame 845 => [0. 1.] || 1 (expected 1)
Frame 846 => [0. 1.] || 1 (expected 1)
Frame 847 => [0. 1.] || 1 (expected 1)
Frame 848 => [0. 1.] || 1 (expected 1)
Frame 849 => [0. 1.] || 1 (expected 1)
850 correct out of 850, Accuracy:1.000000
```

The second test of the model used another video that I had taken myself. The video used was a clip consisting of 133 frames of my baby cousin eating Jell-O. No frames contained fire. The clip was recorded on my iPhone, with the resolution scaled down.

The results for this test were also 100% accurate, showing the datasets provided a good baseline for recognizing a lack of fires.

```
Frame 122 => [1. 0.] || 0 (expected 0)
Frame 123 => [1. 0.] || 0 (expected 0)
Frame 124 => [1. 0.] || 0 (expected 0)
Frame 125 => [1. 0.] || 0 (expected 0)
Frame 126 => [1. 0.] || 0 (expected 0)
Frame 127 => [1. 0.] || 0 (expected 0)
Frame 128 => [1. 0.] || 0 (expected 0)
Frame 129 => [1. 0.] || 0 (expected 0)
Frame 130 => [1. 0.] || 0 (expected 0)
Frame 131 => [1. 0.] || 0 (expected 0)
Frame 132 => [1. 0.] || 0 (expected 0)
133 correct out of 133, Accuracy:1.000000
```

The first test of the model used a PBS Video clip of a monarch butterfly flying through the air. The video used was a clip consisting of 212 frames, in which no frames contained fire.

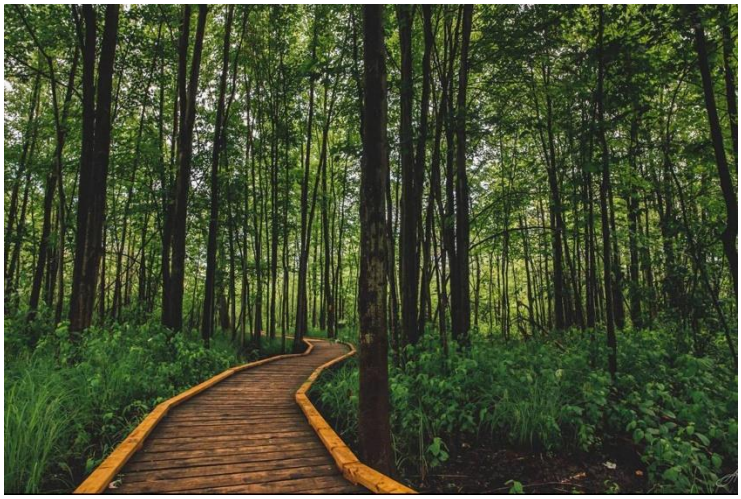
The results for this test were once again completely accurate, showing the model is capable of recognizing a lack of fire, even in images that may appear to resemble a fire at first glance.

```
Frame 200 => [1. 0.] || 0 (expected 0)
Frame 201 => [1. 0.] || 0 (expected 0)
Frame 202 => [1. 0.] || 0 (expected 0)
Frame 203 => [1. 0.] || 0 (expected 0)
Frame 204 => [1. 0.] || 0 (expected 0)
Frame 205 => [1. 0.] || 0 (expected 0)
Frame 206 => [1. 0.] || 0 (expected 0)
Frame 207 => [1. 0.] || 0 (expected 0)
Frame 208 => [1. 0.] || 0 (expected 0)
Frame 209 => [1. 0.] || 0 (expected 0)
Frame 210 => [1. 0.] || 0 (expected 0)
Frame 211 => [1. 0.] || 0 (expected 0)
212 correct out of 212, Accuracy:1.000000
```

The success of these three tests entirely exceeded my expectations, so I decided to investigate which images the model was falsely classifying in the 10% test it performed during model generation.

Running the same testing script I had created for these three videos on the fire and non-fire datasets gave some interesting results.

There were only a few false-positives of the non-fire dataset. They can be seen below. As you can see, most of these false-positive display a striking resemblance to a fire or flame. It’s easy to imagine that wooden path is a flame, that forest glowing with yellow fog is that way because of a fire, or there may be a fire beginning behind that green bush in the third picture.



As for the false-negatives on the fire dataset, there were quite a few more, however most of them have a few things in common. You can see examples below. Specifically, they contained small flames that are all burning in a controlled manner.



Discussion

Test Results

As mentioned, the test results were certainly above my expectations. Especially given the fact that the false categorizations are generally caused by reasonable mistakes.

While the false-negative tests showed flames that were not properly categorized as fires, this is due to the fact that all of these flames were small and controlled, the kind of flames you would expect to see in a house, such as a candle.

The false-positive tests showed non-flames classified as flames. This is not a concern, because in a final product users will be able to report that these are false-positives, allowing the model to be updated accordingly.

Plans for the Future

While the system is showing great results so far, there is definitely much more that needs to be done from here. Not the least of which is integrating the use of the camera modules to the system. It is important that a security system monitors more than one point of view, so there must be some component made to run multiple video feeds against the model. This would be easy enough to do, simply by extracting still frames from them.

Another feature that will need to be implemented into the system is another method for communicating with the user beside the standard terminal output that is currently in place. Whether this takes the form of a web interface, a mobile app, or a simple email system remains to be seen.

Lastly, the final feature that I would like to implement would be detection of more dangers. Since the system is only currently able to detect house fires, there are certainly quite a few options on dangers to add to the system. Broken glass, intruders, and wild animal detections all come to mind. There are two options for how these can be done. Either these are all implemented, along with the fire, to a single "dangers" model, or each frame is processed through multiple models, with each specialized for each danger. The second option would likely be the better one, as it allows for different layers to be used to specialize for each danger type, however this would be much more costly. More testing will need to be done before this decision can be made.

Even with all the modifications that will need to be done in the future, I am happy with where the project stands currently. It is exciting to see the model performing so accurately on the data that is provided to it, and I can't wait to see what the project becomes in the future.

Sources

fire and non-fire datasets: https://www.kaggle.com/phylake1337/fire-dataset

second fire dataset: https://github.com/sulenn/fire-dataset/

PBS Video, monarch butterflies: https://www.youtube.com/watch?v=lWOySU_hAz0