Image Classification for Public Security

Alexandra Pawlak
University of Washington Tacoma
TCSS600
apawlak@uw.edu

1 Introduction

Public safety is an ever-growing concern in people's daily lives. Walking around in metropolitan and high traffic areas can be dangerous for many people as not everyone can defend themselves. People should be able to live their lives and go about their business without the constant fear that someone may try to attack them or steal from them. However, there is no way to stop all criminals since they will always find new ways to commit the acts and crimes that they want to.

This project stems from another project I worked on building a security system that integrated IBM Watson Visual Recognition to recognize criminal activity. The basics of the system is that it is a security system built on a Raspberry Pi equipped with a motion sensor, sound sensor, and camera module. When either the motion or sound sensors are activated, the camera captures images and sends them to IBM Watson for classification. The classification model used is a small custom built model that classifies images either as 'safe' or 'threat'. If a threat is detected, then an alert is sent. The goal of this project was to be able to alert authorities of crimes or attacks in real-time so that they would be able to respond faster and to more crimes and have a better chance at catching criminals.

When building this project, there were some hiccups in creating the classification model. IBM Watson was having trouble classifying more than 20 images in each class, it would freeze during the training process. This resulted in a very small batch of training images, which affects the accuracy of the model when classifying images. And for the project, this could mean miss-classifying images of threats and leaving people vulnerable.

This lead me to wonder if I could build my own image classification model that would be able to train on more images and possibly classify more accurately than the small model on IBM Watson.

This image classification model is Convolutional Neural Network built using Keras with Tensorflow as the back-end. The data set is self-created, so it is still on the small size when comparing to other deep-learning problems. The training dataset is 200 images, the validation dataset is 50 images, and the testing dataset is 30 images; all split between two classes, 'threat' and 'safe'. The goal of the network is to classify images that could be taken in public areas from security cameras and detect for possible crimes that are happening.

2 Dataset and Evaluation Metrics

The dataset used in training and evaluating this CNN consists of 250 images that were hand selected from various Google image searches. These images are evenly split between two classes, 'threat' and 'safe' and split 80/20 for training and validation. There is a separate testing dataset that consists of 30 images evenly split between the two classes. This test dataset is also used for testing the accuracy of the IBM Watson Visual Recognition model for comparison.

The dataset is rather small when comparing to traditional deep-learning problems. This is because there were no available datasets that fit the need of this project available, to my knowledge. Creating a dataset by hand is a very time consuming task. Ideally, there would be thousands of images in each class for training because the more examples a network can train on, the higher the accuracy and the less likely it is to overfit the data.

Images can be represented by three-dimensional tensors of size length-by-width-by-3. The length and width are the size of the image and the 3 represents the RGB color intensities of each pixel in the image. For consistency, the images are all scaled to the same size.

For evaluating the CNN, I looked at the accuracy of both the training and validation data sets as well as their respective loss. I used this information to try to minimize overfitting of the model. For comparison of my CNN to the model built on IBM Watson Visual Recognition, I compared the accuracy of the models ability to predict on the test dataset.

3 Methodology

The model is a Convolutional Neural Network using Keras and Tensorflow. Since the images were hand selected from Google image searches, their sizes can vary. To obtain consistency in the image sizes, they were all re-sized to be 200-by-200 pixels. Additionally, all of the images were augmented by applying shear, zoom, and flipped.

The Convolutional Neural Network was built based on a few examples, including the VGG-16 classifier. The

layers of the network were added and tweaked to try to optimize the accuracy of the model and lower the loss per epoch. By optimizing these metrics, the model becomes more ideal and less over- or under-fitted. The model uses the rmsprop optimizer and categorical-crossentropy. I chose to use categorical-crosentropy as opposed to binary since the model is classifying the image based on two classes and not just a yes or no regarding the image. Then the model takes the higher percentage of the two classes and chooses that for it's classification.

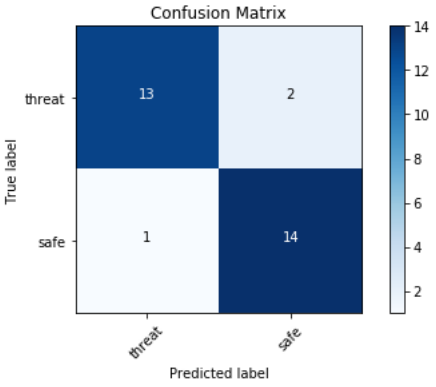
The final model that was producing the best accuracy and lowest loss is shown below.

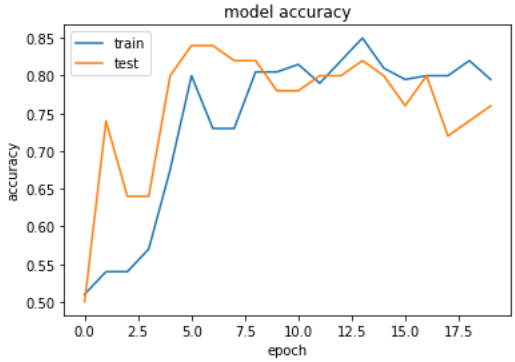
```
model = Sequential()
model.add(Conv2D(25, (3, 3), input_shape=input_shape))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(50, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(75, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(100, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(100))
model.add(Activation('relu'))
model.add(Dense(50))
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(Dense(25))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(2))
model.add(Activation('sigmoid'))

model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

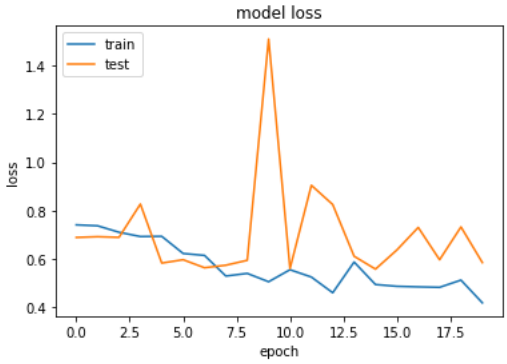
4 Results

The final model is producing results with 90% accuracy on the testing dataset. The accuracy on the training and validation datasets tends to hover around 70-80%. The accuracy of the validation dataset tends to be lower than that of the training, which shows that the model is slightly overfitting. However, it is classifying the test images at a much higher accuracy.

When running the same test dataset through the custom model built on IBM Watson Visual Recognition, the accuracy results were identical at 90%. The only difference in the results between the two models was that the Watson model classified 13/15 'safe' images correctly and 14/15 threat images correctly, while the CNN classified 14/15 'safe' images correctly and 13/15 threat images correctly. Below is the confusion matrix for the CNN, showing the classification results. Additionally, there are graphs showing the accuracy of the training and validation during each epoch, as well as their corresponding loss.







5 Conclusion

The Convolutional Neural Network was built on a very small dataset, however it produced very good results despite of this. The goal was to try to build a model that could out perform the custom model built using IBM Watson Visual Recognition and I was able to construct a model that matched the classifcaion accuracy.

IBM Watson Visual Recognition is a truly powerful tool. However it does have it's problems, especially when using the free version. The goal of this project was to train a Convolutional Neural Network that could out perform the very small model that I was able to successfully create on IBM Watson. In the end, with 10x as many training images I was able to build a model that outperformed the power of the model on IBM Watson. With more time and training images, I think I could continue to improve this model even further.

Despite many hours trying to fine-tune the CNN model, it is still overfitting the training data, which will affect the performance of the model the grand scheme. The purpose of the model is to be able to detect potential crimes and threats via images that are captured on security cameras in public spaces. The model performed exceptionally well on the testing dataset. Solving the problem of overfitting will allow the model to perform even better across a larger and

more diverse test set as well.

6 Future Work

Going forward, I would like to be able to take the time to curate a larger training dataset and fine tune the images. I believe the model could work a lot better if the images were initially the same or similar sizes, so as to not distort them out too much when modifying them in the CNN. The more images I am able to collect to train on, the better my model will be able to perform and the less likely it will be overfitting the data.

In the grad scheme, the reasoning for the creation of this model came from an image recognition security system. Growing the dataset would also allow for training on more scenarios and being able to detect them. Some crimes are harder to see happening, such as pick-pocketing, but this is something that could be tackled with a larger dataset.

Another technique that I found interesting, but did not have time to test out was fine-tuning a pre-trained classifier. I found that you can use models such as the VGG-16 to help enhance a small model. Since my model is so specific to what it's classifying, I would be interested to see if adding in a larder model like this would enhance the accuracy of the model.

References

[1] Francois Chollet. Keras. 2016. https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html
[2] Deeplizard. 2017. https://www.youtube.com/channel/UC4UJ26WkceqONNF5S26OiVw
[3] Deeplizard. Patreon. 2018. https://www.patreon.com/posts/code-for-keras-1-19266488
[4] Jason Brownlee. "Display Deep Learning Model Training History in Keras" Machine Learning Mastery. 2016.
[5] http://cv-tricks.com/tensorflow-tutorial/training-convolutional-neural-network-for-image-classification/