Project:
Security Threat detection

Arnav Ashank
School of Engineering and Applied Sciences
University at Buffalo
arnavash@buffalo.edu

Vinson Chen
School of Engineering and Applied Sciences
University at Buffalo
vinsonch@buffalo.edu

1 Introduction

The purpose of this project is to build a threat detection deep learning model. This model will identify whether the object is that of an employee or not at a given organization. The use case for this model is that of a surveillance system. It will be able to detect whether something or someone is a threat. The model can be implemented as a surveillance system outside the premises of army bases, research centers, government facilities, private properties, museums etc. It can be used for detection for trespassers as an alarm system.

This is done by inputting images into our self-built CNN model. The idea is that these photos would be captured by a security camera setup around an important location or building. Then, the CNN model will determine whether or not a threat is captured in each image. There are two different classes for our combined data set, threat or non-threat. In total, there are 13,880 samples to be used for training and testing purposes. These images were taken from three separate data sets of images. The main purpose of our CNN model is to perform binary classification.

2 Data Sets

2.1 Data Sources and Information

We used three different data sets of images. The first data set contains images of objects, such as animals and vehicles. More specifically, there are images of animals like birds, cats, deer, dogs, horses, and monkeys. There are also images of vehicles like airplanes, cars, ships, and trucks. In total, there are 8,000 images of just animals and vehicles. This data set was retrieved from https://www.kaggle.com/competitions/deeplearningclassificationtask/data. Specifications of each image include the size of 96 x 96 and the format of BMP. These specifications are important for preprocessing which will be mentioned later.

The second data set was retrieved from https://susanqq.github.io/UTKFace/. This data set only contains images of human faces with ages ranging from 0 to 116 years old and sorted by age. In total, there are 20,000 sample images. To get a more equal distribution of image types for the combined data set, we only took 2,000 samples out of the 20,000. This was done by shuffling the data set and taking the first 2,000 samples. Shuffling was performed first so that we can get a more equal distribution of ages from 0 to 116. Specifications of each image include the size of 200 x 200 and the format of JPG.

The final data set was retrieved from https://www.kaggle.com/datasets/dansbecker/5-celebrity-faces-dataset. This data set also contains images of human faces. However, these images are of celebrities. There are 5 different celebrities in total, and there are only a few images for each. These celebrities are Ben Affleck, Elton John, Jerry Seinfeld, Madonna, and Mindy Kaling. In total, there were 118 images. Each is in JPG format but with varying sizes. Images from this celebrity data set are labeled as non-threat and considered as an employee for an organization. These images can be replaced with real ones of employees of an organization. The images from the other two data sets are labeled as threats. We need more images for the celebrity data set as our dataset is imbalanced. To achieve this, we used image augmentation to create more samples. This process will be explained more below.

2.2 Accessing Data Set

There are two options to access the combined data set. Either downloading the final preprocessed version from https://drive.google.com/drive/folders/1MdrKGUMyhiwb1J_BsdUwzEqBSWqk76b0?usp=sharing or by downloading the three original data sets and following the preprocessing steps provided on GitHub. It is recommended to simply download the preprocessed version to save time.

3 Preprocessing

The goal of preprocessing was to combine all three data sets. The two main steps were converting all the images into the same image type, as well as having all images be the same size. Since the second data set of human images were 200 x 200, we decided to convert all images to that size. The first data set contains images of size 96 x 96 and the third data set contains images of varying image sizes. The **ImageMagick** software tool was used to perform the image resizing via command line. This tool was also used to convert the image type of the first data set (BMP) to the same as the other two data sets (JPG).

The celebrity data set had images of varying image sizes. We decided to only keep the images that were square, so that resizing them to 200 x 200 would be easier. This reduced the total number of images from 118 down to 33. Then, we performed data augmentation to increase the number of samples. We were able to increase the total number of images from 33 up to 4983. That means there are 151 variations of each image (150 + original).

Before combining all three data sets, images from each of the three folders were renamed to include their respective labels. The inclusion of "_threat" was added to the end of each image name for the first two data sets of objects and human faces. The inclusion of "_nonthreat" was added to the end of each image for the third data set of celebrities. Then, the JPG extension was added to the end of each image file.

Then, the images from the three different folders were combined into a single folder. Finally, the last step of preprocessing was to shuffle the images. That way it would be easier to split the data set into training and testing sets. The sample code is provided in a separate file on the GitHub link.

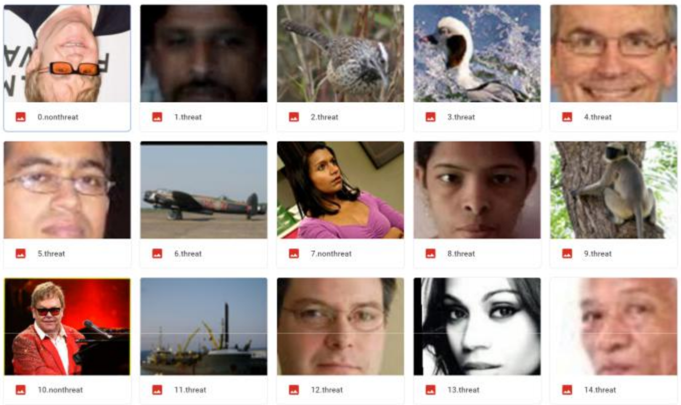


Figure 1. First 15 samples of the combined data set

4 Model Selection

Here we used Convolutional Neural Networks to classify images as threats/non-threats from the dataset. After data collection, data labeling and cleaning the data we tried normalizing the images before applying convolutional neural networks. Due to system constraints we have built one base model with normalized images and hyperparameter tuning has been performed on the model based on unnormalized images. We tested out several different methods, such as Dropout, L1 Norm, L2 Norm, and Early Stopping. The accuracy rates and losses for both training and testing for each method are shown below.

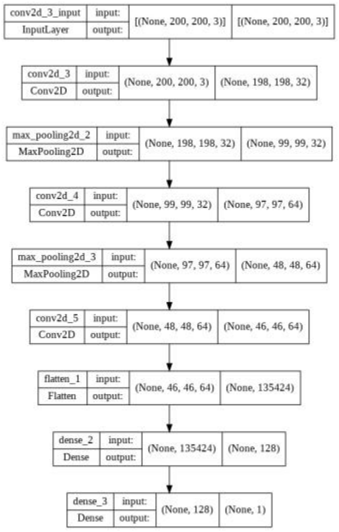

conv2d_3_input | InputLayer | input: | output: | [(None, 200, 200, 3)] | [(None, 200, 200, 3)]
conv2d_3 | Conv2D | input: | output: | (None, 200, 200, 3) | (None, 198, 198, 32)
max_pooling2d_2 | MaxPooling2D | input: | output: | (None, 198, 198, 32) | (None, 99, 99, 32)
conv2d_4 | Conv2D | input: | output: | (None, 99, 99, 32) | (None, 97, 97, 64)
max_pooling2d_3 | MaxPooling2D | input: | output: | (None, 97, 97, 64) | (None, 48, 48, 64)
conv2d_5 | Conv2D | input: | output: | (None, 48, 48, 64) | (None, 46, 46, 64)
flatten_1 | Flatten | input: | output: | (None, 46, 46, 64) | (None, 135424)
dense_2 | Dense | input: | output: | (None, 135424) | (None, 128)
dense_3 | Dense | input: | output: | (None, 128) | (None, 1)

Figure 2: Computational graph of CNN mode,

Figure below depicts the accuracy and loss of training and test set with epochs for the normalized images base model:

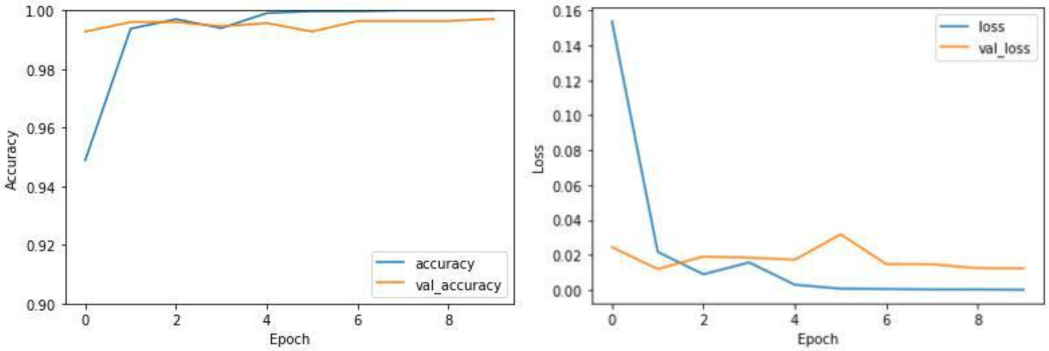


Figure 3: Base model on normalized images (accuracy + loss)

The accuracy on the test set for the normalized images is 99.71%.

Figure below depicts the accuracy and loss of training and test set with epochs for the unnormalized base model:

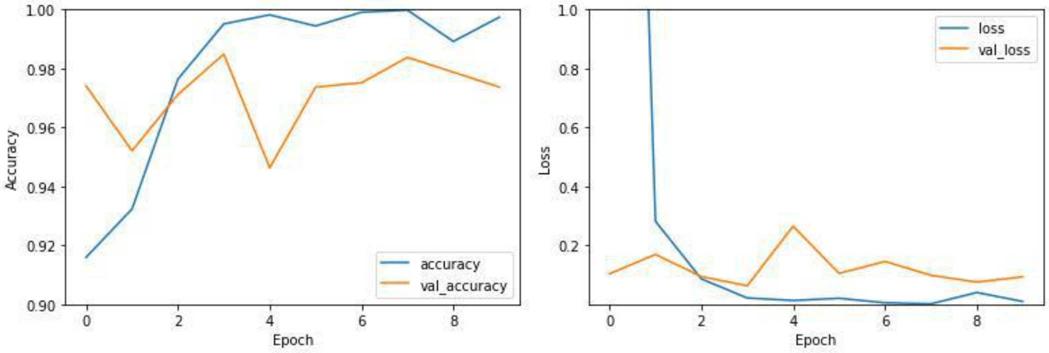


Figure 4: Base model on unnormalized images (accuracy + loss)

The accuracy on the test set for the unnormalized images is 97.23%.

5 Hyperparameter Tuning

For hyperparameter tuning, we tested out several different weight regularizer methods. We generally saw high accuracy rates for both testing and training. However, these accuracies tended to drop as the number of epochs increased. Therefore, we used Early Stopping to get the best accuracy rates for both training and testing.

5.1 Dropout

Dropout is a regularization method that approximates training a large number of neural networks with different architectures in parallel.During training, some number of layer outputs are randomly ignored or “*dropped out*.” This has the effect of making the layer look-like and be treated-like a layer with a different number of nodes and connectivity to the prior layer. In effect, each update to a layer during training is performed with a different “*view*” of the configured layer.

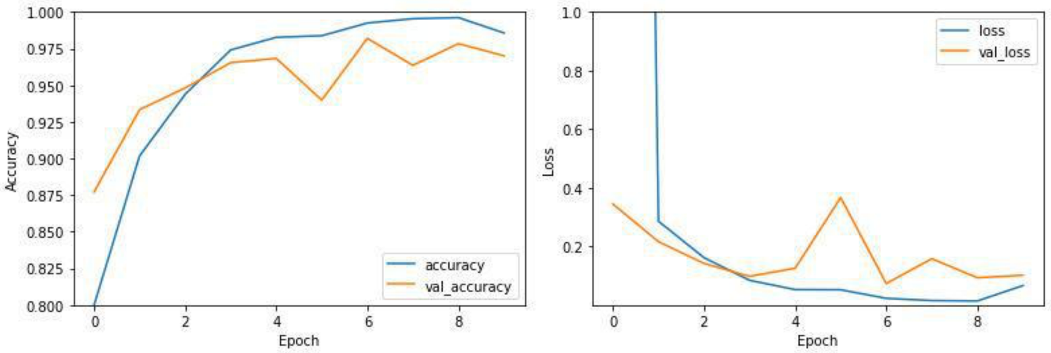


Figure 5: Dropout on unnormalized images using Dropout (accuracy + loss)

The accuracy on the test set for the unnormalized images using Dropout is 97.01%.

5.2 L1 Norm

Regularization is a method that controls the model complexity. Each feature is assigned a certain weight. If there are a lot of features then there will be a large number of weights, which will make the model prone to overfitting. So regularization reduces the burden on these weights. The weights then end up having less impact on the loss function which determines the error between the actual label and predicted label.

$$\boxed{L(x,y) \equiv \sum_{i=1}^{n}(y_i \; - h_\theta(x_i)\,)^2 + \lambda \boxed{\sum_{i=1}^{n}|\theta_i|}}$$

Here theta is our parameter that we multiply with our input to get a prediction. In L1 regularization we take the modulus of all parameters and sum them up. Lambda is set before initiating the training.

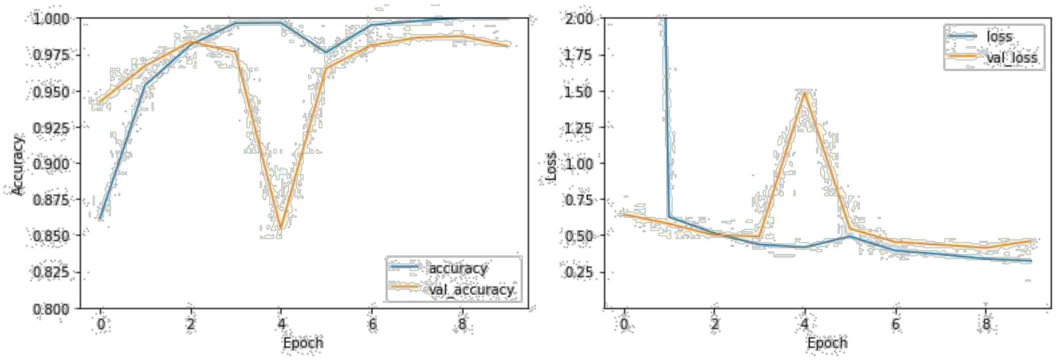
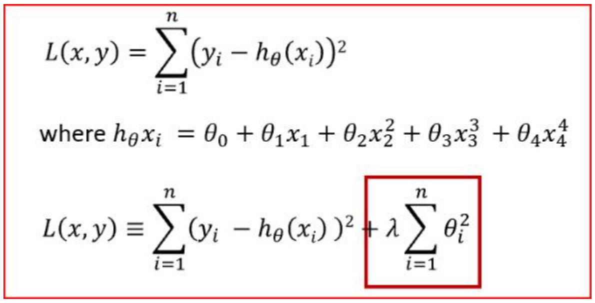


Figure 6: L1 Norm on unnormalized images using L1 Norm (accuracy + loss)

The accuracy on the test set for the unnormalized images using L1 Norm (lambda = 0.01) is 97.84%.

5.3 L2 Norm



$$L(x,y) = \sum_{i=1}^{n} (y_i - h_\theta(x_i))^2$$

where $h_\theta x_i \ = \theta_0 + \theta_1 x_1 + \theta_2 x_2^2 + \theta_3 x_3^3 \ + \theta_4 x_4^4$

$$L(x,y) \equiv \sum_{i=1}^{n} (y_i \ - h_\theta(x_i)\ )^2 \boxed{+ \lambda \sum_{i=1}^{n} \theta_i^2}$$

In L2 regularization we take the sum of all the parameters squared and add it with the square difference of the actual output and predictions. Same as L1 if we increase the value of lambda, the value of the parameters will decrease as L2 will penalize the parameters. The difference is that the weights will not be sparse and we will get much better accuracy values compared to L1.

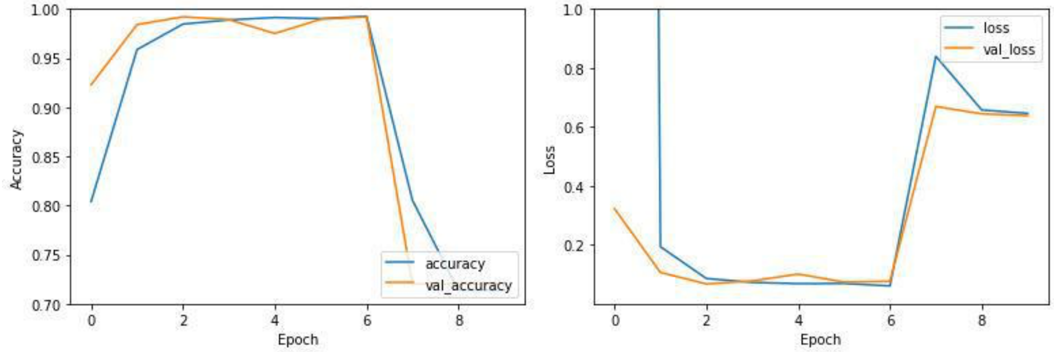


Figure 7: L2 Norm on unnormalized images using L2 Norm (accuracy + loss)

The accuracy on the test set for the unnormalized images using L2 Norm (lambda = 0.01) is 72.04%.

5.4 Early Stopping

Early stopping is a method that allows us to specify an arbitrarily large number of training epochs and stop training once the model performance stops improving on the validation dataset.This requires that a validation split should be provided to the fit() function and a EarlyStopping callback to specify performance measure on which performance will be monitored on validation split.

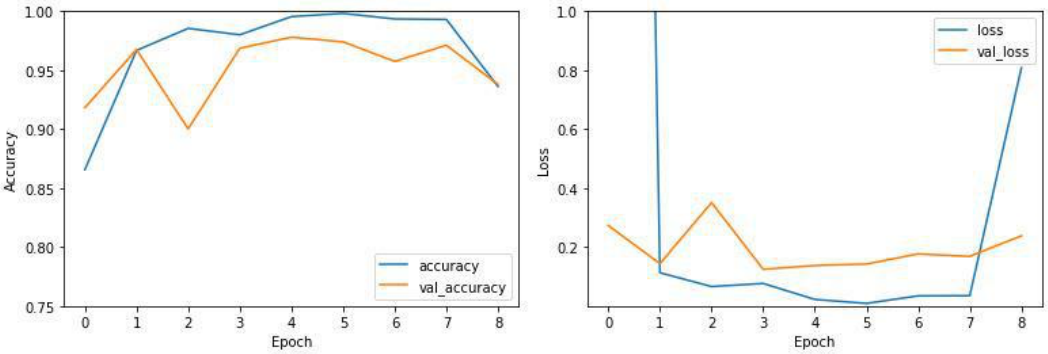


Figure 8: Early Stopping on unnormalized images using Early Stopping (accuracy + loss)

The accuracy on the test set for the unnormalized images using Early Stopping is 93.8%.

6 Challenges

There are some issues that we would like to note. The first is that most of our work was done on Google Colab to make use of the GPU acceleration. However, the amount of RAM is limited to just 12 GB. We specifically run into issues when trying to normalize the data (divide the data set values by 255) and when trying to create more than 150 augmentations of each celebrity image. Our Google Colab session will crash. We had to switch to a local machine to bypass these problems. Unfortunately, it is much slower to train the CNN model on our local machine compared to Google Colab. Therefore, we performed our hyperparameter tuning by training several different CNN models on Google Colab without normalizing the data set. Even so, our instances crash often.

If we didn't face these challenges, then we would have extended the project to include hyperparameter tuning with normalized data and video capturing. We would take a video recording and break it up into image frames. Then, we would feed these images into our CNN model to determine whether an object is a threat or not. The few attempts we made at including video capturing for our project did not pan out well. It was difficult capturing videos using our laptop webcams. The object sizes in the captured videos were too small to distinguish.

7 Conclusion

After hyperparameter tuning using different regularizers, we found that both the normalized and unnormalized CNN models performed better than most of the regularized models. The best performing model for normalized images was the base model with an accuracy of 99.71%. The best performing model for the unnormalized images was the L1 Norm model with an accuracy of 97.84%. It performed just slightly better than the base model. The worst performing model was with L2 Norm with an accuracy of 72.04%. This model can be extended to real-time applications to include every employee of the organization. We would add images of each employee into the data set and label them as a threat or not. Overtime, the model will be able to determine whether someone or something is a threat or not.

GitHub Link: https://github.com/Vinsonch/CSE-676-Project

8 References

- Brownlee, Jason. “Multi-Class Classification Tutorial with the Keras Deep Learning Library.” *Machine Learning Mastery*, 31 Dec. 2020, https://machinelearningmastery.com/multi-class-classification-tutorial-keras-deep-learnin g-library/.
- “Deep Learning Classification Task.” *Kaggle*, https://www.kaggle.com/c/deeplearningclassificationtask/data.
- DanB. “5 Celebrity Faces Dataset.” Kaggle, 10 Nov. 2017, https://www.kaggle.com/datasets/dansbecker/5-celebrity-faces-dataset.
- “Large Scale Face Dataset.” *Utkface*, https://susanqq.github.io/UTKFace/.
- “Convolutional Neural Network (CNN) : Tensorflow Core.” *TensorFlow*, https://www.tensorflow.org/tutorials/images/cnn.
- Akilos, Ryan. “A Simple Example: Confusion Matrix with Keras flow_from_directory.Py.” *GitHub*, 2 Sept. 2017, https://gist.github.com/RyanAkilos/3808c17f79e77c4117de35aa68447045.
- Brownlee, Jason. “How to Visualize a Deep Learning Neural Network Model in Keras.” *Machine Learning Mastery*, 11 Sept. 2019, https://machinelearningmastery.com/visualize-deep-learning-neural-network-model-keras /.
- Early stopping https://medium.com/zero-equals-false/early-stopping-to-avoid-overfitting-in-neural-netw ork-keras-b68c96ed05d9