Safety Helmet Detection using Convulutional Neural Networks

Jordan O'Donovan
School of Computing
National College of Ireland
Dublin, Ireland
x19372016@student.ncirl.ie

***Abstract*—Computer vision is a rapidly evolving aspect of deep learning, with new models and methods constantly being released or updated. This paper will document the results of a convolutional neural network created from scratch to two pretrained models, SGG16 and ResNet50 when attempting to differentiate between images of heads and safety helmets using image classification. Accuracy, recall, precision and loss will be investigated. Overall, from analysing these metrics, the models created from scratch performed best for this task.**

***Keywords*—Convolutional Neural Network, Image Classification, Keras, TensorFlow, Deep Learning, VGG16, ResNet50, Computer Vision**

I. Introduction

The objective of this project is to utilise computer vision and deep learning to develop a neural network which can accurately predict whether a picture of somebody contains just their head or if they're wearing a safety helmet (the type required for construction sites). I will create a Convolutional Neural Network (CNN) to accomplish this image classification task and will gauge the performance of my model by evaluating the accuracy, recall, precision and loss. I will incorporate Dropout layers and regularization if needed to reduce overfitting.

II. Data

This dataset, created by Liangbin Xie in 2019 at Northeastern University in China, contains 7,038 images of varying height and width dimensions in the JPEG format. Each image contains at least one person's head, whether it be with a safety helmet or without. The dataset is split into two folders, Train and Test. Three-quarters of the images are in the train folder with the remainders in Test. Each folder also has a CSV file named "_annotations.csv" which contains the filename for each image, their width and height, the label and the x & y coordinates for a box encapsulating the head/helmet. Each image has on average 3.75 heads and/or helmets. One image in the train set has thirty-five heads/helmets and in the test set the image with the highest number has twenty-eight. In total, there are ten images which have twenty or more heads/helmets.

III. Related Work

A. A Study on CNN Transfer Learning for Image Classification

In 2018, Mahbub Hussain Jordan J. Bird and Diego R. Faria published a paper investigating transfer learning with CNN for image classification [1]. Here, they used two datasets, CIFAR-10 (a dataset containing 60,000 32x32 images and 10 classes) and Caltech Faces (10,524 images of faces with different resolutions and settings). They chose the model Inception-v3 which was pretrained on the ImageNet dataset. They then retrained this pretrained model using their two datasets. From this, they were able to return an accuracy almost twice what another paper was able to achieve (70% vs 38%), where that author created a CNN-CIFAR-10 model from scratch.

From this paper I was able to further my knowledge of CNNs, and it introduced me to the concept of transfer learning, which I now plan to use after I create my own CNN models to compare the results.

B. Medical image classification with convolutional neural network

For this paper Qing Li et al. investigated the effectiveness of using a CNN model to classify lung image patches with interstitial lung diseases (ILD) [3]. They compared their CNN model to three other feature extracting methods: scale-invariant feature transformation (SIFT), rotation-invariant local binary patterns (LBP) feature and unsupervised learning using restricted Boltzmann machine (RBM).

Their model was fairly simple: a Conv2D layer with sixteen neurons, a max pooling layer also with sixteen neurons and finally three fully connected layers. They utilized dropouts in their model to reduce overfitting.

What they found was that this simple CNN model performed better than the other three methods given above, based on recall and precision.

What I'll take away from this paper is that a CNN model does not have to be very complex to outperform other methods, and it enforced the importance of recall and precision upon me.

C. Detecting Affect States Using VGG16, ResNet50 and SE-ResNet50 Networks

In 2020, Dhananjay Theckedath and R. R. Sedamkar published this paper, investigating the performance of three pretrained models, VGG16, ResNet50 and SE-ResNet50. For their data they used the CK+ dataset which contains 2,502 images of people's faces displaying one of seven emotions.

They found that with this dataset, each model returned almost perfect recall and precision values, however it took ResNet50 only one epoch to reach a validation accuracy >95% while SE-ResNet50 didn't reach this accuracy until its twenty-fourth epoch. For recall and precision, ResNet50 returned the best results, with SE-ResNet50 coming in second and VGG16 coming in third. ResNet50 returned a perfect precision value for five of the emotions and three for recall.

After reading this I've further increased my knowledge of transfer learning and I'm now heavily considering using VGG16 and ResNet50 to compare my CNN models to.

D. Face Detection with the Faster R-CNN

In the final paper I read when researching this project, Huaizu Jiang and Erik Learned-Miller used the faster R-CNN model, which is designed for generic object detection, to perform face detection [4].

They based their training on VGG16 pretrained with the ImageNet data. They compared its performance to the R-CNN and fast R-CNN and also compared it to other state of the art face-detection methods.

After running their tests, they found that the effectiveness of faster R-CNN may come from the regional proposal network (RPN) module. The faster R-CNN proved to be highly successful at facial recognition despite the model running very quickly when compared to other popular models.

From this paper I discovered faster R-CNN and its previous two generations, and I learned a lot about object and facial detection. If I decide to incorporate object detection for the safety helmets, I will compare the results of my CNN model to the faster R-CNN model. Hopefully this will not be very difficult on my current hardware limitations, given how quickly faster R-CNN tends to run.

IV. Deisgn Approach and Implementation

To make these images easier to work with, I used the data found in the CSV files to crop each head/helmet from each image and save it as a new JPEG file. As I intended to use Keras' image_dataset_from_directory() function, instead of saving each image into train/test folders, I created a "helmet" and a "head" folder to store the images, as the function expects each image in a folder to belong to the same class. There were now 26,424 images.

Once I had created my variable containing each image and their label as a BatchDataset with the default shape (256, 256, 3) to avoid loading each image into memory, I then created another variable containing this data as NumPy iterators using the .as_numpy_iterator() function. I considered converting the images to their grayscale counterparts using MatLab's rgb2gray() function as the helmets in the images were many different colours, but I decided to keep the images as they were unless my models performed with low accuracy.

Next, I divided each pixel by 255 (the maximum RGB value possible) so that a pixel could only contain a value between 0 and 1. I then split the data into train (70%) validation (20%) and test (10%) sets. I kept the batch sizes at their default to see if my hardware would be able to handle it thus the test set containing 10% of the data was made up of eighty-two batches.

The first model I created was a very simple, five-layer CNN model which I intended to use as my foundation for future models. My next model contained many more layers, some of which were dropout layers to reduce any overfitting present. Model three was taken from a DataCamp example to compare my work to a model which is proven to function correctly.

Each of these Sequential models will contain a final Dense layer with a sigmoid activation and will be compiled with Adam optimiser, with accuracy and binary crossentropy as the loss being measured. Sigmoid and this loss will be used as the model can only predict one of two labels i.e., it's binary. If more labels were to be used a softmax activation would be used instead and the loss would be categorical crossentropy. Any other layer with its activation specified will be using Rectified Linear Unit (ReLU).

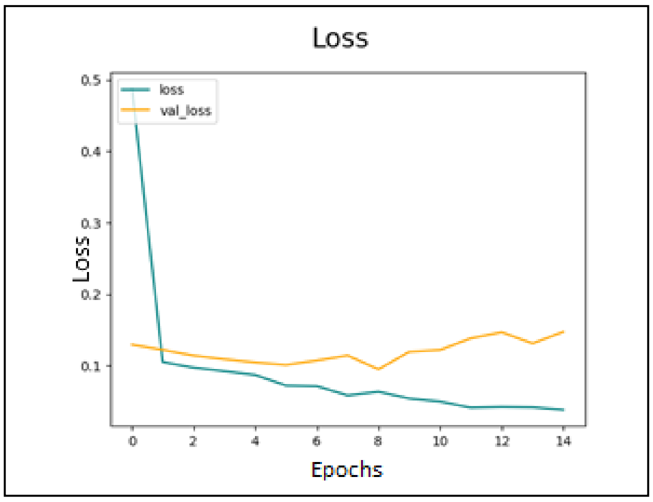
After these CNN models I decided to compare the results seen to two pretrained models, those being VGG16 and ResNet50. For these I had to reshape the data to (224,224,3) as that's what the models require.

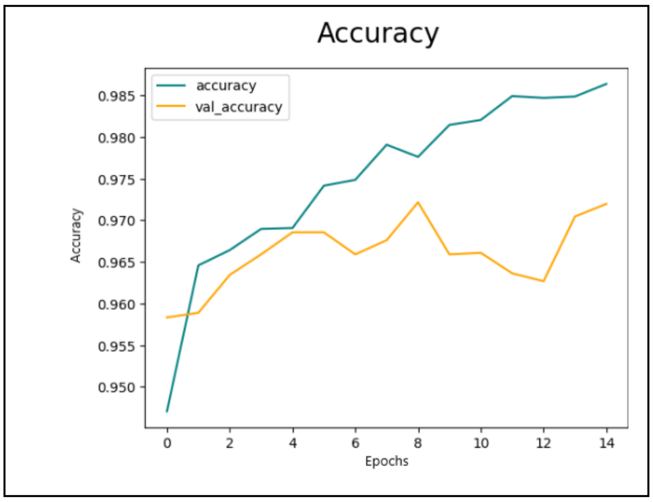
I will perform some hyperparameter tuning for my models, assuming my hardware is able to handle it. Unfortunately, the laptop I'm working from has an AMD GPU, but TensorFlow requires an Nvidia GPU with Cuda cores to run the TensorFlow-gpu package, thus the computational power required to train and test my models will be run through my four-core CPU, significantly hindering the time required to run my Python code.

V. Results and Evaluation

My first model performed much better than I had initially expected. This Sequential model had only five layers, a Conv2D, MaxPooling2D, Flatten, a Dense layer then the final Dense layer with sigmoid.

I trained this model using fifteen epochs. The first had a loss value of 48.81% and an accuracy of 94.71%, but this loss immediately dropped off while the accuracy increased over the following epochs.





The accuracy of the training data increased, and loss decreased with each epoch, but for the validation data, these two values quickly plateaued before trending in a negative direction. From these results I would recommend using eight epochs as the validation loss was the lowest here and it had a high validation accuracy. Using more epochs would improve the accuracy for both sets but the validation loss would increase.

These results indicate that there may be overfitting happening, but once we evaluate the model with the test set, the accuracy returned was 97% with a loss of 13.6%.

To overcome any overfitting, I added more layers, some of which were Dropout layers, into my model. Similar results were seen here, the loss decreased almost linearly after the first epoch, while the validation loss began to increase after the fourth epoch. The accuracy for both was higher here though, with the test set accuracy reaching 99.4% with a loss of 1.59% on the final epoch. Validation accuracy was 97.41% with a loss of 15.98%. Results from the test set were very close to the previous, with an accuracy of 97.18% and loss of 14.55%

To compare these results to somebody else's model, I ran a model from DataCamp with my data. As it took over an hour for each epoch to run on my CPU and returned an out of memory (OOM) error multiple times, I decided to stop it after one epoch. Here, it returned an accuracy of 94.14% with a loss of 22.26%. Validation data performed better with an accuracy of 96.21% and loss of 11.41%.

My next three models were expansions of the first (which is why they're placed out of numerical order in the Jupyter Notebook).

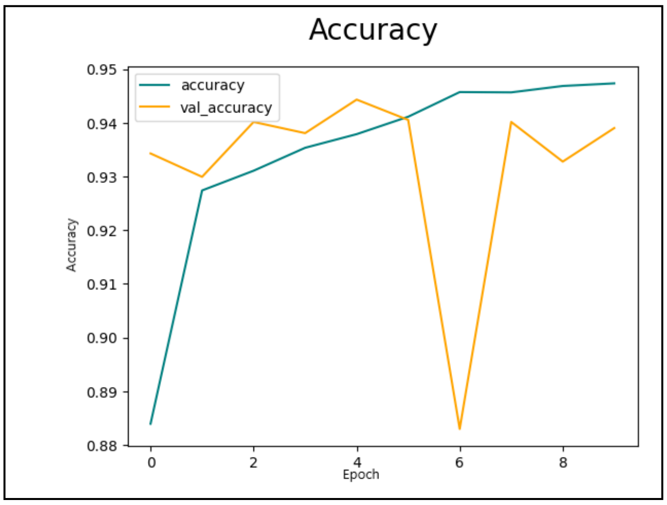
To reduce the validation loss, I added regularization (0.01 L2) to the second last Dense layer of the first model. This increased the accuracy of both train and validation and greatly improved the loss of train data, but it did not stop the validation loss from increasing.

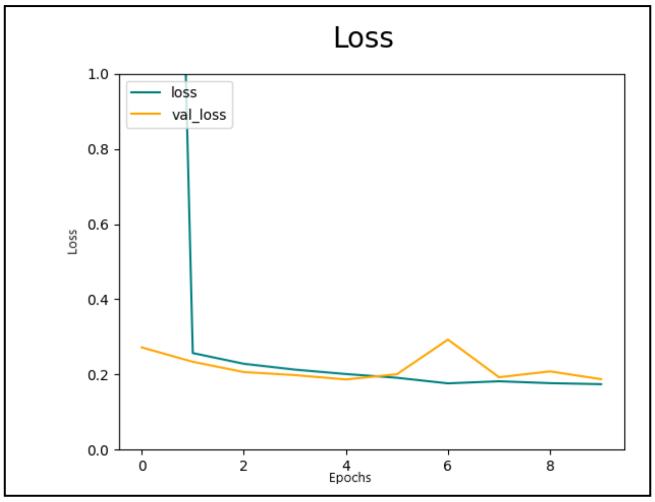
For the second last CNN model I created from scratch, I moved the regularization from the previous model to the first layer. I also added dropout layers to the model, with the first being 25% and the second being 50%.

This decreased the accuracy by a few percentage points and increased the loss for both subsets, although the losses each experienced were almost identical. Epoch six was the outlier for this model however. I suspect if it were to be run again, the output would be much more similar to the epochs before and after it.

While this model did not return the most ideal results seen so far, it was possibly the most robust model, given its dropout and regularisation layers, reducing the chances of overfitting significantly.

This model achieved a precision score of 95%, a recall score of 97.3% and a binary accuracy of 94.2%. The F1 score of this model was 96%.





For my final CNN model, I used the model just written about, but I changed the learning rate in Adam from the default 0.001 to 0.01. After doing this, the accuracy for training and validating was between 74.5% and 75.1% with the losses for each being 56-57% after the first epoch.

Next, I decided to compare the results of my CNN models to VGG16. Using the pretrained model, I tried to get it to predict what it was looking at. For most images it thought it was a mosquito net (with low confidence). Next, out of curiosity, I tried running it without training using the training and validation data with various hyperparameters. What I found was the higher the steps_per_epoch value was, the better the results. For the best results seen, the accuracy for train and validation were usually almost 80% with a loss in the low 40's. In total there were 14,715,201 parameters for this model. I then tried it with training turned on. Here, the accuracy was about the same, but the loss increased to the mid 60's. There were 138,358,545 parameters here, all trainable.

Following this, I moved onto the ResNet50 model. For this model, the accuracy returned for the training set and validation were usually in the mid 70's and the loss was usually between 35-40%.

VI. CONCLUSIONS

From these results, it is clear that the CNN models I created were more accurate than VGG16 and ResNet50 models. This may be due to my inexperience with these two models or with transfer learning. Adding dropout and regularisation layers to my models may not have been necessary given the results from my test data but a slight decrease in performance may be acceptable if it means that the chances of overfitting a reduced significantly.

If I were to do this again or if I had more time, I would experiment more with hyperparameter tuning. I would try out my CNN models with varying values of learning rates, regularisation and dropouts, and I would increase the steps taken by VGG16 and ResNet50. I would also experiment with regularisation and learning rates with these models, to see if it's possible to make them much more accurate for this image classification task. I would also like to use the Faster R-CNN model as it has a reputation for being very quick to run while remaining accurate.

The largest factor holding me back while undertaking this assignment was my hardware limitation. Due to the lack of a compatible GPU, running these models took a very long time, unless I reduced the number of images used which made the results less accurate. I also would've been able to investigate the results from my models more, as my Jupyter Notebook crashed many times and I didn't have time to train each model every time I wanted extract more information from them. In hindsight, I should've used a Python module like Pickle to export and import my models to overcome the need to retrain them.

References

[1] Hussain, M., Bird, J.J., Faria, D.R. "A Study on CNN Transfer Learning for Image Classification." *In: Lotfi, A., Bouchachia, H., Gegov, A., Langensiepen, C., McGinnity, M. (eds) Advances in Computational Intelligence Systems. UKCI 2018. Advances in Intelligent Systems and Computing*, vol 840. https://doi.org/10.1007/978-3-319-97982-3_16

[2] Theckedath, D., & Sedamkar, R. R. "Detecting Affect States Using VGG16, ResNet50 and SE-ResNet50 Networks." In 2020 *SN Computer Science, 1, 79 (2020).* doi:10.1007/s42979-020-0114-9

[3] Q. Li, W. Cai, X. Wang, Y. Zhou, D. D. Feng and M. Chen, "Medical image classification with convolutional neural network," *2014 13th International Conference on Control Automation Robotics & Vision (ICARCV), 2014*, pp. 844-848, doi: 10.1109/ICARCV.2014.7064414.

[4] H. Jiang and E. Learned-Miller, "Face Detection with the Faster R-CNN," 2017 *12th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2017), 2017*, pp. 650-657, doi: 10.1109/FG.2017.82.