Measuring the Effects of Applying OpenPose to Distracted Driver Detection

Carson Forsyth
George Mason University
cforsyt@gmu.edu

Michael McNamara
George Mason University
mmcnama@gmu.edu

1. Abstract

Detecting distracted drivers is an emerging field of study. According to the CDC Motor Vehicle Safety Division, over 3,000 people die in a crash involving a distracted driver every year.[3] State Farm has provided a data set and hosted a Kaggle Competition to see if dashboard cameras could effectively classify a driver's behavior while driving. In our testing, we have applied OpenPose [1, 2, 6, 10] to see if applying pose estimation to the images would result in a better-performing model than those that only use the stock-provided data set. Our results show that pose estimation was unable to beat our baseline results, which we believe is due to pose estimation being inconsistent. We implemented various custom CNN's and also a wide range of transfer learning from pre-trained models to ensure that this was not the result of poorly implemented models but rather due to OpenPose hindering our models performance rather than improving it.

2. Introduction

Over 3,000 people die every year from distracted driving. In 2020 there were 3,142 deaths caused by a distracted driver and over 2,880 fatal accidents involved a distracted driver. Resulting in a total of 8% of fatal accidents being due to a distracted driver.[3] Given these statistics, being able to detect distracted drivers would allow State Farm and other insurance companies to better insure and prevent drivers from engaging in distracted driving. Our aim was to score within the top 100 on the Kaggle leader board for the State Farm Distracted Driver Detection competition while using our modified data sets that we curated by applying OpenPose pose to our given data. We initially believed that applying pose estimation would make it easier for the model to identify the differences in the poses of the various distractinos or lack there of, however, we found that the inconsistencies of OpenPoses pose estimates led to a decrease in the performance of our various models.

We experimented with a variety of models such as custom CNN's and models that used transfer learning to achieve better results. We then compared the performance of models that were trained and evaluated on the datasets which had pose estimations with those that were trained and evaluated on the base dataset.

3. Data

The data set that we used was provided by the State Farm Kaggle competition. The training data is broken into 10 classes c0-c9, with c0 being safe driving and c1-c9 being forms of distracted driving. See table 1 for more information. All metadata such as creation dates has been removed and each driver can only appear in either the training or testing set. There are a total of 22,424 training images and a total of 79,726 testing images. We found it odd that there was such a significant difference in the size of the training and testing data. The effect of the imbalance between the train and test sets is that our models are very prone to over fitting due to the limited amount of data. Please view figures 2, 7 and 6 for samples from the dataset.

4. Related Work

The Kaggle user JACOBKIE described his implementation of his first-place model which achieved a score of 0.08739. He explains the unique challenges that this data set provides, first, the test set is much larger than the training set making it very easy to over fit. Second, the images are correlated since they are frames from a video thus images from the same video segment are part of the same classifi-

Class	Distraction	Count
c0	safe driving	2489
c1	texting - right	2267
c2	talking on the phone - right	2317
c3	texting - left	2346
c4	talking on the phone - left	2326
c5	operating the radio	2312
c6	drinking	2325
c7	reaching behind	2002
c8	hair and makeup	1911
c9	talking to passenger	2129

Table 1. Data Classes and Counts

cation category however when viewed as individual frames they may appear as if they are part of another class due to the subject being in motion. [9]

This user used a modified version of VGG-16 to achieve his first-place score. The modifications include using K nearest neighbor averaging on the last maxpool layer (pool5) with pretrained weights to map each test image to a 51,277-dimensional coordinate resulting in a weighted average of the predictions of each image. [9] With the 10 nearest neighbors in pool5's feature space, ensemble averaging was then applied to each class separately. Due to some models being able to confidently predict a single class while still performing poorly on the rest of the class. Each model that was used in the ensemble averaging was within the top 10 percent of the cross entropy loss of their respective class. [9] Finally the predictions for individual test images are divided into small groups according to their similarities in the 51277-dimensional space from the pool5-feature space described above. If all images in a single group portray consistent and confident predictions then this implies that all images in that group should be re-normalized to share the same predictions. [9]

5. Approach

We used OpenPose to apply pose estimation to our data set to attempt to outperform the common CNN-based models that are popular on the Kaggle leader board. There are two main parts of our implementation. First is our data set manipulation and second is our model. We initially had a lot of trouble trying to get OpenPose to work. This was due to our attempt to build it from source on a Linux-based operating system. We faced many issues with dependencies being mismatched and ultimately it was unable to view our OpenCV library and forced us to switch to a Windows-based OS and use the pre-built binaries. We used three separate commands to generate our our datasets. First we generated our "non-max accuracy" dataset which only produces a general skeleton of the subject. Next we generated our "max-accuracy" dataset which included all of the "non-max accuracy" estimations but added detection of fingers and faces. Finally we generated our "no-background" dataset which is the same pose estimations as those found in the "non-max accuracy" but with the original image stripped away and replaced with a black image. OpenPose allows for many flags to be used when running allowing for us to generate our 3 versions of the data set. We also found out how to accelerate our pose estimations through the use of disabling the rendering of the images upon estimation. We tested a wide variety of models across all 4 data sets and we will discusses our findings in the results section later on in this paper.

5.1. Custom CNNs

Our first attempt at creating a model was to design one from scratch and train it using the provided data. The CNNs that were tested mostly followed the template: convolution, pool, batch normalize until a low resolution, and then flatten and pass through a few dense layers, eventually ending up in a softmax activation that predicted the probabilities for each of the ten classes. Our thought process for these models was to convolve the images and decrease their resolution until a small enough resolution followed by flattening and passing through dense layers. After testing them and tweaking values, we found that these architectures were not performing well. Despite having a very high accuracy in the upper 90s, over fitting and not having high confidence in their predictions resulted in these models performing very poorly on the Kaggle submissions.
The first model, Custom CNN A table 2, is a very simple model of alternating 2D convolutional layers with max-pooling layers. This model was designed to get a baseline ranking in the competition and then improve from there. The next model was Custom CNN B table 3. This model added additional layers in the form of batch normalization. Also, the dense layers at the end were enlarged to have more connections. The hope was that increasing the number of parameters in these layers, it would allow for better predictions. The third and last of the custom models is Custom CNN C table 4. This one was similar to the second model, except that it has one less convolution and pooling layer. The rationale behind this model was that maybe the previous one was too complex and causing it to over fit. This rational led us to use a simpler model to help generalize. None of these models performed particularly well. Their exact rankings and performance will be discussed in the Results section.

5.2. Transfer Learning

Moving on from custom, trained-from-scratch models, we implemented and tested many different pretrained models through the use of transfer learning. We tested VGG-16, VGG-19, EfficientNet (multiple versions including b3, b4, b5 and b7) and ResNet152V2. The results of these models can be viewed in the results section of this paper. Implementing transfer learning allowed us to benefit from models that have taken weeks if not months to train on large distributed systems. These models have millions of trainable parameters that we would have no way of training due to a lack of time and hardware. Although these models are pretrained there is still a significant amount of time that needs to be dedicated to training the remaining dense and softmax layers of these implementations. Due to the amount of training data available to us, we still spend about 2-6 hours training each model even with a GPU. The pretrained mod-

els were all trained on the ImageNet data set that has become popular through its use in object detection training. For each model, the dense layers were removed and we add our own. This allowed us to leverage the pretrained convolution layers, but utilize our own design for the classification layers. The weights on the convolution layers were frozen so that they would not be altered as we fine-tuned the models using the competition data.

5.2.1 VGG

The VGG architecture was initially introduced in 2014 as part of the ImageNet classification competition. It is a relatively simple model compared to others. It is comprised of convolution layers, pooling layers, fully-connected layers, and use ReLU as its main activation function besides the last layer which uses softmax. VGG-16 is a model architecture with 16 weight layers, and the VGG-19 has 19 weight layers. This architecture along with its variants rose in popularity near the beginning of the deep learning revolution due to its effectiveness as well as its simplicity [7].

5.2.2 ResNet

The original Residual Network also appeared early on as deep learning became more popular. The network used special blocks called residual blocks that added the input tensor to the output of a layer a few steps farther into the architecture. This was implemented to help guard against the vanishing gradients issue that is more prevalent the more layers are added to an architecture. The ResNet152V2 architecture has a total of 152 layers made up mostly of 3x3 convolutional layers. and 1x1 layers to change the number of filters [5].

5.2.3 EfficientNet

EfficientNet is a very interesting set of architectures that is thoroughly planned out to increase model complexity to fit a specific budget in terms of resources such as memory. The main component of the family of architectures is a special residual block called an MBConvolution or Inverted Residual Block. The EfficientNets are designed to be more lightweight than other state-of-the-art architectures and still beat them in terms of performance. There are eight total variants in the EfficientNet family, B0 to B7. Each increasing value has a more complex model and thus a higher number of parameters. B7 is the most complex model but also performs the best on the ImageNet benchmark [8].

6. Results

The leaderboard positions of the Kaggle competition are defined by the use of multi-class logarithmic loss. Each image in the test set must have a set of predicted probabilities. The log loss is defined below.

$$logloss = -\frac{1}{N}\sum_{i=1} N \sum_{j=1} M y_{i_j} log(p_{i_j})$$

N is the number of images in the test set while M is the number of class labels. It is important to note that the natural log is being used throughout this equation. y_{i_j} is 1 in the case that i belongs to class j and 0 otherwise. p_{i_j} is the predicted probability that observation i belongs to class j.[4] Our goal was to achieve a leaderboard position within the top 100 with any of our new data sets. However, we were unable to do so, we found that all versions of our pose estimated data sets were incapable of outperforming the base data set no matter which model was applied

Layer	Filters	Kernel Size	Stride
Conv2D	32	5	1
MaxPool2D	-	2	1
Conv2D	64	5	1
MaxPool2D	-	2	1
Conv2D	64	3	1
MaxPool2D	-	2	1
Conv2D	128	3	1
MaxPool2D	-	2	1
Conv2D	128	3	1
MaxPool2D	-	2	1
Conv2D	256	3	1
MaxPool2D	-	2	1
Flatten	-	-	-
Dense	64	-	-
Dense	64	-	-
Dense	10	-	-

Table 2. Custom CNN A

Layer	Filters	Kernel Size	Stride
Conv2D	32	5	1
MaxPool2D	-	2	1
Conv2D	32	3	1
BatchNorm	-	-	-
MaxPool2D	-	2	1
Conv2D	64	3	1
MaxPool2D	-	2	1
Conv2D	64	3	1
BatchNorm	-	-	-
MaxPool2D	-	2	1
Conv2D	128	3	1
MaxPool2D	-	2	1
Conv2D	128	3	1
BatchNorm	-	-	-
MaxPool2D	-	2	1
Flatten	-	-	-
Dense	512	-	-
Dense	128	-	-
Dense	10	-	-

Table 3. Custom CNN B

to it. We also observed very odd behavior during the testing of our VGG-16 transfer learning model. We found that although it was our best-performing model on our "non-max accuracy" and "max-accuracy" datasets, it had training accuracy's that were below 20% during training. Figure 1 contains the loss and accuracy graph for this model. This was reproduced with both the max-accuracy data set and also with the non-maximum accuracy data set. We also were applying random rotation and random zoom for data augmentation When training. This model achieved a score on Kaggle of 2.30224 which lands at 1197 on the Kaggle leaderboard.

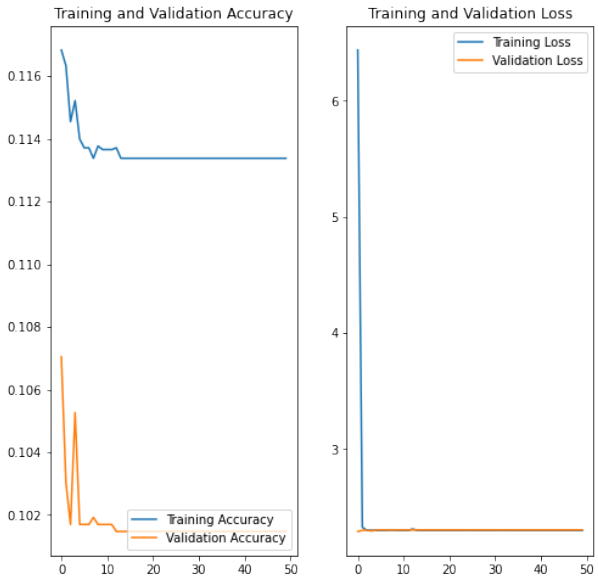

Figure 1. accuracy and loss during training

We found it very odd how the accuracy dropped during training and the loss was near 0 from the very beginning. We were able to repeat this behavior many times and have no explanation as to why it's occurring or why it performs relatively well on Kaggle.

When comparing the best model's performance on each of the data sets in Figure 5 there is a very large gap between the performance of our best model on the base data set and the performances of the best models on all of the other data sets. We believe there is such a large discrepancy between the performances because of the inconsistent pose estimations that OpenPose applies to the images. For example in figures 2 and 3 below are two poses for class 0 with no background.

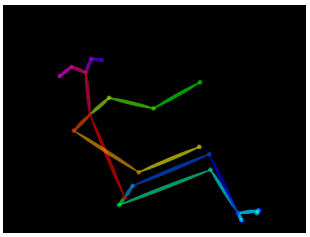

Figure 2. image 1 of class 0

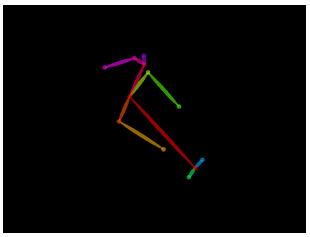

Figure 3. image 2 of class 0

As seen above the two images are extremely different, the inconsistencies with OpenPose in our opinion make it more difficult for the model to correctly classify whether or not someone is distracted with confidence. The majority of our models are still able to correctly classify with over 95% accuracy however their confidence when doing so decreases. These inconsistencies are not limited to the data set which has no background, it is apparent in both the max accuracy and non-max accuracy data sets as well.

Layer	Filters	Kernel Size	Stride
Conv2D	32	3	1
MaxPool2D	-	2	1
Conv2D	32	3	1
BatchNorm	-	-	-
MaxPool2D	-	2	1
Conv2D	64	3	1
MaxPool2D	-	2	1
Conv2D	64	3	1
BatchNorm	-	-	-
MaxPool2D	-	2	1
Conv2D	128	3	1
MaxPool2D	-	2	1
Flatten	-	-	-
Dense	512	-	-
Dense	128	-	-
Dense	10	-	-

Table 4. Custom CNN C

6.1. Custom CNNs

We implemented 3 separate CNNs before moving on to transfer learning. The three CNNs that we created are detailed above and can be referenced in tables 2,3, and 4. Out of the three CNNs CNN-B performed the best scoring 2.54461, followed by CNN-A and CNN-C. Their results can be referenced below in Figure 4.

These CNNs performed relatively poorly in comparison to our transfer learning results detailed below. Scoring a whole 2.0 higher using the log loss defined above than the

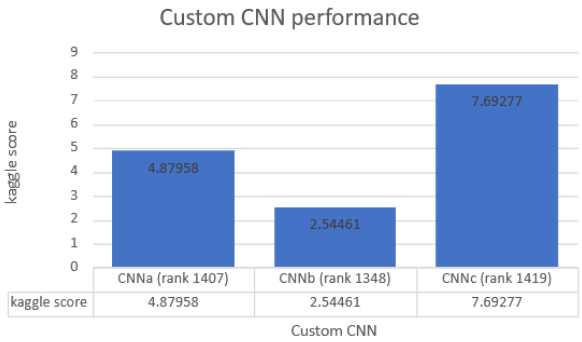

Figure 4. Comparison of Custom CNN performance

transfer learning did on the base data set. This leads us to shift our focus to transfer learning due to showing the most potential early on.

6.2. Transfer Learning

Below in figure 5 are the results of our best-performing models on each of the data sets.

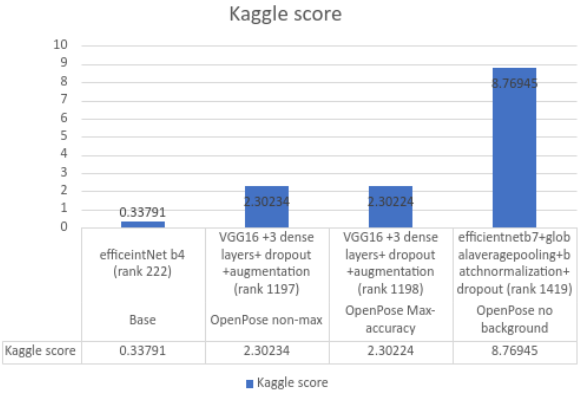

Figure 5. Comparison of best-performing models on each dataset

We found that different implementations of transfer learning were the best-performing models across all data sets. As seen in figure 5 EfficientNet and VGG-16 resulted in the lowest log loss across the baseline, non-max accuracy, max accuracy, and no background data sets. EfficientNetB4 was the best performing on the base data set while EfficientNetB7 (with slight modifications with the addition of global average pooling, batch normalization, and dropout) was the best performing on the no background data set. VGG-16 was the best performing on the max accuracy and non-max accuracy data sets (with slight modifications adding dense layers along with dropout and data augmentation to help with the over-fitting). We think one of the reasons that the transfer learning models perform so much better than the custom models is that the transfer learning models are pretrained on ImageNet and then fine-tuned on the Kaggle competition, whereas the custom models are only trained on the competition data. This means the pretraining on ImageNet allows the models to learn key patterns and then apply that feature learning to detect distracted driving.

6.3. Original Images

As shown in figure 5, the models trained on the original data set performed much better than the other data sets. The model architecture that performed the best was the EfficientNetB4. The full list of results of our models on the original data is shown in table 5, which shows the architecture, the score received from Kaggle upon submission, and the ranking of that score. The EfficientNetB4 model scored 0.33791 which achieved a top 15% rank with position 222 out of 1438. Our custom CNN models performed quite poorly with the best one achieving a score of 2.54461 at rank 1348 (top 93%).

6.4. Non-Max vs Max Accuracy

We were surprised by how close the performance between the non-max accuracy and max-accuracy data sets was with our VGG-16 implementation of transfer learning. A possible reason the non-max and max accuracy data sets resulted in such similar values is the difference of having individual fingers and faces identified by OpenPose did not carry much weight and were not learned much by the model. In addition, it is possible that not enough of the images actually had the driver's fingers and faces identified by OpenPose resulting in the model not considering it important. Our results for both of these datasets can be found in tables 6 for nonmax accuracy results and table 7 for max accuracy results. One interesting note about one of the results from the non max accuracy dataset is that we achieved a new highest (lowest performing) score on Kaggle with a score of 15.59965 with a transfer learning approach of EfficientNetB7 landing it in last place with a ranking of 1439. This model was trained for 50 epochs and was severely over fit.

6.5. No Background

We were not surprised by how poorly the data set with no background performed because of how much information is lost by removing the original image and only keeping the inconsistent OpenPose estimations. By removing any context from the estimated skeletons the model loses key information about the driver such as a device they could be holding or where their heads are looking. The arms and legs, if they were detected, gave very little information besides where they were located. The legs give almost no information since leg positions do not usually correlate strongly with distracted driving. The arms do give some information about distracted driving but can be confusing without the

ability to detect objects in the hands such as phones. The heads of the skeletons make it very difficult to detect which way they are facing and so it would be hard for the model to detect if they are looking straight ahead or if they have turned their head.

The model that performed the best on this data was a modified version of the EfficientNetB7 with global average pooling, batch normalization, and dropout. This model received a Kaggle score of 8.76945 which would rank it at position 1419 out of 1438 (Top 98%) as shown in figure 5. Interestingly we were able to perform even worse than our worst model on the non max accuracy dataset. We managed to get a score of 31.29878 which is almost twice as bad as our previous worse score. This horrible performance was achieved with VGG 16 with 3 dense layers + 1 dropout layer for each and data augmentation in the form of random rotation and zoom. This shows how poorly this combination of data and model selection performed compared to the other data sets. This is why we believe removing the background is not a good choice for data prepossessing for this problem domain. For further results from this dataset reference table 8.

7. Conclusion

This project was very interesting and allowed us to learn a lot about applying a combination of deep learning and computer vision to try and improve upon existing results for classifying distracted drivers with high confidence. We hoped the use of OpenPose on the provided data set would allow for increased classification accuracy with high confidence. However, we were unable to be able to improve upon existing results. Ultimately we were only able to achieve competitive results and rank on the Kaggle leaderboard when the original data set was used, any modifications made to the data set made through the use of OpenPose resulted in a decrease in performance.

We believe that future works combining the use of OpenPose with some existing works that segment and crop the image around an object such as a phone or cropping the region around the head may remove some of the inconsistencies that OpenPose exhibited during our testing. Using advanced techniques like cropping areas of the image that are of high interest may result in a significant boost in performance.

Furthermore incorporating the use of ensemble methods showed a lot of promise amongst the other Kaggle competitors so it may be possible to use OpenPose in conjunction with the ensemble methods described by others to potentially develop a competitive model.

References

[1] Z. Cao, G. Hidalgo Martinez, T. Simon, S. Wei, and Y. A. Sheikh. Openpose: Realtime multi-person 2d pose estimation using part affinity fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.

[2] Z. Cao, T. Simon, S.-E. Wei, and Y. Sheikh. Realtime multi-person 2d pose estimation using part affinity fields. In *CVPR*, 2017.

[3] B. S. J. Christy Bieber, J.D. Distracted driving statistics facts in 2023, 2023. ECCV06 submission ID 324. Supplied as additional material `eccv06.pdf`.

[4] S. F. K. Competition. State farm distracted driver detection evaluation, 2016. https://www.kaggle.com/competitions/state-farm-distracted-driver-detection/overview/evaluation.

[5] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition, 2015.

[6] T. Simon, H. Joo, I. Matthews, and Y. Sheikh. Hand keypoint detection in single images using multiview bootstrapping. In *CVPR*, 2017.

[7] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[8] M. Tan and Q. V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks, 2020.

[9] K. user Jacobkie. A brief summary, 2016. https://www.kaggle.com/competitions/state-farm-distracted-driver-detection/discussion/22906.

[10] S.-E. Wei, V. Ramakrishna, T. Kanade, and Y. Sheikh. Convolutional pose machines. In *CVPR*, 2016.

8. Appendix

Figure 6. non max sample

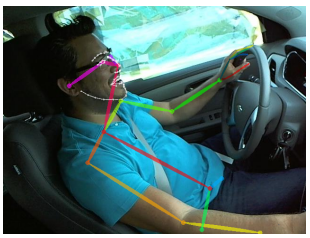

Figure 7. max sample

Architecture	Kaggle score	Rank
EfficientNetB4	0.33791	222
EfficientNetB4 + 1 dense Layer	0.38913	256
EfficientNetB5	0.40516	264
EfficientNetB3	0.43549	281
ResNet152v2	0.89533	495
VGG-16 + 4 dense layers	1.13135	579
VGG-16 + 3 dense layers	1.18985	599
VGG-19	1.48117	713
VGG-16	1.70411	989
VGG-16 + 6 dense layers	1.78596	1078
Custom CNN B	2.54461	1348
ResNet50 + 2 dense	2.69283	1362
Custom CNN A	4.87958	1407
Custom CNN C	7.69277	1419

Table 5. Results on Original Images

Architecture with non max accuracy	Kaggle score	Rank
VGG-16+ 3 dense layers + random rotation and random zoom	3.7477	1395
EfficientNetB3+ global average pooling batch normalization + dropout	8.78428	1419
EfficientNetB7+ global average pooling batch normalization + dropout+ 10 epochs	10.0002	1422
EfficientNetB7+ global average pooling batch normalization + dropout + 50 epochs	15.59965	1439
VGG-16 + 3 dense + 3 dropout + augmentation	2.30234	1198

Table 6. Results on non-max accuracy data set

Architecture with max accuracy	Kaggle score	Rank
EfficientNetB7 early stopping	12.85355	1423
EfficientNetB7+ global average pooling batch normalization + dropout+ 10 epochs	9.48186	1421
VGG-16 + 3 dense + 3 dropout + augmentation	2.30234	1198

Table 7. Results on max accuracy data set

Architecture with no background	Kaggle score	Rank
EfficientNetB7++ global average pooling+ batch normalization + dropout + early stopping	16.19463	1427
EfficientNetB7 + global average pooling batch normalization + dropout + 10 epochs	8.76945	1419
VGG-16 + 3 dense + 3 dropout + augmentation + 34 epochs	31.29878	1439

Table 8. Results on no background data set