Eyes on the Road: Vision Transformers for Multiclass Distracted Driver Detection

Justin Lee
Stanford University
justinl8@stanford.edu

Aaron Li
Stanford University
aaronli9@stanford.edu

A. Abstract

In this paper, we introduce EyesOnTheRoad: a vision transformers(ViT) that performs multiclass distracted driver detection. We also explore explainability in AI by exploring methods that demystify deep learning models.

B. Introduction

Mercedes made a storm last month with the recent unveiling of the first conditional self-driving system approved for European public roads. Level 3 vehicles, as defined by SAE, have "environmental detection" capabilities and can make informed decisions for themselves. But—they still require a human override. The driver must remain alert and ready to take control if the system is unable to execute the task. Therefore, there needs a way to identify drivers' attention level and their ability to take back control of the vehicle. As headlines had shown countless examples of accidents in Tesla vehicle stems from drivers' complacency when using the company's advanced driver assistance system (Level 2 self-driving). A monitoring mechanism has to be set in place to monitor driver attentiveness while using conditional self-driving features.

The objective of this project is to apply state-of-the-art CNN and Vision Transformer on distracted driver detection. Conditional self-driving still relies on the driver paying attention as a failsafe, so accurate detection is important for self-driving car operation. Further, the project has commercial applications because delivery drivers often drive long hours and can easily get distracted at some point during that time, meaning reducing distracted driving saves lives and reduces the failure rate of package delivery.

In this project, we implement data augmentation techniques, train our pre-processed data on state-of-the-art deep learning models, and lastly implement explainability methods to help further understand and answer the black box of deep learning models.

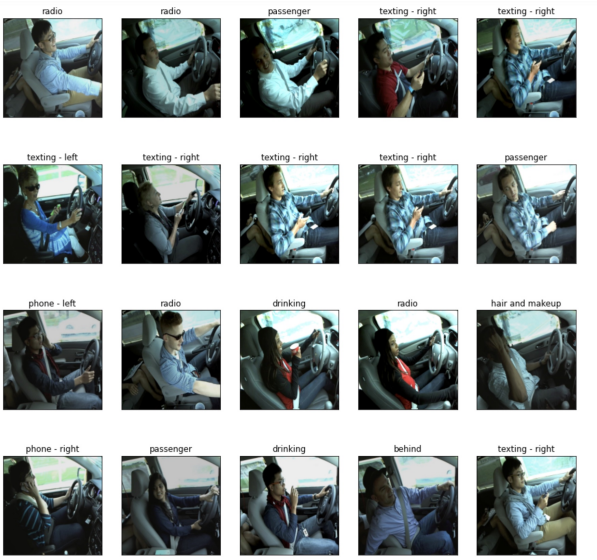


Figure 1. Distracted Driver Training Dataset

C. Dataset and Features

The Kaggle distracted driving dataset consists of 22422 training images, and about 79000 testing images. However, as the labeling of Kaggle's testing dataset was not accessible due to this being a Kaggle competition, we only had the training images to work with. The images given by Kaggle are colored and are 640 x 480 pixels in resolution. The image consists of a driver behind the wheel, and the network must assign percentages to one of ten classes predicting what the driver is doing (driving safely, texting right, texting left, talking on phone right, talking on phone left, operating radio, drinking, reaching behind, doing hair/makeup, or talking to a passenger). The images were taken with 26 unique subjects in different cars, each image is 640 x 480 pixels. A representation of the dataset could be found in figure 1.

To preprocess the data, we first set aside 1/5 of these images (about 4000) to be purely part of our own test set,

and we did not touch or look at these images. For VGGNet, we normalized our training data to be 224 pixels x 224, and for vision transformers, we normalized our training data to be 64 pixels x 64. Also, we divided all input values by 255 so that instead of RGB values, each pixel would have an input value between 0 and 1.

Intending to create a more accurate while reducing overfitting during training, we have implemented a few data augmentation techniques to increase the amount of data by adding slightly modified copies of already existing data or newly created synthetic data from existing data. Some of the data augmentation techniques we used are random translation, zooming in and out, and color jitter. We did not apply rotation or inversion because some classes within our dataset are directional sensitive as the images would be classified as texting-right and texting-left for example.

D. Related Work

See the references section for bibliographies of related work. In their paper, "Real-time Distracted Driver Posture Classification" [1], Abouelnaga et. al. train ensemble CNN models on the same Statefarm distracted driver dataset that we are working on. Their methods inspire us to further improve our training because they found a way to find more images to train on. For their model, they trained neural networks to be able to separately detect hands, and separately detect faces. These distinctions helped because texting or calling, for example, can be identified by seeing whether a person has a phone in hand, and where a person is looking can be seen on their face. The researchers combined their networks for hand images and the networks for facial images into an ensemble classifier that would predict the original dataset with a softmax loss function.

In their work "Autoencoder Based Inter-Vehicle Generalization for In-Cabin Occupant Classification" [2], Dias Da Cruz et. al. use an auto-encoder based convolutional neural network architecture on datasets of car interiors (specifically, backseats of cars that may or may not contain passengers) to identify whether passengers are infants, children, adults, or if the seat is occupied by baggage. Such classification is similar to our project because it uses car interior datasets and must identify people as well.

In their work "Very Deep Convolutional Networks for Large-Scale Image Recognition", Simonyan and Zisserman present VGGNet, a convolutional neural network architecture they used to perform well in the ImageNet classification challenge. Our initial attempt at training the Statefarm dataset used VGGNet, as we thought this paper's performance training and evaluation methods could help our project.

Transformers [3] first found their initial applications in natural language processing (NLP) tasks, as demonstrated by language models such as BERT and GPT-3. By contrast, the typical image processing system uses a convolutional neural network (CNN). Well-known projects include Xception, ResNet, EfficientNet, DenseNet, and Inception.

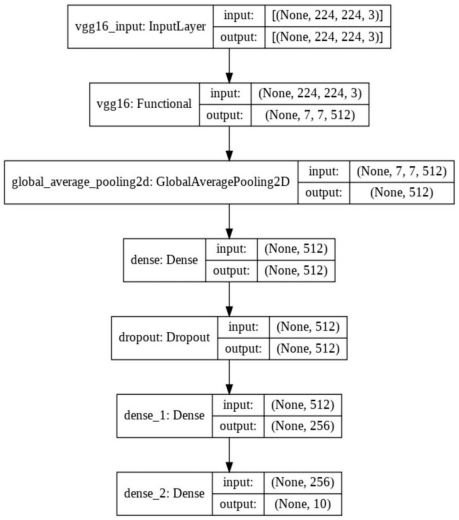


Figure 2. Our VGG Net Architecture

Transformers measure the relationships between pairs of input tokens (words in the case of text strings), termed attention. The cost is exponential with the number of tokens. For images, the basic unit of analysis is the pixel. However, computing relationships for every pixel pair in a typical image is prohibitive in terms of memory and computation. Instead, ViT computes relationships among pixels in various small sections of the image (e.g., 16x16 pixels), at a drastically reduced cost. The sections (with positional embeddings) are placed in a sequence. The embeddings are learnable vectors. Each section is arranged into a linear sequence and multiplied by the embedding matrix.

E. Methods and Models

After the data augmentation steps performed in Section C, we have set to train the pre-processed data on both of the following architectures described below.

E.1. VGGNet

We are using VGGNet as our baseline model. VGG is an innovative object-recognition model that supports up to 19 layers. Built as a deep CNN, VGG also outperforms baselines on many tasks and datasets outside of ImageNet. VGG relies on using small filters, specifically 3x3 convolutional layers, the smallest unit that allows filters to still capture a notion of surroundings in all directions, as well as some 1x1 convolutional filters that act as linear transformations.

Our VGGNet achieved a training accuracy of 87.94 and

a training loss of 5.012 (see Table 2). We did not end up testing our VGGNet on the test set.

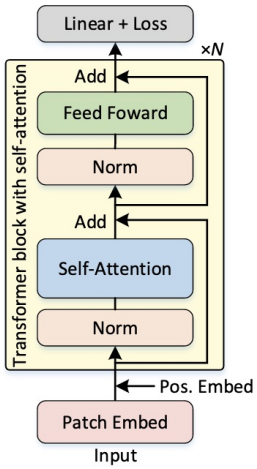


Figure 3. Vision Transformer (ViT) Architecture

E.2. Vision Transformers (ViT)

E.2.1 Overview of Vision Transformers (ViT)

Figure 4. ViT Patch Generation

A vision transformer model is composed of three main components, a linear layer for patch embedding, a stack of transformer blocks with multi-head self-attention and feed-forward layers for feature encoding, and a linear layer for classification score predictions, as depicted in Fig 3.

Vision Transformer adapts the transformer architecture to process 2D images with minimal changes. In particular, ViT extracts N non-overlapping image patches, $x \in \mathbb{R}^{H\times W\times C}$, and turn it into a sequence of patches $x_p \in \mathbb{R}^{N\times P\times P\times C}$, where (H, W) is the height and Width of the original image, C is the number of channels, (P, P) is the resolution of each image patch, and $N = HW/P^2$ is the resulting number of patches, which also serves as the effective input sequence length for the Transformer.

Hyperparameter	Value
transformer_layers	6
patch_size Net	4
hidden_size	64
num_heads	4
mlp_dim	128

Table 1. Hyperparameter tuning of Vision Transformer

In order to optimize our vision transformer on the log loss function, we used AdaGrad optimization. Adagrad is particularly good for image recognition because images often have sparse features that don't get updated so much during gradient descent. The AdaGrad algorithm specifically prioritizes giving more updates in directions that have had few updates by making those learning rates bigger, as opposed to stochastic or batch gradient descent, which simply keeps one learning rate and updates weights based on the steepest gradient values. This difference allows AdaGrad to be a very efficient optimizer.

$$\text{Adagrad} \Rightarrow \mathrm{w_t} = \mathrm{w_{t-1}} - \eta_t^{'}\frac{\partial \mathrm{L}}{\partial \mathrm{w_{t-1}}}$$

$$\text{where } \eta_t^{'} = \frac{\eta}{\sqrt{\alpha_t + \varepsilon}}$$

$$\varepsilon \text{ is a small } +\text{ve number to avoid divisibilty by } 0$$

$$\alpha_t = \sum_{i=1}^{t}(\frac{\partial \mathrm{L}}{\partial \mathrm{w_{t-1}}})^2 \quad \textit{summation of gradient square}$$

Figure 5. AdaGrad Optimizer Equation

F. Results and Discussion

F.1. Cross-Validation and Hyperparameter Tuning

Before trying our model, we implemented 10-fold stratified cross-validation on the training data (the images we did not set aside to be test data) in order to create validation sets, thus allowing us to tune hyperparameters. Stratified 10-fold cross-validation works by ensuring that each of the folds has about the same number of images corresponding to each of the possible labels, addressing any data imbalances. After tuning our hyperparameters with cross-validation, we found that the best hyperparameters are as shown in Table 1.

Classifier	Training Acc.	Testing Acc.	Loss
MobileNet	72.45	-	8.23
VGG 16 Net	87.94	-	5.012
Vision Transformer	93.45	92.01	0.28

Table 2. Training, Testing and loss accuracies of different models

F.2. Evaluation Metrics: Accuracy/Log Loss on Test Set

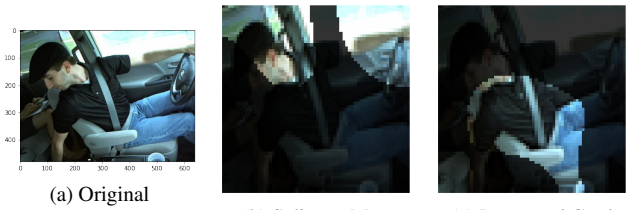
After tuning our hyperparameters and training our vision transformer model, we wanted to gain insights into what our model did well and what could be improved. As metrics to decide this, we decided to use test accuracy (accuracy on the validation images that we had set aside from the original training data), as well as log loss on those validation images.

Our first model was evaluated on the log loss function shown on Figure 7.

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

Figure 6. Formula for multi-class logarithmic loss

At the end of training our fully tuned vision transformer for 50 epochs on the training set, we achieved a test accuracy of 92.01 and a log loss of 0.28 on our test set (see Table 2).



(a) Original (b) Saliency Map (c) Integrated Grad

Figure 7. Three different masking on driver looking behind

(a) Original (b) Saliency Map (c) Integrated Grad

Figure 8. Three different masking on driver texting

F.3. Result Visualization: Truelens

Trulens [4] is a deep learning library that exposes which specific parts of an input a neural network is looking at when it makes a prediction. In our case, we didn't want to just see an accuracy and loss; we wanted to see which pixels in images were helping our vision transformers make a decision and output a class. Within Truelens, there were two methods that we used: mask visualization and integrated gradients. Below, we explain each method and show an example of each:

F.3.1 Attribution Methods

[] Attribution methods allow us to quantify the contribution of particular variables in a model towards a particular behavior of the model. In many cases, for example, this may simply measure the effect each input variable has on the output of the network. The first methods that we will examine directly measure the importance of input features on the classification of the model. These are the most straightforward type of attributions to compute, as there are relatively few choices to make.

F.3.2 Example of Saliency Map

[] Saliency maps take the gradient of the network's predicted class on a given input. To visualize the attributions, we use MaskVisualizer from the Trulens visualizations module. The MaskVisualizer class takes a blur and threshold argument and allows us to overlay a partially-opaque mask over a given image that reveals the top-threshold percentage of pixels by attribution, after applying a Gaussian blur of the given radius.

F.3.3 Example of Integrated Gradient

Turning to Integrated Gradients, the workflow for obtaining attributions is nearly identical. The only difference is that the AttributionMethod instance we construct. It is of IntegratedGradients rather than InputGradients.

F.3.4 Analysis on two attribution methods

Visualizing the results in Figure. 8 and Figure. 9 it is apparent that Integrated Gradients in this case are better able to focus on the pixels corresponding class of concern. In figure 8, the Integrated Gradient can highlight the body of the driver turning backward while in Fig. 9 it was able to highlight the driver's hand and phone, explaining how the model makes its decision for classification based on the highlighted features. Integrated gradients are superior because it can address some issues with saliency maps where it reflect the importance of edges more accurately than vanilla

saliency maps because they capture the gradient information in a relative global manner.

F.4. Confusion Matrix

In classification problems, confusion matrices are a good way to see which classes a model is predicting well and what the model has trouble with.

In figure 10, we have our confusion matrix of our fully trained vision transformer model. The labels on the bottom axis represent the true labels, and the labels on the left axis represent our transformer's predicted labels. Our model mostly predicted well across the board; however, there are a couple of classes that drew our attention. The model had trouble differentiating between texting left and phone (calling) left; there were 18 images in which the driver was texting on the left side, but the model predicted calling left, and there were 24 images in which the driver was calling on the left side, but the model predicted texting left. Such a prediction mistake makes sense because both calling and texting involve holding a phone, so a model could easily mix those up. However, the model doesn't make nearly as many mistakes between texting on the right side and calling on the right side, which leads us to believe that the mistake isn't based on both texting and calling involving a phone, but the distance and camera angle. Since the images are all taken from the center of the front dash, the driver's left side is farther from the camera and more on the edge of the image than the driver's right side. We think this creates more mistakes because, on the left side, the details of what a driver is doing with a phone are obscured by other parts of the image, such as the driver's head. This hypothesis is supported by the fact that the model mainly incorrectly predicted safe driving with the phone left and texting left classes, suggesting that any phone actions on the left side could sometimes be obscured.

F.5. Classification Report

In figure 11, we have the classification report. Key evaluation metrics here are precision, recall, and F1 score for each of the 10 classes. Precision tells us the amount of true positive predictions for a class, divided by the number of total positive predictions for a class. For example, a precision of .9 in the normal driving class means that 90% of the time the model predicted a driver was driving safely, the model is correct. The other 10% of the time, the driver isn't driving safely. Recall tells us the amount of true positive predictions for a class, divided by the number of true positive predictions for a class plus the number of false negatives (times when something should have been predicted as a part of the class but got predicted as something else). Finally, F1 score is determined using the following formula:

$$\frac{\text{precision} * \text{recall}}{\text{precision} + \text{recall}} \quad (1)$$

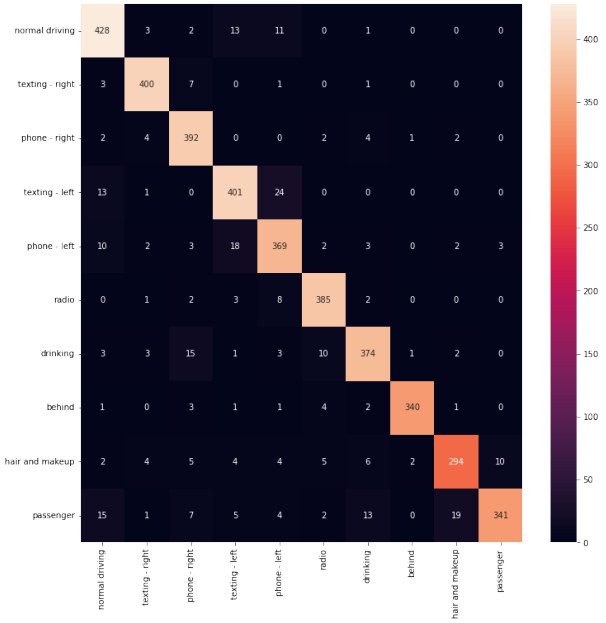


Figure 9. Formula for multi-class logarithmic loss

Classification Report:

	precision	recall	f1-score	support
normal driving	0.90	0.93	0.92	458
texting - right	0.95	0.97	0.96	412
phone - right	0.90	0.96	0.93	407
texting - left	0.90	0.91	0.91	439
phone - left	0.87	0.90	0.88	412
radio	0.94	0.96	0.95	401
drinking	0.92	0.91	0.91	412
behind	0.99	0.96	0.98	353
hair and makeup	0.92	0.88	0.90	336
passenger	0.96	0.84	0.90	407
accuracy			0.92	4037
macro avg	0.92	0.92	0.92	4037
weighted avg	0.92	0.92	0.92	4037

Figure 10. Classification Report

Seeing that all classes have an F1 score of around .9 is an alternative evaluation to the .9 accuracy that our model achieved.

We wanted to separate these statistics for each class to get insights into which classes are easier or harder for our model. It seems the model can easily tell when a driver is looking behind, achieving a .98 F1 score in that class. That makes sense because a driver looking behind is such a big movement, so it's not hard to tell. However, a class like a phone - left is harder for the model, with a .88 F1 score in that class. This also makes sense, because having a phone out is more subtle, and can be confusing with other classes such as phone-right and texting - left.

G. Conclusion/Future Work

In conclusion, we successfully implemented a Vision Transformer(ViT) with a solid performance on the distracted driver detection task. At 50 epoch, it was able to

achieve a test accuracy of 92.01% and also a loss value of 0.28, beating MobileNet and VGG16 Net by a margin. Vision Transformer, while new, has many advantages in classification tasks compared to traditional machine learning techniques.

Moving forward, we could utilize the remaining 70,000 test images without labels on a semi-supervised learning method, so that we could utilize some of those images for training, increasing data robustness.

Furthermore, we could improve our data augmentation, preprocessing, and explainability techniques. We found that some classes were giving the model more trouble than others, plausibly because the distinguishing features were being obscured by other, less relevant parts of the image. However, to prove such a theory, we would need a better Trulens visualization that allows us to see whether there are images where the deciding feature for prediction was off due to the true feature being obscured. If we can prove that, then preprocessing data to reduce the obscurity of key features will allow the model to perform even better.

H. Contributions

Aaron was responsible for hyperparameter tuning, cross validation, and data preprocessing/augmentation. Justin was responsible for the VGGNet and vision transformers models, as well as Trulens data visualization. The paper was written jointly by Justin and Aaron. We did not get help from non-CS231N students.

References

[1] Yehya Abouelnaga, Hesham M. Eraqi, and Mohamed N. Moustafa. Real-time distracted driver posture classification, 2017. 6

[2] Steve Dias Da Cruz, Oliver Wasenmüller, Hans-Peter Beise, Thomas Stifter, and Didier Stricker. Sviro: Synthetic vehicle interior rear seat occupancy dataset and benchmark, 2020. 6

[3] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2020. 2

[4] Klas Leino, Rshih32, Divya Gopinath, , Anupam, Shayaks, MacKlinkachorn, and Caleblutru. truera/trulens: Trulens, 2021. 4

[5] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015. 6

[6] R Zaheer and H Shaziya. A study of the optimization algorithms in deep learning, 2019. 6

[1] [2] [5] [6]