Road Sign Detection Through Transfer Learning

Matthew Gonzalez (mig42), Samuel Qian (scq5), Rachel Yue (ry263)

May 2023

Link to Presentation: Presentation
Link to GitHub Repo: Road-Sign-Detection

AI Keywords:

Computer Vision, Transfer Learning, Faster-RCNN, Data Augmentation

Application Setting:

Our project is designed to classify four major road signs: a traffic light, a stop sign, a crosswalk sign, and a speed limit sign. The data set used in training and testing is comprised of photos taken from European streets, and as there are some slight differences between road signs in the United States and Europe, the main application setting for our project is roads in Europe.

Part 1: Project Description

The Plan

When brainstorming ideas for this project, we were ambitious: we wanted to create a computer vision program that could take an input video feed or an image, detect what road signs were present, and alert the driver so that they could take the expected actions. Ideally, we wanted our road signs to be detectable in many different countries around the world, regardless of language and specific designs for the road signs, as long as they followed a specific general design principle. Now, these were enthusiastic goals, but at the core of what we really wanted was an application that could correctly and quickly classify road signs when they were present (i.e. only recognize signs when there were signs present and not detect any when there were no signs). We wanted this because this would be a useful application that could actually be used in practice.

The Execution (Attempt 1)

Our initial approach to creating this application was to start small, focusing only on images instead of both images and videos and using road signs from European countries instead of road signs all over the world. Later on, we hoped to add on these additional features once we got a high quality model completed on this simplified task. Using this simplified goal we started developing our road sign detector in what we thought was the best way to go about it, first creating a classifier that, given an image centered and framed around a road sign, would output the type of road sign that was in the image, and then creating a detector that would detect when an image had a road sign in it and crop the image around the road sign. We then planned to pass the initial image through the detector, and if it found a road sign in the image, take the cropped image that was output from the detector and input it into the classifier to get the final output.

We spent the first half of the semester working with this approach, getting as far as having a fully implemented classifier that worked correctly classified road signs with a 95% test accuracy and the beginnings of a road sign detector; however, after receiving feedback from the mid-semester status report, we realized that our plan was flawed and what we should have done was do both detection and classification in one step. Using the advice of our project advisor, we turned to transfer learning and Faster-RCNNs to implement our idea.

The Execution (Attempt 2)

The first part of this new plan was preprocessing the data. The data we were using came in two parts: the images themselves and .xml files containing all of the metadata for the images, including the classification of each image and the location of each road sign in the image. So, using this we created a dataframe to be more accessible and usable place to store the data. Then, since all of the data that we used contained road signs in them, we created a collection of images that had no road signs in it, as part of our goal was to be able to detect when there are no road signs in an image. We then merged the original data with the new no-road-sign data to create our complete dataset to which we would train and test on.

After preprocessing, we started working on the detector. Since we were doing transfer learning, our first step was to find a pretrained model and use that architecture and its trained weights as a convolutional base in our detector. After some research, we found two architectures that we thought would work well: VGG16 and ResNet50. Later on in our finetuning, we found that we had a better validation accuracy and test accuracy using the ResNet50 architecture as a convolutional base. For both of these architectures, we used the model that was trained on the imagenet dataset and used those precalculated weights in the convolutional base. After splitting the data into training, validation, and testing sets, we ran each set on this pretrained model to extract the features from the images.

Then, we implemented a classifier that took in the features output by the convolutional base and the labels and output the classification of the image. In creating this classifier, we used the Adam Optimizer to train the model and for a loss function, we used a custom cross entropy loss as we wanted to more heavily penalize classifying no sign when there was indeed a sign in the image and this loss function allowed us to more harshly punish this type of misclassification.

With all of the computer vision parts completed, we added a small user interface to make using the application more pleasant and deployable.

Key Aspects of AI Used

There are numerous key aspects of artificial intelligence that are crucial in the implementation of our project. First, and perhaps the most important for this project, is transfer learning. While this project could have been done in a lot more time and a lot more effort using a model from scratch, our project is built around the concept of transfer learning. We take a model that was previously trained on the imagenet data set and use its weights and architecture as the convolutional base of our feature detection model. Using transfer learning allowed us to save time training while also likely providing better performance, as our version of ResNet50 was trained using millions of images for data while our data set only contains about one thousand five hundred images. To get this pretrained model to work effectively on our data set, we used another key aspect of ai: fine-tuning.

Closely related to fine-tuning and transfer learning, another crucial feature of artificial intelligence that we included in our application was deep learning and neural networks, in this case specifically, Faster-RCNNs and Multi-Layer Perceptrons. As part of transfer learning, the pretrained model that we used was a version of Faster-RCNN and we used this to extract features from the images. We then took these features and ran them through our own Multi-Layer Perceptron. So, we had two instances of deep learning in our project.

Finally, we utilized data augmentation in our project. As our project was overfitting and our dataset was relatively small, we wanted to combat these problems by generating new images to train on with slight variations to make our model more representative of the true underlying distribution.

Part 2: Evaluation

When looked at purely from the perspective of the ambitious goal we set at the beginning of the semester, this project was a failure. We wanted to be able to input a dashboard video of a car driving and be able to detect road signs and alert the driver on what to do in real time. We say this is failure because our project cannot even take in a video; it only works on images. That being said, our project still accomplished a lot and there is still much to evaluate our project on.

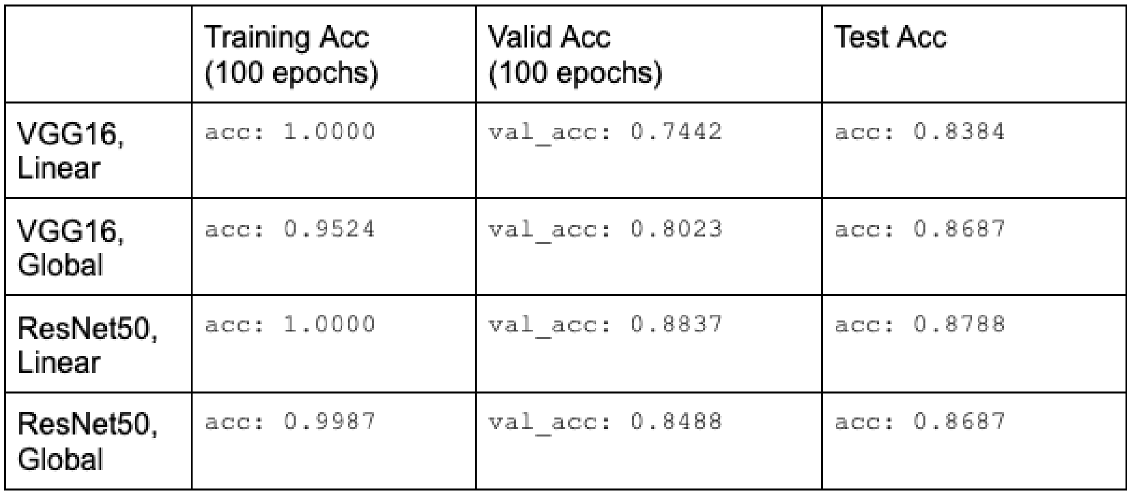
After we determined the idea for our project and came up with a rough outline, we devised a number of questions that would help us evaluate the success of our project. The following is our findings to these questions.

Accuracy

Of course, one of our primary objectives was to have an **accurate** classifier, as a road sign classifier that works in real time is useless if its hardly ever correct. Naturally then, one of the questions that we asked ourselves was "How accurate can we get our classifier?". Feeding off of the ambitious project idea, we set the ambitious goal to implement a classifier with a 95 percent test accuracy.

Since we were doing transfer learning, we had a couple of options on how we wanted to go about designing the architecture. After reading a few articles on the main ideas of transfer learning and ways to go about it, we knew we needed to find a pretrained model that we would use as a convolutional base and then build a classifier on top of that that would take in the features output from the convolutional base and output a prediction on the kind of road sign. In CS 4670–Computer Vision, a course the three of us all took this semester, we learned briefly about VGG16 and ResNet50 as convolutional neural network architectures. Knowing this, we decided to try out both architectures and pick the better of the two. On top of this, we also needed to pick the design for our classifier that would run on top of the convolutional base. From our reading, we discovered that a fully connected neural network or a neural network with a global max pooling layer would be simple, yet effective designs. So with two possibilities for the convolutional base and two possibilities for the classifier, we had four different options to try out and test.

Once we had the ideas in place for the designs, we got to work implementing each of the models and with them all up and running, we achieved the following training, validation, and test accuracies:



	Training Acc (100 epochs)	Valid Acc (100 epochs)	Test Acc
VGG16, Linear	`acc: 1.0000`	`val_acc: 0.7442`	`acc: 0.8384`
VGG16, Global	`acc: 0.9524`	`val_acc: 0.8023`	`acc: 0.8687`
ResNet50, Linear	`acc: 1.0000`	`val_acc: 0.8837`	`acc: 0.8788`
ResNet50, Global	`acc: 0.9987`	`val_acc: 0.8488`	`acc: 0.8687`

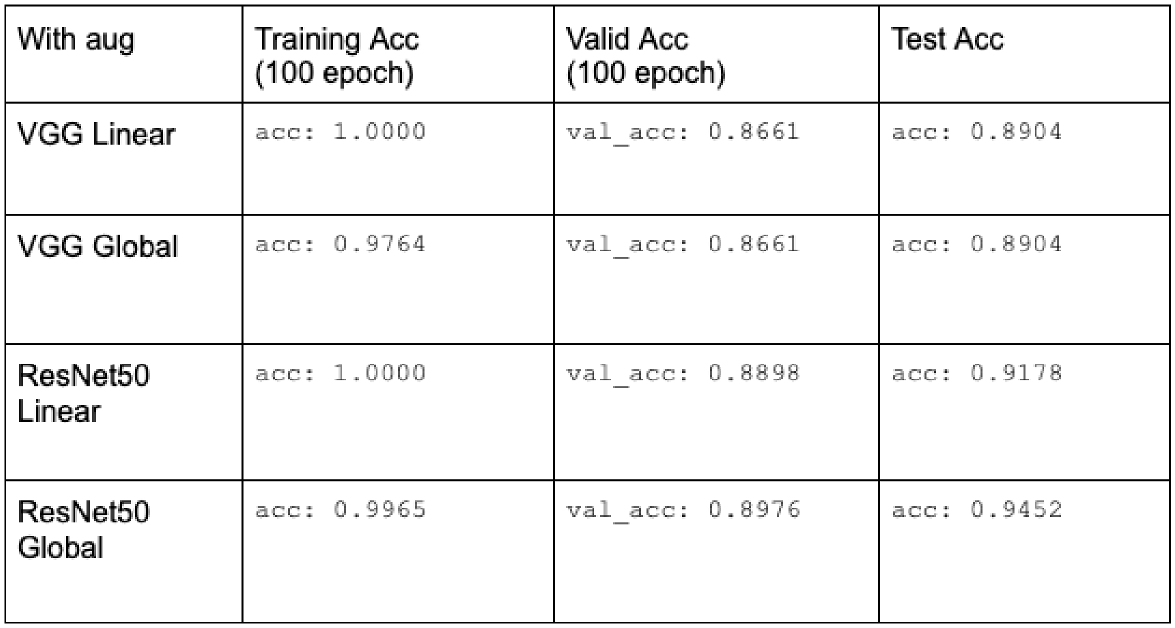
Figure 1: Metric: Accuracy (No Data Augmentation)

Clearly, we were pretty far from our goal and we had a lot more work to do. The training accuracy for all of the possibilities were nearly perfect while the validation accuracy and test accuracy were only classifying correctly around 85 percent of the time, with some combinations performing slightly better than others. Still, even for the best performing model, a ResNet50 convolutional base with a fully connected linear classifier, we were over seven percent away from our goal of 95 percent test accuracy.

To address this, we did a few things. First, we took a closer look at the data set we were using. Comprised of 877 road sign images split into four classes combined with 100 'false' road sign images created by us from the images in the dataset, we had 977 images to train on to be classified in to five classes. 'False' here means an image with no road sign in it. Unfortunately, when actually looking at the images, we noticed some noise, specifically that some images had multiple different road signs in the same picture. For example, an image labeled as 'Stop Sign' may have a stop sign *as well as a speed limit sign in it.* While these did only occur infrequently, it was enough for us to have more lenience toward the 95 percent goal, as we did not take this noise into consideration when coming up with the goal. While we would have liked to switch data sets to one without this problem, all of our preprocessing code was written specifically for this data set and as we were time constrained, it was not feasible for us to go hunting for cleaner data.

Next, heeding the advice from office hours, we implemented data augmentation. Since our data set was quite small, by generating new images from the original images, we would bolster the size of our data set, which is generally a good idea to improve accuracy. Also, by adding images with slight variations (our augmented images had variations in rotation, shear, zoom, horizontal and vertical shift, horizontal flip, and brightness), we believed it would help our model with its overfitting problem as the variations would make our model more invariant to these changes.

With our new modifications, we again trained and tested the four combinations of models, this time achieving the following accuracies along with the following training/validation accuracy plots across all 100 epochs:



With aug	Training Acc (100 epoch)	Valid Acc (100 epoch)	Test Acc
VGG Linear	`acc: 1.0000`	`val_acc: 0.8661`	`acc: 0.8904`
VGG Global	`acc: 0.9764`	`val_acc: 0.8661`	`acc: 0.8904`
ResNet50 Linear	`acc: 1.0000`	`val_acc: 0.8898`	`acc: 0.9178`
ResNet50 Global	`acc: 0.9965`	`val_acc: 0.8976`	`acc: 0.9452`

Figure 2: Metric: Accuracy (With Data Augmentation)

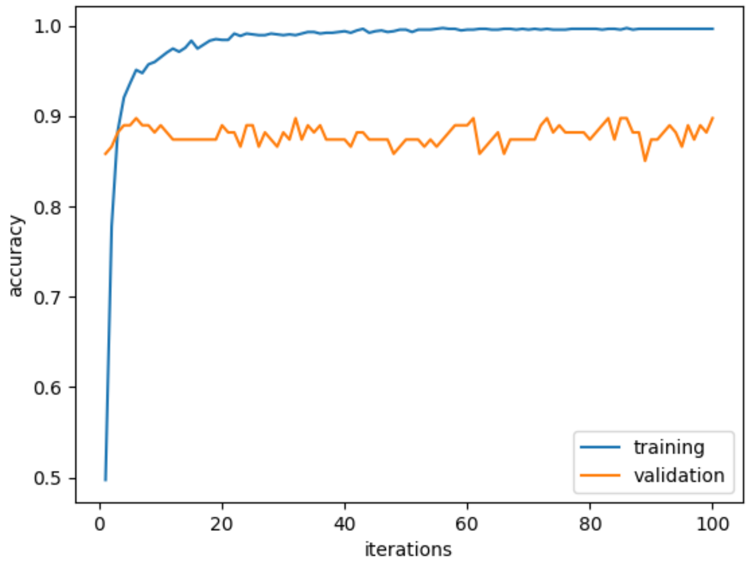


Figure 3: ResNet50 Base with Global Max Pooling Accuracy

With data augmentation in place, we got much better results across the board, with a validation accuracy right around 90 percent for the classifiers with a ResNet50 base and around 87 percent for the classifiers with VGG16 and all four options having a test accuracy above 89 percent. To our delight, using a ResNet50

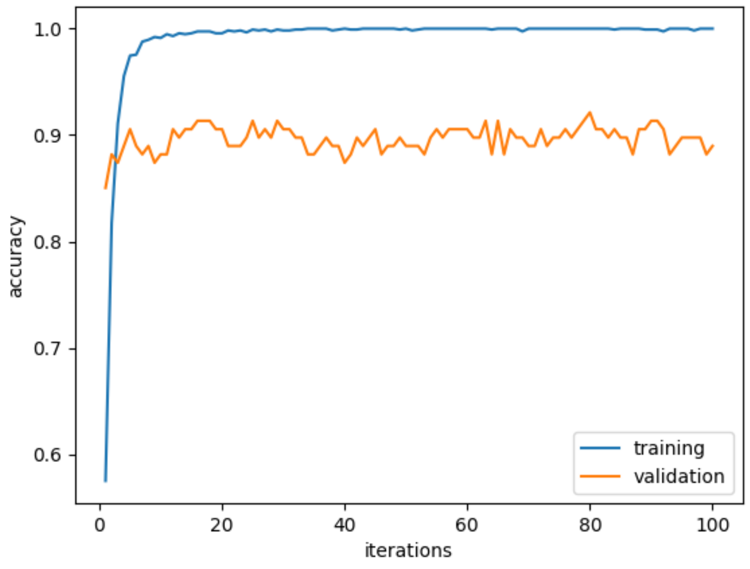


Figure 4: ResNet50 Base with Fully Connected Linear Accuracy

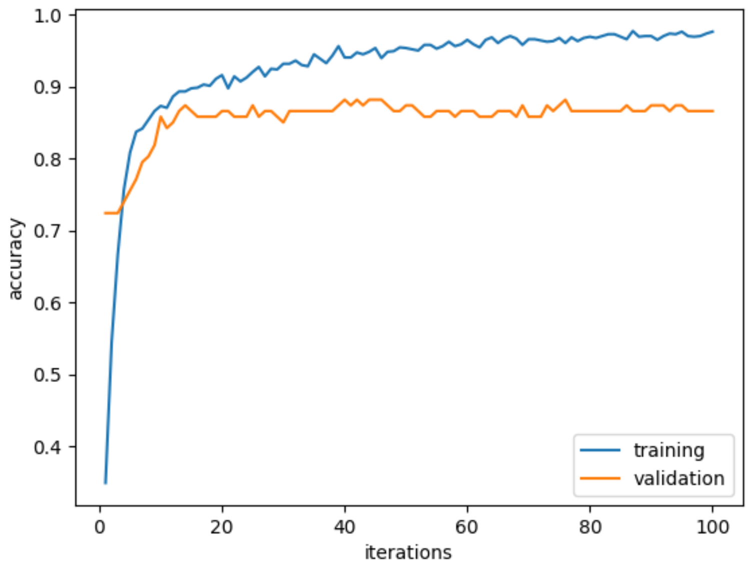


Figure 5: VGG16 Base with Global Max Pooling Accuracy

base and a global max pooling classifier, we achieved a 94.52 percent test accuracy, which, when looked at with two significant figures, rounds up to our goal of 95 percent test accuracy. While 94.52 percent is not exactly 95 percent, considering the noise in our data set, we were very pleased with this result.

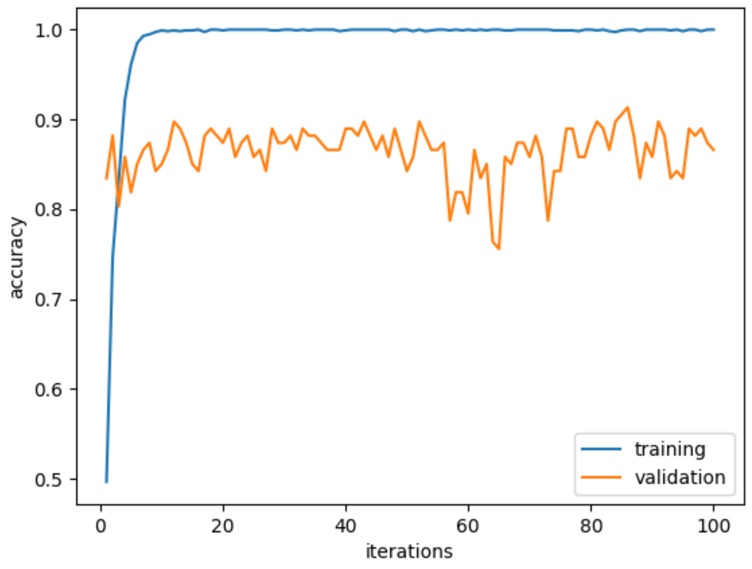


Figure 6: VGG16 Base with Fully Connected Linear Accuracy

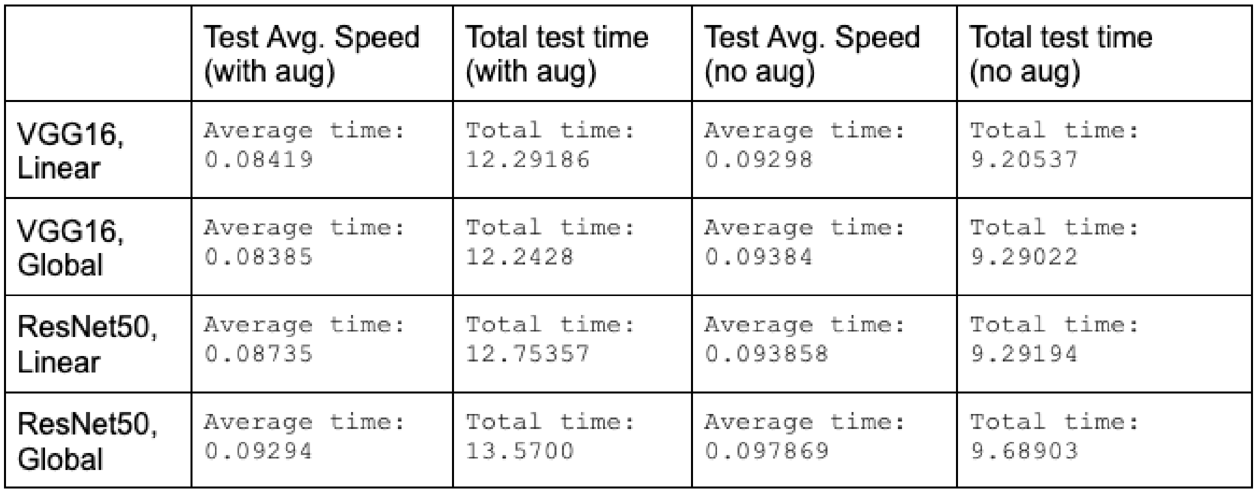
For a breakdown between the four models, we saw that the models using a ResNet50 convolutional base significantly outperformed those using a VGG16 base by over 2.5 percent. Between the two using the VGG16 base, there was no difference in test or validation accuracy but between the two using a ResNet50 base, the one using a global max pooling classifier had an accuracy nearly 3 percent better than the one using a fully connected linear classifier. Our hypothesis for why the ResNet50 based models performed better is that because the ResNet50 architecture is much deeper than the VGG16 architecture and contains many more parameters, it is able to capture more meaningful features than the VGG16 model.

From these results, we decided the best model to deploy in our application was the model with a ResNet50 convolutional base and a global max pooling classifier, as this was the model with the best validation and test accuracy and, as later shown, still met our speed requirement.

Speed

One of the driving forces behind our project idea was to have a road sign detection and classification application that would have the potential to be deployed. This meant that we would have to classify the road signs within a certain time frame for it to be usable. According to the National Library of Medicine, the average human reaction time to detect visual stimuli is 180ms - 200ms. So, we determined that if a human can detect to a road sign in, on average, about 0.18 seconds, then a road sign classifier that can classify in under 0.18 seconds should be fast enough to have the potential to be deployed. Our question in this aspect of the project then is ”Can we build a classifier that outputs predictions faster than 0.18 seconds?”.

As seen from the previous subsection, we tried four different combinations of architectures: VGG16 as a convolutional base with a linear classifier, VGG16 with a global max pooling classifier, ResNet50 as a convolutional base with a linear classifier, and ResNet50 with a global max pooling classifier. On all four options, we evaluated the wall clock time to evaluate all samples in the test set (99 total samples for before data augmentation, 146 images after data augmentation). We evaluated each images one at a time (i.e. batch size of one), measured the total time to classify all images, and then took the average. The results are displayed in the table below.



	Test Avg. Speed (with aug)	Total test time (with aug)	Test Avg. Speed (no aug)	Total test time (no aug)
VGG16, Linear	Average time: 0.08419	Total time: 12.29186	Average time: 0.09298	Total time: 9.20537
VGG16, Global	Average time: 0.08385	Total time: 12.2428	Average time: 0.09384	Total time: 9.29022
ResNet50, Linear	Average time: 0.08735	Total time: 12.75357	Average time: 0.093858	Total time: 9.29194
ResNet50, Global	Average time: 0.09294	Total time: 13.5700	Average time: 0.097869	Total time: 9.68903

Figure 7: Metric: Speed (in seconds)

Note: One reason why the classification with data augmentation is significantly faster on average than without augmentation is that more powerful GPUs were being used on those; however, all architectures with augmentation were evaluated on the same hardware and all architectures without augmentation were evaluated on the same hardware, although it is a different hardware from the classifiers with augmentation. This is the reason why no comparison between classifiers with and without data augmentation is done.

Right off the bat, its clear that we surpassed this goal by a large margin. Now, is this the *fastest* road sign classifier that has ever created? Probably not. But, in terms of our goal in trying to get it to classify road signs faster than the average human can, we greatly exceeded this goal, classifying images roughly twice as fast as our goal time on average. For this reason, as well as the fact that all classifiers are within less than 0.009 seconds of one another on average speed, our choice of architecture laid out in the previous section was much more strongly based in accuracy than in speed. That being said, the inter-model speed difference is something worth looking at.

One aspect of significance is the difference between classifiers with a VGG16 convolutional base and a ResNet50 convolutional base. When holding the classifier (linear or global) constant and changing the convolutional base, in three out of the four comparisons, the model with a ResNet50 convolutional base is significantly slower, with the most drastic being the difference between VGG16 with a global max pooling classifier and data augmentation and ResNet50 with the same, as the one using VGG16 was about 10 percent faster on average than ResNet50. Our hypothesis for the reason behind this is, compared to VGG16, ResNet50 is a huge convolutional base, using 50 layers, of which 48 of them are convolutional layers, while VGG16 has 16 total layers, of which 13 of them are convolutional layers. So, when extracting features from ResNet50, the input must pass through many more layers and more computation must be done.

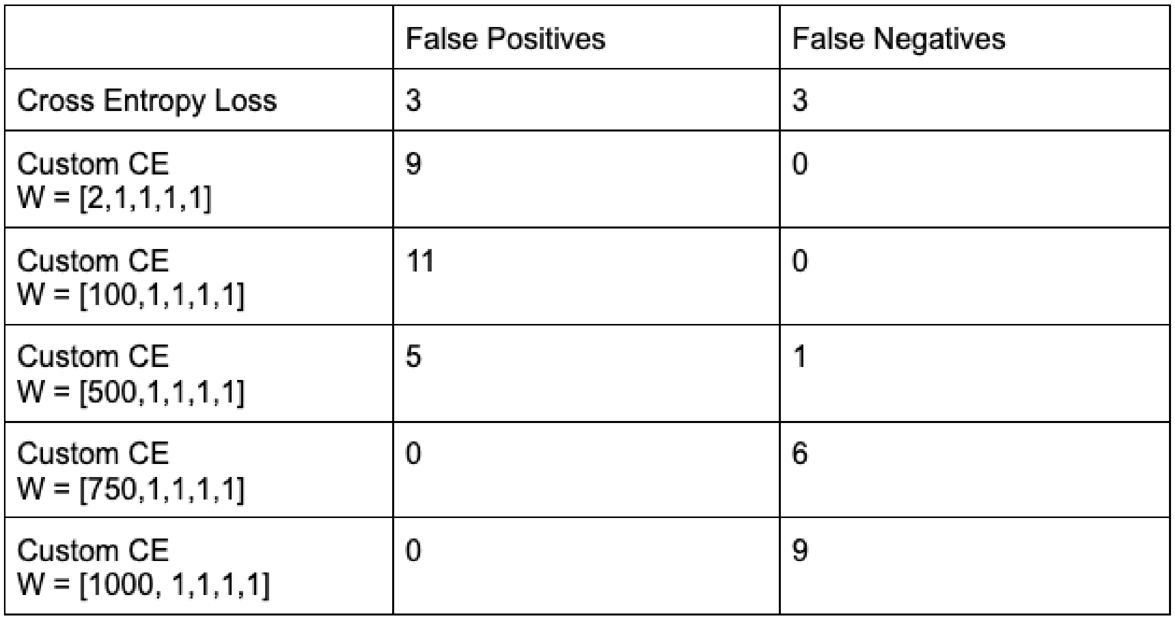
Interestingly, when using VGG16 as a convolutional base, the difference in speed between using a linear classifier and a global max pooling classifier is negligible; however, when using ResNet50 as a base, there was a notable difference in speed, with the model using a global max pooling classifier taking on average more than 0.004 seconds longer in both cases. We are not sure why this is the case, and it is something we would like to look into more when we have the chance.

To close out this subsection, we would like to acknowledge the vast difference in prediction speed when run through Google Colab, where we were running our test, and our user interface. When using the interface, the wall clock prediction time is usually around one whole second, ranging from as fast as 0.75 second to as slow as 1.25 seconds. We are not sure why this is the case as it is the exact same model being used in Colab. Our hypothesis is that is has to do with the user interface module we are using, streamlit, as we know that the classifier can and does work much faster when not using the UI. This is why during evaluation we focused on the speed outside of the user interface.

Loss

The final area of evaluation that we wanted to focus on was limiting the number of images that contained a road sign being classified as a false image. The reason we cared about this goes back to our original goal. If a road sign detector does not detect a road sign, then the driver cannot take any action towards it, so if there is a road sign that it misses, it could lead to accidents. The reason we did not care as much about any other type of misclassification is because the actions that can be associated with stop signs or cross walks for example would be to slow down, and slowing down when it is not necessary is much better in this case than not slowing down when it is necessary. So, the question we asked ourselves was, "Can we build a model that minimizes the number of false positives of non-road sign images while also keeping a high test accuracy?".

To address this goal, we decided to build a custom loss function that weighed each class differently. By changing the weight values, we hoped that we would be able to decrease the number of false positives. In our experiment, our custom loss function was a weighted multi-class cross entropy loss function and our control was a standard built-in multi-class cross entropy loss function. Because we wanted to see the effect our custom loss had on the number of false positives, we chose our worst-performing model, the VGG16 convolutional base with a fully connected linear classifier, to run the experiment on, as this one had the worst accuracy and would allow for the greatest amount of improvement. In the table below, our results are displayed. False positive in this case means the number of images with a road sign in it classified as a false image and false negative means the number of false images classified as an image with a road sign in it. A weight array $W = [x, y, y, y, y]$ has a weight of magnitude x corresponding to a false image and a weight of magnitude y corresponding to a non-false image.



	False Positives	False Negatives
Cross Entropy Loss	3	3
Custom CE W = [2,1,1,1,1]	9	0
Custom CE W = [100,1,1,1,1]	11	0
Custom CE W = [500,1,1,1,1]	5	1
Custom CE W = [750,1,1,1,1]	0	6
Custom CE W = [1000, 1,1,1,1]	0	9

Figure 8: False Positives and Negatives per Weight

As displayed in the table, the custom loss function technically does cause the model to have fewer false positives, however, in order to achieve this, the input weight for a false image must be multiple orders of magnitude larger than the weights for the other classes. Because of this, when training and testing with these extreme weights, the accuracies were horrible, making the results useless. Of course, we expected having a trade off between maximal accuracy and minimal number of false positives, but with our custom loss function, the trade off is not worth it. This trade off can be easily seen by the steady increase in number of false negatives as the weight on the non-road sign class increases.

We consider this part of our project a failure as we were not able to more harshly punish this type of false positive while maintaining a respectable test accuracy. After running these tests and realizing that our loss function was not up to our standard, in the deployed model in the application, we ended up using the built-in multi-class cross entropy loss, as it managed to keep the false positive number much lower than ours even when our weights were set to equal values.

Wrap Up

Overall, we say this is a failure because, when looking back to our original goal, our project cannot even take in a video; it only works on images. Also, we were unable to create a custom loss function that heavily punishes false positives of false images while maintaining a high accuracy. That being said, our project did accomplish a lot, including many of the implicit goals that we set, like the speed and accuracy goals. And, as the semester moved along, we had new ideas that we thought would be interesting to implement but just did not have the time, like putting a bounding box around the detected road sign. So, overall, we classify this project as a good start, as we managed to accomplish a lot, but unfinished, as we fell well short of our original goals and we believe that we could have achieved more if we had more time.

References

Data augmentation: https://www.geeksforgeeks.org/python-data-augmentation/

Preprocessing: https://docs.python.org/2/library/xml.etree.elementtree.htmlmodule-xml.etree.ElementTree
Transfer Learning: https://vijayabhaskar96.medium.com/tutorial-on-keras-flow-from-dataframe-1fd4493d237c https://towardsdatascience.com/transfer-learning-from-pre-trained-models-f2393f124751
Custom Loss: https://towardsdatascience.com/how-to-create-a-custom-loss-function-keras-3a89156ec69b
Software Resources:
Keras, Pandas, StreamLit
Data Resources:
https://www.kaggle.com/datasets/andrewmvd/road-sign-detection