Street Signs!

presented by:

Joseph Linton

Copeland Carter

Camilo Hozman

Philip Marvin

# I. Models

Custom

The custom network took a decent amount of tweaking, but ran a lot faster. We noticed that when we used RELU as the activation function on the final layer caused it to get accuracies of ~5% (akin to random guessing). We switched to using softmax as the final activation function which performed much better.

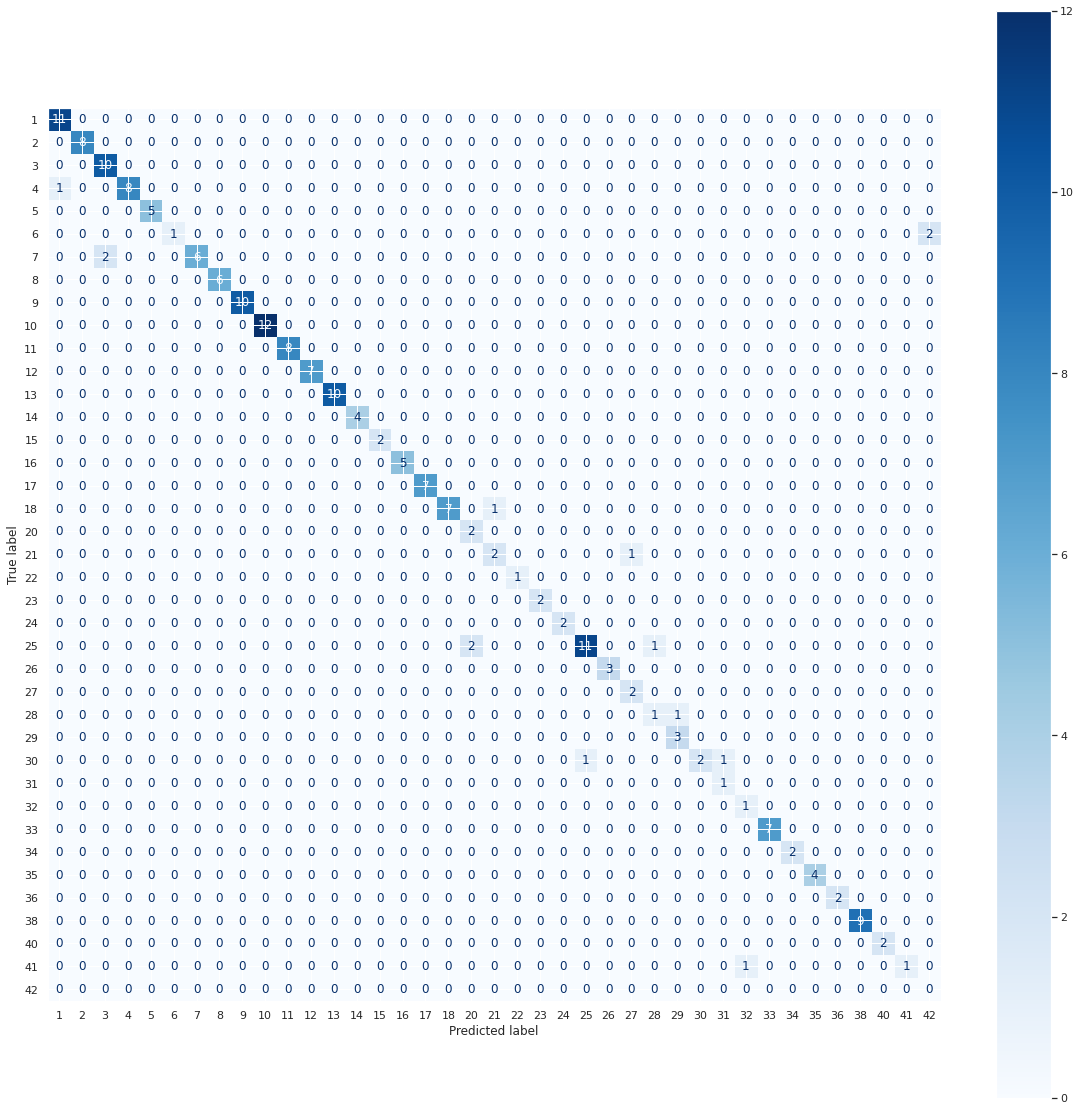
We ran the custom model for ~1,440,000 images (~150 epochs @ 300 batches/epoch @ 32 images/batch), and the model seemed to plateau at ~93% accuracy, 95% precision, 93% recall, and 93% F1 score. We suspect this is because the model has become “saturated” and has learned all it can. Because a model with 50 parameters could not possibly learn how many bikes are going to be rented in Washington DC, a minimum amount of parameters are required for certain accuracies in certain tasks. We suspect that our custom model (with 8,799,595 trainable parameters) reached its upper limit as to how much it could learn.

precision recall f1-score support

accuracy 0.93 201

macro avg 0.89 0.89 0.87 201

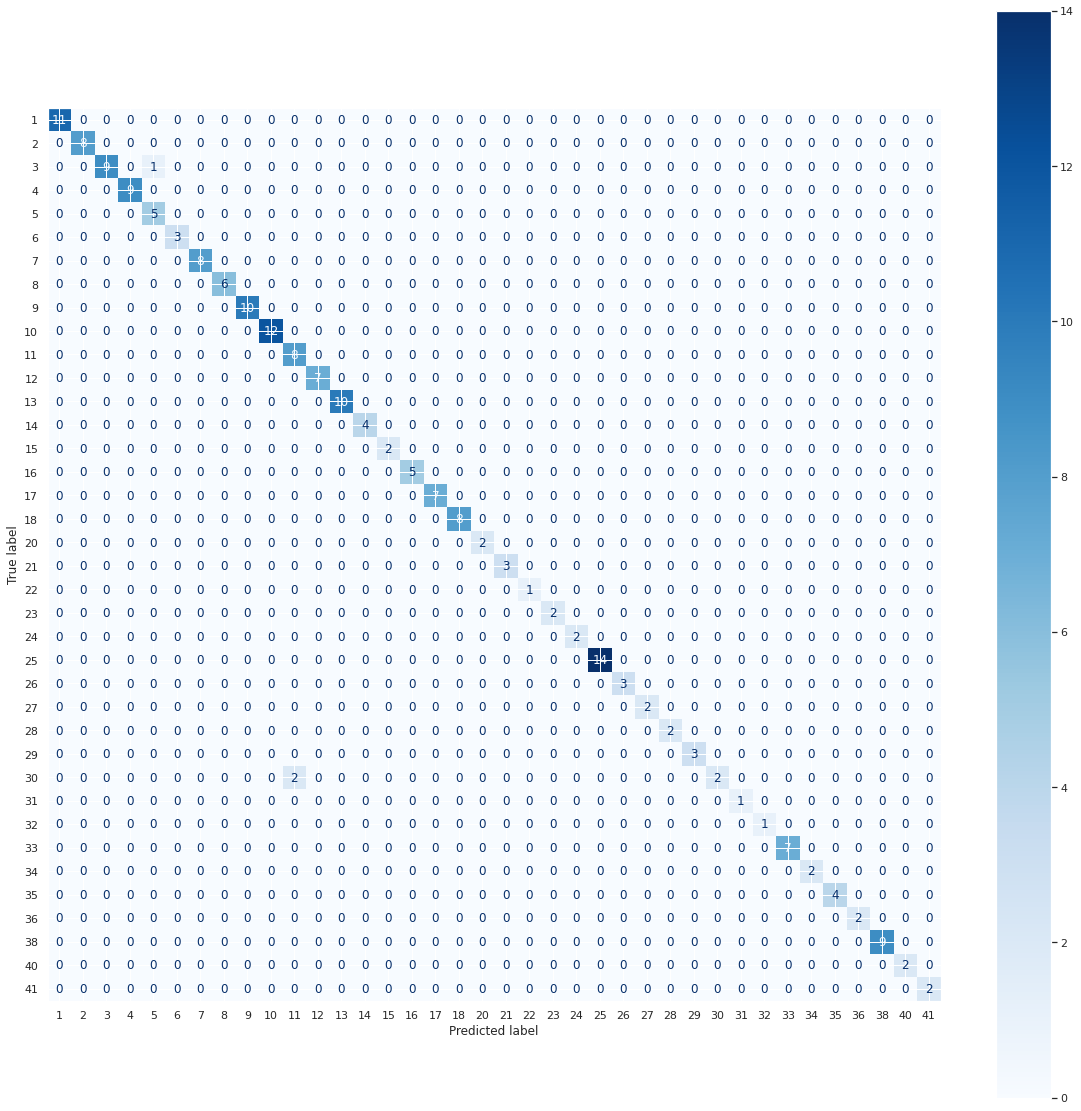
weighted avg 0.95 0.93 0.93 201



VGG-16 Model:

We also tried a VGG-16 model from the tensorflow.keras.applications module. It seemed to work significantly better after only 96,000 images, getting val\_accuracy scores in the 90’s. We ran it for ~1,000,000 images and got an accuracy of 98.5%, a recall and precision of 99%, and an F1 score of 98%.

The VGG-16 model is a highly significant tool in the field of machine learning due to its exceptional ability to accurately classify and recognize objects. Its unique architecture, with multiple layers of convolutional and pooling operations, followed by fully connected layers, makes it highly effective in tasks such as object detection and segmentation. Moreover, the VGG-16 model's pre-trained weights enable transfer learning, making it a valuable tool for researchers and practitioners who work with limited amounts of data. Its simple and modular architecture also ensures that the learned features and activations in each layer are easy to understand and interpret, making it useful for debugging and optimizing the model's performance for specific applications. Therefore, the VGG-16 model's exceptional performance, unique architecture, transfer learning capabilities, and interpretability make it a crucial tool for researchers and practitioners in the field of machine learning.



precision recall f1-score support

accuracy 0.99 201

macro avg .99 0.98 0.98 201

weighted avg .99 0.99 0.98 201

# II. Data Augmentation

We did several operations for data augmentation/preprocessing:

* Brightness
  + This is an important factor because it’s impacted by weather. We want to be able to recognize signs both at night and in bright sunlight.
  + Hard: 60% - 150% change
  + Easy: 80% - 120% change
* Shift
  + This is important because if we see only part of a sign with our image, or the sign is blocked by a wall or some other obstruction, we still have to be able to recognize it.
  + Hard: ±22% shift
  + Easy: ±10% shift
* Shear
  + We figured shear would be very important, because frequently images would be taken as the car is making a turn or is coming up on a sign.
  + Hard: ±20°
  + Easy: ±10°
* Zoom
  + Zoom seems important, especially if the vehicle is going to take multiple pictures as it approaches the sign.
  + Hard: ±40%
  + Easy ±15%
* Rotation
  + If the car or camera is off by a little or the sign is tilted a little, it’s helpful to be able to recognize that, however it likely doesn’t need to be excessive (we’re not going to have to recognize upside down signs).
  + Hard: ±10°
  + Easy: ±4°
* Flip
  + We decided not to use horizontal nor vertical flipping, as that’s simply not a situation we’re going to have to deal with (a backwards sign is not the same sign).

Something we did for training was to start the model on easy image augmentation, so the model could have a chance to learn the basics of what a sign looks like, then after about 32,000 images increase the image augmentation to make the model learn more complicated patterns in the images. We compared it to a human child trying to learn. You typically don’t make a child who just learned to read try to immediately read Shakespeare and legal contracts. You probably want to start off with something easy like “Hop on Pop”. However to become a really good reader, you can’t always read picture books, you probably do need to read Shakespeare at *some* point.

We also noticed that with the “hard” data augmentation, the model tended to improve slower, but was able to improve more than with the “easy” augmentation. We hypothesize this is because it’s learning more difficult material, but also is gaining more insights from the material being harder. Like in our own lives, neural networks tend to learn more when things are harder than if they’re always easy.

| Examples of images with “Easy” augmentation | Examples of images with “Hard” augmentation |
| --- | --- |
|  |  |

III. Implementation

**# Custom Model**

**custom = ks.Sequential([**

**layers.Conv2D(32, (3,3), input\_shape=image\_size, activation='relu'),**

**layers.MaxPooling2D(),**

**layers.Conv2D(64, (3,3), activation='relu'),**

**layers.MaxPooling2D(),**

**layers.Conv2D(128, (3,3), activation='relu'),**

**layers.MaxPooling2D(),**

**layers.Dropout(.15),**

**layers.Conv2D(256, (3,3), activation='relu'),**

**layers.Flatten(),**

**layers.Dense(512, activation='relu'),**

**layers.Dropout(.2),**

**layers.Dense(num\_classes, activation='softmax'),**

**])**

**custom.compile(optimizer=ks.optimizers.Adam(learning\_rate=.000115), loss=loss, metrics=['accuracy', loss])**

**custom.summary()**

# The model architecture we have created is a simple Convolutional Neural Network (CNN) consisting of two convolutional layers with pooling, followed by a fully connected layer and an output layer. Technically, the closest type of model that resembles this architecture would be a basic CNN model. However, the specific hyperparameters and layer configurations can vary greatly between CNN models, so it's still considered a custom model. The configuration that causes the greatest difference is the following:

1. Number and size of the filters in each layer
2. Number of neurons in the fully connected layers
3. Learning rate of the optimizer
4. Batch size
5. Number of epochs

# Premade Model

premade = app.vgg16.VGG16(

include\_top=True,

weights=None,

input\_shape=image\_size,

classes=num\_classes,

)

premade.compile(optimizer=ks.optimizers.Adam(learning\_rate=.0001), loss=loss, metrics=['accuracy', loss])

premade.summary()

# 

# IV. Conclusions and Retrospective

There are several qualities in the dataset that could be improved to help train the model better:

1. More data: Increasing the amount of training data can help the model learn more patterns and make better predictions. Collecting more images of each class or generating new images through data augmentation techniques can help in this regard.
2. Balanced dataset: The dataset appears to be imbalanced, with one class having significantly fewer images than the other classes. This can lead to bias in the model and affect its ability to correctly classify images from the minority class. Balancing the dataset by collecting more images from the underrepresented classes or using techniques like oversampling or undersampling can help.
3. High-quality images: Poor quality images can make it difficult for the model to correctly identify patterns in the data. Ensuring that images are of high quality, properly cropped, and oriented can help improve model performance.
4. Image normalization: Normalizing the image data can help improve the model's ability to learn from the data by reducing the effects of differences in lighting and color. This can be done by standardizing the pixel values or using techniques like histogram equalization.
5. More diverse images: The current dataset appears to have images of road signs that are primarily taken in a single setting, such as on a sunny day with a clear background. However, road signs can appear in a wide range of conditions, such as in bad weather or in crowded scenes. Including images that capture this variety can help the model learn to recognize signs under different conditions.

# 

# V. Discussion Questions

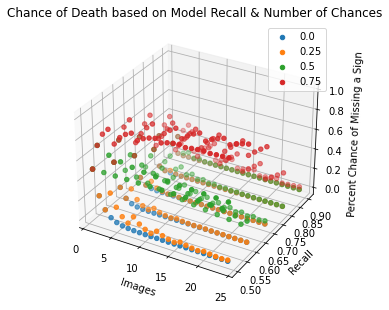
Our team felt that VGG-16 and a custom model would be good architectures to explore with this data.

For preprocessing, at minimum we needed to rescale the color values from 0-255 to 0-1 in order to input into the model. We also did a bunch of preprocessing steps as mentioned above.

In order to recognize signs that don’t look exactly like the ones in the training set, we used the preprocessing steps mentioned above to help give the model the “concept” of what any given sign looks like, instead of just an accurate representation of what *our* signs look like.

We determined that the most important metric here was recall. We figure i t’s better to mis-identify a stop sign and stop in the middle of the road than to miss a stop sign and plow through an intersection.

Before we get into the vehicle, we decided we would like to know more about the particular design of the vehicle. Say the vehicle takes multiple images (or video) of a sign as it approaches a sign, and has to recognize a certain percentage of them in order to recognize it as a sign. If we assume the vehicle takes 12 images as it approaches a sign, our model has a recall of 95%, and you only have to get 25% (3) images to identify as a sign before the car recognizes it as a sign, there's only a 3.743×10-8% (P(Binomial(12, .95) <= 3)) chance that the car will *not* recognize the sign, and even with a 90% recall model given 5 images, there’s still a 0.046% chance of missing a sign



# VI. Python Notebooks

Below are Github Gist links to the notebooks we used during this case study:

(Note: It was decided that we’d each create our own Notebooks)

Joseph: <https://colab.research.google.com/drive/1qp1q1Wqw-dXNgeiNA7CHEMsn4FSAxB59?usp=sharing>

Copeland: [Gist](https://colab.research.google.com/gist/smartycope/ed1786829438a91a16da17fd6796e334/signsproject.ipynb), Custom & VGG-16 models, data augmentation, and statistical analysis  
Philip: <https://colab.research.google.com/drive/19zDdK9NESXB6HIiUAmX0tpSGuD7wdAO6>   
Camilo: <https://colab.research.google.com/gist/hozmancj7/e6ae9b8baebce6f01278983c6e395da4/untitled11.ipynb>