Rice University - COMP 340

M&M Street Number Readers

Muthu Chidambaram & Melinda Ding

Final Project Report

December 6, 2019

Abstract

In this paper, we examine different convolutional neural network architectures for the Street View House Numbers classification problem. The chosen architecture results in a 95% test accuracy.

Table of Contents

Overview

The goal of this project was to develop a convolutional neural network that could correctly identify digits in the Street View House Number (SVHN) dataset. The SVHN dataset is comprised of over 600,000 real world images of house numbers from Google Street View, and is an excellent dataset to build and test performance of varying neural network architectures. In this report we will attempt to extend the work of existing frameworks to improve the performance of our model.

Data

The data set we are using was obtained from house numbers in Google Street View images. The data was preprocessed and labelled by a research group at Stanford[1]. The dataset we chose to use was the MNIST-like images, where each image is centered around a single character but still contain distractors at the sides. Each image had a dimension of 32x32. The data comes labelled with 10 classes, 1 for each digit. There are 73257 digits for training, 26032 digits for testing, and 531131 additional, somewhat less difficult samples, to use as extra training data.

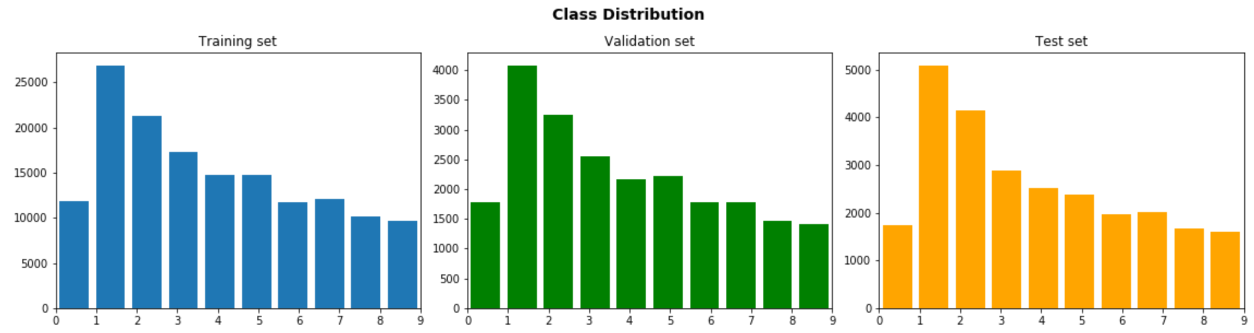
We used the entire training data set, plus an additional 100,000 images from the extra data set. We then used 13% of the training data set for validation. The following table shows the number of images in the train, validation, and test set.

	Number of Samples
Train	150733
Validation	22524
Test	26032

When splitting the data for training, testing, and validation, we were careful to set the random state seed so that on repeated runs of the notebook our model was being trained on the same split of training, validation, and test data. This is done to control for the data split as a variable that might affect model performance. It is arbitrary which random split we choose, but we must be consistent when testing different model architectures to ensure that the model architecture is actually influencing performance.

[1] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, Andrew Y. Ng Reading Digits in Natural Images with Unsupervised Feature Learning NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011. (http://ufldl.stanford.edu/housenumbers/)

We then conducted EDA on the dataset to see the distribution of digits (depicted below). Considerations like this are valuable because they inform us of that the model might be biased simply because it is over or under represented in the training samples. Future work can be done here to ensure the training set is an even distribution of digits.



Model

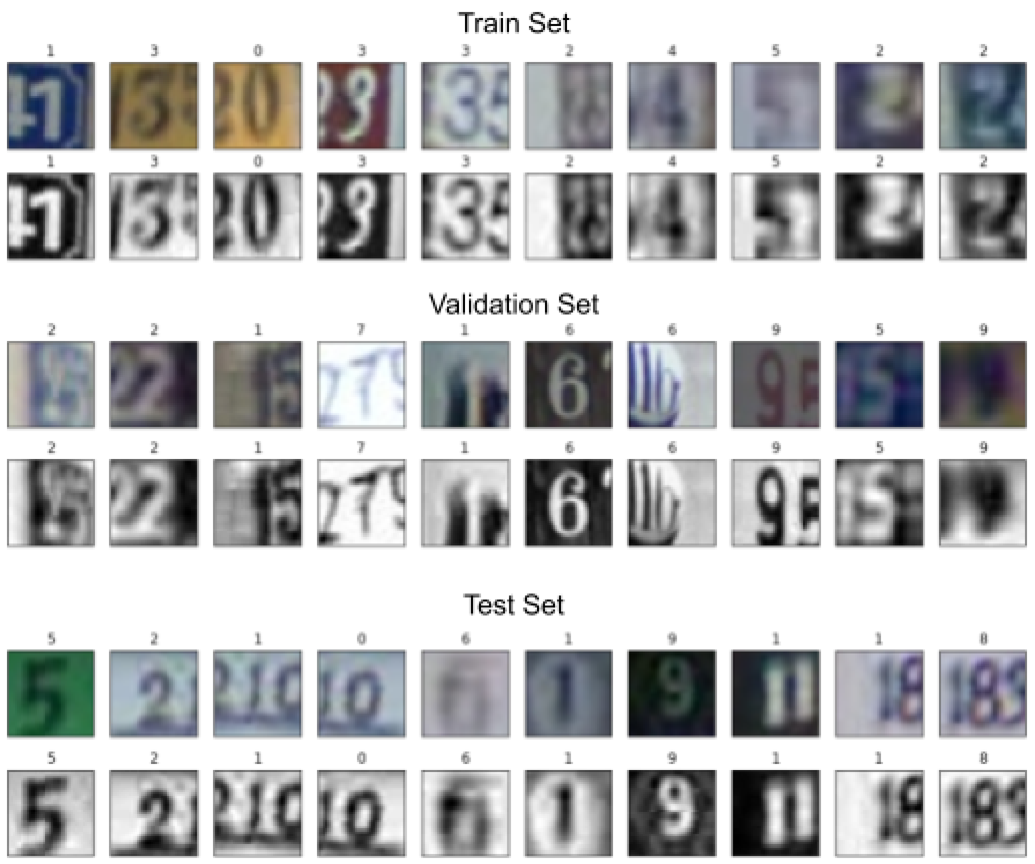
Procedure

The first thing we did with the cropped images was convert them to grayscale. This is for a couple reasons. First, the color features are relatively unimportant in order to distinguish which digit is present - the shape is more primary. Thus, reducing the image to simply grayscale prevents the model from learning arbitrary variations in color as features for classifying digits. Furthermore, color adds to the dimensionality of the image pixels so reducing them to grayscale vastly improved the training speed performance for the models. For the purposes of our project, model speed was highly prioritized because we needed to test multiple different model architectures in a short period of time.

Next we normalized the images so that they have zero mean and unit variance. This was done by subtracting the training mean equally from all splits and then dividing by the standard deviation. The reason for doing this is because we train our neural network using a constant learning rate for all of the weights. If we did not scale these weights and normalize them in some manner, then we might be compensating for these inherent feature differences by altering the fit of the network. The fit generated with non normalized weights might not accurately reflect the underlying structure and patterns in the data set. Normalization is key in any gradient descent algorithm to prevent oscillation when trying to find a maxima or slow moving convergence[2].

[2] Stats StackExchange: https://stats.stackexchange.com/questions/185853/why-do-we-need-to-normalize-the-images-before-we-put-them-into-cnn

Below are a few examples of the original and grayscale images from each set.



Finally, in order to prepare our output labels for the neural networks we had to one hot encode the categorical results in a matrix. This was easily done using numpy util's to_categorical method with the number of possible classes (10).

Methodology

Comparing Training Dataset Sizes

To compare the effect of the training dataset size on the model, we used the same model, the chosen model (page 11), for each training set. We created a small, mid, and large sized dataset. The sizes and performances of each dataset are summarized in the table below:

Training Data Size	Validation Data Size	Test Loss	Test Accuracy
63733	9524	0.44760306440216513	0.912338660110633
107233	16024	0.24081448260989538	0.9343116164720344
150773	22524	0.20034723480677258	0.9516748617086662

We see that with each 50,000 increase in training, there is a ~2% increase in accuracy. This was the best improvement we saw in the model from a single attribute.

Comparing Network Density & Depth

For purposes of comparing models, we define the following to represent the architecture of different networks:

Layer	Representation
Convolutional Layer	Conv-{number of filters}-{filter size}-{stride size}
Max-Pool Layer	MaxPool-{filter size}-{stride size}
Dropout	Dropout-{probability}
Fully-Connected Layer	FC-{number of nodes}

We then defined the following simple network (from Sharma's notebook[3]) as our base model: Conv-32-5-1 → MaxPool-2-2 → Conv-64-5-1 → FC-1000 → FC-10. Each convolutional layer used relu activation, and the final fully connected layer of 10 nodes used softmax activation. Sharma's model also used a batch size of 128, 10 epochs, and adam optimizer. The model achieved around 92% accuracy with our dataset.

We then researched and experimented with different ways to improve our model including adding more data, transfer learning with the VGG16 network, varying depths and density of the convolutional layers, varying filter size and stride and different loss optimizers (Adam vs. SGD).

For the purpose of the report, we selected a few key models. For all of the models, except the VGG16 model, we used an Adam optimizer, batch size of 128, maximum of 25 epochs, and early stopping with patience of 3. The VGG16 model used a SGD optimizer with

[3] Sharma, Aditya, SVHN-CNN, March 2018, https://github.com/aditya9211/SVHN-CNN

learning rate of .001, momentum of 0.9, batch size of 128, and 100 epochs. All the models used the same training, validation, and test set. The models are defined as:

Model	Description	Architecture	Total Num. of Params.
Base	Aditya's Model	Conv-32-5-1 → MaxPool-2-2 → Conv-64-5-1 → MaxPool-2-2 → FC-1000 → FC-10	1,663,106 (0 untrainable)
A	VGG16 pretrained network	VGG16 network → FC-1024 → Dropout-0.2 → FC-512 → Dropout-0.2 → FC-10	15,769,930 (14,714,688 untrainable)
B	Goodfellow	Conv-48-5-1 → Batch-Norm → MaxPool-2-2 → Conv-64-5-1 → Batch-Norm → Dropout-0.1 → MaxPool-2-1 → Conv-128-5-1 → Batch-Norm → Dropout-0.1 → MaxPool-2-2 → Conv-160-5-1 → Batch-Norm → Dropout-0.1 → MaxPool-2-1 → Conv-192-5-1 → Batch-Norm → Dropout-0.1 → MaxPool-2-2 → Conv-192-5-1 → Batch-Norm → Dropout-0.1 → MaxPool-2-1 → Conv-192-5-1 → Batch-Norm → Dropout-0.1 → MaxPool-2-2 → Conv-192-5-1 → Batch-Norm → Dropout-0.1 → MaxPool-2-1 → FC-1024 → Batch-Norm → Dropout-0.1 → FC-10	4,545,418 (4,384 untrainable)
C	Sparser 4 layer network	Conv-48-5-1 → MaxPool-2-2 → Conv-64-5-1 → MaxPool-2-1 → Conv-128-5-1 → MaxPool-2-2 → Conv-256-5-1 → MaxPool-2-1 → FC-1024 → FC-10	7,667,370
D	5 layer network	Conv-48-5-1 → MaxPool-2-2 → Conv-64-5-1 → MaxPool-2-1 → Conv-64-5-1 → MaxPool-2-2 → Conv-128-5-1 → MaxPool-2-1 → Conv-256-5-1 → MaxPool-2-1 → FC-1024 → FC-10	2,264,810
E	Denser, 4 layer network	Conv-48-5-1 → MaxPool-2-2 → Conv-128-5-1 → MaxPool-2-1 → Conv-128-5-1 → MaxPool-2-1 → Conv-256-5-1 → MaxPool-2-2 → FC-1024 → FC-10	10,832,618
F	Denser, 4 layer, with more fully connected layers at the end	Conv-48-5-1 → MaxPool-2-2 → Conv-128-5-1 → MaxPool-2-1 → Conv-128-5-1 → MaxPool-2-2 → Conv-256-5-1 → MaxPool-2-1 → FC-1024 → Dropout-0.5 → FC-512 → FC-10	8,468,714
G	**Same as F, but with**	**Conv-48-5-1 → MaxPool-2-2 → Conv-128-5-1 → Batch-Norm → Dropout-0.1 → MaxPool-2-1**	**9,700,202**

	dropout & batch-normalization **(poster)**	**→ Conv-256-5-1 → Batch-Norm → Dropout-0.1 → MaxPool-2-2 → Conv-256-5-1 → Batch-Norm → Dropout-0.1 → MaxPool-2-1 → FC-1024 → Dropout-0.5 → FC-512 → FC-10**	**(1,280 untrainable)**

Model A is significant because it is our base model. Model B is significant because it is the transfer learning VGG16 model. Model C was created as a result of adding depth to the base model until performance stopped increasing; model C is when we realized a depth of 4 was the best. Model D, with 5 layers, reinforces the notion that a depth of 4 was the best. After realizing a depth of 4 was the best, we created Model E to tune the density of the model. We then created Model F to experiment with adding more fully connected layers after the convolutional layers. Model G was created after experimenting with dropout only, batch normalization only, and both (see next section).

The performance results of the models are summarized in the table below.

Model	Description	Test Loss	Test Accuracy
Base	Aditya's Model	0.4817130405970542	0.9230562384757222
A	VGG16 pretrained network	1.8437621228593803	0.3735018438844499
B	Goodfellow	0.7296997932803067	0.7639443761524278
C	Sparser, 4 layers	0.2462518504458281	0.9404194837123541
D	Sparser, 5 layers	0.23914086382026706	0.9397280270436386
E	Denser, 4 layers	0.24861236794530595	0.9412261831591887
F	Denser, 4 layers, more fully connected layers	0.2469909702741155	0.9394975414874002
G	**Denser, 4 layers, more fully connected layers, batch normalization and dropout**	**0.20034723480677258**	**0.9516748617086662**

It is important to note that we tried to set the seed for the models to ensure reproducible results. However, because we ran all the experiments in Google Colab with a GPU, we are not able to completely control the randomness. The backend may be configured to use a sophisticated stack of GPU libraries, this may introduce a source of randomness that we cannot account for.[4]

[4] J. Brownlee, Machine Learning Mastery, https://machinelearningmastery.com/reproducible-results-neural-networks-keras/

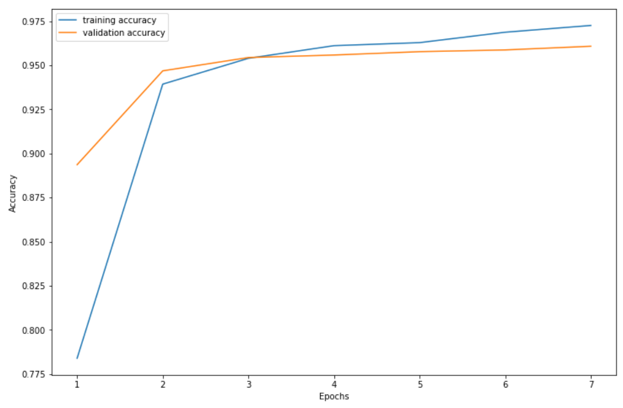
Comparing Dropout & Batch Normalization

After settling on a depth and density of our network, we then added dropout and batch normalization layers to improve our network. For the batch normalization only model, we added a batch normalization layer between each convolutional and maxpool layer. For the dropout only model, we added a dropout layer with probability 0.2 between each convolutional and maxpool layer. The networks are detailed below:

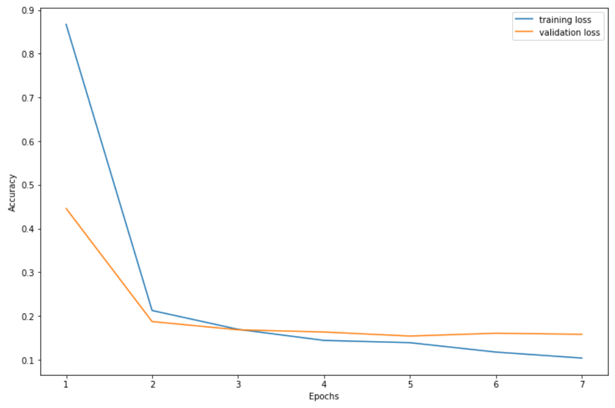
Model	Architecture	Number of Params.
Batch Normalization Only	Conv-48-5-1 ➝ BatchNorm ➝ MaxPool-2-2 ➝ Conv-128-5-1 ➝ BatchNorm ➝ MaxPool-2-1 ➝ Conv-256-5-1 ➝ BatchNorm ➝ MaxPool-2-2 ➝ Conv-256-5-1 ➝ BatchNorm ➝ MaxPool-2-1 ➝ FC-1024 ➝ Dropout-0.5 ➝ FC-512 ➝ FC-10	Total: 9,700,394 Non-trainable: 1,376 Trainable: 9,699,018
Dropout Only	Conv-48-5-1 ➝ Dropout-0.2 ➝ MaxPool-2-2 ➝ Conv-128-5-1 ➝ Dropout-0.2 ➝ MaxPool-2-1 ➝ Conv-256-5-1 ➝ Dropout-0.2 ➝ MaxPool-2-2 ➝ Conv-256-5-1 ➝ Dropout-0.2 ➝ MaxPool-2-1 ➝ FC-1024 ➝ Dropout-0.5 ➝ FC-512 ➝ FC-10	Total: 9,697,642 Non-trainable: 0 Trainable: 9,697,642
Both	(see model G)	Total: 9,700,202 Non-trainable: 1,280 Trainable: 9,698,922

We then plotted the training and validation loss and accuracy plots for each model to determine how dropout and batch normalization affected the models.

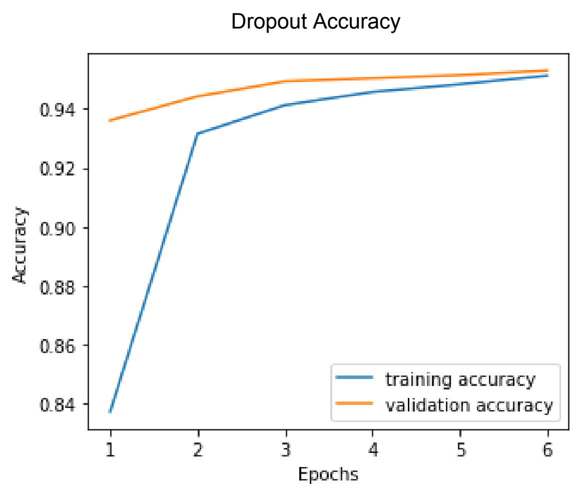
Batch Normalization Accuracy

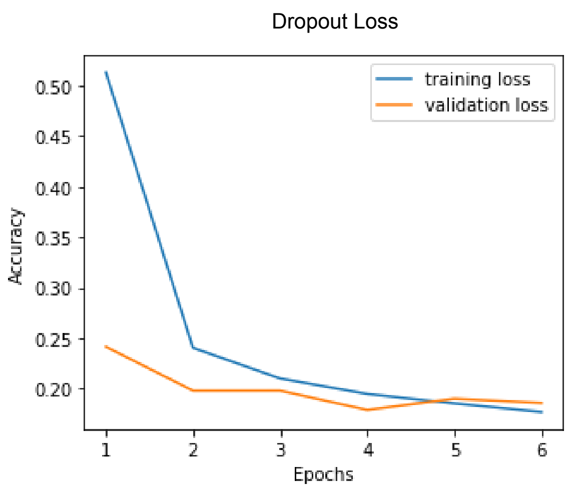


Batch Normalization Loss

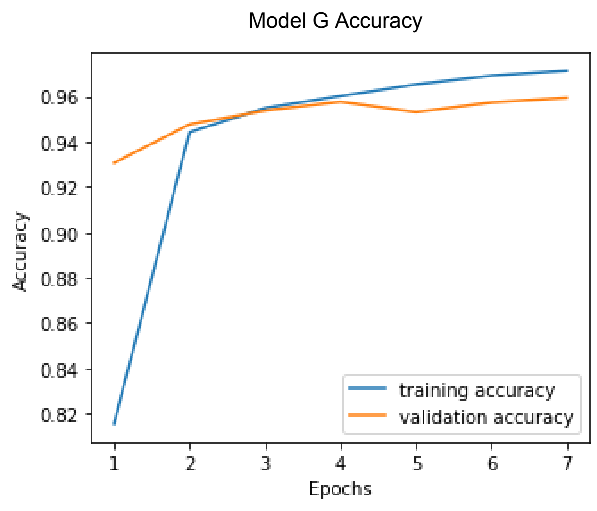


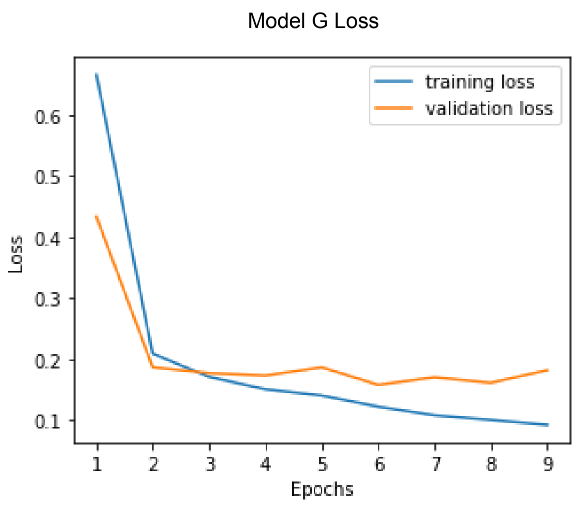
For the Batch Normalization, we see pretty good accuracy and loss curves. The validation and training curves converge at some point, and remain relatively close.





The Dropout model performs pretty strongly as well, with the validation loss curve showing almost no overfitting and excellent convergence.





Model G, with both dropout and batch normalization, appears to do the best. The accuracy curves converge for a longer time. The drifts at the ends of the curves can be explained by the patience allowing another few runs, and the model becoming overfit.

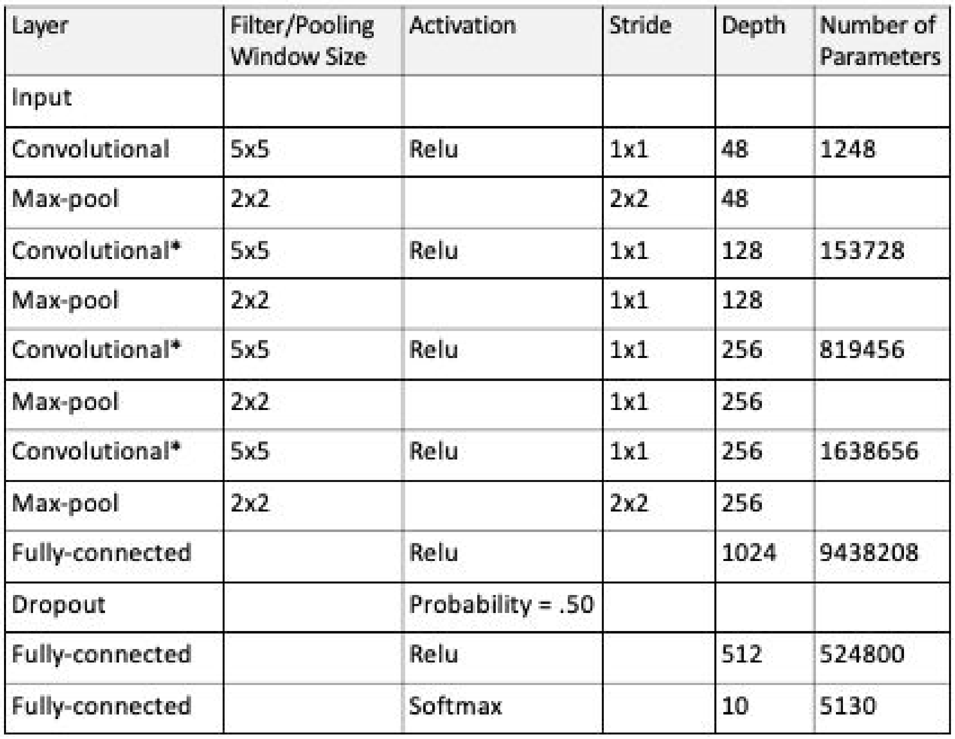
The test loss and test accuracy results for each model type is summarized below:

Description	Test Loss	Test Accuracy
Batch-Normalization Only	0.2454454548317497	0.9396896127842655
Dropout Only	0.2522263968158479	0.9416487400122926
Both	0.20034723480677258	0.9516748617086662

We see that the model with both dropout and batch normalization performs the best.

Chosen Model

The architecture of our best model, model G, is detailed below:



Layer	Filter/Pooling Window Size	Activation	Stride	Depth	Number of Parameters
Input					
Convolutional	5x5	Relu	1x1	48	1248
Max-pool	2x2		2x2	48	
Convolutional*	5x5	Relu	1x1	128	153728
Max-pool	2x2		1x1	128	
Convolutional*	5x5	Relu	1x1	256	819456
Max-pool	2x2		1x1	256	
Convolutional*	5x5	Relu	1x1	256	1638656
Max-pool	2x2		2x2	256	
Fully-connected		Relu		1024	9438208
Dropout		Probability = .50			
Fully-connected		Relu		512	524800
Fully-connected		Softmax		10	5130

Convolutional* = convolutional layer followed by a batch normalization step and a dropout layer with probability of 0.1

The total number of parameters is 9,700,202. The total number of trainable parameters is 9,698,922, and the number of untrainable parameters is 1,280. The difference in the number of parameters is due to the batch-normalization layers; this is because batch-normalization layers are updated with mean and variance, but not "trained with backpropagation". [5] The batch size was 128, and the max number of epochs was 25. The model was implemented with early stopping and a patience of 3.

Evaluation & Results

Test Loss and Accuracy

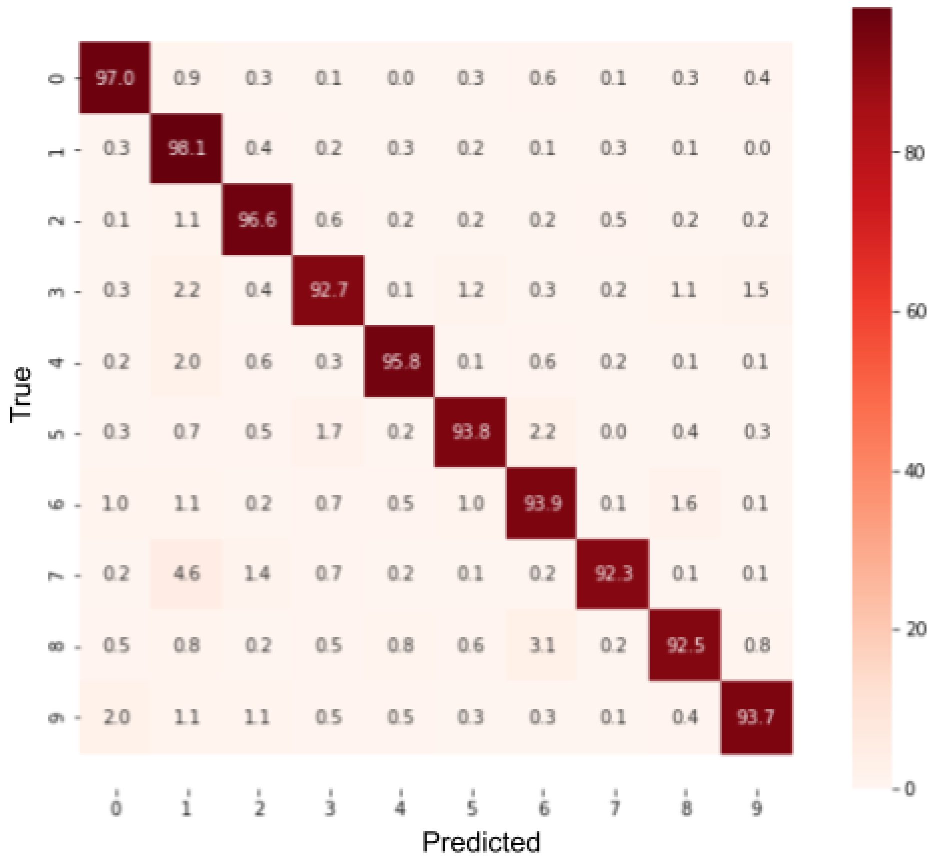
The metric we used to compare the models were test loss and test accuracy. Our chosen model had the following results:

[5] Stack Overflow: https://stackoverflow.com/questions/47312219/what-is-the-definition-of-a-non-trainable-parameter

Test Loss	0.20034723480677258
Test Accuracy	0.9516748617086662

We see that the chosen model achieved a ~3% increase from the base model, which is a significant improvement.

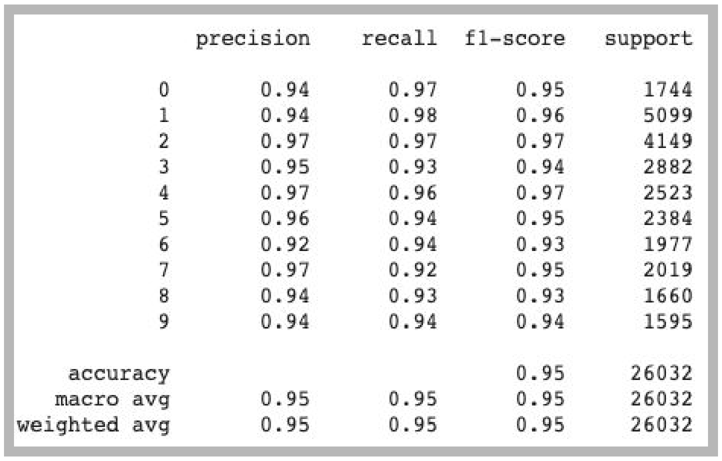
Confusion Matrix



True \ Predicted	0	1	2	3	4	5	6	7	8	9
0	97.0	0.9	0.3	0.1	0.0	0.3	0.6	0.1	0.3	0.4
1	0.3	98.1	0.4	0.2	0.3	0.2	0.1	0.3	0.1	0.0
2	0.1	1.1	96.6	0.6	0.2	0.2	0.2	0.5	0.2	0.2
3	0.3	2.2	0.4	92.7	0.1	1.2	0.3	0.2	1.1	1.5
4	0.2	2.0	0.6	0.3	95.8	0.1	0.6	0.2	0.1	0.1
5	0.3	0.7	0.5	1.7	0.2	93.8	2.2	0.0	0.4	0.3
6	1.0	1.1	0.2	0.7	0.5	1.0	93.9	0.1	1.6	0.1
7	0.2	4.6	1.4	0.7	0.2	0.1	0.2	92.3	0.1	0.1
8	0.5	0.8	0.2	0.5	0.8	0.6	3.1	0.2	92.5	0.8
9	2.0	1.1	1.1	0.5	0.5	0.3	0.3	0.1	0.4	93.7

The highest rate of misclassification arose with 7s and 1s, with the model very frequently misclassifying 7s as 1s. This is probably because the dataset was naturally biased towards having more 1s overall, so the model disproportionately chose 1s as the labels for unknown training data. Interestingly, 1s were not really misclassified as 7s by the model, indicating that the model learned one side of the feature discrimination much better than the other.

Precision, Recall, and F1 Scores per Class



```
              precision    recall  f1-score   support

           0       0.94      0.97      0.95      1744
           1       0.94      0.98      0.96      5099
           2       0.97      0.97      0.97      4149
           3       0.95      0.93      0.94      2882
           4       0.97      0.96      0.97      2523
           5       0.96      0.94      0.95      2384
           6       0.92      0.94      0.93      1977
           7       0.97      0.92      0.95      2019
           8       0.94      0.93      0.93      1660
           9       0.94      0.94      0.94      1595

    accuracy                           0.95     26032
   macro avg       0.95      0.95      0.95     26032
weighted avg       0.95      0.95      0.95     26032
```

The table above shows the different metric scores for all the unique classes in the dataset. Overall the performance is pretty good.

For precision, we are examining the number of correctly predicted digit labels out of all predicted digit photos for that class. For example, the precision of 2 would be the number of correctly predicted 2’s out of all predicted 2’s[6]. We see that 2, 4, and 7 had the highest precision (0.97) and 6 had the lowest precision (0.92).

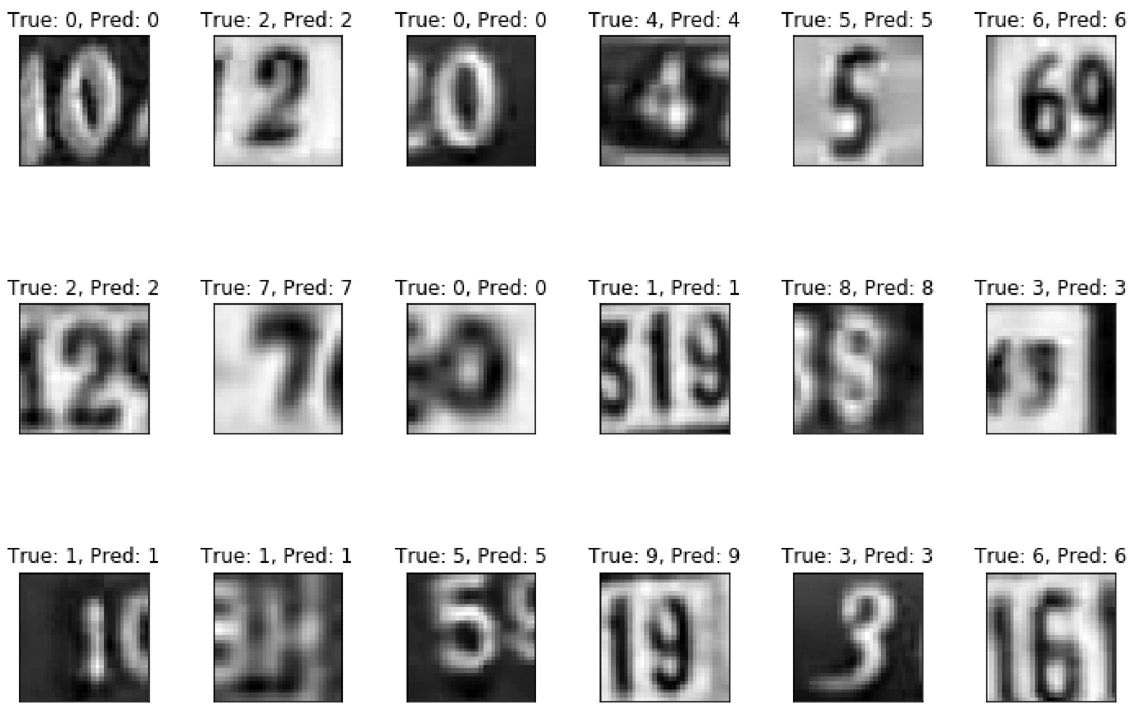
For recall, we are examining the number of correctly predicted digit labels out of the number of actual digit images. For example, the recall of 2 would be the number of correctly predicted 2 photos out of the number of actual 2 photos. We see that 7 had the lowest recall with 0.92 and 1 had the highest recall of 0.98.

The highest F1 score was found for the digits 2 and 4, whereas the model struggled the most with classifying 6 and 8. This is not very surprising as 6 and 8 look very similar, and in fact are most frequently misclassified with each other (from the confusion matrix data). 2 and 4 are both pretty unique looking digits so the model had little trouble classifying them.

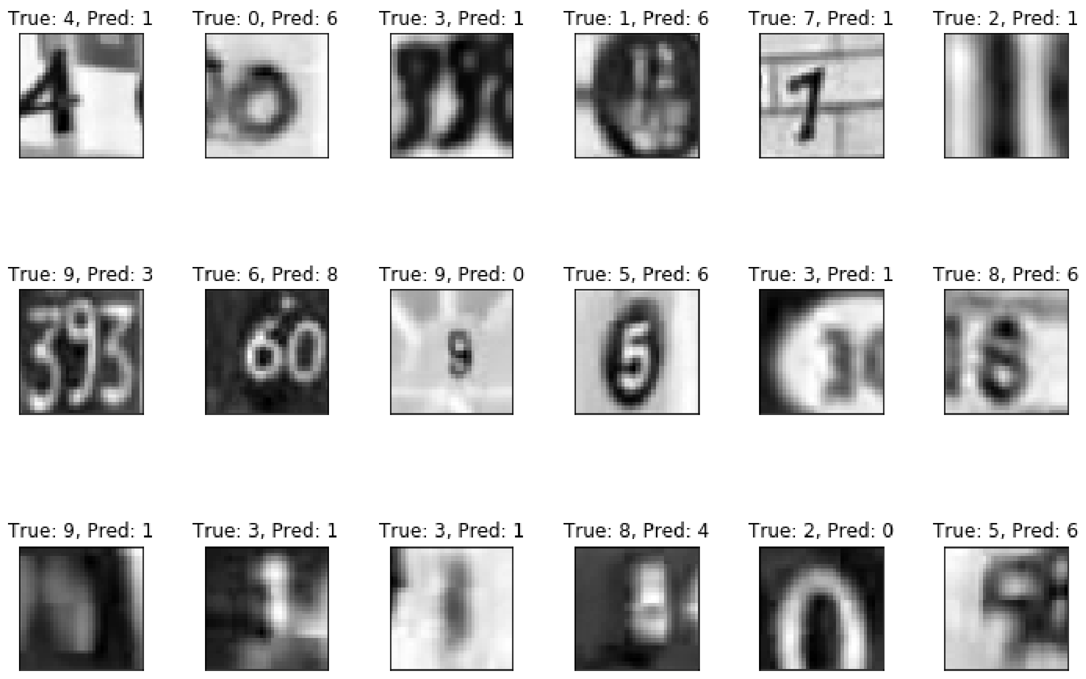
Correct and Incorrect Samples

We wanted to examine further the samples the model correctly and incorrectly predicted. Below is a sample of correctly labeled images:

[6] Shmueli, Boaz, Towards Data Science, https://towardsdatascience.com/multi-class-metrics-made-simple-part-ii-the-f1-score-ebe8b2c2ca1



Below is a sample of incorrectly predicted images:



Findings

Training Data Augmentation

The best way to improve the accuracy of our model was by feeding it more training data. This was successful because smaller training dataset sizes cause the model to overfit. So the introduction of new samples allowed the model to better learn the distinguishing features between digits. Even though the bigger dataset consisted of "easier" samples, they still allow for enhanced feature detection by the neural network. Furthermore, having more data allows you to build a more complicated model with more interesting architecture because there is more data from which the parameters can be updated. This is especially true considering that we were able to nearly double the amount of training data with the introduction of extra images. The extra images were preprocessed in the same way as the original training data to ensure consistency. The results from the earlier sections showed that with each 50,000 increase in training, there is a ~2% increase in accuracy.

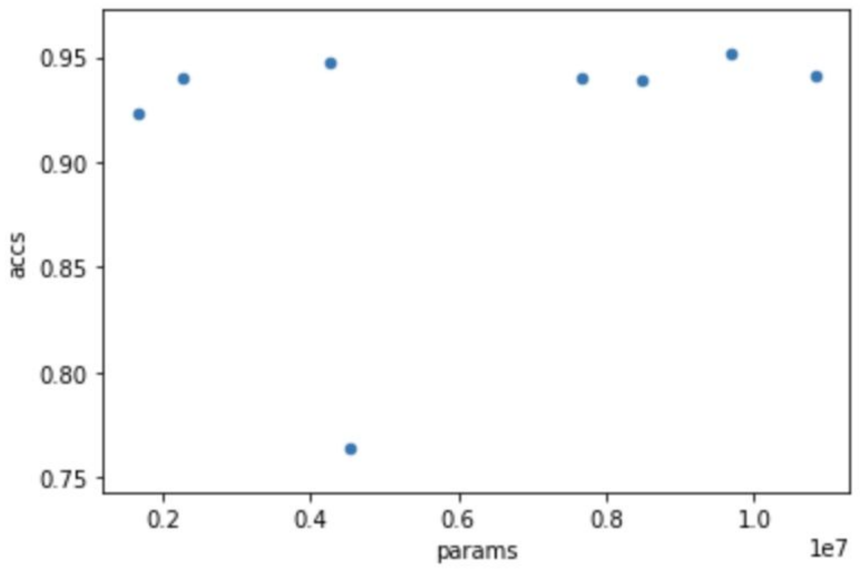
Network Architecture

We experimented with a variety of network sizes and depths. At first, we tried using the pretrained VGG16 model. However, we found that the performance was worse than our simple base model. This may be because the VGG16 model weights are too generalized for our dataset, as the VGG16 model is trained on the ImageNet dataset which contains far more than just digit images. We then tried implementing Goodfellow's 8-layer model. We found that this model's performance was also poor because we did not have enough data to fully train the model; in Goodfellow's paper, he mentions that he trains for 6 days. We then went back to our base model and starting adding layers until the performance stopped improving; we found that 4 layers was the best. This can be explained by the fact that smaller, shallower networks are better at memorizing features but not so good at generalization. Multiple layers are better at generalizing, but need lots more data to learn underlying structure instead of overfitting[7].

We were also interested in how the total number of parameters in the model influenced the test accuracy. To help summarize our findings, we created a plot (shown below) of the number of parameters (in ten millions) vs the test accuracy.

[7] Stats StackExchange: https://stats.stackexchange.com/questions/222883/why-are-neural-networks-becoming-deeper-but-not-wider

Number of Parameters vs Test Accuracy



The number of total parameters was of particular interest because it illustrates the potential of your model to be overfit on the training data; having an excess of model parameters than features in the dataset is highly indicative of overfitting. This plot illustrates how the number of model parameters usually doesn't influence test accuracy too much, and tells us that we do not need to simply increase the number of parameters and overfit just to increase accuracy. We should prioritize finding a model that has a lower number of overall parameters but still performs fairly accurately, as a less complicated model is more likely to understand the general features of the dataset and perform better when presented with unknown data.

Filter Size and Stride Length

The size of the images in our dataset was fairly small (32x32), so building a deeper network required the use of more padding to maintain the necessary amount of information for convolution. We first started with the same filter (5x5) size and maxpool stride (alternate 1x1 and 2x2) length that was used in the Goodfellow paper, achieving test accuracy of 0.936616. We then attempted to increase the filter size, giving us a test accuracy of 0.936 and test loss 0.282. We then decreased the filter size and got a test accuracy 0.927 and test loss of 0.300. We hypothesized that increasing the stride length with model depth would allow the model to focus on larger, more significant features as it learned more of the data. However, this did not improve performance. As a result, we decided to stick with the Goodfellow architecture and keep a filter size of 5x5 and alter the maxpool stride lengths from 1x1 and 2x2 at each layer. The paper also mentions that this helps with preserving spatial information, as only the 2x2 layers actually lose information when the data is fed through. Overall, these findings are not surprising given that filter size and stride are pretty finely tuned parameters that depend more on the characteristics of the classification objects (digits) rather than how we preprocessed the data differently from Goodfellow or which sample of training data we chose to use. As a result, identical parameter values will yield similarly strong performance.

Optimizers

Another we noticed when training the models was the superior performance of the Adam optimizer when compared with SGD. This came as a bit of a surprise as literature suggests that SGD (in general) tends to perform better in terms of finding an optimal solution when compared to Adam[8]. However, Adam does tend to find a solution more optimally for sparse data and converge quicker in general. For our particular dataset, the Adam optimization solution (Test accuracy: 0.952) consistently performed better when compared to the SGD optimization (Test accuracy: 0.948) for identical model architectures. This could be because Adam is better suited for quick convergence, and when trained on such few epochs, the SGD optimizer does not have enough time to learn the data.

Batch Normalization and Dropout

We implemented batch normalization and dropout in order to prevent the overfitting we found in the earlier models. As the data flows through the neural network, the weight values are constantly being updated and changed to fit the data. However, this can cause the values to skew too big or too small as the algorithm progresses. Batch normalization functions to renormalize the dataset at intermediate steps in the neural network so that we can use higher learning rates and the data can converge faster. By resetting the weight values to a scaled version, we also allow the model to more accurately learn the underlying structure of the data, thus avoiding overfitting.

Dropout is another technique we used in order to prevent overfitting of the data[9]. It functions as a way to mimic the building of multiple neural networks with differing architectures. This is accomplished by randomly dropping out nodes, by not updating them, during training in the hopes of better learning the data while improving generalization error. In conjunction with dropout, it is highly desirable to feed in more training data because the dropout is essentially 'losing' some of the learned behavior at each step, so starting off with more available information to learn helps the model perform better. Without dropout, models often run into the problem of correcting for previous layers by adapting their weights to mask the mistakes of previous layers. These quickly leads to overfitting because the layers are self fulfilling and making bandage adjustments to improve accuracy rather than learning the underlying structure. By introducing dropout, we break up these situations and make the model more robust.

Overall, as discussed earlier in the methodology comparison results section, the model with both batch normalization and dropout performed the best with the least amount of overfitting.

[8] A. Wilson, R. Roelofs, M. Stern, N. Srebro, B. Recht, arXiv:1705.08292

[9] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov; 15(Jun):1929−1958, 2014.

Future Work

A future task we could implement is augmenting our training data set via image transformations. This would include rotations, stretches, shifts, and salt & pepper flakes. Augmenting our training data set in this way would artificially increase the size of our dataset with unique samples, which we already saw greatly enhanced the performance of our model. Not only is there more data, but this data would be more complicated and introduce greater distractors so the model can better learn the most important features to classify digits.

Another future task would be to continue experimenting with other parameters such as batch size and learning rate. During the scope of our project, we did not have time to experiment with these parameters. Learning rates are highly correlated with the optimizers that we tested, and trying out newer or more robust optimizers could result in better convergence. Fine tuning of the batch size, learning rate, momentum, and initialized weights can be implemented using sklearn's GridSearchCV and cross-validation. We could also further tune the model to prevent overfitting via adjustment of batch normalization layers and dropout rates. We already began to see more generalized fits with the introduction of default batch normalization/dropout, but sweeping these parameter values over a range could allow us to better tune the model.

Another future task is implementing sliding detection to read in and label non-centered images. The actual SVHN dataset comes in as raw uncentered images, and a majority of the papers and projects we read on this started by implementing the sliding detection into the learning parameters of the algorithm. By doing so, some researchers were able to build demos displaying real time probabilistic determinations of digits in large, arbitrary real-world images. For the purposes of our project, we believed it would be most interesting to focus our efforts on building a network to perform digit classification on already centered images. Sliding detection also allows for this project to extend far beyond street view house numbers, and instead move towards arbitrary classification of numbers in any image (license plates, credit cards, etc.). Furthermore, building a model that can naturally parse out relevant information from distractors allows for potential classification of both numbers and letters in arbitrary images.

Extension: Smaller Model

After our poster presentation, we were presented with a challenge to simplify our chosen model to have half the number of parameters. At first we dropped one convolutional-maxpool layer, but that actually resulted in more parameters (10,944,106). This is because of how the total number of parameters for fully connected layers depends on the previous layer[10]. When we

[10] Rakshith Vasudev, 11 Feb. 2019, https://towardsdatascience.com/understanding-and-calculating-the-number-of-parameters-in-convolution-neural-networks-cnns-fc88790d530d

broke up the architecture, we actually exposed a larger layer to the first fully connected one, so the first fully connected layer shot up in number of total parameters. We then decided to leave the convolutional layers as they are, but modify the fully connected layers at the end because those seemed to always have the largest number of params. We took out the fully connected layer of 1024 nodes and the dropout (prob = 0.5) layer, and we reduced the fully connected layer of 512 nodes to 256. After these modifications, we had about half of the parameters that the original model had.

The following table outlines in further detail the architecture of the two networks. Model G is the chosen model and model H is the reduced model.

Model	Description	Architecture	Total Num. of Params.
G	Same as F, but with dropout and batch-normalization (poster)	Conv-48-5-1 → MaxPool-2-2 → Conv-128-5-1 → Batch-Norm → Dropout-0.1 → MaxPool-2-1 → Conv-256-5-1 → Batch-Norm → Dropout-0.1 → MaxPool-2-2 → Conv-256-5-1 → Batch-Norm → Dropout-0.1 → MaxPool-2-1 → FC-1024 → Dropout-0.5 → FC-512 → FC-10	9,700,202 (1,280 untrainable)
Smaller CNN	Model after presentation feedback	Conv-48-5-1 → MaxPool-2-2 → Conv-128-5-1 → Batch-Norm → Dropout-0.1 → MaxPool-2-1 → Conv-256-5-1 → Batch-Norm → Dropout-0.1 → MaxPool-2-2 → Conv-256-5-1 → Batch-Norm → Dropout-0.1 → MaxPool-2-1 → FC-256 → FC-10	4,256,874 (1,280 untrainable)

The following table outlines the performance of the two models:

Model	Description	Test Loss	Test Accuracy
G	Denser, 4 layers, more fully connected layers, batch normalization and dropout	0.20034723480677258	0.9516748617086662
Smaller CNN	Denser, 4 layers, more	0.2429417632493635	0.9479486785494776

We see that performance is comparable; the difference in test accuracy is ~0.004, which suggests that the performance is about the same but with the chosen model still performing better. The test loss in model G is also slightly better (by 0.04).

The F1 Scores of Model H are reported below:

```
              precision    recall  f1-score   support

           0       0.94      0.94      0.94      1744
           1       0.96      0.97      0.96      5099
           2       0.96      0.96      0.96      4149
           3       0.95      0.92      0.93      2882
           4       0.97      0.96      0.97      2523
           5       0.92      0.96      0.94      2384
           6       0.96      0.92      0.94      1977
           7       0.93      0.96      0.94      2019
           8       0.95      0.90      0.93      1660
           9       0.90      0.93      0.92      1595

    accuracy                           0.95     26032
   macro avg       0.94      0.94      0.94     26032
weighted avg       0.95      0.95      0.95     26032
```

Once again, this model performed very well in classifying 4s (0.97 F1 score). Interestingly enough, this more generalized model struggled the most with 3s (0.93 F1 score), 8s (0.93 F1 score), and 9s (0.92 F1 score). All of them have pretty related shapes so this does not come as too much of a surprise. Humans would struggle to delineate between these numbers just like our model does. It is interesting to note that the model performed so well in classifying 4s, even though 4s could potentially be written in multiple different typefaces with pretty drastically different structures.

References

1. A. Wilson, R. Roelofs, M. Stern, N. Srebro, B. Recht, “The Marginal Value of Adaptive Gradient Methods in Machine Learning”, https://arxiv.org/abs/1705.08292

2. Goodfellow, Ian J. et al. (2013). “Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks”. In: arXiv:1312.6082 [cs]. arXiv: 1312.6082. url: http://arxiv.org/abs/1312.6082.

3. J. Brownlee, Machine Learning Mastery, https://machinelearningmastery.com/reproducible-results-neural-networks-keras/

4. Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov; 15(Jun):1929−1958, 2014. (http://jmlr.org/papers/v15/srivastava14a.html)

5. Rakshith Vasudev, 11 Feb. 2019, https://towardsdatascience.com/understanding-and-calculating-the-number-of-parameters-in-convolution-neural-networks-cnns-fc88790d530d

6. Restrepo, Ronny, Real-Time Multi-Digit Recognition and Localization, February 2016, http://ronny.rest/portfolio/multi_digit

7. Sharma, Aditya, SVHN-CNN, March 2018, https://github.com/aditya9211/SVHN-CNN

8. Shmueli, Boaz, Towards Data Science, “Multi-Class Metrics Made Simple, Part 1”, https://towardsdatascience.com/multi-class-metrics-made-simple-part-ii-the-f1-score-ebe8b2c2ca1

9. Stack Overflow: https://stackoverflow.com/questions/47312219/what-is-the-definition-of-a-non-trainable-parameter

10. Stats StackExchange: https://stats.stackexchange.com/questions/185853/why-do-we-need-to-normalize-the-images-before-we-put-them-into-cnn

11. Stats StackExchange: https://stats.stackexchange.com/questions/222883/why-are-neural-networks-becoming-deeper-but-not-wider

12. Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, Andrew Y. Ng Reading Digits in Natural Images with Unsupervised Feature Learning NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011. (http://ufldl.stanford.edu/housenumbers/)

Considerations

We changed our project to the Street View House Numbers on Monday, November 25, 2019. Before changing our project, we had made significant work on other project proposals as detailed below.

Movie Reviews

Our original project proposal was to predict box office success of movies based on movie data. This would include features like genre, actors/actresses, budget, directors, music, or release date to uncover what governs a movie's revenue.

Before switching our proposal, we had gathered, cleaned, and prepped all the data. We first downloaded 4 csvs files of review data from GroupLens, and cleaned and joined all of the csvs. We then used the IMDb movie ID from the GroupLens data to query movie information from the OMDb API. We then joined the OMDb and IMDb dataframes. Lastly, we scraped production cost and revenue numbers from thenumbers.com. We then cleaned and joined the data with the OMDb and GroupLens data to create our entire dataset.

After cleaning and preparing all the data, we did some initial EDA. We found that there was a weak positive linear trend between production cost and domestic gross (Figure 1). We found that domestic and worldwide gross are strongly, positively correlated (Figure 2). We also found that there was no apparent correlation between imdb rating and domestic gross (Figure 3).

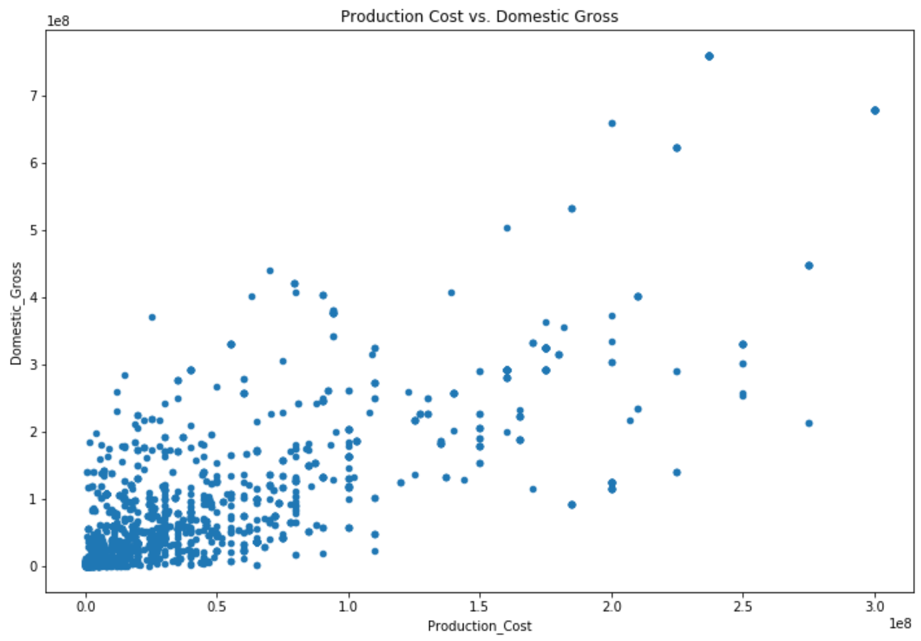


Figure 1

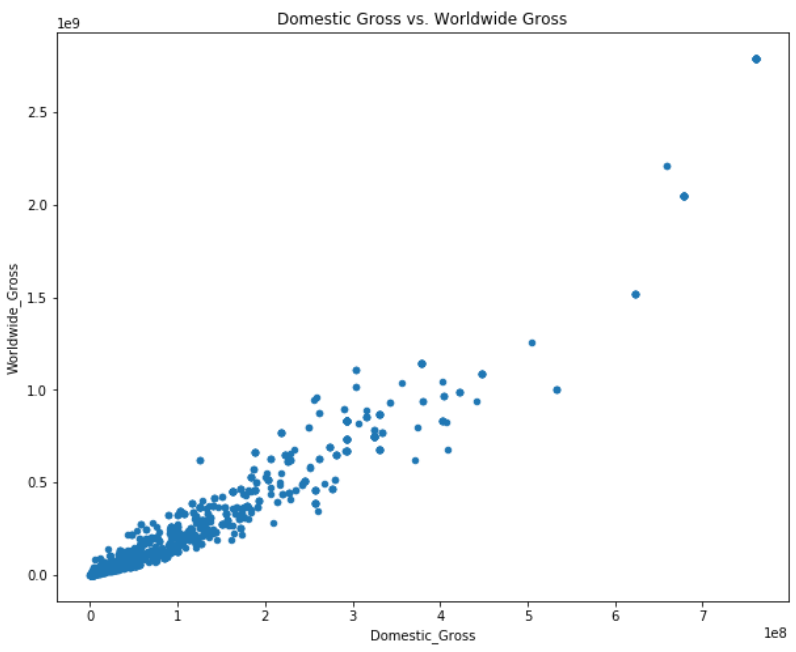


Figure 2

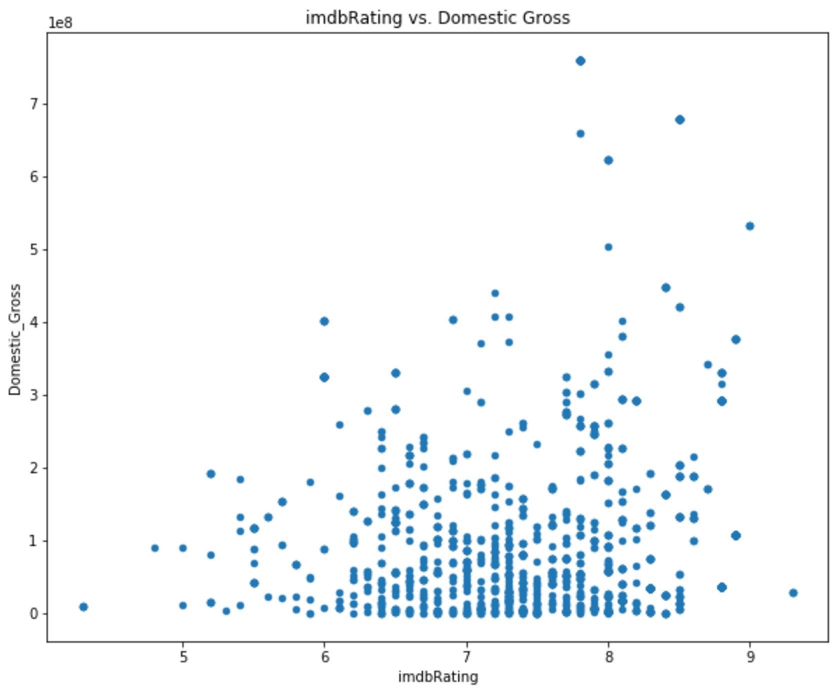


Figure 3

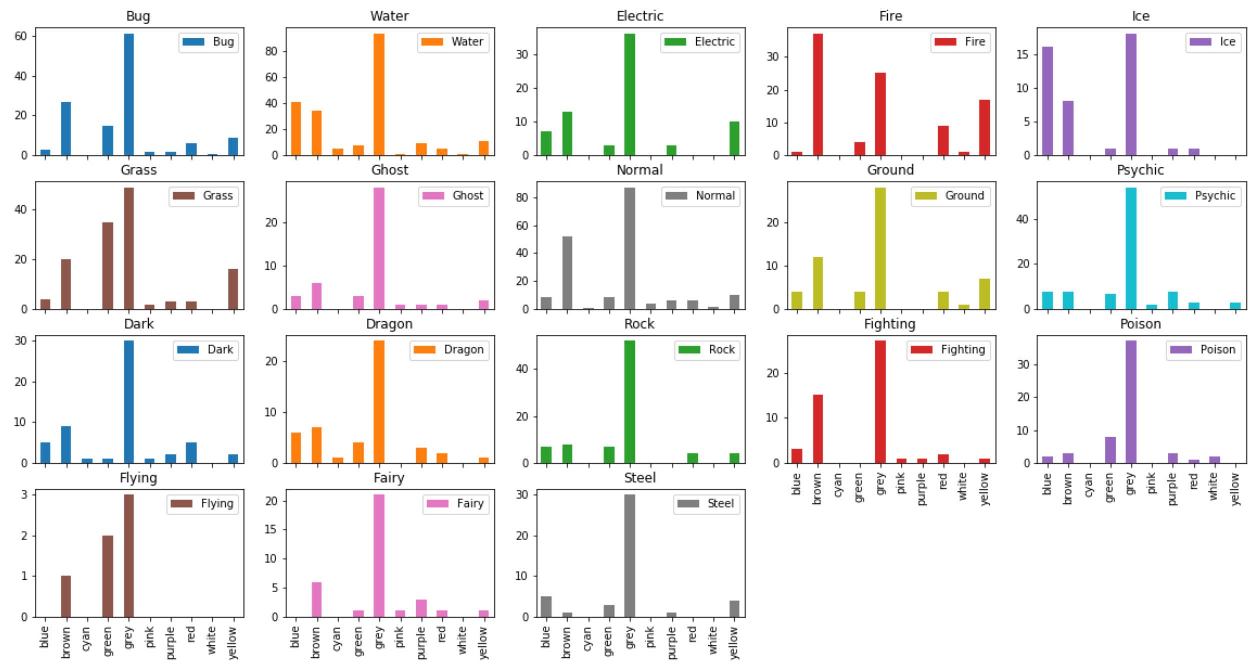
After meeting with Professor Subramanian, we decided to modify our project proposal to a movie recommendation model. However, after looking back at the lab 9 classwork and homework, we realized we had already worked on the exact same problem. Our new project would use the same model but with a bigger dataset. Because we felt like this wasn't very interesting, we decided to explore different problems and datasets.

Pokemon

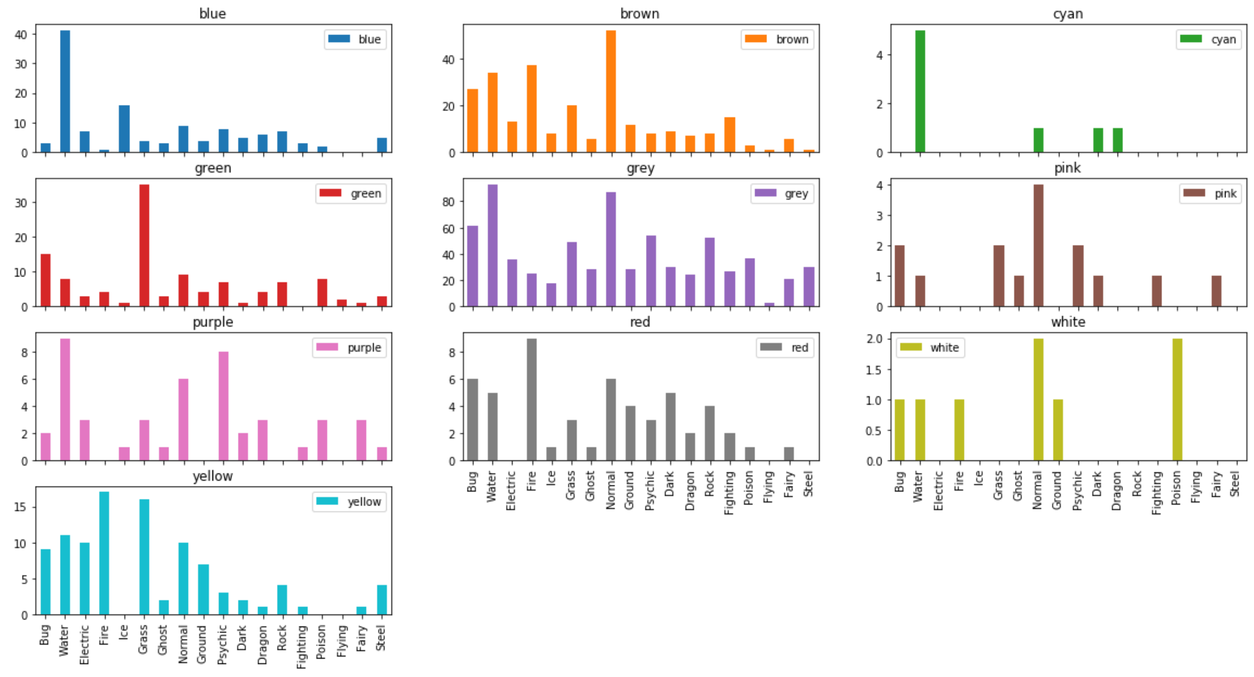
Our second project proposal was to determine whether a Pokemon's type can be determined from an image of the Pokemon, or from the statistics of the Pokemon. This project was motivated by the recent release of a new Pokemon installment.

For this project, we first downloaded a dataset of pokemon images and pokemon statistics. We then worked on image pre-processing. We used a library called colorThief to get the top 3 colors per image; this library returned the colors as an RGB vector. We then used webcolors to convert the RGB vector into the closest categorical color. Lastly, we scraped w3schools to get a mapping from webcolor to general color. (ex: cornflowerblue to blue). We then grouped the pokemon by type and counted the number of times each color appears. We then graphed the distribution of colors per type, and the distribution of types per color.

Color Distribution per Type



Type Distribution per Color



We then met with Professor Subramanian again to discuss the project proposal. In the Pokemon universe, there are only 800 Pokemon, which is too small of a dataset for a neural network to properly train. Thus, we decided to change our project proposal to the Street View House Numbers.