Final Project: Classification and Detection with Convolutional Neural Networks Discussions

Michael Ryan Henderson
mhenderson43@gatech.edu

Presentation
https://drive.google.com/file/d/10MrnPDPqptSUBqTlfF_8GK4cSxxetdzj/view?usp=sharing

Sample video
https://drive.google.com/file/d/1KNVK7oh87inYjQGjdgNVVFrB9e0SLMKK/view?usp=sharing

Overview
Two different networks were tested, a simple network with convolution layers with batch normalization and relus, and FractalNet. FractalNet has an elegant design compared to many other state-of-the-art networks. It is also a wider network, which have been shown to increase performance without increasing depth of the network. Less depth allows for faster batch processing due to increased parallelization. This FractalNet implementation also contains local drop path. This prevents co-adaptation of feature detectors

The final implementation has one class per digit with a threshold. If none of the classes are above the threshold, it is considered “no digit”. This was opposed to using eleven classes with the last class indicating no digit. The threshold is an additional hyperparameter that can be tuned.

Pipeline Overview

Preprocessing
Images were centered and normalized using standard deviation during training and while testing real images. Centering and normalizing will mitigate the effects of different lighting conditions. There was no additional preprocessing performed, such as finding edges, points of interest, etc.

Noise Management
Noise in training samples was not removed. This will help to make the network more robust to noise without preprocessing noise. Real images were passed through a denoising function prior to prediction.

Location, Rotation and Scale invariance
The network was trained using the ImageDataGenerator to provide rotation and scaling during training. This made the network more robust to these changes. On real images, a sliding window and image pyramid were used, providing additional robustness to the pipeline

Model Variation

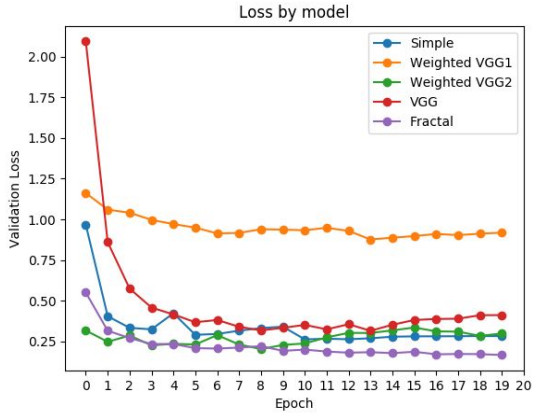
I tested four networks- VGG16, VGG16 pretrained, simple network (3 conv levels with batch normalization between each), and a FractalNet with a single block.

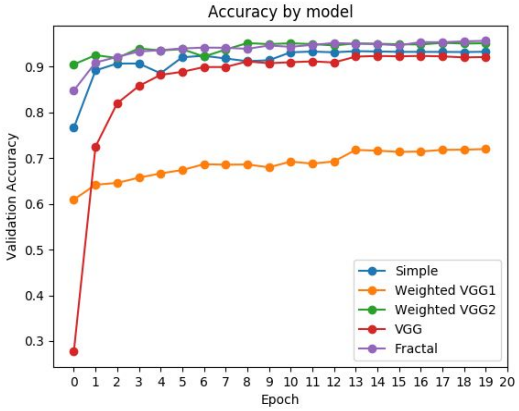
These four networks were tested on 11 classes where one label represented “no digit”. The best network was then used to classify real data.

For VGG with pretrained weights, the weights prior to the fully connected layers were attempted both frozen and not frozen (these are marked Weighted VGG1 and VGG2 in the graph). Allowing the layers to continue to train resulted in a lower loss. This could be attributed to the model continuing to optimize to this dataset. The fully connected layers were fully trained.

The fully connected layers in the VGG are used to classify the data. Since the data that was classified with the original weights were different than the classes in this project, they were not reused.

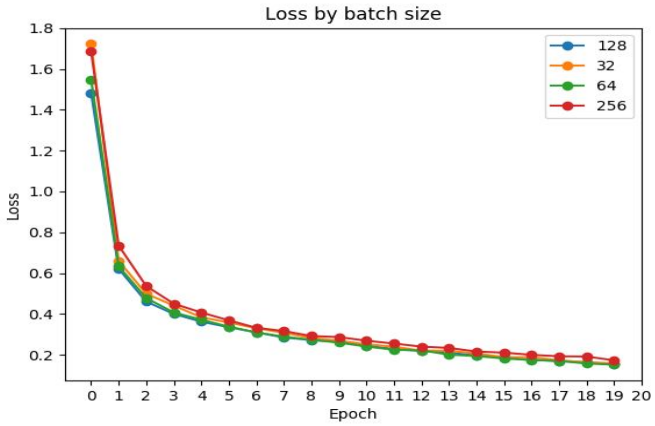
VGG without pretrained weights required the number of samples in each class to be very close. If one of the classes makes up a disproportionate number of samples, the VGG will fall into a local minimum that classifies everything as this class.





Batch Size

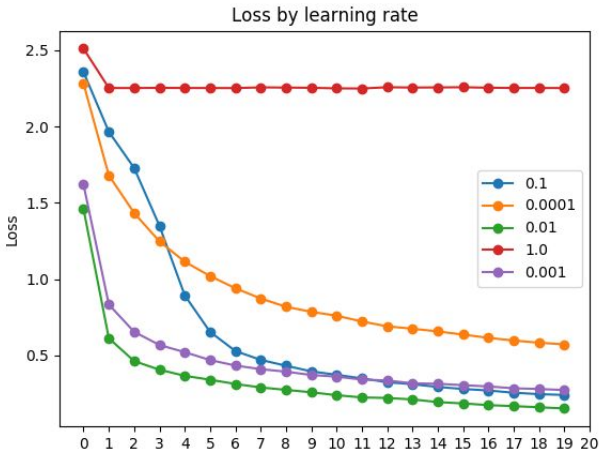
The batch size is how many sample images are tested against labels for each update to the model weights. I settled on a batch size of 128. There is a tradeoff between number of steps which adjust weights, learning time, and the amount of normalization per batch. Larger batch size allows each epoch to complete faster and more robust batch



normalization. From the graph at right, the 128 is slightly better than 64, was about 50% faster. 256 was slightly faster than 128 but does not learn as quickly

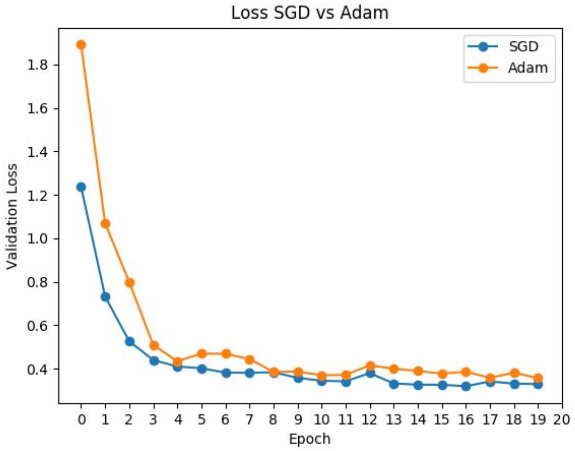
Learning Rate

The learning rate is how large the update to weights is with each step. Several learning rates were tested and 0.01 gave the best initial rate for the FractalNet model. On the extended training for real images, the code used a callback to reduce the learning rate once it plateaued. Switching to the lower learning rate once it plateaued allowed the model to get into the narrower parts of the “valleys” of global minima.



Optimizers

Optimizers decide which direction (via weight updates) will move the network towards zero loss. SGD with momentum and Adam are two common optimizers. The models were originally tested with Adam and had validation loss and accuracy that was much worse than with training. A research paper has shown that Adam generalizes worse than SGD. Switching to SGD caused the difference between training and validation to shrink significantly. VGG didn’t converge with SGD (it stalled at 0.39 accuracy), so VGG was trained with Adam.



Performance Considerations

The simple convolution network with relu activation and batch normalization between three convolution layers, as well as the FractalNet, both performed better than the VGG models. The assumption is that the batch normalization has a large effect on performance because the VGG model was developed prior to batch normalization.

During training, all networks were allowed to run for up to 20 epochs, but were stopped after 10 consecutive epochs of no improvement in loss if that occurred earlier.

Using 11 classes, where one class represents no digits, had very good accuracy on training data, but did not perform very well on real data. There were a lot of false positives. This was likely due to the use of softmax, where one class was chosen regardless of how strongly it was classified. See example below

11 classes is the left image and 10 classes with thresholding is the right image

Updating the model to use 10 classes with a sigmoid activation on the last layer allowed for a threshold hyperparameter on the last layer. If none of the classes were above the threshold, it would be considered no digit.

For the sliding window, the implementation used the method of convolving over a larger area to act as the sliding window. This technique is discussed in Andrew Ng's lecture on CNNs. This technique reuses the calculations from a CNN layer between the sliding window positions, rather than recalculating all values each time. This resulted in roughly a 50x improvement in performance from doing the sliding window using this method. The pipeline uses an image pyramid to run the sliding windows at different scales. The drawback to this method is that same padding doesn't work, so each convolution layer reduces the window size, limiting the max depth by an order of magnitude. It may be possible to switch to same padding at some point in the model, but this was not resolved during this project.

Discussion

With eleven classes, the models used softmax activation with categorical crossentropy loss to force a single class for each label. With ten classes, the models use sigmoid activation function with a customized binary crossentropy so each class was activated independently. The loss function was customized because binary crossentropy loss function weighting would favor classifying all outputs as 0 to achieve 90% accuracy, which caused the training to get stuck in a local minimum. The solution was to weight 1 labels more heavily than 0 labels.

The implementation uses convolution to slide the windows. This provided roughly a 50x improvement in speed on large images. Convolutional sliding windows had the drawback of requiring the network to use valid padding instead of same size. This reduces the potential number of layers and columns in the fractal network. Ideally, the model would use recursion for sliding with same padding.

This implementation had some differences from state-of-the-art solutions. The bounding boxes could be calculated as part of the network or an even more advanced technique like YOLO.

Also, the thresholding parameter could be tuned more optimally, potentially with different thresholds for each class.

There are other contextual improvements that could be made as well. There are many cases where the letters I or O are mistaken for the numbers 1 or 0. Also, vertical edges are often mistaken as 1s.

The size of the input could be increased while maintaining the dimensions of the digit, while using images that have other digits around the labeled digit. This could train the network to expect digits to be next to other digits. Also, using negative training examples containing letters would train it to ignore letters like A that can be mistaken as a 4.

Citations

FractalNet: https://arxiv.org/pdf/1605.07648.pdf
Convolving sliding window: https://www.coursera.org/learn/convolutional-neural-networks/lecture/6UnU4/convolutional-implementation-of-sliding-windows
Adam vs SGD: https://arxiv.org/abs/1705.08292
Batch normalization: https://arxiv.org/pdf/1502.03167.pdf