Classification and Detection with Convolutional Neural Networks: CS 6476

Josh Adams
jadams334@gatech.edu

***Abstract*—This report will show the creation of a machine learning pipeline which will be able to detect and classify housing numbers. The dataset used is the Street View House Numbers (SVHN) dataset (Netzer et al, 2011), which consists of over seventy-three thousand training images and over twenty-six thousand testing images. The pipeline will use a convolutional neural network (CNN), maximally stable extremal regions (MSER) and non-maximum suppression (NMS) to produce a robust solution for detecting and classifying house numbers.**

1 EXISTING METHODS

Sambasivam discusses ways to handle imbalanced datasets in their article "A predictive machine learning application in agriculture" such as Synthetic Minority Oversampling Technique. Synthetic Minority Oversampling Technique (SMOTE), (Krawczyk, 2016) is a technique used to reduce the negative impact of imbalanced datasets. A variation of SMOTE was implemented which does not continually resample, rather reduced the sampling of the over-sampled classes. This was implemented by finding the lowest represented class and randomly sampling the remaining classes with replacement, limited by the number of samples in the minority class.

2 SVHN DATASET

The Street View House Numbers (SVHN) dataset was obtained from Google Street view and contains over 600,000 images (Netzer et al, 2011). The dataset is broken up into three parts, training, testing and extra. The training set has over seventy thousand samples and the testing has over twenty-five thousand samples. A script was created to extract just the numbers from each of the provided images as well as extract negative examples from each image. The negative examples were generated by splitting up the image into many segments. Any segment which shared information with any of the digits, was discarded. For each image, three of the remaining segments were randomly chosen and added to the training set. All images in the training set and testing set were resized to be sixty-four pixels in height by thirty-two pixels in width. The training set was then artificially augmented in three ways. The first was adding gaussian noise to each sample, a rotation of fifteen degrees in both positive and negative directions. This resulted in a training set which contained almost six times the number of samples as the original. The reason for doing these augmentations was to make the network invariant to noise and pose. The dataset was very imbalanced with label one having nearly three times the samples of the label with the least. To resolve this issue a function was created to randomly sample equally from the distributions between the labels. It worked by finding the label with the least number of samples, it would establish that as the maximum number of samples used for all other labels. For each label, the maximum number of samples were obtained through randomly sampling with replacement. This would generate a dataset which was evenly distributed and randomly sampled, prior to the function returning this dataset, it would shuffle all of the samples as to not introduce any biases into the set.

3 CONVOLUTIONAL NETWORK OVERVIEW

Two different CNN's were trained, compared and the best performing network was used in the final pipeline. VGG16 (Simonyan et al, 2014) and LeNet5 (LeCun et al, 1998), both involve the use of convolutional layers. Both networks required modifications which would allow them to be used on the SVHN dataset. The networks were trained for fifty sessions where each session would have two epochs. Each epoch would have at least twenty thousand samples and for every session the number of samples in each epoch was increased by five hundred. Stopping training early can help to counteract the networks abilities to overfit the dataset. Training was taking an exceedingly long time and VGG16 was above 93% testing accuracy very quickly, while LeNet5 was not making noticeable progress for many sessions, thus training was stopped after fifty sessions. Looking at the learning curves in *figures one, two* would suggest LeNet5 should continue to train as the divergence of the training and testing accuracy has yet to reveal itself. The batch size determines the number of samples the network is trained on at one instance. Each epoch consisted of at least twenty-thousand samples and these samples were broken up into mini-batches of thirty-two samples. The reason thirty-two was chosen was to minimize memory usage, to help with the stability of the network as well as help to converge faster. The way a small batch size helps with converging faster is that the networks weights are updated more frequently. The learning rate determines how much impact each individual weight update has. Lower learning rates increase the training time but also in cases like VGG16 help to slow the rate at which it begins overfitting. Having a complicated network like VGG16, the lower learning rate helps the network avoid local minima. The optimization method used was 'Adam' for both networks, which is a variant of stochastic gradient descent. The main difference that comes with 'adam' is the adaptive learning which uses an adaptive moment estimation using running average of gradients and second moments of gradients (Kingma 2014).

3.1 VGG16

The VGG16 network consists of five stages of convolution layers then pooling layers. The final three layers are fully connected. The modifications required were the input and output layers. The input images to the networks by default were 224x224x3, due to the changes made in the preprocessing of the datasets, the input had to be changed to accept a 64x32x3 image. The output of the network was developed with the idea to be used with the 'imagenet' dataset which has over one thousand labels. The SVHN dataset has ten labels, zero through nine. The preprocessing of the dataset added a label for 'non-digit' images, the output would need to have eleven total outputs. The weights for the VGG16 network were initialized using the weights previously trained on the 'imagenet' dataset because the 'imagenet' dataset is much larger and has many more labels. Each layer's weights were set to be trainable as such the weights were trained over during training.

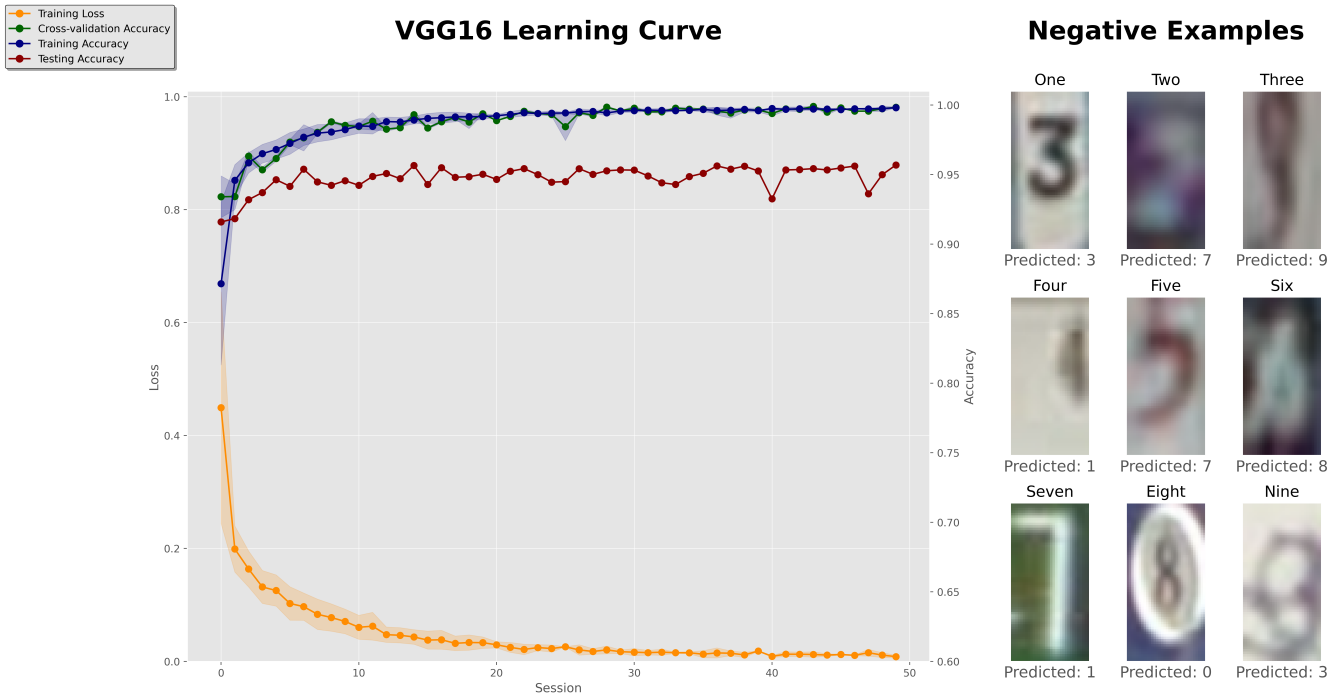

Figure 1 — Learning curve and negative examples of the VGG16 network.

The VGG16 network used a learning rate of 0.00005. After each epoch, five percent of the data was used in validation and that accuracy was recorded. The results for the validation during each session were averaged and shown in *Figure 1*. VGG16 is a deep convolutional network which was able to capture the variance within the training set quickly, as shown by the high accuracy in both the training set and cross-validation sets. VGG16 was able to get 95.7 % accuracy on the unseen testing set. This is near the limit of accuracy for this network with the datasets used, as shown in *figure 1*, by the low and unchanging loss as well as the high and unchanging accuracy on both the training and cross-validation sets.

Figure 1 also shows some examples where the network predicted an incorrect label. Image four was incorrectly selected in the SVHN and part of the four was cut off, this led the network to predict a seven. Other images such as image seven, clearly has a seven in it but that seven is also surrounded by a circle, this adds difficulty for the network to decipher between a zero and a seven.

3.2 Modified LeNet5

The standard architecture of LeNet5 contained three stages of convolutions, each convolution layer has a pooling layer after and 'tanh' activation functions throughout. The standard input to LeNet5 was a 32x32, my images were 64x32, so the input layer had to be changed to reflect that. The final layer was changed to have eleven outputs, one for each of the digits zero through nine, and the tenth label was for non-digit areas. LeNet5 is a basic network when compared to VGG16 and would have trouble competing with VGG16. An expanded convolutional segment was added between the first and second convolutional stages in LeNet5. The expanded segment contained three convolutional layers with much smaller kernels and strides, followed by 'relu' activations.

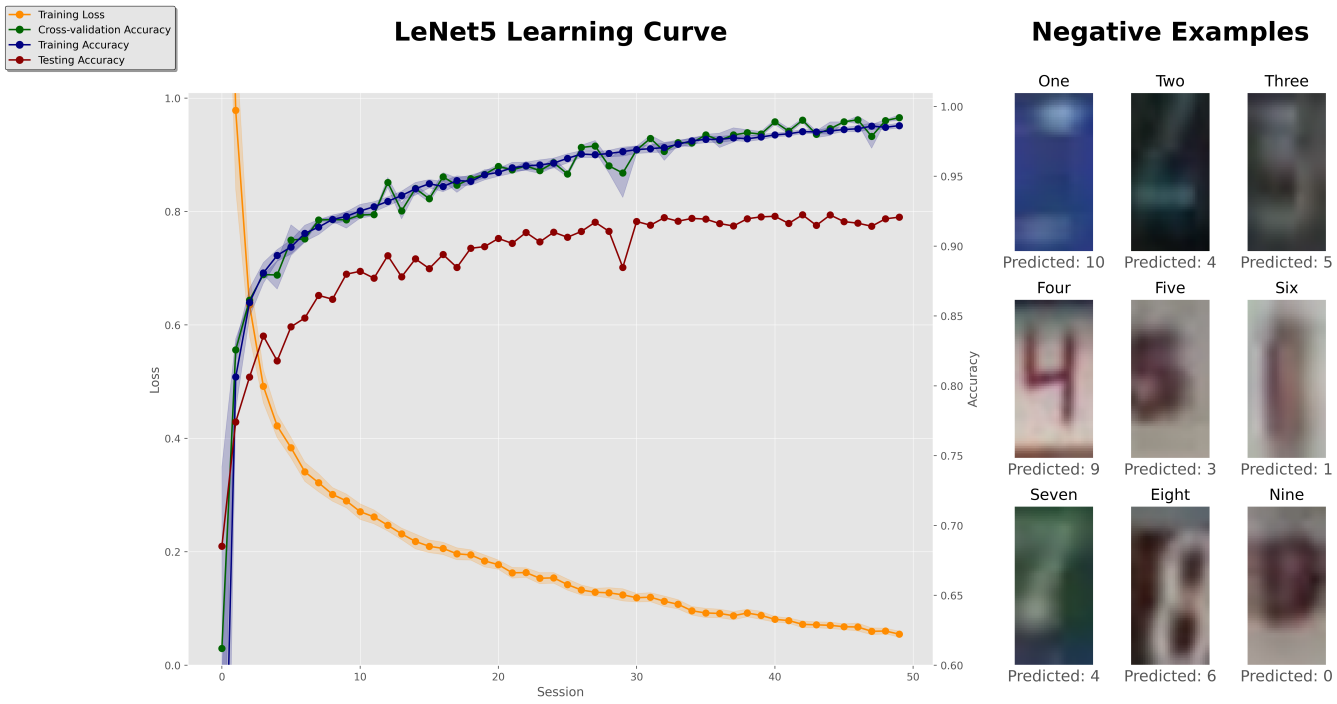

Figure 2 — Learning curve and negative examples of the LeNet5 network.

This expanded convolution layer was fed into the old stage two convolution layer. After this layer, three fully connected layers with 1048, 512 and 64 nodes, respectively. The final layer had eleven outputs and used a 'soft-max' activation function to facilitate the prediction process. The weights used in this modified LeNet5 were randomly initialized and then trained. *Figure 2* shows the learning curve for the modified LeNet5 network. This network was able to achieve 92% accuracy when using a learning rate of 0.001 and tested on the completely unseen testing set. Training was stopped after fifty sessions for both networks. The modified LeNet5 was not able to surpass VGG16 in fifty sessions, *figure 2*, suggests it would have come much closer given more sessions of training. The training loss is steadily lowering as the sessions progress and the accuracy for both training and cross-validation are steadily increasing.

Figure 2 also shows examples where the network predicted incorrectly, based on the label present in the SVHN dataset. For image one, the network predicted that as a ten which corresponds to 'not-a-digit' and in my opinion was more correct than the SVHN dataset. Image two was incorrectly marked in the dataset which cut part of the two off. This makes the network think this is a four, verses image four where it is clearly a four, but the network predicted it was a nine.

Table 1 — Performance Comparison between VGG16 and modified LeNet5 after fifty sessions of training.

Name	Training Accuracy	Cross-Validation Accuracy	Testing Accuracy	Loss
LeNet5	98.6 %	99.2 %	92.07 %	0.0847
VGG16	99.8 %	99.8 %	95.68 %	0.00839

4 PIPELINE OVERVIEW

The pipeline created for this project has multiple stages such as the image pyramid stage, NMS stage and filtering stage. Each stage building on results from the previous to produce a solution that is moderately robust to noise,

location, scale, font, pose and lighting. The primary sources for pose and noise invariance come from the preprocessing steps of the SVHN dataset, described in the SVHN Dataset section. Location and scale invariance were resolved during the pyramid stage of the pipeline. Rather than using a sliding window as it is computationally expensive, MSER was able to reveal areas which should be searched. This reduced the search space from being the entire image to some percentage less than thirty percent of the original space.

The pipeline receives the entire image, such as the one in *figure 3*. The image is sent to the image pyramid stage where it is reduced in size by half, at each level of the four-level pyramid. At each level of the pyramid, MSER is used to find regions where a digit may be present. Those regions are then sent to the trained VGG16 model where it predicts the label and gives a confidence for that label. The resulting regions and confidences are combined and then passed to the NMS stage. The NMS stage will go through and process these results, eliminating regions which have some threshold of Intersection over Union (IoU) and keeping the one with the highest confidence. The NMS stage significantly reduces the number of regions to be processed. The final stage is the filtering stage. During this stage two different heuristics are calculated for the region of digits. The first measures how vertically aligned the digits are and the second how horizontally aligned. These two methods use vertically and horizontally aligned IoU, meaning how many rows or columns are shared between all digits found in a region as a percentage of rows or columns where a digit is present. These two values are subtracted to produce an absolute difference between being vertically or horizontally aligned. Taking the argmax of the difference array gives the index where best choice of digits is located. This works well when housing numbers are either vertically or horizontally aligned.

5 RESULTS

Figure 3 shows the pipeline worked correctly as it detected and classified the house numbers five, one, seven and nine, between the two garage doors.

Figure 3 — Image three, showing the detection and classification of 5,1,7,9 near garage door using the VGG16 network. Source: Houzz.

There were many instances of failures from most were not due to the classification rather the detection aspect. *Figure 4* shows the digits were all classified correctly, with the exception of the nine at the bottom.

Figure 4 — Failure image four, showing incorrect detection and classification of 2,1,3,9, using the VGG16 network. Source: Houzz.

The nine was detected and classified correctly but the pipeline also found a zero within that nine, the upper loop. One of the main reasons the pipeline works well on some images and not as well on others is due to using the image pyramid. The image pyramid makes the images progressively smaller and during this process, some data is lost. The at the lower levels of the image pyramid, numbers become less discernible, example in the case of the nine. The

6 REFERENCES

1. Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., & Ng, A. Y. (2011). Reading digits in natural images with unsupervised feature learning.
2. Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.
3. LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278-2324.
4. Sambasivam, G., & Opiyo, G. D. (2021). A predictive machine learning application in agriculture: Cassava disease detection and classification with imbalanced dataset using convolutional neural networks. *Egyptian Informatics Journal*, *22*(1), 27-34.
5. Krawczyk, B. (2016). Learning from imbalanced data: open challenges and future directions. *Progress in Artificial Intelligence*, *5*(4), 221-232.
6. Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
7. Team, Keras. "Keras Documentation: VGG16 and VGG19." *Keras*, keras.io/api/applications/vgg/.
8. All Required Files: https://1drv.ms/u/s!AvQKh521Q29JhNcRCYaagw72qM_W9A?e=w2Dtyk