Final Report: Deep Learning with Street View House Numbers (SVHN) Dataset

Hina Arora, Olivia Bernstein, & Sacha R. Uritis
Department of Statistics
University of California, Irvine
{hinaa, obernste, srrobbin}@uci.edu

Abstract

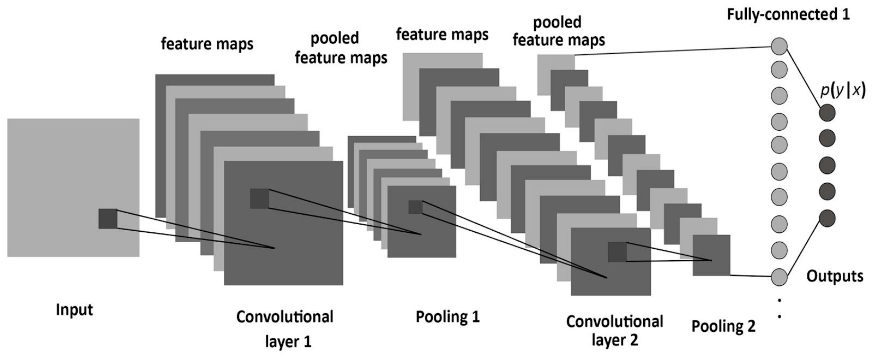
CNN was a topic that this course glossed over briefly. In this report, we will discuss what CNNs are, how they can be applied to image data, how we implemented a model to the Street View House Number (SVHN) dataset, and how we improved our accuracy while controlling for complexity and overfitting. Our model was fit using Tensorflow [1] and Keras on a virtual machine (VM) hosted by Google Cloud. We will present the accuracy of our final model through confusion matrices on predicted values from our test dataset. We will also investigate the robustness of our hyperparameters. Our final model was able to obtain an accuracy of 90.07% on our test data.

1 Introduction

We used deep learning via a convolutional neural network (CNN) model that determined numbers from 32 x 32 pixel images of cropped digits from the SVHN dataset provided by Stanford University. For image processing and classification, CNN is a considered an effective tool for three reasons: (1) it is infinitely flexible, (2) it allows for all-purpose parameter fitting, and (3) it is fast and scalable.

Deep learning is a type of machine learning method, and within its architecture are neural networks. A neural network contains many layers of linear and nonlinear functions. The name neural network was inspired by the biological structure of filters and synapses in the brain because NNs work very similarly. The word *convolutional* in CNN refers to the type of linear function layers applied to the image data. Examples of convolutions applied to images include adjusting brightness, adjusting RGB levels, rotating, flipping, etc.

The combination of linear and nonlinear layers allows the NN the ability to solve any type of problem up to close accuracy as long as you set enough parameters. In this regard, NNs are considered infinitely flexible functions. Stochastic gradient descent (SGD) gives us all-purpose parameter fitting and is used in the nonlinear function layers of the NN. Through SGD, we can experiment with different sets of parameters to check their efficiency and figure out what set works the best for our situation. Lastly, CNN can obtain fast and scalable computational power by utilizing graphics processing units (GPU).

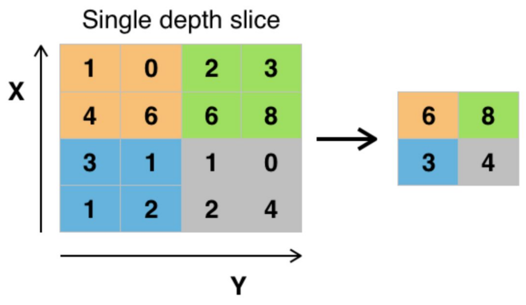


[2] Figure 1.1 : Basic CNN Architecture with two convolutional layers and two pooling layers.

There are three main layers to a CNN: input layer, hidden layer(s), and output layer. The hidden layers are made up of the linear (convolutions) and nonlinear layers. In general, the more hidden layers an NN has, the more computational power it needs, but the better accuracy the model will have.

Mathematically, a convolutional layer is a linear operation applied to the data of an image. Each image in our SVHN dataset was size 32 x 32 pixels. The model takes a kernel of smaller size (typically 3 x 3) and scans the entire image. Each 3 x 3 kernel is a 3 x 3 matrix with certain values. It takes each 3 x 3 section of the entire image and performs a linear transformation with its kernel values. Every convolution has a purpose, but overall the concept is to find patterns, or features, within the images.

Beyond the linear and nonlinear layers, hidden layers also consist of activation functions (e.g. rectified linear unit, ReLU) and pooling layers. We use ReLU, $R(z) = max(0, z)$, for our activation functions because they are simple, the most used, and results in faster learning than other activation functions such as sigmoid function. The pooling layer, or downsampling layer, is optional addition to the NN. The intuition behind pooling is that the exact location of a feature on an image is not as important as the relative location to other features. Figure 1.2 shows an example of how maxpooling layer reduces the amount of parameters, which decreases the necessary computational power by reducing the spatial dimensions and controls overfitting.



[3] Figure 1.2 : Example of Maxpool with a 2x2 filter and stride of 2.

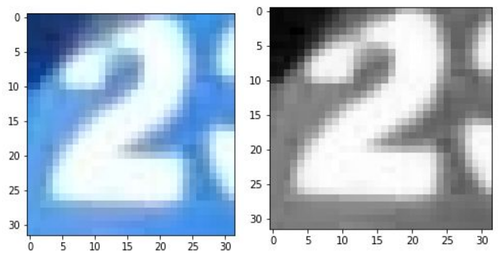
2 Datasource

2.1 Description of the Data

We used The Street View House Numbers (SVHN) dataset with the goal of identifying numbers from images. SVHN includes images of address numbers on houses obtained from Google Street View Images. The images are more difficult to classify then the handwritten letters in the traditional MNIST dataset because the photographs are taken in a natural environment.

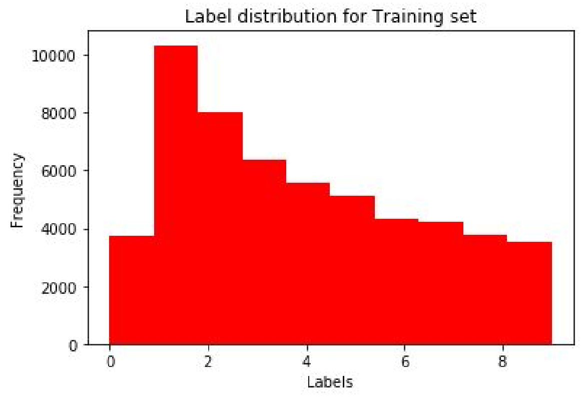
The images of the full address have been cropped so that each image in the final dataset contains one number with a fixed 32 by 32 pixel resolution. Because each number is part of a larger house number, there are partial numbers on the edges of each image which could potentially be distracting. [4]

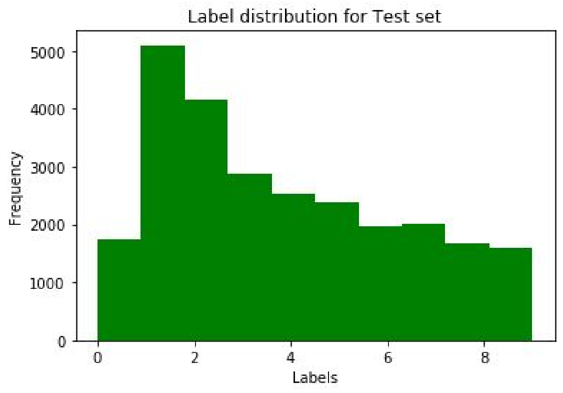
2.2 Preprocessing and Exploratory Data Analysis



There was some preprocessing involved before we could train out model. Since color is not a useful feature in identifying digits (say a 0 can be any color) and for ease of processing, we converted all images to grayscale. The data imported from the .mat files is in a 4D array that has the dimensions (length, width, channels, size), where length and width are 32 each in our case, 1 channel since we had already converted the images to grayscale and size is the total number of training images. So we reshaped the data to get into the format Keras expects it to be in which is (size, length, width, channels). The 0 label had been labeled as 10 in the dataset, so we changed that to 0 and then converted all labels (0-9) to a one-hot encoding such that each label is a vector of length 10 with a 1 at the index of the label and rest 0s. We normalized the data, as was suggested in class, where we subtracted the mean of the training data from the data points and scaled each datum with the standard deviation.

We had 73257 training examples each with dimension (32(length), 32(height), 3 (RGB channels)). The test set consisted of 26032 examples with the same dimensions. We converted the test set to grayscale as well before getting our accuracy metrics. As mentioned before, the



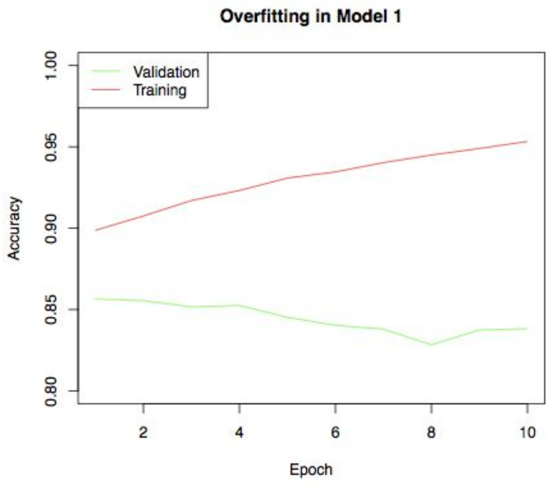


labels range from 0-9, so there are 10 unique values for it. We plotted the distribution labels in training and test data.

3 Model Implementation

3.1 Model 1

The first model we used had two convolutional layers, a flatten layer, and a dense layer. The first convolutional layer has 64 filters, a 3x3 filter matrix, and a rectified linear activation function. The second convolutional layer had 32 filters, a 3x3 filter matrix, and a rectified linear activation function. The flatten layer connects the convolutional layer and the dense layer. The dense layer is a common final layer used for neural networks. Our dense layer had 10 filters for the ten



classes (0-9) and a softmax activation function which outputs the probabilities for each class. [5]

In order to prevent overfitting, we split our training data into validation (20%) and training (80%) sets. We trained our model on the training set and estimated the accuracy on the validation set. We compared the validation and training error at each epoch as the model was trained. The training accuracy continued to increase with the number of epochs, but the validation accuracy did not improve after epoch 4 as can be seen in the plot above. We ran the model several times and the training accuracy seemed to increase until around epoch 3 or 4. This changed every time we ran it because the model building is non-deterministic. The model will be slightly different every time it is trained because it uses stochastic gradient descent. Thus, we decided to use four epochs to train our model.

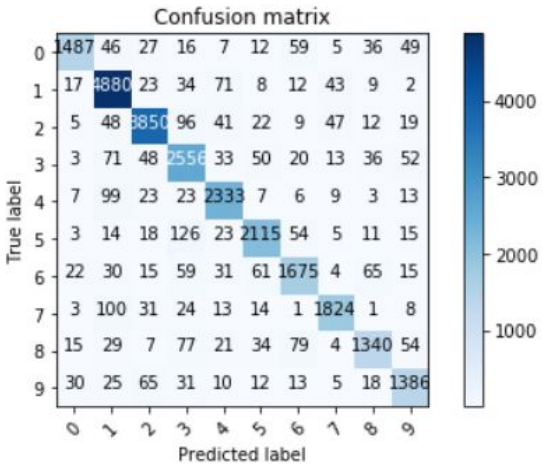
3.2 Model 2

The second model we had three convolutional layers interspersed with maxpooling layers followed by a flatten layer, a dense layer, and dropout layer. This architecture was recommended by a Keras blog article [6]. The two convolutional layers had 32 filters, a 3x3 filter matrix, and a rectified linear activation function. The third convolutional layer had 64 filters, a 3x3 filter matrix, and a rectified linear activation function. Our dense layer and softmax activation function was the same as in Model 1.

We included an extra step before fitting the model with the training dataset: data augmentation. Usually with small dataset, but we wanted to try it for pedagogical reasons and to see if it in fact improves our model accuracy and decreases overfitting. Augmenting the data means put our training images through a number of random transformations and then taking that augmented data through our model for a stage called *pretraining.* The model learns from a larger variety of images before seeing the actual training dataset. The idea is to decrease overfitting and have the model generalize out of sample better. Though, augmented samples are still highly correlated, so this is why we implemented the maxpooling layers and dropout. The dropout layer prevents the model from seeing the same exact pattern more than once; it disrupts the random correlations that occur in the data. We did five random augmentations to the training data : rotation, width shift, height shift, horizontal flip, and brightness. There are many augmentations that make sense for certain types of images. For example, we included the horizontal flip because certain numbers are symmetrical (1, 8, and 0) while others are not.

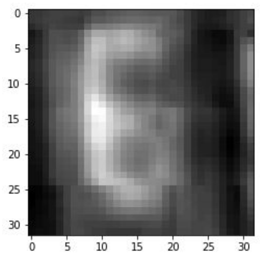
When we fitted Model 1, after epoch 3, the training loss became much smaller than the validation loss, which told us that our model was overfitting. When we fitted Model 2, the training loss remained larger than the validation loss, which indicated that one or more of our added techniques solved the overfitting issue.

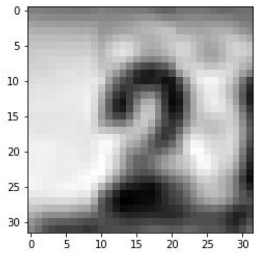
4 Results

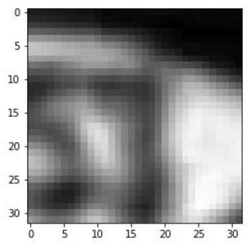


We ran two models one preliminary and the final model (model 2), the confusion matrix from which can be found to the right. Note the model seems to often confuse 3's and 5's. It also seems to misclassify 7's and 2's. This seems reasonable because the shape of the numbers is similar.

Our final model yields 90.07% accuracy on the test set. An example of where our model misclassified a 6 to a 5 is shown to the left. An example where our classifier was very confident (probability>0.99) is shown in the middle. An example where it is not too confident (probability<0.6) is shown on the right.







Overall, our model was very accurate in predicting street view numbers captured in a natural environment. Increasing the complexity of our model allowed us to improve the accuracy.

6 Contribution

Hina Arora was in charge of exploratory data analysis, preprocessing the data, setting up the initial model and getting the confusion matrix. Olivia Bernstein was in charge of setting up the iPython notebook and model on Google Cloud. [7] She also figured out how to output the prediction values. Sacha R. Uritis investigated models that have been previously successful and implemented Model 2.

References

[1] Martin Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mane ́, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Vie ́gas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

[2] Albelwi S, Mahmood A. A Framework for Designing the Architectures of Deep Convolutional Neural Networks. *Entropy*. 2017; 19(6):242.

[3] Karpathy, Andrej. (2018). Convolution Neural Networks (CNNs/ConvNets). Retrieved from URL.

[4] 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. URL

[5] Allibhai, Eijaz. (2018). Building a Convolutional Neural Network in Keras. *Towards Data Science.* URL

[6] Chollet, Francois. (2016 June). Building Powerful Image Classification Models using very little data. *The Keras Blog.* Retrieved from URL.

[7] Aankul, Amulya. (2017). Running Jupyter Notebook in Google Cloud Platform in 15 minutes. *Towards Data Science.* URL

Chollet, Francois, et al. (2015). Keras. URL

Sharma, Aditya. Google Street View House Number (SVHN) Dataset, and classifying them through CNN. *Github.* URL