Final Project Report

Anna Hampton, Bradley Johnson, Moisess Rodriguez, Ipshita Aggarwal

***Abstract*—Traffic signs installed on roads play a critical role in assisting and regulating drivers and their behavior. This report discusses traffic sign classification using convolutional neural networks (CNN). The main task of the CNN model is feature extraction, such that it can differentiate images with traffic signs and label them with respect to their unique categories. We give a brief overview of image classification and how CNNs can be used for this purpose. We then discuss the CNN model used for this project, the training as well as the testing process. Finally, we cover various experiments conducted on the model including model hyperparameter tuning and model performance testing.**

I. Introduction

Image classification is the process of designating images with labels from one of the several pre-defined classes. Some of the main problems seen in this task are that object variability is dependent on viewpoint and data-set with inconsistent values (high in-class variability) [1]. Our focus here is on convolutional neural networks (CNNs) for image classification, where developments in computing in the form of power, performance, data availability, and enhanced algorithms have contributed to the expansion and application of CNNs in the world of neural networks. Our goal is to develop a machine learning system to classify traffic signs. The training data-set has a total of 10 traffic signs, which is equivalent to 10 classes. We also attempted to identify an "unknown" class, where the given test image does not belong to any of the classes.

Feature extraction for image representation is a major task in image recognition systems. Before deep learning was introduced, these features were handcrafted which is quite difficult and expensive, both in terms of time and expertise. In deep learning, the extraction of features and learning them is automated. CNN is one such deep learning architecture which is highly efficient for classifying image data, in our case traffic sign images. Convolutional Network first identify and try to recognize any low-level features of the image, such as bright and dark areas, edges, backgrounds etc., and then combine this information related to the features to learn more complex shapes and patterns [2]. All these different features are identified from multiple independent layers in the network. Each of these layers also have certain number of neurons. They even have their specific height, width, depth [3]. Therefore, the first part of CNN is focused on feature extraction or convolution. The image is processed via a series of filters or convolution kernels, and the output of this step is an intermediate image, also known as convolution maps [2]. An example of one such filter is the reduction of image resolution.

Considering the time, equipment, data-set availability, and even expertise, creating a new CNN is expensive [2]. The first step is to define the number of layers, their size, and the respective operations. Once the architecture is defined, the training begins. But even here, we need to optimize the network parameters so that we have the most performance and the least error. In the subsequent sections, we have discussed the architecture of our CNN as well as how we tuned the network parameters for best results.

For our experiments, preprocessing steps including min-max normalization, image resizing, and relabeling were done. Following preprocessing, we did 4-fold cross validation to determine the optimal hyperparameters. To further confirm the performance of the chosen hyperparameters and the trained network, predictions and subsequent accuracy were computed on a test set.

II. Implementation

A. The Model

We implemented our CNN with PyTorch. The CNN is comprised of 3 convolutional layers followed by 2 linear layers.

The first convolutional layer takes in an RGB image and pads all 4 sides with 1 zero. Then the layer performs 2-dimensional convolution to produce 200 feature maps. The padded images are convolved with kernels of size 5x5 and a stride of 2. The output is then 2-dimensionally maxpooled with a kernel size of 3x3. Next, the output passes through a ReLU activation function. Finally, 60 percent of the neurons are dropped out, and the result is passed to the next layer.

The second convolutional layer takes in the 200 feature maps after the first dropout. Then, the layer pads all 4 sides of the feature maps with 1 zero and performs 2-dimensional convolution to produce 650 feature maps. The padded feature maps are convolved with kernels of size 5x5 and a stride of 1. The output is then 2-dimensionally maxpooled with a kernel size of 3x3. Next, the output passes through a ReLU activation function before 60 percent of these neurons are dropped out. This output moves on to the third convolutional layer.

The third convolutional layer takes in the 650 feature maps after the second dropout. Then, the layer pads all 4 sides of the feature maps with 1 zero and performs 2-dimensional convolution to produce 350 feature maps. The padded feature maps are convolved with kernels of size 5x5 and a stride of 1. The output is then 2-dimensionally maxpooled with a kernel size of 3x3. Next, the output passes through a ReLU activation function before being flattened. The flattened features undergo a 50 percent dropout. This output of 1400 features is passed to the first linear layer.

The first linear layer takes in the 1400 features after the third dropout. This layer produces 128 outputs that are then passed through a ReLU activation function. After the activation function, there is a dropout of 60 percent. This output of 128 features is passed to the final layer of the model.

The final layer, also linear, takes the 128 features after the fourth dropout and produces 10 outputs, corresponding to the 10 possible classes. The 10 outputs are then passed through a log softmax activation function before being labelled.

B. Training

Before any training is done, the hyperparameters (apart from those in the model itself) are defined. The number of epochs is set to 100; the loss function is cross entropy loss; the learning rate is set to 0.0004; and the training batch size is set to 64. (The learning rate, batch size, and kernel size were determined from cross validation.) Also, the model will be moved to a GPU if one is available; otherwise, it will remain on the CPU, and training will take significantly longer.

To detect overfitting and get a measure of generalization, the training dataset of 6195 samples was split 80/20 into a training and a validation set. Because 20 percent of the training data was used for validation, the training data set contains 4956 samples, and the validation set contains 1239 samples.

Each epoch of training looks the same. At the beginning of an epoch, the model is set in train mode, which effectively tells our model to enable dropouts. After setting the model to train mode, all mini-batches are looped through with a batch size of 64. For each mini-batch, the gradients are zeroed (so we do not accumulate gradients from each backpropagation); the mini-batch is fed forward to get new predictions; the new predictions are passed (along with the true labels) into a cross entropy loss function; backpropagation occurs; and finally, the parameters of the model are updated. After all the mini-batches of the training data have been looped through, the model is set to evaluation mode (disables dropouts) and gradients are disabled (because no parameters will be updated based on the evaluation). When evaluation mode is turned on, predictions are made for all the samples in the validation set. The predictions are used to calculate the accuracy and loss (cross entropy loss) of the validation set. The accuracies and losses are saved for the training and validation sets for all 100 epochs.

After all the epochs are over and the training is over, the training and validation accuracies and losses are plotted.

C. Testing

To test the model, the model is set to evaluation mode and gradients are disabled. When evaluation mode is turned on, predictions are made for all the samples in the test set. Finally, the predictions are used to calculate the accuracy on the test set, and they are used to produce a confusion matrix. The test accuracy and confusion matrix are then printed.

In an attempt to distinguish images within the specified 10 classes from images that fall outside the 10 classes, we look at the result of the feed forward. Feeding an image through the model produces an array of length 10, corresponding to the 10 classes. All the results are less than or equal to zero, due to the final activation function being a log softmax function. Usually, the label is said to be the argument with the maximum value. Therefore, we look at the value at the argument with the maximum value when deciding whether or not an image belongs to any of the 10 classes; if the value falls below some threshold, then it is determined to not belong to any of the 10 classes. We chose this threshold to be -1.

III. Experiments

A. Data Preprocessing

The dataset of 6195 sign images was used for training. As a first step, all images for each class were visualized to identify mislabeled instances. Incorrectly labeled images were reassigned to the correct class. Post relabeling, images were resized to be 175x175 using bicubic interpolation, which is a type of cubic interpolation used for resizing that considers the 4x4 grid of neighbors, to decrease run time and space complexity [4]. The final preprocessing step consisted of min-max normalizing the dataset to create a common scale and improve convergence.

B. Cross Validation Experiments

A grid search with 4-fold cross validation was used to tune the learning rate, the convolutional kernel size, and the batch size. A grid search was chosen because it exhaustively checks every possible combination of inputted hyperparameters and chooses the one that maximizes validation accuracy. Learning rates of .0004, .0006, .0008, .001, kernel sizes of 3x3 and 5x5, and batch sizes of 256, 128, 64, and 32 were tested.

TABLE I
Top ten hyperparameter combinations

Accuracy	Learning Rate	Kernel Size	Batch Size
$96.6 \pm .3$ %	0.0004	(5,5)	64
$96.6 \pm .5$ %	0.0006	(5,5)	32
$96.4 \pm .3$ %	0.0008	(5,5)	64
$96.3 \pm .6$ %	0.0004	(5,5)	128
$96.3 \pm .9$ %	0.0006	(5,5)	256
$96.2 \pm .3$ %	0.0006	(5,5)	64
$96.2 \pm .6$ %	0.0004	(5,5)	32
$96.1 \pm .7$ %	0.0008	(5,5)	32
96.1 ± 1 %	0.0008	(5,5)	256
$96.0 \pm .1$ %	0.0008	(5,5)	32

Based on the cross validation results shown in Table I, the best performing hyperparameter combination with an average validation accuracy of 96.6% was a learning rate of 0.0004 in conjunction with a (5,5) kernel and a batch size of 64. Despite overlap between the top score and the next best score given the displayed standard deviations, we chose to use the top result since it had a smaller standard deviation and the highest mean accuracy. Our final experiment was performed using the CNN with this hyperparameter configuration.

C. Train and Test Split Experiments

To confirm the performance and investigate potential overfitting of the model with learning rate of 0.0004, kernel size of (5,5), and batch size of 64, the provided training set was split into a training and test set with 80% of the data in the training set and 20% in the test set. The network achieved a test accuracy of 95.9% on the test set. Thus, the model performed very well and was able to quite accurately classify all classes

on never before seen data. Our final test to ensure that the model was not overfit was plotting the learning curves visible in 2. The learning curves do not display too many jumps and peaks, indicating that the model is a good fit and not overfit to the training images.

Lastly, to identify where the bulk of the misclassifications were coming from, the confusion matrix was plotted and can be seen in Figure 1. From the confusion matrix, it appears that the network struggled most with roundabout signs since they received both the most false positives and false negatives. This potentially stems from the variation in the way that these signs present. For example, some were blue where as others were yellow.

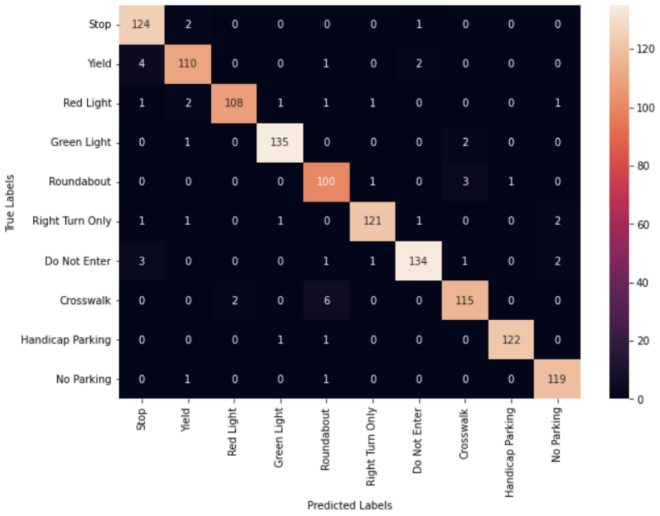

Fig. 1. Test set confusion matrix

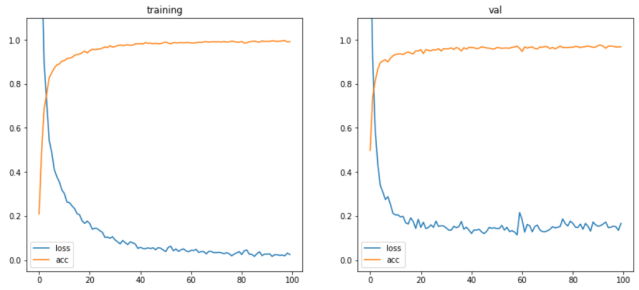

Fig. 2. Training learning curves: epoch accuracies and losses for training and validation sets

IV. Conclusion

In conclusion the objective to implement a Convolutional Neural Network was achieved and the results were able to be displayed accordingly, and it was made possible by utilizing the PyTorch framework. The model that was implemented was a CNN with 3 convolutional layers and 2 linear layers. As explained in section 2-part A, by initially passing in an image, we are then able to produce feature maps to pass to the following layers, resulting in a set of outputs that is proportional to the number of classes in the data set. As the data traverses through the layers, after the fourth layer we have 1400 features which is then sent back to the first linear layer. The features or outputs are eventually reduced down to 10 outputs. This is made possible by using multiple methods such as padding feature maps, convolving feature maps with kernels of a set stride, as well as using the ReLU activation function. During the training phase, prior to training the network, the hyper parameters were determined through the cross-validation method. The hyper parameters being the learning rate, batch size, and kernel size. The network trained faster while using a graphics processing unit (GPU) rather than using a central processing unit (CPU). GPUs tend to be more efficient with large or resource taxing applications, and this can be related to the architecture of the two different components. However, during the training phase, overfitting was able to be determined by splitting the training data into another subset of training and test data, where the distribution was 80/20 respectively. After splitting the data, we began training by setting the model into train mode where the 4956 samples of the training data is utilized to allow mini batches to loop through with a set batch size. After feeding the mini batch forward, we are able to get new predictions, which are then passed into a cross entropy loss function, which allows us to measure the classification performance of the network. Once this is complete, the model will begin backpropagation process to finally update the parameters of the model. After we are done training the model, we then set the model into evaluation mode, where it will then make predictions for the samples. Once we have the predictions, we can then determine cross entropy loss of the validation set and store the data for both the training and evaluation phases. So, by using the known training data, and splitting it into training and validation sets it was possible to tune the model's parameters, which will allow our model to be more accurate when working with unknown data. When testing with unknown or test data, the model will behave similarly to when it was evaluating the validation. Once predictions are made for all of the samples in the test data set, the performance metrics are then calculated. This allows us to determine the accuracy of the network, and we are able to evaluate the performance matrix to see what classes tended to be more accurate, or what classes were less accurate. In the actual experiment prior to training the data we resized, and then corrected any mislabeled images. This allowed for improved accuracy, as well as reducing run time and space complexity. To determine the optimum hyper parameters, a grid search algorithm with k-fold (where k is 4) cross validation was implemented. During this phase we were able to see the accuracy of the model with different learning rates, kernel sizes, and batch sizes. From table I, we noticed that the accuracy was very close at those given values, however by using the row with the smaller standard deviation and highest mean accuracy, we were able to ensure that these hyper parameters will provide the most accurate and efficient predictions. After we split the data into training and test sets, we determined that the model scored an accuracy of 95.9%. By outputting a confusion matrix, we saw that the model classified 124 samples correctly into the stop class, 110 into yield, 108 into red light, 135 into green light, 100 into

roundabout, 121 into right turn only, 134 into do not enter, 115 into crosswalk, 122 into handicap parking, and 119 into no parking. The class that scored the highest was do not enter, and the one that scored the lowest was round about. Potential factors for these results can be related to the characteristics of the images, being consistency, image quality, background noise, bold colors, etc. To increase the accuracy of the classes that performed the worst we can add more data of the same specifications as the ones that are already in the training set, or we can train the network only on images that are of the same variation.

REFERENCES

[1] W. Rawat and Z. Wang. Deep convolutional neural networks for image classification: A comprehensive review. 9 2017.
[2] S. Zayen N. Jmour and A. Abdelkrim. Convolutional neural networks for image classification. 2018.
[3] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. 10 2010.
[4] Anil Gavade and Prasana Sane. Super resolution image reconstruction by using bicubic interpolation. 10 2013.