Image Classification Using a CNN

Justin Gilmore
Dept. of ECE
University of Central Florida
Orlando, FL, US

Ryan Heras
Dept. of ECE
University of Central Florida
Port Saint Lucie, USA

Paul W Davis
Dept. of ECE
University of Florida
Jacksonville, FL, US

***Abstract*—In this paper, a CNN is built from a LeNet-5 style architecture for the purpose of classifying color images of traffic signs and traffic lights. The model grows to include stacked convolutional layers, overlapping max pooling, and drop out layers, similar to the AlexNet CNN. Regularization and overfitting reduction techniques are explored. A brief analysis of misclassification of targets is done.**

I. Introduction

EEL 5840 introduced a subset of machine learning known as deep learning. As described in [1], an advantage of deep learning over classical machine learning is that a certain task may be implemented by a deep learning model (neural network, or NN) without many of the data preprocessing steps required, prior to implementing that task by a non-NN machine learning model. One major reason such preprocessing steps may be avoided when employing a NN is that its deep architecture is able to operate on unstructured (non-tabular) data. The architecture comprises an automatic model parameter update system that works according to a back-propagation of output error through the network. Upon each epoch – or pass of a batch of data through the network – an error is calculated, and a chain of derivative (gradient) calculations are made at each layer, contributing to the optimization of each layer's weight matrix (model parameters). Hereby, and with successive batches, the focus of the NN on its task is sharpened.

In the context of image classification, and in the interest of this group, it was determined that candidate classification models which required the group to learn image data preprocessing techniques should be avoided for the sake of convenience. Further, and learned in [1], a convolutional neural network would be a sound model to predict on unstructured image data.

Deep learning model architectures for image classification, such as AlexNet [2], provide a template for state-of-the-art networks, capable of handling a data set of one million images and outputting a thousand class predictions. A model of similar purpose, but very much scaled down, was sought for predicting one of ten labeled image classes, using a training set of 6,195 images. Experimentation with architecture began with our version of the LeNet-5 network [1], the precursor to the much more complex AlexNet. An early NN design, LeNet-5 predicted one of ten classes from a single channel (black and white) image set. The assumption was that this model would provide a simple starting point for building the complexity required for classification of color images.

Challenges to designing a NN that would achieve at least 90 percent accuracy on a held-out test set were described in EEL 5840 lectures [1]. These mainly reduced to choice of CNN hyperparameters to avoid overfitting the training data, number of samples per batch, and number of training epochs to optimize training sessions, The CNN, by virtue of its use of convolution matrices between layers, forces a reduction in weight connections per layer by sharing weights and eliminates the value of other connections. Therefore, a CNN inherently has a lesser tendency toward overfitting than other NNs such as multi-layer perceptron models.

Three-by-three convolutional filters were used in each layer empirically and due to the shape of the input image. Additionally, consideration for capturing text features, which help differentiate similarly shaped signs (such as No Parking and Handicap Parking Only), invited the use of 3x3 filters. It was intended that this size filter would capture these locally important text features. ReLu activation functions were used instead of saturating functions, as it was recommended for better training speed in [2]. The use of overlapping MaxPooling after convolutional layers was implemented for dimensionality reduction while imputing translational invariance to the reduced feature maps, and also as empirical suggestion to avoid overfitting (overlapping MaxPool). MaxPooling was selected over AvgPooling due to its tendency to extract edge-type features over smoothed-type features [3]. The goal is to identify features of traffic signs, which usually present edges as a major feature. Finally, a standard ten-neuron output layer, using SoftMax activation was used. Additional layers are described in Part III of this paper.

A simple learning rate application was performed by inspection of training epoch trends in accuracy and loss. So, a learning rate 0.002 was applied for the first training session (100 epochs), and a learning rate of 0.001 was applied at subsequent training sessions.

The strategy for developing a CNN that would predict above 90 percent on a held out test set of images was one mostly based on empirical suggestion – the iterative stacking of convolutional layers combined with pooling layers and adding a dense layer connected to an output layer – combined with methods to reduce overfitting (which would inevitably occur with such aggressive expansion to increase complexity) which were learned in course EEL 5840 lectures. The design effort respected the theory of Occam's Razor. Number of convolutional layers and dense layers were added in order to reach a satisfactory accuracy score in testing. There was no effort to increase complexity beyond attaining scores of 95 and 96 percent in testing.

GridSearchCV was done for number of units in the first and second dense layer upon their respective addition to the network. GridSearchCV was done for batch number and number of training epochs at the point of the network reaching ten layers. The Glorot Uniform weight initialization is the default for Keras Conv2D and was used somewhat arbitrarily. No attempt to optimize weight initialization was made.

Increasing the number of feature maps occurred at intervals, with the increasing number of network layers. The concern for increasing feature maps simultaneously with increasing network depth came after inspection of those extracted from the first convolutional. Upon increasing the first convolutional output maps from 16 to 32, it was noted than most of the 32 maps were simply very dark, lacking many typical feature activations. It seemed counterproductive to pass these maps through the network, so the number of maps was increased to 64 and a dropout layer was added in an attempt to reduce the number of dark monochromatic maps, while increasing the number of apparently activated maps. That increase by two cascaded through the network, as is typical in standard CNNs.

A technique used to reduce some overfitting early in model design was L1-L2 regularization. L1-L2 was to substantially reduce overfitting. L1-L2 regularization is helpful in CNNs because it penalizes the weight matrices and causes them to decrease in value. This simplifies the model so that overfitting the data is less likely to occur. L2 regularization causes some weights in the model to decay towards zero, but not exactly zero. With L1 regularization, some weights will become zero. This is useful when compression of the model is desired [4].

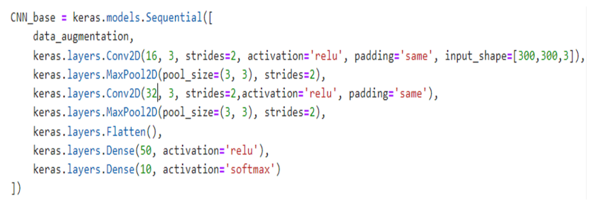
This paper is divided into five parts. Part I is the introduction, where the strategy of CNN design is described with respect to expert reference material. Part II describes the general form of implementation of the experiments used in design of the network. Part III details the experiments conducted to reach the final model. Part IV highlights the conclusions reached in the overall effort to design the CNN.

II. Implementation

The project NN was implemented using Keras version 2.7 with a Tensorflow 2.7 backend. The HiPerGator (University of Florida's supercomputer cluster) environment was used for hyperparameter tuning and model training/testing. A NVIDIA A100 GPU with 80,000 MB of memory and two cores powered the project.

The given training data set consisted of 6,194 color images of shape 300-by-300-by-3. The training data was split into training and testing data, with a testing portion of 15 percent. The training data was further split into train and validation sets. Validation consisted of 20 percent of the training data.

It was noted that some of the training samples corresponded to wrong labels. A manual inspection of the entire training set found thirty incorrectly labeled samples/targets. These were corrected. Further, inspection of the training set revealed images which might be more trouble than what they were worth,



```
CNN_base = keras.models.Sequential([
    data_augmentation,
    keras.layers.Conv2D(16, 3, strides=2, activation='relu', padding='same', input_shape=[300,300,3]),
    keras.layers.MaxPool2D(pool_size=(3, 3), strides=2),
    keras.layers.Conv2D(32, 3, strides=2,activation='relu', padding='same'),
    keras.layers.MaxPool2D(pool_size=(3, 3), strides=2),
    keras.layers.Flatten(),
    keras.layers.Dense(50, activation='relu'),
    keras.layers.Dense(10, activation='softmax')
])
```

Fig. 1. Base model

with respect to successful training sessions. The preprocessing step for cleaning the image set is described in the Experimental section of this paper. The pre-standardized (noisy) training data set was tested on after training on the standardized training data and obtained a satisfactory accuracy.

Due to kernel death associated with tensor conversion of data larger than 5,000 samples, training data was limited to 3,400 samples, validation limited to 850 samples, and test data limited to 750 samples during a training/testin session. No extra data was used in training. Keras RandomFlip and RandomRotation methods comprised a data augmentation layer which was added to the model after the first training session, using the base model. Image data was normalized by dividing by the entire training set by 255.

The LeNet-esque base model, described in Table 1, was used as a standard for base accuracy and constituted a foundation for cautiously expanding the model complexity. Steps for deciding an change in the model were either (1) adding a component for complexity and then training (using batch size of 50, over 100 epochs), or (2) adding a regularizing method or dropout, when step (1) was shown to cause model overfitting. Assessment of overfitting was made according to a learning curve review at the end of every training session.

The model was compiled each time using a sparse catagorical crossentropy loss function, due to the ten integer-valued classes. The Adam optimizer and first training session learning rate of 0.002 was used empirically. On subsequent training sessions, a learning rate of 0.001 was used. This technique proved useful and avoided a lengthy grid search or scheduling process to find optimal learning rates. In model fitting, early-stopping with a patience of thirty was used and callbacks were set to save-best-only = "True."

The value for batch size (50) and number of epochs (100) was obtained using GridSearchCV, once the model reached ten layers. These were also the settings prior to GridSearchCV. Default kernel initialization (Glorot Uniform) was used in each convolutional layer and not contested by a GridSearch. A classification report, confusion matrix, and learning curve were obtained after prediction on the training and testing set. Loss and accuracy for training and validation predictions were plotted in each learning curve after training and validation predictions.

III. Experimentation

A. Data preprocessing

Of the 6,194 original training images, 30 were discovered to be mislabeled by inspection. These labels were corrected. During inspection of the training image set, 97 images were determined to be more trouble than what they might be worth for training purposes. Reasons for discarding some images were:

- non-target objects crowding the target image, rendering the target under-represented in the image
- multiple class representation in the same image
- extremely rare instances of a class

A significant percentage of the images were rotated or flipped so that it was necessary to make the model robust to these translations. A data augmentation layer using Keras RandomFlip (horizontal and vertical) and RandomRotation of 0.2 was added as an input layer to the model.

B. Building the model

The Base model described in Table 1 would provide a lowest measure for accuracy in the course of experimentation. This model represented a version of the LeNet-5 model, built for predicting on ten classes of images, that would be carefully added to in order to predict on very noisy color images. Initial runs showed overfitting (seen in Figure 1) and unsatisfactory accuracy of 88 percent.

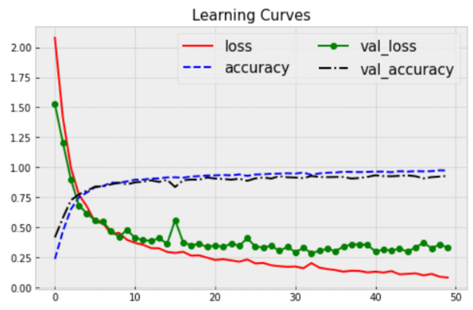


Fig. 2. Base model learning curves

At this point, the overfitting needed to be addressed. L1-L2 regularization was added to the dense layer connected to the output layer. The default tuning of L1 = L2 = 0.001 was used. The learning curve after adding regularization showed a reduction in overfitting. Model accuracy was at 91 percent after regularization. Overfitting had been abated, and adding model complexity was the next step in the attempt to increase model accuracy.

An additional convolutional layer was placed below the flatten layer. A 3x3 kernel was used and the layer output 64 feature maps. This addition slightly reduced accuracy yet showed no significant overfitting. More complexity was added by doubling the number of feature maps over the network. Meaning that 16 maps in the first convolutional layer increased to 32. Thirty-two maps in the second and third convolutional layer increased to 64. And 64 maps in the fourth convolutional layer increased to 128. This addition showed neither an increase in overfitting nor an increase in accuracy, so that another convolutional layer was added on top of the last Conv2d layer. So that there were two stacked Conv2D layers, similar to the technique used in AlexNet. A slight increase in accuracy as well as a slight increase in overfitting resulted.

At this point, there was an interest in optimizing the dense layer. A GridSearchCV over 25 to 100 units showed that 80 units should be the optimal number in the dense layer, connected to the output layer. A second dense layer of 80 neurons was stacked on the existing dense layer of 80 neurons. A drop out layer (Dropout = 0.1) was added to the output of each dense layer to protect against possible overfitting. Several values [0.1, 0.2, 0.25, 0.3, and 0.4] for drop out were tried, with 0.1 providing the top accuracy at the end of 100 training epochs. The resulting learning curve after these additions and reductions showed no significant overfitting and model accuracy remained at or above 90 percent.

It was considered that the first convolutional layer output feature maps had not been inspected during experimentation. Most of these 32 feature maps, as shown in Figure 2, appeared completely void of detectable features. It seemed that the network could be supplied more and better information if these feature maps were improved. Therefore, the first convolutional layer output was doubled to 64, and this doubling was cascaded through all convolutional layers in the network.

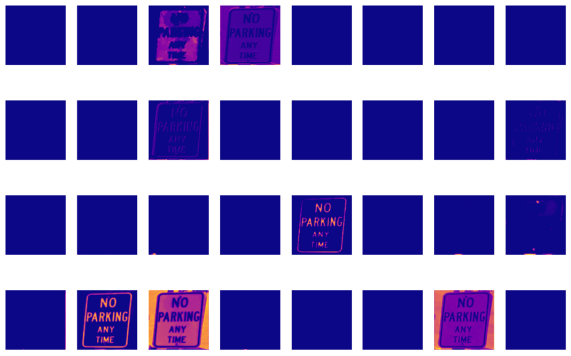
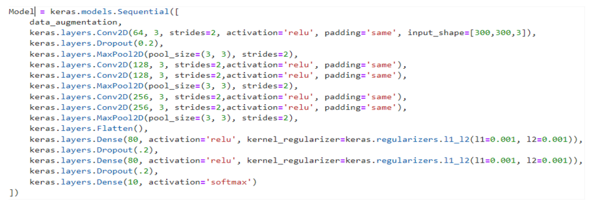


Fig. 3. 32 feature maps, first convolutional layer

The increase of feature maps could mean an increase in those regarded as featureless, as seen in the 32 maps previously. Therefore, a GridSearcCV was done to find a suitable value for a drop out layer connected to the output of the first convolutional layer. Grid search optimized the drop out percentage at 20. The resulting learning curve showed some overfitting, but was corrected with L1-L2 (L1 = 0.001, L2 = 0.001) regularization added to the second dense layer, connected to the output layer. This corresponded to reduction in overfitting.

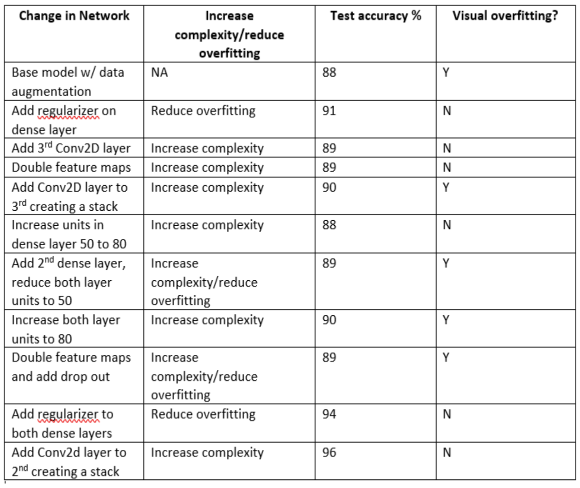
It was considered that the current model prediction accuracy may not be high enough. To reach an accuracy percentage closer to 100, yet not adding more complexity than deemed necessary, two more stacked convolutional layers, which output 256 feature maps each, were placed on top of the two stacked convolutional layers which output 128 feature maps each. These new additions were placed in between max pool layers, to help protect against overfitting. The final model, reaching a testing accuracy of 96 percent on the preprocessed data (with images removed) and 94 percent on the original (correctly labeled) data is shown in Figure 4.



```
Model = keras.models.Sequential([
    data_augmentation,
    keras.layers.Conv2D(64, 3, strides=2, activation='relu', padding='same', input_shape=[300,300,3]),
    keras.layers.Dropout(0.2),
    keras.layers.MaxPool2D(pool_size=(3, 3), strides=2),
    keras.layers.Conv2D(128, 3, strides=2,activation='relu', padding='same'),
    keras.layers.Conv2D(128, 3, strides=2,activation='relu', padding='same'),
    keras.layers.MaxPool2D(pool_size=(3, 3), strides=2),
    keras.layers.Conv2D(256, 3, strides=2,activation='relu', padding='same'),
    keras.layers.Conv2D(256, 3, strides=2,activation='relu', padding='same'),
    keras.layers.MaxPool2D(pool_size=(3, 3), strides=2),
    keras.layers.Flatten(),
    keras.layers.Dense(80, activation='relu', kernel_regularizer=keras.regularizers.l1_l2(l1=0.001, l2=0.001)),
    keras.layers.Dropout(.2),
    keras.layers.Dense(80, activation='relu', kernel_regularizer=keras.regularizers.l1_l2(l1=0.001, l2=0.001)),
    keras.layers.Dropout(.2),
    keras.layers.Dense(10, activation='softmax')
])
```

Fig. 4. Final model

The final model used five convolutional layers – a single layer and two stacks of two – with adjacent max pooling layers. The model used two dense layers – each followed by a drop out layer – in the fully-connected upper region. The total number of parameters was 1,136,330. The dense layers used L1-L2 regularization. All layers used ReLu activation, except the last dense layer (output) which used SoftMax activation for the purpose of normalizing a probability for class prediction. Experiments are summarized in Figure 5, according to (a) the change in the network, (b) whether the change was to increase complexity or reduce overfitting, (c) test accuracy after the change, and (d) resultant overfitting (Y/N) by inspecting learning curves.



Change in Network	Increase complexity/reduce overfitting	Test accuracy %	Visual overfitting?
Base model w/ data augmentation	NA	88	Y
Add regularizer on dense layer	Reduce overfitting	91	N
Add 3rd Conv2D layer	Increase complexity	89	N
Double feature maps	Increase complexity	89	N
Add Conv2D layer to 3rd creating a stack	Increase complexity	90	Y
Increase units in dense layer 50 to 80	Increase complexity	88	N
Add 2nd dense layer, reduce both layer units to 50	Increase complexity/reduce overfitting	89	Y
Increase both layer units to 80	Increase complexity	90	Y
Double feature maps and add drop out	Increase complexity/reduce overfitting	89	Y
Add regularizer to both dense layers	Reduce overfitting	94	N
Add Conv2d layer to 2nd creating a stack	Increase complexity	96	N

Fig. 5. Summary of model changes

C. *Target Classification*

Classification reports including precision, recall, and f-1 score were recorded after each change in the model. Precision was chosen as the statistic to analyze over the 16 changes in the model, to quantitatively describe the difficulty which the model had in classifying the ten targets. Mean and standard deviation of the precision score at completion of each initial training session (100 epochs) was recorded and is shown in Table 1.

TABLE I
CLASSIFICATION PRECISION OVER 16 MODEL CHANGES

Target	Mean	St Dev
Stop sign	92.48	3.62
Yield sign	89.88	5.06
Red light	91.88	4.37
Green light	94.06	6.37
Roundabout	91.0	3.35
Right turn only	86.81	7.19
Do not enter	88.06	4.37
Crosswalk	89.13	4.09
Handicap park	88.88	4.83
No park	93.63	4.58

IV. CONCLUSIONS

Building a CNN modeled on an existing standard architecture like LeNet-5 provides a fairly straightforward approach to solving a similar, yet somewhat more complex problem. However, crucial to accomplishing the required model complexity for obtaining at least 90 percent in model accuracy is implementation of overfitting countermeasures. These which proved effective are the combination of (a) drop out, after a fully-connected layer and/or convolutional layer, (b) L1-L2 regularization at the fully-connected layer, and (c) overlapping max pooling of feature maps.

The simple data augmentation strategy used in the experiments proved successful, in that, additional augmented data wasn't needed to obtain satisfactory accuracy scores in the end. Therefore, more data isn't required to be stored, freeing up resources. Further, robustness against image translations was accomplished within the model.

A decrease in learning rate, from 0.002 to 0.001, when training a model on its first exposure to training data and then with subsequent exposures seemed useful overall. Inspection of a given model's history over 100 epochs for a subsequent training session on mostly seen data indicated that a decrease in learning rate was useful; however the optimized learning rates were not sought in the interest of time.

Deducing from the table of classification precision, targets with larger standard deviation in precision caused variation or instability in the model as it attempted to classify that target. A counter example however was the "green light" target; the precision plot for which was a steep increase upon testing the model in its second change. Precision for the green light increased to 96 percent on the second change to the model and never went far below 96 percent. This, in conjunction with a very high mean precision, indicates that the green light remained the easiest target for the model (in any form) to correctly classify.

REFERENCES

[1] C. Silva. (2022). Fundamentals of Machine Learning Lectures 1-21 [Online]. Available: https://github.com/EEL5840-EEE4773-Summer2022.

[2] Krizhevsky, I. Sutskever, and G. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1, 2017. pp. 1097-1105, [Online]

[3] M. Basavarajaiah. “Maxpooling vs minpooling vs average pooling.” Medium.com. https://medium.com/@bdhuma/which-pooling-method-is-better-maxpooling-vs-minpooling-vs-average-pooling-95fb03f45a9 (accessed Jul. 25, 2022).

[4] J. Shubham. “An Overview of Regularization Techniques in Deep Learning (with Python code).”Analyticsvidhy.com. https://www.analyticsvidhya.com/blog/2018/04/fundamentals-deep-learning-regularization-techniques/ (accessed Jul. 26, 2022).