1. **Introduction**

In Project 2 of this course we were tasked with classifying images contained in the German Traffic Sign Dataset. To accomplish this task various deep learning architectures were trained using a significant portion of the dataset. These architectures were then validated on a smaller portion of the dataset and finally the best performing architecture was tested on the last portion of the dataset. Different preprocessing, sampling and dataset augmentation techniques were also tested in the same way. The main performance metric used to compare architectures was accuracy on the test and validation sets. An analysis of the classifier’s mistakes is included along with reflections on the process in the conclusion.

In the first part of this article we will review machine learning concepts utilized in this work, with a focus on the methods and tools used for convolutional network design. In the second section, we will explore the data and dataset augmentation methods used in this work. In the third section, we will explore model evaluation techniques, compare architecture variations and their relative performance on our dataset and then demonstrate the final architecture used. In the fourth and final section, we will find out exactly where our classifier went wrong and make some suggestions for future work.

1. **Machine Learning**

A machine learning algorithm usually consists of a model and on optimization scheme that together are able to accurately predict or provide insight to some targeted output(s). While the technical details of how these algorithms operate are often dense and incomprehensible to the average person they operate in much the same way humans and other living things do. It is no wonder then that the basis of many machine learning and especially deep learning algorithms can be traced back to neuroscience, the study of the structure and operation of the brain.

One often refers to the ‘capacity’ of a machine learning algorithm. Informally, capacity refers to the set of possible functions (or hypotheses) a model may represent. Models with a high capacity may overfit a set of data by memorizing the training set while models with low capacity may underfit the training set. In both cases the model’s capacity is not well suited to the task’s complexity and it will not perform well on unseen data. The goal of model design is then to control the capacity by choosing its hypothesis space in such a way that, when the model is exposed to training data, it is able to select the correct function that maps input to output. There are many methods and tools used to control a model’s capacity, some of which are described and demonstrated here.

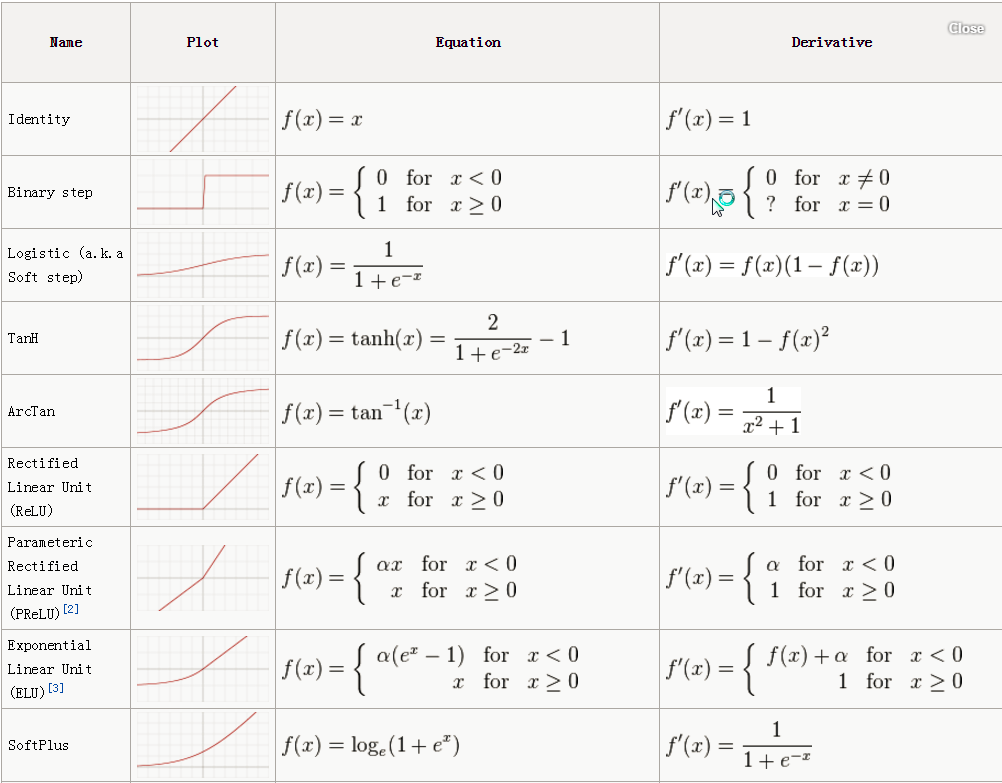
* 1. **Feedforward Networks**

The OG neural network is the feed-forward network. A feedforward network and its corresponding optimization function learn a set of parameters which approximate a function mapping input to the desired output. These types of models are called feedforward because information flows in a forward direction (acyclic) and as networks because they consist of many connections between input and output of many smaller functions. A network which contains backwards connections (cycles) between intermediate functions is referred to as a recurrent neural network. Finally, these networks are referred to as neural because their structure is similar to the structure of connected neurons in the brain. Each neuron (function) in each layer of the network is connected to many inputs and computes its own activation in parallel.

Deep feed forward networks are created when multiple feedforward layers are combined in a linear fashion. As the depth or number of hidden layers increases, the capacity of the model increases, allowing the model to approximate more and more complex functions. It is this depth that gives deep learning its almost uncanny potential to learn complex tasks which are extremely hard to define explicitly.

* 1. **Activation Functions**

If we constrain the output of each neuron to be some linear function of the inputs, the capacity of the network will remain linear no matter how many layers are added. One option to extend the capacity is to include non-linear transformations of the output at each layer. Non-linear functions applied at the output are called activation functions and are often preceded by a linear transformation of the input with learned parameters. Examples of popular activation functions are shown in the below figure.



The most popular is the rectified linear unit (ReLU)and it close variant the leaky-ReLU/parametric-ReLU (pReLU). In choosing an activation function we aim to achieve the universal approximation property, which is accomplished through depth and non-linearity, and computational efficiency. The universal approximation property is necessary to ensure that we are able to create reasonably good approximate functions (models) which fit the observed data. Computational efficiency is necessary to maintain low training and evaluation time. The ReLU function accomplishes both of these tasks.

* 1. **Convolutional Layers**

This work utilizes convolutional layers, which were popularized by Yann LeCun in his work on LeNet-5. Convolutional layers are often used when the input data has some underlying structure that may be exploited. In this case that structure is spatial meaning each pixel in an image is related to the pixels around it in some way. Another example is the temporal structure of time-series data in which the data collected at one time-step is often related to previous time-steps. In using convolutional layers we hope to tell our model something about the data it sees and restrict its capacity in such a way that makes it easier to learn a proper representation of the output

A convolution is any operation on two functions of a real-valued argument. The moving average of a time-dependent signal, used in a wide array of industries, is a very simple convolution of the time-dependent signal and a weighted average over the past *n* signals a weight of 1 and any signal older than that a weight of 0. In the context of convolutional neural networks the time-dependent signal is referred to as the input, the weight function is referred to as the kernel, the magnitude of *n* is the window size, and the output (i.e. single moving average value) is a feature map. The feature map may be more than one value but must be less than the number of inputs.

In this work and many other image recognition tasks the input is a multi-dimensional image and the kernel is a multi-dimensional matrix of values. Convolutional networks typically keep the size of the kernel much smaller than the size of the input image which allow small-scale features such as edges or corners of objects in the images to be detected. Smaller kernels achieve sparse connectivity between the input units and the outputs which reduces the memory required to store and computation required to evaluate the network by decreasing the number of learnable parameters.

Parameter sharing refers to the method of using the same weights across multiple input units. In this work and many others, we use a sliding-window method in which the same kernel (which is smaller than the input image) is used uniformly across the entire image. This creates an output which is also 2-dimensional with each output ‘pixel’ being the output of the kernel function which, in this work, is matrix multiplication. The output dimensions vary depending on the border conditions (padding) and the distance in each dimension the kernel is shifted on each evaluation (stride).

The sliding-window method also results in the convolutional layer having a property called equivariance to translation. Since the same kernel is used to evaluate all windows in the image, if the same feature in an image moves to a different window the feature created in the output feature map will shift the output the exact same way. This allows relevant features, such as edges or corners, to be detected all over the image with the same set of kernel weights. While convolutional layers are equivariant with respect to translation, they are not equivariant with respect to other translations such as dilation or rotation.

* 1. **Pooling Layers**

Pooling layers are typically used after activation to perform down-sampling. The output of a pooling layer is attained in much the same way as the convolutional layer with the key difference being that there are no learnable parameters. Each output of the pooling layer is created by computing a summary statistic over a small neighborhood of the input. For example, the max-pooling operation outputs the maximum input value within the neighborhood defined by the pooling kernel. Other popular pooling functions include average-pooling, L2-pooling, and weighted average-pooling.

Pooling helps to make the model approximately invariant to small translations of the input. Invariance to translation implies that the output will remain the same when the input is shifted slightly. This complements the equivariance of the convolutional layer and can be a very useful property if one cares more about whether some feature is present than exactly where it is. Because of the loss of information when feed through a pooling layer, these types of operations are often called down-sampling.

* 1. **Regularization**

[To be determined]

1. **Dataset Exploration**

The dataset used in this study contains approximately 40,000 32x32 pixel RGB images (instances) of 43 different kinds (classes) of German traffic signs. The normalized distribution of instances over their respective class labels may be seen in Figure 1.

FIG

Figure 1: Distribution of instances over their respective class labels.

Figure 1 illustrates the non-uniformity of the distribution of instances over labels. We hypothesize that this non-uniform distribution will lead to more common labels being classified more consistently than less-common labels.

Since the training set contains far too many images to inspect each one individually we will use the pixel-by-pixel mean over all instances belonging to a certain class (mean-pixel image) to estimate the variance of perspective, contrast, color, brightness, etc. for each class label.

FIG

Figure 2: Pixel-by-Pixel mean over all instances belonging to each class

All mean-pixel images shown in figure 2 appear to retain a lot of the structure required to identify the signs they represent. This indicates that the image properties for each class are relatively well-behaved and that the classification task should not be too difficult. However we must remain wary of overfitting the dataset.

1. **Model Design**

In designing a model to classify the images in our dataset we use the LeNet-5 architecture as a baseline and evaluate variations of this architecture using a series of experiments. Then we combine the best-performing variations into a final architecture we will use to classify images in the test set.

* 1. **LeNet-5**

The LeNet-5 architecture was created by Yann Lecun et al to classify handwritten numeric digits. The input to the original network was 32-pixel grayscale images and the output was a one-hot encoding which corresponds to a probability distribution over the digits 0-9. The architecture of LeNet-5 is illustrated in Figure x.

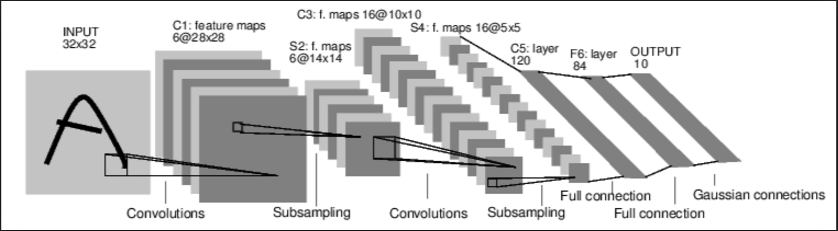


Figure x: LeNet-5 architecture

The LeNet-5 architecture consists of two convolutional layers each followed by a ReLU activation and max-pooling layer. The output of the second max-pooling layer is fed into 3 fully-connected layers, the first two of which are followed by ReLU activations and the last of which is fed into a SoftMax function which creates the desired distribution over the digits. In the implementation used in this work the number of initial input channels had to be increased to three to accommodate the color images in our dataset. Also, the output had to be increased from a 10-unit to a 43-unit vector to accommodate for the number of classes in the traffic sign dataset.

* 1. **Metrics**

The evaluation metrics we use are accuracy, precision, recall, f1-score, and computation time taken to evaluate the test set. We aim to optimize our network by maximizing test set accuracy, minimizing computation time and will use the other metrics to guide us. Precision and recall are often calculated using a confusion matrix. A confusion matrix used in binary classification can be seen below. The confusion matrix illustrates the distribution of error over the instances in the test-set.



Figure x: Binary classification confusion matrix; (clockwise from top-left) True-Positives, False-Negatives, True-Negatives, False-Positives

In multi-class classification, the confusion matrix is of size *n\*n* where n is the number of classes in the test set. The sum over rows is the distribution over actual classes in the testing set and the sum over columns will be the distribution over predicted classes output by our classifier. The confusion matrix gives a good visualization of the classifiers weaknesses and may be used to guide optimization of the network and/or future data collection.

* 1. **Experiments**

Each experiment is done on the optimal network from the previous experiment allowing the best variations to carry forward. Each variation is trained for 25 epochs with an initial learning rate of 0.001, batch size of 128 and the same augmented training set. Perturbations to the training set were random but drawn from the same distribution for each experiment. In an attempt to fairly account for optimal learning rate differences the rate is halved whenever the optimization fails to increase the accuracy by more than 5% of the 5-epoch moving-average over 3 epochs. In the following table we compare three activation functions, two types of pooling layers, and two types of regularization.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Activation Function** | **Pooling** | **Regularization** | **Test Accuracy** | **Computation Time** |
| ReLU | Max | None | 90.5% | 2.22s |
| pReLU | Max | None | 90.9% | 2.73s |
| tanh | Max | None | 91.6% | 2.19s |
| tanh | Average | None | 92.5% | 2.17s |
| tanh | Average | L2 | 94.6% | 2.07s |
| tanh | Average | Dropout | 92.5% | 2.65s |
| tanh | Average | L2 + Dropout | 92.6% | 2.68s |

Table x: LeNet-5 Experiment Results

\*

1. **Recycling**

Machine learning is often separated into two groups, supervised and unsupervised. Supervised algorithms, the ones we focus on in this work learn from experience. In supervised machine learning the goal is to create a model from some set of data that is able to efficiently extrapolate to unseen data. The simple case that we will focus on here is image classification with the goal being to correctly classify traffic signs based on images. We will create a network of functions (a neural network) that maps images, represented by a matrix of pixels, to their labels. We will create this model by first making high-level observations about the structure and representation of the data. Then we will use this knowledge to structure our initial network and optimization functions. Finally, we will expose our model to the training set and allow the optimization to find the most efficient function.