Comp 4730, Machine Learning
Project 1 - Convolutional Neural Network

Group Members:
Anthony Graziano, Cole Fuerth, Mathew Pellarin

Github
https://github.com/colefuerth/4730-Project-I

Abstract

In this report, we present a comparison between two convolutional neural networks one derived from a prebuilt Keras model and the other handcrafted with hardcoded hyperparameters. We experimented with differing amounts of layers and tuning hyperparameters to adjust our overall accuracy score and improve loss while keeping the model trainable within a set amount of time. The added complexity of adding more layers adds more time to training and uses more ram usage.

Our model, after having built a convolutional neural network from scratch, achieved a result of 70% on the test image segment, compared to our implementation of a Keras model with prebuilt layers, which performed at a rating of 95%.

Introduction

Convolutional neural networks (CNNs) are generally sparsely connected feedforward networks that possess convolutional layers which employ kernels (a.k.a. filters) that scan over the inputs or content of a previous layer and produce an output activation layer by computing a tensor through a particular type of linear operation [1]. Like Multilayer perceptrons, a type of densely connected feedforward network, and neural networks broadly, they are classes of composed parameterized and differentiable functions, referred to as layers, linked together with similar stages or heterogeneous structures to form outputs [2]. Such networks form computational graphs of dependencies between nodes (neurons) that expose parameters to continuous optimization [3]. These optimizations often take the form of backpropagated gradient descent with automatic differentiation on a loss function (which measures performance) with respect to the weights, or the kernel in the case of a CNN [7]. Deep learning is a category of machine learning hypothesis model types that stack many successive layers each retaining transformed, increasingly refined representations [4]. CNN's accomplish this through the architectural application of interleaved convolutional, pooling, and non-linear activation functions, ending in a fully connected (dense layer) for classification.

Inspiration for one of the earliest models, Fukushima’s Neocognitron, which later CNN's adapted from, came from the neuroscientific experimental results of Hubel and Wiesel in which the cat’s visual cortex contained a hierarchy of cell types that feature extracted edges, shapes,

and motion from visual stimuli [3]. Although deep, the Neocognitron's weights were not optimized by supervised backpropagation, but instead were either handset or based on unsupervised, local, winner-take-all learning algorithms [5]. Through the years between 1979 and 2012, there was much progress in developing backpropagation with advanced gradient descent and it was applied by LeCun to CNNs in 1989 [5, 8]. However, modern visual classifiers were not in vogue until AlexNet put deep learning in the foreground of the minds of machine learning researchers as it outperformed all other competing models by a large margin on the ImageNet Large Scale Visual Recognition Challenge. As of now CNNs can be found in many areas and have come to proliferate executing all sorts of visual classification tasks [6].

Contributions

All members were responsible for implementing this convolutional network, code review and the report write-up. Specific tasks include:

Anthony G

- Implemented backpropagation
- Code overview

Cole F

- Implemented Dense Layer, Flattened layer and developed base code
- Code cleanup and optimization

Mathew P

- Implemented Convolution layer and Pooling layer.
- Code debugging

CNN Theory Review

Convolutional neural networks are usually made up of four major filters. The convolutional layer, pooling layer, flattening layer, and a fully connected layer. These layers work together to build a fully developed neural network. Convolutional neural networks are great for categorizing photos and detecting patterns within an image.

Convolution is the application of a weighted average at every stride (the distance the kernel is translated across the image) with the weights being contained in the kernel itself. Displacement of the superimposed passing kernel over the image (or previous layers activation function) produces a feature map whose multi-dimensional array entries are the summed products of the displaced kernel entries element-wise with the image and an added bias [1]. Technically the kernel must be convolved by rotating 180 degrees, however, it is sufficient in neural networks to simply take the cross-correlation as described above. In a normal feedforward neural net, the outputs are each fully connected to all inputs through a matrix multiplication operation, but CNN's are equipped with sparse connections through the application of a kernel smaller than the image. With this as the case, CNN's keep fewer cached parameters which means lower operational complexity and improved efficiency as a result [1]. The receptive field (the

parameterized neurons connected convergently over the layers to the neuron under inspection further downstream) for each deep-layered neuron keeps them connected to most of the image. Secondly, shared parameters between kernel operations ensures further reduction in storage usage yielding increased efficiency compared to fully connected networks. This sharing also causes there to be equivariance, a type of input-to-output coupling whereby the change in input changes the output closely [1].

Pooling is another associated activation function performed in a CNN with the primary objective of compressing the convolutional feature map into statistical array entries that summarize neighbouring values in the feature map tensor. Max pooling in particular takes smaller rectangular arrays sectioned from an array of unit activations and downsample into a layer composed of the maximally active units compiled from their respective rectangular arrays [5]. Invariance to transformations in the image result from pooling as the post-pooling parameters are activated strongly by various detector units each having responded to differing orientations of the image captured by the filters [1]. Added efficiency is also gained statistically and computationally as there are statistical summations and fewer operations to perform respectively.

These components are two of the pivotal components responsible for the architecture and overall success of CNNs alongside the rectified linear unit non-linear activation layer and the important backpropagation with gradient descent.

Experimental setup and Methodology

The Setup

Our code starts off by importing the mnist data set from the TensorFlow library. We shrink the data set to one-thousand training data points and one–thousand testing points. This new set is large enough to train with but not obscure where training takes a long time.

The default model we chose to implement has a total of five layers. The image is passed through each of the five layers starting with the convolutional layer. The next layer the image is passed through is the pooling layer, this keeps all the significant data within the image but reduces the number of pixels the network must go through. The next layer the image goes through is the flattening layer. This layer takes the array of the image and makes it into a single dimension. The last 2 layers the image is transposed to are the dense layers. This layer is the neurons and the fully connected layer.

```
model.append(Conv2D(32, 2, 1, 1, activation='relu'))
model.append(Pooling(2, 2, 'max'))
model.append(Flatten())
model.append(Dense(np.prod(dims.shape[1:]), 128, activation='relu'))
model.append(Dense(128, 10, activation='relu'))
```

Convolutional Layer

The convolution layer has 5 input parameters, num_filters, spatial_extent, stride, zero_padding, and activation. The first parameter, num_filters, is an integer value designated in creating the number of filters that will be applied to the image. The spatial_extent parameter is the size of the filter in a 2d array. The stride parameter is used to control how much the filter moves across the image. The zero_padding parameter is used to add padding to the image. The activation parameter is used to determine what activation function will be used. The convolution layer is used to apply filters to the image and to extract features from the image. The filters are applied to the image by sliding the filter across the image. The filter is moved across the image by the stride parameter. The filter is applied to the image by multiplying the filter by the image and adding the bias. The output of the convolution layer is the result of the filter being applied to the image. The output of the convolution layer is then passed to the pooling layer. padding is also applied to the image to make sure the image is the same size as the original image.

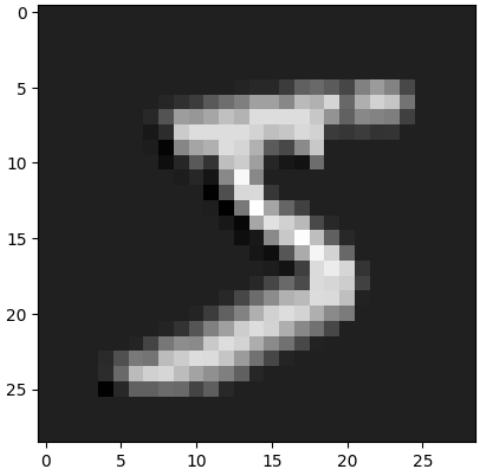

Figure 1: Feature map of the digit after a pass through a convolutional layer

Pooling Layer

The pooling layer has 3 input parameters, spatial_extent, stride, and mode. The spatial_extent parameter is the size of the filter in a 2d array. The stride parameter is used to control how much the filter moves across the image. The mode is which filter to apply whether the max filter or average filter is used. The pooling layer is used to reduce the size of the image by sliding a filter across the image. The filter is moved across the image by the stride parameter. The filter is applied to the image by either getting the max value of its array or the average of the array. The output of the pooling layer is the result of the filter being applied to the image. The output of the pooling layer is then passed to the flattening layer.

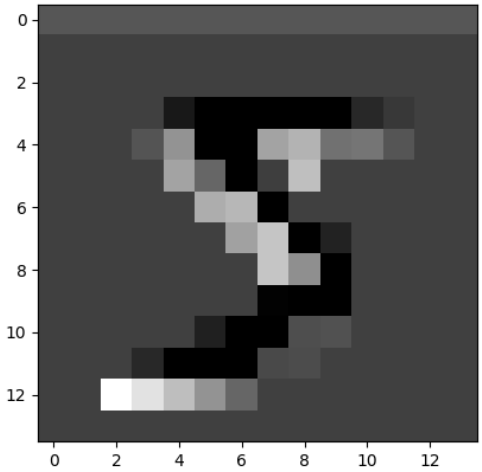

Figure 2: Feature map of the digit after a pass through a pool layer

Flatten Layer

The flattened layer has no input parameters. The flattening layer is used to flatten the image into a single dimension. The output of the flattened layer is then passed to the dense layer.

Dense Layer

The dense layer has 3 input parameters, input_size, output_size, and activation. The input_size parameter is the size of the input array. The output_size parameter is the size of the output array. The activation parameter is used to determine what activation function will be used. The dense layer is used to connect the neurons together. The weights of each neuron are randomly generated and then the output of the neuron is calculated by multiplying the weights by the input and adding the bias.

Testing

During our testing, our neural network achieves a testing accuracy of about 70%. This is not the best achievable result and our neural network could use more layers and optimization. Comparing our model to a completed neural network such as Keras, a test result of 95% accuracy can be achieved when training the same dataset.

```
Train: X=(10000, 28, 28, 1), y=(10000,)
Test: X=(1000, 28, 28, 1), y=(1000,)
loss = 14.736452312321356   accuracy = 49.27%
loss = 8.735814235556202   accuracy = 75.12%
loss = 7.3109634963510794   accuracy = 80.80000000000001%
loss = 6.620557965528036   accuracy = 83.32000000000001%
loss = 6.189683138953189   accuracy = 84.83999999999999%
loss = 5.882243207349753   accuracy = 85.70999999999998%
loss = 5.648579267915426   accuracy = 86.49%
loss = 5.461908518294826   accuracy = 87.13%
loss = 5.307371794883893   accuracy = 87.74%
loss = 5.175245833299928   accuracy = 88.27999999999999%
Test accuracy: 66.60%
```

Figure 3: Results from CNN

Conclusion

Our convolution neural network is fully established and can decipher the majority of test images that are tested. The overall network hyperparameters can be tweaked as well as more layers can be added to improve the accuracy.

References:

1. Goodfellow, I., Bengio, Y., Courville, A. (2016). *Deep Learning*. Chapter 9, pages 321 - 361.
2. Chen, T., Kolter, Z. (2022). *Deep Learning Systems: Algorithms and Implementation* (online course Carnegie Mellon University). Lecture 3 - Manual Neural Networks/Backprop.
3. Aggarwal, C.C. (2018). *Neural Networks and Deep Learning*. Preface, VIII, Chapter 1, page 40.
4. Chollet, F. (2018). *Deep Learning with Python*. Chapter 1, pages 1 - 23.
5. Schmidhuber, J. (2014). *Deep Learning in Neural Networks: An Overview*. Neural Networks, 61: pages 85 - 117.
6. Lecture Slides .(2022). *Introduction to Convolutional Neural Networks-4*.
7. Karpathy, A. (2016). Lecture 4. *Backpropagation Neural Networks I*.
8. LeCun, Y., et. al. (1989). *Backpropagation applied to handwritten zip code recognition*. Neural Computation. 1(4): pages 541 - 551