James Taylor

Assignment 5

University of Maryland University College

DATA640

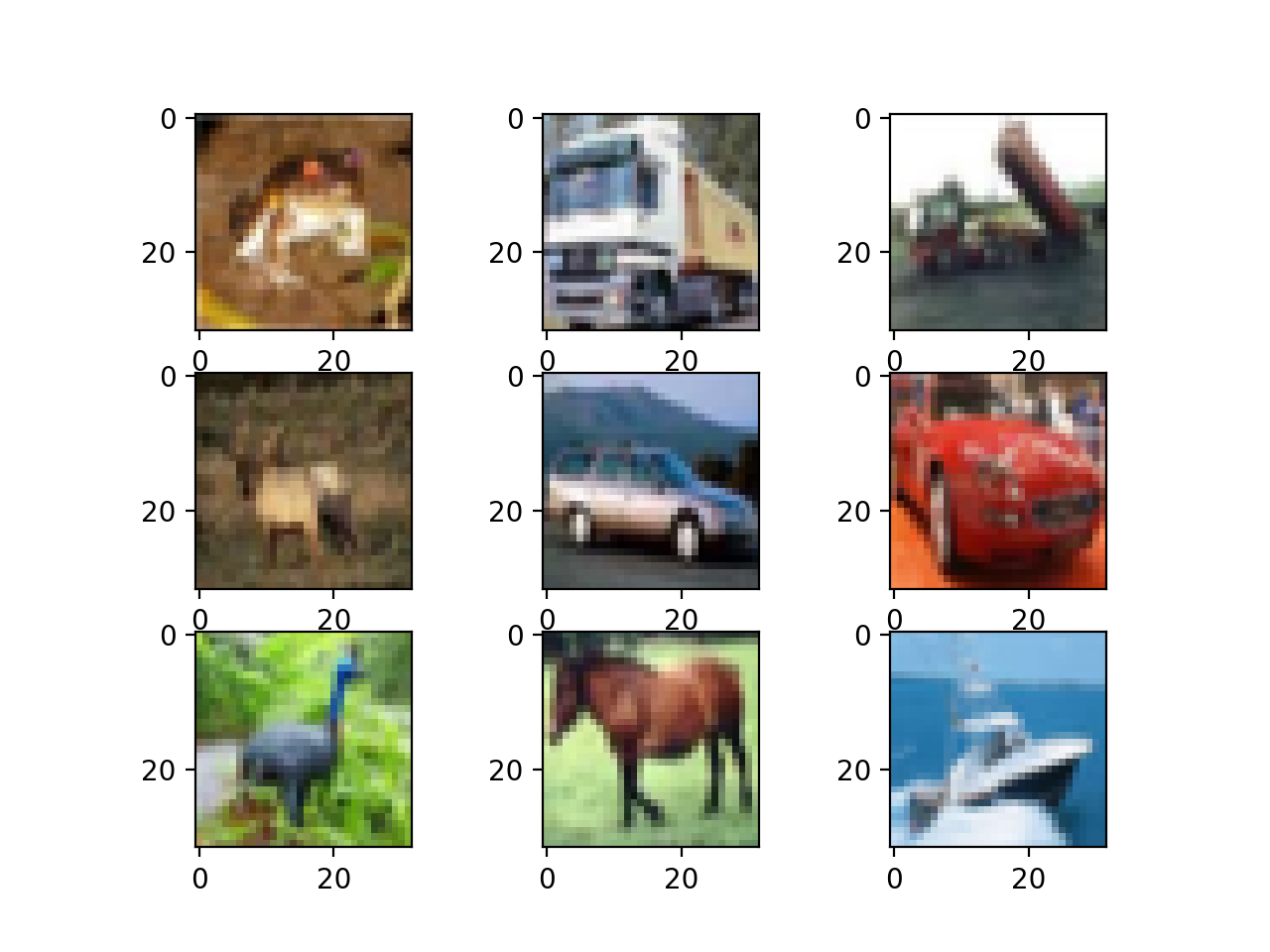
Dr. Knode

james.taylor.ps3@gmail.com

**Introduction**

The goal of this paper is to use deep learning techniques to make accurate predictions on images. A variety of types of convoluted layers and structures will be used to achieve more accuracy. This is to help understand how CNN’s work and what works best.

The dataset used for this paper was the CIFAR-10 image set. It consists of 60,000 images that are 32x32 pixels in size. They are all labeled as airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. There are 6,000 images for each category. The data has four dimensions. Firstly, is the individual image, the next two are the x-y position of the pixel in the image, and the last is the red-blue-green value of the individual pixel. Below are some example images.



**Data Preparation**

All the variables had the same scale which is required for neural networks. The dataset was partitioned into training and test sets of 50,000 and 10,000 respectively. There were no missing values or outliers.

At model 4 the input values were scaled to between zero and one. The rescaling was done to help the neural networks computation for gradient descent. All values where divided by the maximum possible value an individual channel a pixel has which was 255. This set the maximum possible value after the transformation to one.

**Predictive Models Developed**

The models used were all built using the Tensorflow library in a Python 3 environment. All the models were from the Keras library of the Tensorflow library. All models were 2-dimensional convolutional neural networks.

The most important layer that will be used is a 2-dimenstional convolutional layer. This technique uses a filter that slides over the inputted image’s pixel values. A filter is like a grouping of pixels that are close to one another. An equation is applied to the values in the filter which is then placed into a feature map. The predictive value is the back propagation. This is where the equation is tuned that delivers the best results in distinguishing between categories. This feature map is what a model looks at to make a prediction.

A pooling layer is another important layer. This takes the average or maximum of a section of the feature map and creates a less-detailed feature map. This can help with accuracy and computational effort.

The Appendix has the model descriptions in *Figure 1*. A variety of combinations of layers were used. They are listed sequentially as they were in the model. The models are listed sequentially as well, each model is an attempt to improve on the previous one’s accuracy. In most cases the models become more complex in an attempt to fit the data better.

The number of epochs also increased as more models were developed. This was also a means to improve the accuracy.

The overall approach that was taken was to train models with low number of epochs. The assumption is the best performing model at 100 epochs will be the best performing one at 400 (this may not be correct to assume). Working from this assumption would save time because the training all the models to 400 epochs would be a large computational effort.

The kernel and pooling sizes will be tuned to see what returns high accuracy. Because these images are low resolution with few pixels, large kernels may not extract the details necessary to distinguish between categories. The image is only 32x32 pixels in size so the more detail we can extract from it will likely lead to more accurate models. Pooling size has the same concern, where if a pool is too large it makes the details of the image not available to the model.

**Results**

The clearest action that lead to improvements was clearly increasing the epochs. With 10 epochs, our models got up to 40% accuracy. With 100 epochs the models were more complex and achieved 57%. The one model we built with 400 epochs was 68% accurate. While the complexity and layers did change while the epochs were increased, the connection to increasing the epochs appears to be the main cause of increased accuracy. The layer tuning alone would not achieve such results. Looking at the difference between models 4 and 5 where the only is the difference between them is the number of epochs, increasing from 10 to 100 had accuracy increase by 18%.

The models that performed better in the different epoch levels were the ones with many layers with small kernels and pool size. Compare models numbers 1,2 and 3 where the only difference between the two are the kernel size and pool size. The first model had small kernels and small pool size. The accuracy decreased somewhat when the kernel size increased from 2x2 to 6x6 in the second model. From the second to the third model the pool size was increased and accuracy took a significant dive. Another example is the difference between models 7 and 8. Decreasing the kernel and pool sizes increased accuracy about 5%.

The best performing model with 100 epochs was selected to have 400 epochs. This achieved an accuracy of 68% which was 11% higher than its 100-epoch performance. That level of accuracy is impressive compared to randomly guessing which would expectedly return a 10% accuracy.

**Conclusions and Takeaways**

The more layered models with smaller kernels and pooling sizes performed better. This is likely because of the smallness of the image to begin with. Larger kernels boil down the details of an image into smaller feature maps. Pooling layers reduce that map even more. The more categories a model has to predict the more numbers it needs to distinguish between them. If this model was built to have a True/False detection it would be able to achieve similar results with a less layered model, or higher results with a similarly-complex model.

Large pooling sizes appear to make a significant negative impact on accuracy. This may be because the model need lots of detail to decide between the 10 categories it’s meant to predict on. The more layered models did perform better overall. While there is diminishing returns, more exploration could be done on how layered a model like this should be.

Appendix A

Figure 1 – Models

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Model # | Layers Sequential Listed | epochs | Total train time | Loss | Accuracy (test data) |
| 1 | 1. Convolutional 2D    1. Filters = 128    2. Kernel size = 2x2 2. Max Pooling 2D    1. Pool size = 2x2 3. Dropout    1. Rate = 0.2 4. Batch Normalization 5. Flatten 6. Dense    1. Units = 10    2. Activation = softmax | 10 | 922 seconds  (No GPU used) | 1.718 | 0.408 |
| 2 | 1. Convolutional 2D    1. Filters = 128    2. Kernel size = 6x6 2. Max Pooling 2D    1. Pool size = 2x2 3. Dropout    1. Rate = 0.2 4. Batch Normalization 5. Flatten 6. Dense    1. Units = 10    2. Activation = softmax | 10 | 30 seconds | 1.765 | 0.392 |
| 3 | 1. Convolutional 2D    1. Filters = 128    2. Kernel size = 6x6 2. Max Pooling 2D    1. Pool size = 5x5 3. Dropout    1. Rate = 0.2 4. Batch Normalization 5. Flatten 6. Dense    1. Units = 10    2. Activation = softmax | 10 | 20 seconds | 1.885 | 0.326 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Model # | Layers Sequential Listed | Epochs | Total train time | Loss | Accuracy (test data) |
| 4 | 1. Convolutional 2D    1. Filters = 128    2. Kernel size = 6x6 2. Max Pooling 2D    1. Pool size = 5x5 3. Dropout    1. Rate = 0.2 4. Batch Normalization 5. Flatten 6. Dense    1. Units = 10    2. Activation = softmax | 10 | 20 seconds | 1.852 | 0.336 |
| 5 | 1. Convolutional 2D    1. Filters = 128    2. Kernel size = 6x6 2. Max Pooling 2D    1. Pool size = 5x5 3. Dropout    1. Rate = 0.2 4. Batch Normalization 5. Flatten 6. Dense    1. Units = 10    2. Activation = softmax | 100 | 200 seconds approx. | 1.406 | 0.510 |
| 6 | 1. Convolutional 2D    1. Filters = 128    2. Kernel size = 6x6    3. Activation = ReLU 2. Max Pooling 2D    1. Pool size = 5x5 3. Dropout    1. Rate = 0.2 4. Batch Normalization 5. Flatten 6. Dense    1. Units = 10    2. Activation = softmax | 100 | 200 seconds approx. | 1.378 | 0.527 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Model # | Layers Sequential Listed | Epochs | Total train time | Loss | Accuracy (test data) |
| 7 | 1. Convolutional 2D    1. Filters = 128    2. Kernel size = 6x6    3. Activation = ReLU 2. Convolutional 2D    1. Filters = 128    2. Kernel size = 3x3    3. Activation = Relu 3. Max Pooling 2D    1. Pool size = 5x5 4. Convolutional 2D    1. Filters = 64    2. Kernel size = 3x3    3. Activation = ReLU 5. Dropout    1. Rate = 0.2 6. Batch Normalization 7. Flatten 8. Dense    1. Units = 10    2. Activation = softmax | 100 | 700 seconds approx.. | 1.352 | 0.528 |
| 8 | 1. Convolutional 2D    1. Filters = 128    2. Kernel size = 2x2    3. Activation = ReLU 2. Convolutional 2D    1. Filters = 128    2. Kernel size = 3x3    3. Activation = Relu 3. Max Pooling 2D    1. Pool size = 2x2 4. Convolutional 2D    1. Filters = 64    2. Kernel size = 2x2    3. Activation = ReLU 5. Dropout    1. Rate = 0.2 6. Batch Normalization 7. Flatten 8. Dense    1. Units = 10   Activation = softmax | 100 | 864 seconds | 1.211 | 0.577 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Model # | Layers Sequential Listed | Epochs | Total train time | Loss | Accuracy (test data) |
| 9 | 1. Convolutional 2D    1. Filters = 128    2. Kernel size = 2x2    3. Activation = ReLU 2. Convolutional 2D    1. Filters = 128    2. Kernel size = 3x3    3. Activation = Relu 3. Max Pooling 2D    1. Pool size = 2x2 4. Convolutional 2D    1. Filters = 64    2. Kernel size = 2x2    3. Activation = ReLU 5. Dropout    1. Rate = 0.2 6. Batch Normalization 7. Flatten 8. Dense    1. Units = 10    2. Activation = softmax | 400 | 3,520 seconds | 0.910 | 0.683 |