ECE594BB Homework 2 Report

Nikita Buzov

02/02/2019

1. CodaLab username: NikitaBuzov; score: 0.7785.
2. **Problem:**

In this exercise, we were given a data set that contains 50,000 32x32-pixel RGB images, each corresponding to one of the 10 categories. Using this data, we had to develop and train a convolutional neural network (CNN) such that it would be able to classify the future images into one of these 10 categories with accuracy above 80%. A set of testing data (10,000 images) was provided to make predictions and test our neural network.

1. **My CNN architecture:**
   1. **Preprocessing.**

The data set was given as a 50,000x3072 numpy array. In order to feed the images into the neural network this data had to be processed. First, I reassembled each row of that array so the values would be arranged pixel by pixel, instead of first red values, then green, and lastly blue. For this purpose, I wrote a python program preprocessing.py which saved new numpy arrays: test\_data\_v2.npy and train\_data\_v2.npy. Then these files where uploaded to the Google Drive along with the labels data and python CNN code.

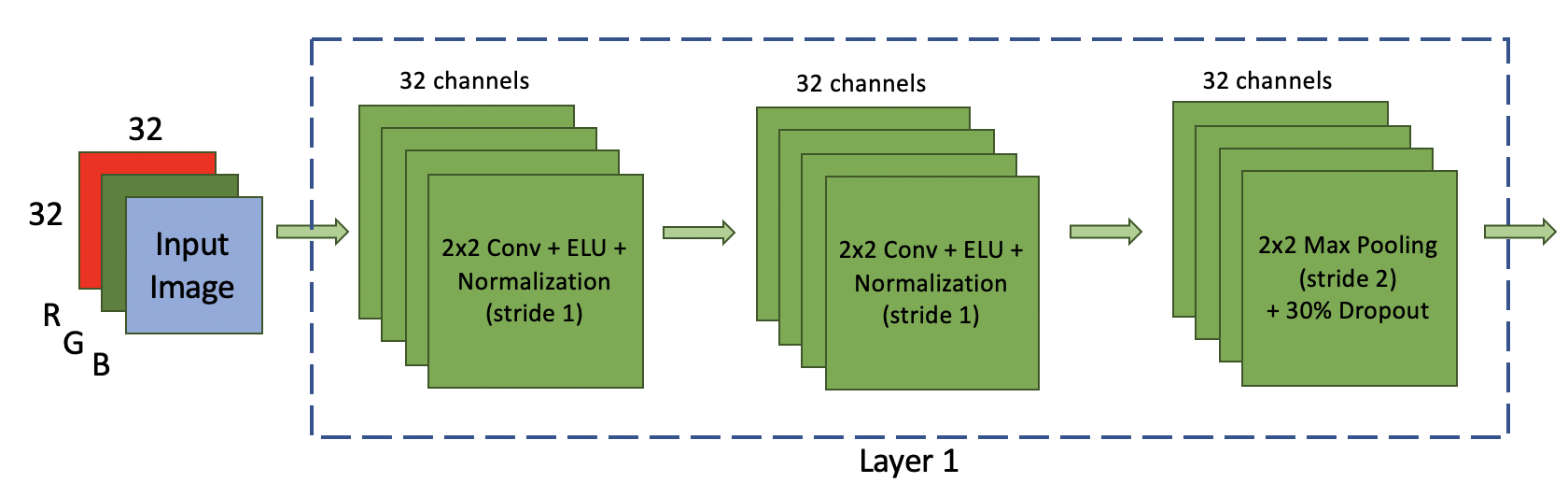
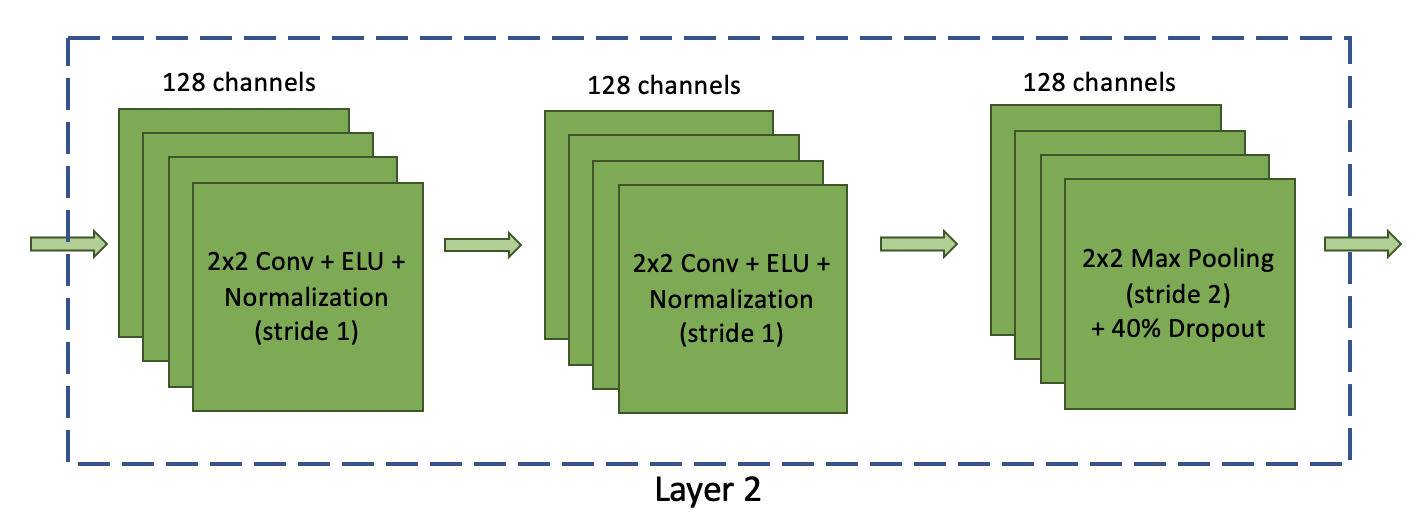
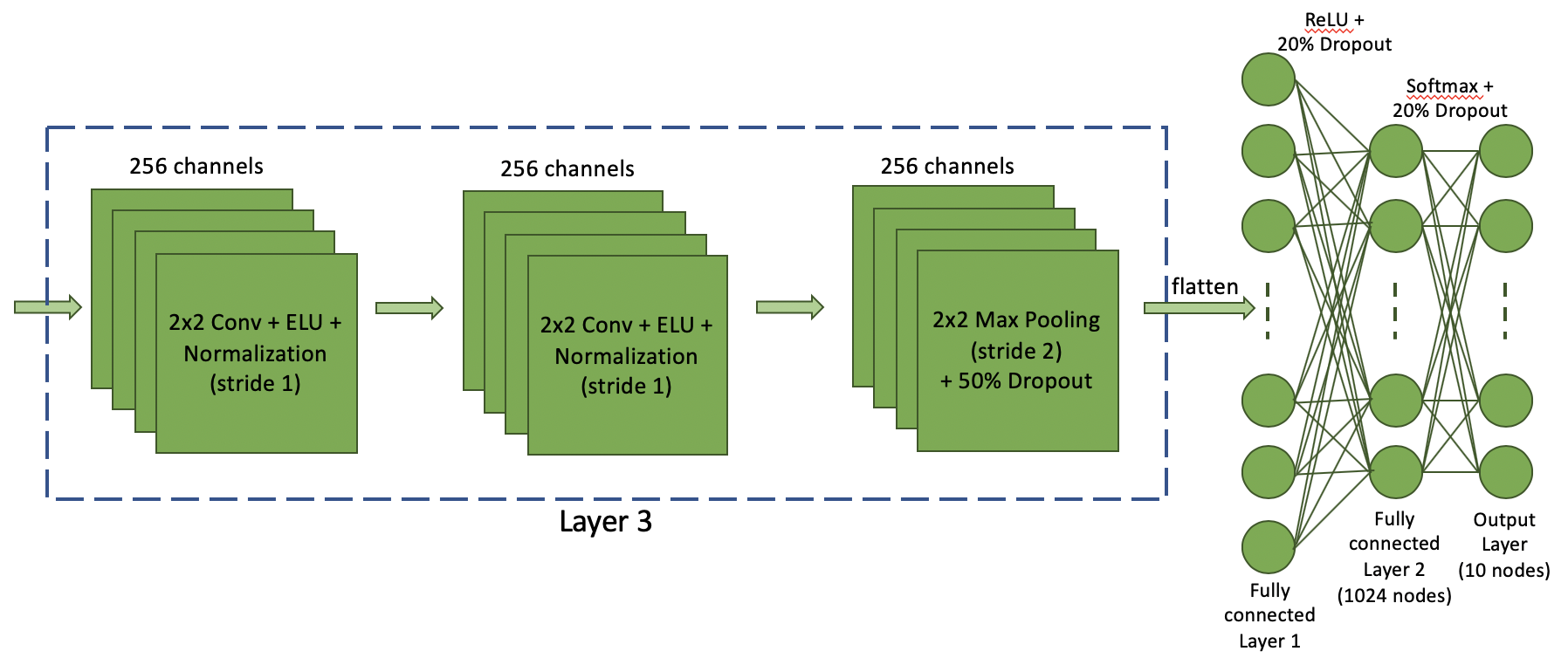
The next step in preparing the data was splitting the training data in two sets: one for actual training and the other for model validation. I trained my model with 40,000 images and used the remaining 10,000 to check the validation accuracy.

Third, the training and validating data had to be reshaped from 2D array, with 3072 inputs in each row, to a 4D tensor of shape (number of samples, x image size (32), y image size (32), 3 channels for RGB)).

The next step was to convert the data to the right type. Each data point was converted into float32 type and then divided by 255, so pixel values would range from 0 to 1.

And the last step was to convert labels into binary class matrices for categorical cross-entropy loss calculations.

* 1. **CNN model.**

For this problem, I used a sequential model, which is basically a linear stack of layers. The convolutional neural network that I have developed consisted of 3 main layers each consisted of two convolutional steps, two activation steps, two batch normalization steps, one max pooling step, and then a dropout regularization step; then I had 3 more fully connected flat layers: input flattening, 1024-node layer, and finally the 10-node output layer. These last layers also had dropout regularization between them. The diagram of the full convolutional neural network is presented below.

* 1. **Challenges and Solutions.**

The greatest issue during the training was overfitting. Generally, it is better to get first get the model from underfitting to overfitting and then use different techniques to make the model more generalized. To overcome this challenge, I used primarily dropout regularization. I started with doing a 20% dropout every step of the way, which decreased overfitting. I later discovered that it is better to increase the dropout rate as the model proceeds through training, so for each new layer I’d increase dropout rate by 10%. For the fully connected layers I used flat 20% dropout regularization between each layer.

Another technique that improved the accuracy of my model was increasing the number of convolution channels in each layer. This increased the amount of time to train and how quickly the model improves upon each subsequent epoch. However, ultimately the accuracy got better.

I also found that doing convolution on a 2x2 image section yields better results compared to larger areas. Though this also slowed down the training process.

Lastly, increasing the number of epochs also improved the accuracy. However, this was true when overfitting wasn’t too bad. Otherwise, the model’s cross validation accuracy converged to a constant value and would not improve any more, no matter how many epochs more it trained trough.

* 1. **Results**.

After training the model with 30 epochs, I was able to achieve the validation accuracy of 82.78%, while the training accuracy was at 86.37%. My model surpassed 80% accuracy rate after 17 epochs and began overfitting around 15th epoch. However, the final validation accuracy is not too far from the training one, so I can say that overfitting was overcome for the most part. The final test loss was at ~0.7721.