# DEEP LEARNING FINAL REPORT

# BY


# CHRIS LEE, JIAHUI LI, MINGHAN SUN & WILLIAM SALAS


# MAY 2020

# MSA 8600

# Motivation / Problem Description:

Not too long ago, if we were to give a computer an input of images and asked the computer to tell us what is in the image, the computer would not be able to comprehend. However, it is now showing up in all sorts of different kinds of software applications. Our motivation is to learn more about and how to, use neural networks to build a customized image recognition system.

# Abstract:

We will be using the CIFAR-10 dataset. This dataset includes thousands of pictures of 10 different types of objects: airplane, automobile, bird, etc. Each image in the dataset has a matching label so we know what type of image the picture is. Using this data set, we will create a neural network and train it to recognize any of these 10 different label classes.

# Dataset:

CIFAR-10 Data consists of 60,000 color images of size 32 pixels x 32 pixels, which are considerably low-resolution images. In the data, we have 10 label classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. Of the 60,000 images, we have 6,000 images of each class. The data we are given is split into two categories: training, and testing. The training data consists of 50,000 images. The testing data consists of 300,000 images: 290,000 of which are false images, and 10,000 of which are the remaining images of our 60,000-image dataset. Using the training data set of 50,000, we will be randomly splitting it once again into a training and validation set. Training on 50,000 of them and validating on 10,000 of them. What is nice about these images is that they are mutually exclusive, in which no single image would be classified under two classes.

# Pre-Processing:

Before we can use the data to train our neural network, we first have to normalize the data. Neural networks work best when the input data are normalized, a floating value between 0 and 1. To normalize our data, we converted our images to floating type and then divided the floating value by 255. We started first by converting to float because images tend to be stored as an integer, this is just a precaution. We then divided by 255 because our images' pixel data is between 0 and 255. We have now normalized and scaled our data to be between 0 and 1. We have done this for both the training and testing datasets.

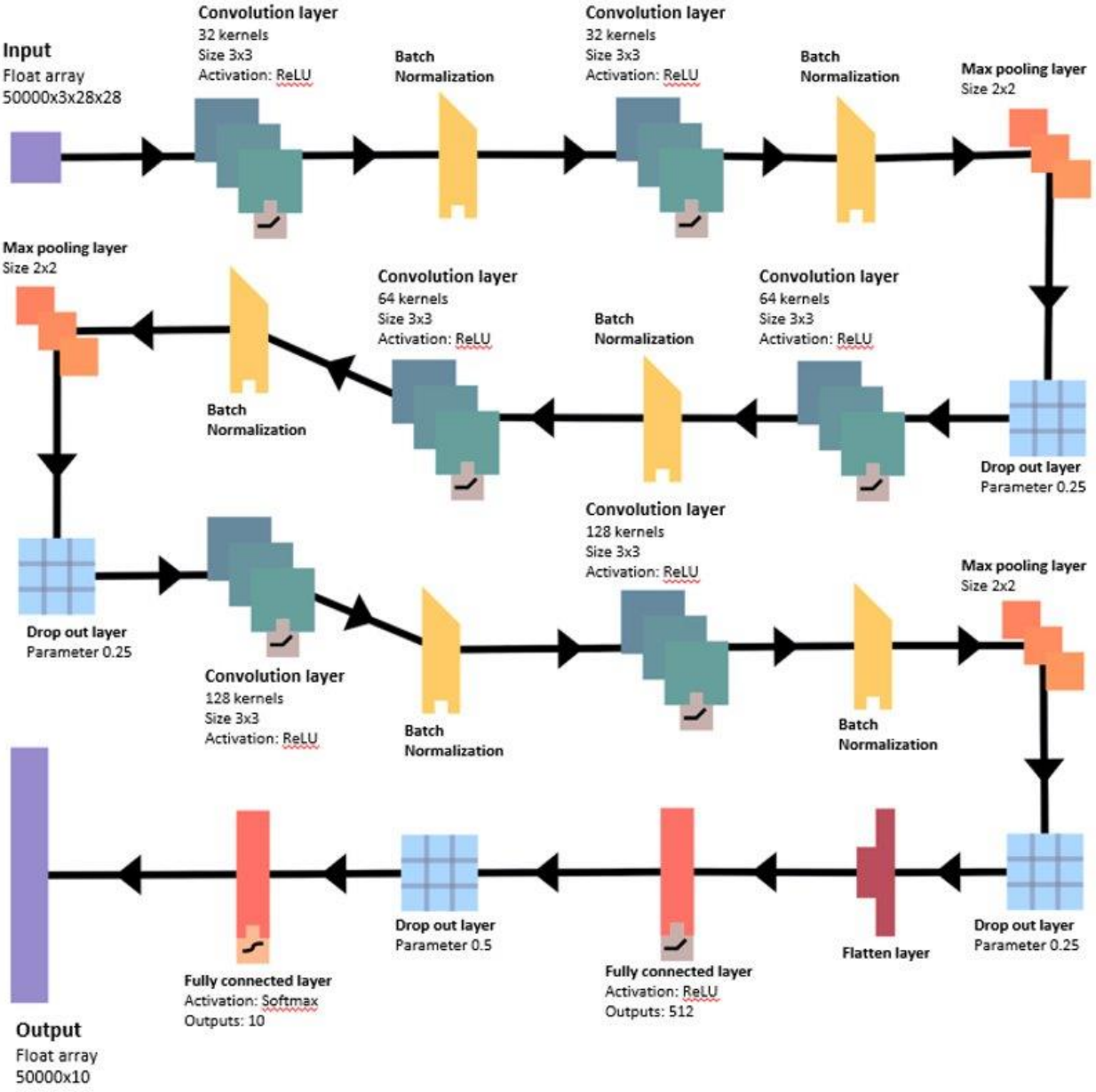
Instead of having our model only being able to detect one kind of class/label for the image, we want to build our model such that it can produce more than just a single output. To do this, we separated our expected value for each of the images. Since we know that CIFAR-10 has 10 classes,

we will be creating a neural network of 10 outputs (the 10 classes). To separate the expected values, we converted each of the labels/classes into an array with 10 elements. Of the 10 elements, one element is set to 1, and the rest will be set to 0.

# Final Algorithm's Structure:

- One Convolutional Layer with 32 filters of size 3 x 3 with padding, and an activation function with 'ReLU'. Input Shape of size 32 x 32 with 3 channels.
- One Batch Normalization.
- One Convolutional Layer with 32 filters of size 3 x 3 with padding, and an activation function 'ReLU'.
- One Batch Normalization.
- A max-pooling of size 2 x 2.
- A dropout rate of 0.25.
- One Convolutional Layer with 64 filters of size 3 x 3 with padding, and an activation function with 'ReLU'.
- One Batch Normalization.
- One Convolutional Layer with 64 filters of size 3 x 3 with padding, and an activation function 'ReLU'.
- One Batch Normalization.
- A max-pooling of size 2 x 2.
- A dropout rate of 0.25.
- One Convolutional Layer with 128 filters of size 3 x 3 with padding, and an activation function with 'ReLU'.
- One Batch Normalization.
- One Convolutional Layer with 128 filters of size 3 x 3 with padding, and an activation function 'ReLU'.
- One Batch Normalization.
- A max-pooling of size 2 x 2.
- A dropout rate of 0.25.
- Flatten
- One connected hidden dense layer of size 512, with activation function 'ReLU'.
- A dropout rate of 0.50.
- One connected hidden dense layer of size 10, with activation function 'Softmax'.
- Size of mini-batches = 64.
- Epochs = 60.

**The model we just described looks as follows:**



# Neural Network Structure Explanation:

To begin building our neural network, the complexity of our model structures gradually increases as we add more complex layers. To construct the neural network, we first created a new neural network object so that layers can be added later on. The sequential model allows us to add layers with any sequence we desire, and the number of layers we would like to try. From there we begin to build the neural network from simple to complex. To start off with any neural network, we have to define an input shape, which is dependent on the original image size. In our case, a 32-pixel by 32-pixel image. Throughout our neural network, we will be using the activation function, Rectified Linear Unit (ReLU), which is a standard typical choice for an activation function when dealing with image data. It is proved to be less computationally expensive and efficient in separating image classes.

One of the first types of layers that we have implemented is the convolution layer. Since we are working with images, we want a two-dimensional convolutional layer. Although this is one of the first types of layers we have added to our model, we have used this layer multiple times throughout to make our neural network much more powerful. In each of our convolution layers, we split the input image with a specified amount of filters into 3 x 3 tiles, and in case our image and input are not divisible by 3, we have added padding to the output. Padding is added in each of the convolutional layers to make sure our image data can be completely separated by the window.

Along with the two-dimensional convolution layers, to further improve our model's performance, we have added Normalization layers throughout our neural network, generally right after each convolutional 2d layer. Batch Normalization standardizes the output from the previous con2d layer. The results will be the new input with a mean of 0 and a standard deviation of 1 for the next layer to the process. By normalizing the output, each feature/column is on the same scale for comparing the importance, so that the following layer, Max Pooling, can better select the most useful data without bias. Our model's performance drastically increased after adding the normalization layer.

As just recently mentioned, the next type of later we have added to improve efficiency is the max-pooling layer. By adding max-pooling we are able to scale down the output and only extract the most useful data. It not only speeds up the process, but also makes the network more efficient since it only deals with the most useful data and ignores less valuable data.

A general problem in neural networks is that neural networks can tend to memorize the input values instead of actually learning how to tell objects apart. To deal with this, we added Dropout layers. Dropout layers are usually directly after max-pooling layers or after a group of dense layers. In our case, we have added them after each max-pooling layer. By adding the Dropout layer, we force our neural network to work harder to learn without memorizing the input data. The idea is

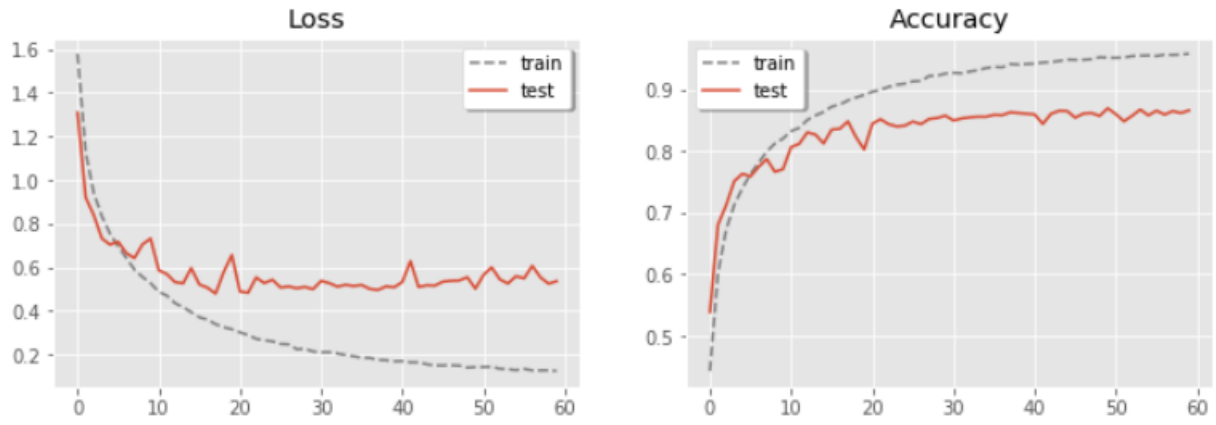
that we will randomly leave out a proportion of output coming from the previous layer (usually 25% or 50%). Therefore, the next group of layers will learn better and harder about how to separate the objects well instead of focusing too much on learning the connection between layers and memorizing the original data.

Every time we transition from a convolutional layer to a dense layer, the Flatten layer is added in order to exit from working with 2D data. The simplest layer is dense layers. In dense layers, the inputs required are the number of nodes and the type of activation function. After we told our neural network we are no longer working with two-dimensional data, we added in the two dense layers, with the final dense layer having the number of nodes to be the number of classes of prediction (10 in our case because there are 10 distinct types of classes we would like to cluster). Softmax is almost always used as an activation function in the output layer when the output clusters are more than two because the softmax activation function can guarantee that all of our output values add up to 1.

Finally, we are ready to compile our model. Categorical Cross-entropy is used as the loss function to evaluate our network. Adaptive Moment Estimation (adam) is used as an optimization algorithm, as it is often the choice for image data modeling.

# Model Training:

When training the model, 64 images will be randomly selected during each training (i.e., batch_size = 64), and a maximum of 60 pass-throughs of the entire data ( i.e., epochs = 60). During each pass-through training, the validation set will be used to test the model accuracy. The optimal epoch number is picked at the point where validation set accuracy is highest.



# Results / Predictions:

| Real \ Pred | Plane | Car | Bird | Cat | Deer | Dog | Frog | Horse | Ship | Truck | Total |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Plane** | **902** | 12 | 14 | 6 | 7 | 0 | 6 | 9 | 29 | 15 | 1000 |
| **Car** | 6 | **933** | 0 | 2 | 1 | 1 | 3 | 0 | 7 | 47 | 1000 |
| **Bird** | 50 | 1 | **769** | 35 | 56 | 34 | 30 | 19 | 5 | 1 | 1000 |
| **Cat** | 14 | 3 | 34 | **756** | 33 | 75 | 42 | 19 | 9 | 15 | 1000 |
| **Deer** | 14 | 1 | 25 | 32 | **874** | 16 | 14 | 18 | 4 | 2 | 1000 |
| **Dog** | 7 | 4 | 19 | 143 | 24 | **764** | 8 | 28 | 2 | 1 | 1000 |
| **Frog** | 3 | 2 | 27 | 32 | 16 | 5 | **908** | 2 | 3 | 2 | 1000 |
| **Horse** | 8 | 2 | 10 | 26 | 24 | 23 | 5 | **898** | 0 | 4 | 1000 |
| **Ship** | 47 | 8 | 3 | 3 | 2 | 1 | 2 | 2 | **922** | 10 | 1000 |
| **Truck** | 18 | 29 | 1 | 6 | 0 | 0 | 0 | 2 | 12 | **932** | 1000 |
| **Total** | 1069 | 995 | 902 | 1041 | 1037 | 919 | 1018 | 997 | 993 | 1029 | 10000 |

# Conclusions & Suggestions:

Thanks to deep learning, image recognition systems have widely improved and are now basically used almost everywhere. From this project, we have learned to build and structure a deep neural network to classify a set of images into specified classes and labels. The model that we have built does a very good job at differentiating animals from transportation vehicles. However, some problems that our model has is that there is a slight problem in distinguishing cats from dogs. 14% of the dogs are classified as cats and 7.5% of the cats are classified as dogs.

To improve our model, we have some suggestions. Firstly, we can perform feature engineering and feature selection techniques to further modify the data prior to modelling. For example, PCA can be used to reduce the dimension of the image so that data is more representative of each label. Secondly, we could have always modified some pre-trained neural networks, such as exploring some cloud-based image recognition systems APIs to use as some alternative to remodel from. However, we did not want to start from a pre-trained, pre-existing model, we wanted to start from scratch and see how well we could make our own image recognition system.