Machine Learning Final Project

Convolution Networks on STL-10 Dataset



**Fathima Kamaludeen**

**Justin Landay**

**Vijayshri Rangasamy**

04.25.2018

ML2 - Data Science

# INTRODUCTION

Our final project is about using Deep Learning algorithms on a computer vision problem. The main objective of our project is to compare the architecture and performance between a Convolution Neural Net (CNN) and a Multi-Layer Percepetron (MLP) neural net as well as compare CNNs with each other that differ by various hyper-parameters such as kernel size, stride, and number of kernels.

The designed neural networks and its code can be seen in the GitHub folder below. The main tool used in this project is Pytorch.

<https://github.com/jlanday/Final-Project-Group-7>

# DATA DESCRIPTION

We chose to use the STL-10 dataset for our project. One of our reasons was due to the similarities of CIFAR-10, another common benchmark set used for supervised learning and computer vision.

The STL-10 dataset contains 96x96 pixels color image data divided into 10 classes: airplane, bird, car, cat, deer, dog, horse, monkey, ship, truck. The dataset was developed for unsupervised and weakly supervised learning algorithms, and is divided as follows:

• Unsupervised Train: 100.000 unlabelled instances containing data from all 10 classes and some variations.

• Supervised train: contains 500 labelled images per class

• Test: 800 images per class.

More information on the STL-10 dataset can be found on their website at <https://cs.stanford.edu/~acoates/stl10/> .

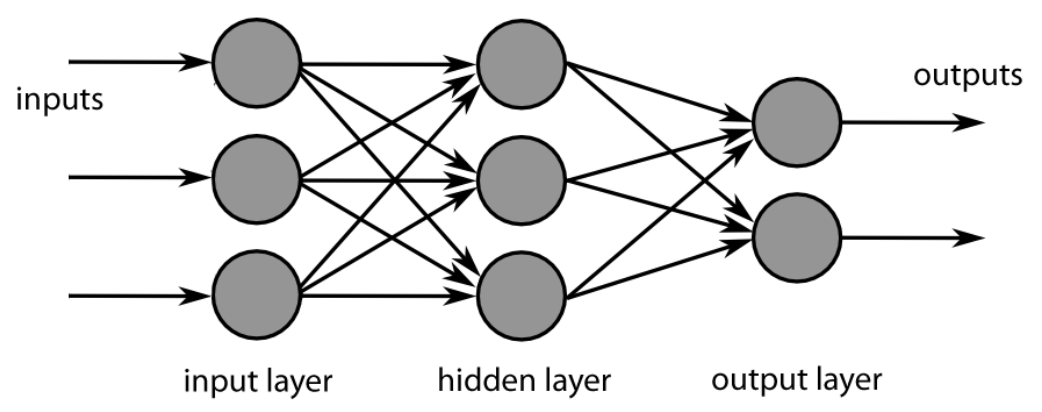
# MODELS AND TRAINING PROCEDURES

**Multi-Layer Perceptron**

Multi-Layer perceptron is a supervised learning algorithm that trains on a set of input-output pairs and learn to model the correlation between those inputs and outputs. Training involves adjusting the parameters, or the weights and biases in order to minimize the error. Backpropagation is used to make the weight and bias adjustments relative to the error and the error can be measured by root MSE.

In the forward pass, the signal moves from the input layer through the hidden layers to the output layer and the decision of the output layer is measured against the ground truth labels.

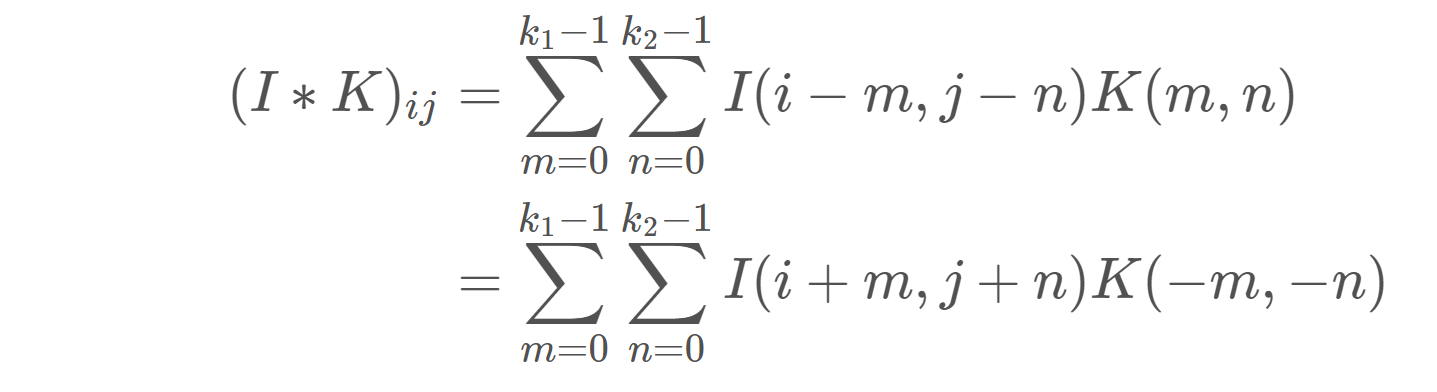
In the backward pass, using propagation and the chain rule of calculus, the partial derivative of the error function wrt to the weight and bias are back propagated through the multi layer perceptron. This act of differentiation gives the gradient and can be adjusted with the parameters as they move the MLP to the error minimum which can be done with any gradient based optimization algorithm such as stochastic gradient descent algorithm. When the network reaches a state, where the error can go no lower, it is called convergence.



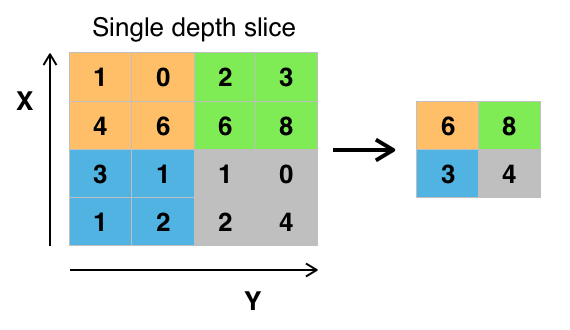
**Convolutional Neural Network**

A convolutional Neural Network (CNN) shares certain similarities with a MLP net however where they differ gives CNN’s a much more robust fitting power. To start with CNNs are a feed forward neural network however unlike with the MLP they preserve the matrix structure of the image. A typical CNN is made up of convolution blocks which feed into each other and then few fully connected layers and a softmax activation function. The difference between CNNs and MLPs are best exemplified by the convolution block. The block will typically consist of a convolution layer(kernel/filter transformation), a pooling layer, and then an activation function.

The first part of the block in the convolution layer is kernel transformation. The kernel transformation can be thought of applying a series of filters to the image. In controlling the kernel we can set the filter size (typically 2x2, 4x4, and 5x5), the stride (the number indices that the kernel moves through the image by) and padding (adding extra 0s to the edges of the image to prevent it from shrinking due to stride setting). Mathematically the convolution/filter transformation can be expressed as follows for an image I and a kernel K of dimensions k1 x k2:



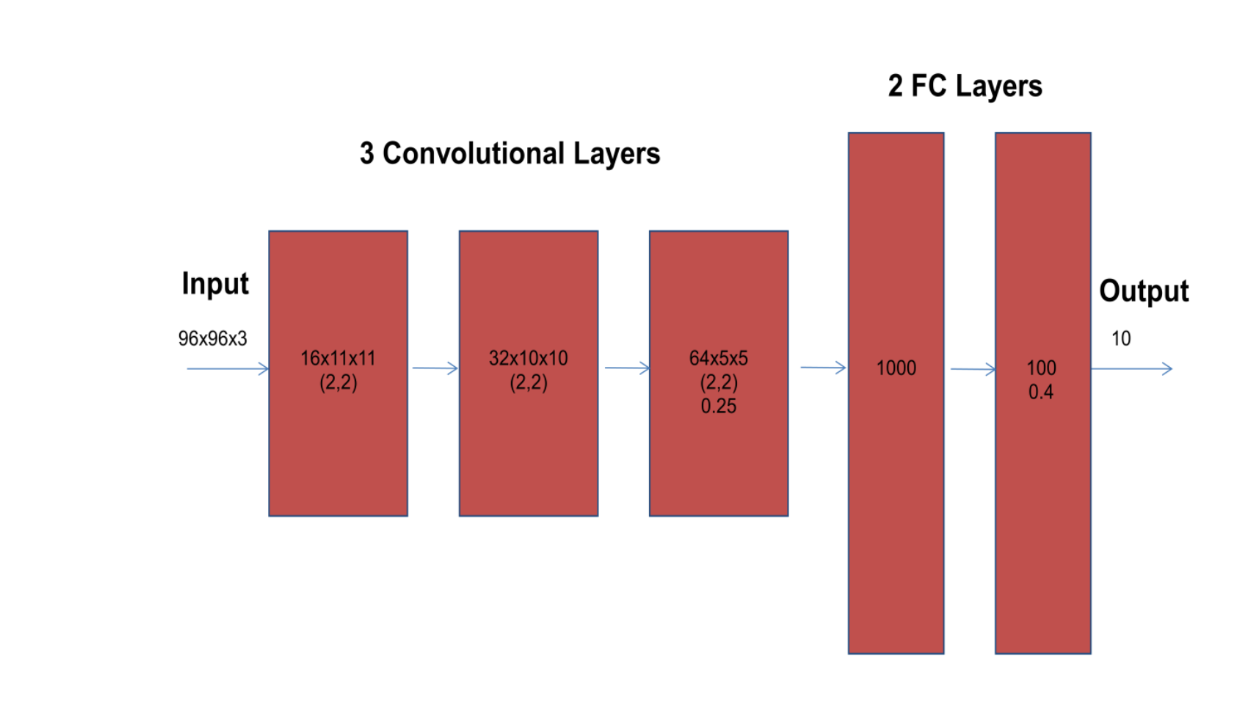
The convolution layer is extremely useful for identifying individual features that can be used in combination with each other in the fully connected layer to make accurate predictions. This is then followed by a pooling layer (typically max pooling). The pooling layer is a form of downsampling where in the case of max pooling it selects the maximum pixel value in a certain filter region. Max pooling is incredibly useful for preventing overfitting as well as reducing the number of parameters by downsampling the images. Below is an example of max pooling and the effects it would have on a sample 4x4 image. The final step of the block is the ReLU layer which just applies the ReLU activation function to the output of the pooling layer. In mathematical terms ReLU can be expressed as f(x) = max(0,x). This activation function is used because it decreases the nonlinear properties, is very easy to differentiate (used in the training process) and is quicker to implement than other nonlinear activation functions.



An example of max pooling on a sample 4x4 image. This sample has a filter size of 2x2 and a stride of 2 as well.

Typically there are a few convolutional blocks followed by an unraveling of the last output so that it can be fed into a few fully connected layers. The fully connected layers have similar design to the MLP and fed into a soft-max activation layer in the last step to perform the classification.

Similar to the MLP, a CNN is trained using the backpropagation algorithm. This involves first doing a forward pass where you calculate the output using the initialized weights and biases. This is followed by minimizing the chosen loss function and then updating the weights. This is typically accomplished by some variant of a gradient descent method.



A sample CNN architecture

# EXPERIMENTAL SETUP

Multilayer Perceptron Neural Network Architecture:

Convolutional Neural Network Architecture 1:

The first CNN architecture implemented contained two convolution layers followed by three fully connected layers. The first layer contains 8 kernels, has a filter size of 2x2, stride = 1, and padding equal to 1. It is followed by a max pooling and a ReLU activation. The second layer has 16 kernels, a filter size of 2x2, stride=1 and padding =1 and is also followed by max pooling and ReLU. The output from the second convolution layer is then reshaped and fed into a series of three fully connected layers containing 96\*96\*16 (147K), 1000, 10 neurons. The first two fully connected layers have a ReLU activation. Additionally batch dropout is implemented between the first and second fully connected layer to prevent overfitting. In training this model, I minimized the cross entropy using the ADAM optimizer with a fixed learning rate = 0.001. I also used a minibatch size = 4 and trained over 10 epochs.

These hyper-parameters as well as the number of layers and neurons were chosen largely by trial and error. Initially, there were three convolution layers having 8, 16, and 32 kernels respectively with a 5x5 kernel size each. This model performed incredibly well on the training set (~98% accuracy) however failed on the test set leading me to conclude we were overfitting. Some of the steps took to reduce overfitting was remove the third convolution layer, shrink the filter size and add dropout to the fully connected layer.Additionally, we were originally using 50 epochs and reducing the number of training epochs lead to a significant gain in the test accuracy giving further indication of overfitting. Other attempts to address overfitting included changing the learning rate, the optimizer, and introduce batch normalization however we did not see noticeable gains in test accuracy from these. The final model archeticture as defined in PyTorch can be seen below.



Convolutional Neural Network Architecture 2:

# RESULTS

Feed-forward Neural Network Result

Convolutional Neural Network 1:

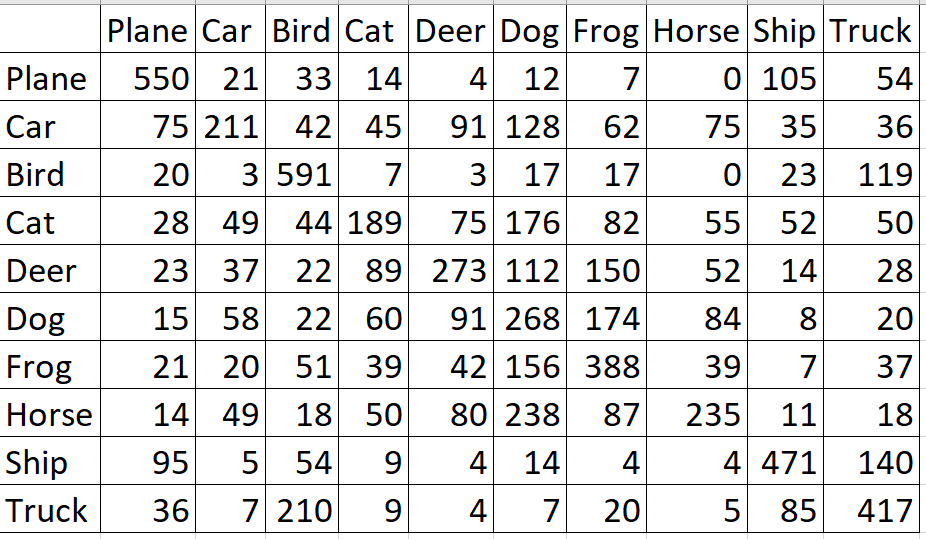
Below we display our final results for this architecture. In total it took 1824 seconds (30 minutes) to train this model on the GPU. After 10 epochs of training the training accuracy was 99% and the test accuracy was 44%. This is a strong indication that even after shrinking our model, introducing dropout, and decreasing the number of epochs our model is still suffering from overfitting.

Below we present the per class accuracy for both the testing and the training.

Training: Training:

Accuracy of plane : 98 %  
Accuracy of car : 98 %  
Accuracy of bird : 100 %  
Accuracy of cat : 98 %  
Accuracy of deer : 99 %  
Accuracy of dog : 98 %  
Accuracy of frog : 100 %  
Accuracy of horse : 100 %  
Accuracy of ship : 100 %  
Accuracy of truck : 100 %

Accuracy of plane : 70 %  
Accuracy of car : 26 %  
Accuracy of bird : 79 %  
Accuracy of cat : 25 %  
Accuracy of deer : 32 %  
Accuracy of dog : 33 %  
Accuracy of frog : 43 %  
Accuracy of horse : 27 %  
Accuracy of ship : 63 %  
Accuracy of truck : 49 %

Additionally, we present the confusion matrices for both the test data 

Confusion Matrix for Test Data

Convolutional Neural Network 2:

# CONCLUSIONS

Most of our conclusions aligned with our initial hypothesis that the CNN models would perform better during both the testing and training phases. We can see this by directly comparing either their accuracies or their confusion matrices.

In the future there are a few modifications that we would implement to better improve our models. We are still experiencing tremendous over fitting for the CNNs, even after drastically reducing the model size and the number of free parameters. We believe that the overfitting is primarily a result of how small our training set compared to our test set. In order to get really good results that will also generalize we would need many more images. However, beyond augmenting the data there are a few other steps that we could deploy to also reduce overfitting. One of those is including batch normalization and drop out at the same time. We tested both of them individually and decided upon implementing drop out only because of the performance gained however the combination of the two perhaps could have been advantageous. Additionally, we did not do a detailed experiment regarding our choice for the optimizer. In future work, we could implement a few different optimizers with various learning weights ( cyclical learning rates in particular) and use K-Fold Cross Validation to find the optimal values.

# REFERENCES