CMP_SC 8725 : Supervised Learning

Project Proposal

Project Title: Analysis of Deep Residual Networks (ResNets) on MNIST and CIFAR10 datasets and a comparative study with the classical Convolutional Neural Networks (CNNs)

Farhan Quadir, Raj Shekhor Roy, Mariam Alshehri, and Sonam Phuntsog

Introduction:

A single layer perceptron (SLP) takes as input a vector ***X*** and outputs a result ***Y***. The SLP computes the result by multiplying vector ***X*** with a set of weights- the vector ***W***- to obtain a result vector ***V***. An activation function φ is applied to ***V*** to get our final output which is ***Y***. This is the forward pass.The learning of SLP occurs when we perform the backpropagation. In order to do that we need to compute the error or loss based on a loss function. Some common loss functions include squared-error(SE), mean squared-error(MSE), weighted mean squared-error(WMSE), mean absolute-error(MAE), binary cross entropy(BCE), categorical cross entropy (CCE), etc. The purpose of backpropagation is to update each of the weights such that during the next cycle, the value of the output *y* becomes closer to its actual value. In other words, we need the error to be minimum. So in order to know by how much to update the weights, we need to perform gradient descent. In a multilayer perceptron (MLP), the error backpropagation works similarly but with the accumulation of errors from every output node to the hidden nodes.

CNNs are very similar to neural networks with certain criteria adaptations. CNNs mainly introduce the concept of shared weights for each filter as we attempt to learn these weights and biases, through sliding a kernel of a fixed size through the entire input, in fixed strides. The convolution layer consists of filters which attempt to learn a particular feature. If we use 64 different filters, we are learning 64 different features. Each filter will produce a different feature map. The filters perform a matrix multiplication (dot product) between the weights in the filter and the input matrix as the filters slide in fixed strides along the input. Since the weights are shared and the filter slides along the input, the filters capture a weighted sum of the features. This allows the features to become more concentrated from layer to layer as the dimensions of the output starts to decrease as we move to the next layer. In order to keep the dimensions of the input and output the same at every layer, we can perform zero padding at every layer. During backpropagation, the weights in the filters are changed similar to a normal MLP but since weights are shared, a sum aggregate of the error for every shared weight is taken. This allows a larger error to be passed since error backpropagates through every neuron the weight was connected to. This also leads to a bigger change in the weight update provided learning rate does not change.

The problem starts to occur when we try creating deeper networks or stacking networks or layers over one another. Deep Neural Networks are very difficult to train. Stacking more layers tend to introduce vanishing/exploding gradients. This is mainly due to the squashing or activation functions like Tanh, sigmoid, etc. which can lead to slower or no convergence. Although this can be solved by introducing Normalization between layers or chosing activation functions that do not squash like ReLU, deeper networks introduce the problem of accuracy degradation. When networks become deep, it tends to lead to higher training error due to saturation of accuracy. A higher training error means that weights are updated with errors thus leading to higher testing error. So this causes degradation in the accuracy of the predictions. The degradation problem is addressed by introducing a deep residual architecture (Figure 1) where adding the input information to the output of a hidden layer neuron (shortcut connections) via identical mapping prevents loss of input information through layers. This does not introduce extra parameters and also does not require more computation. So network does not get saturated thereby preventing degradation. Also, the gradient gets backpropagated through two pathways- through the weights layers and through the shortcut connections. So even if the gradient diminishes through the weights layers, the shortcut adds the original gradient to it making the gradient larger again.

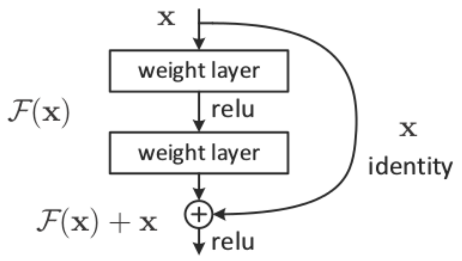
x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 1. Adapted from original ResNet paper. Shows how the input information is propagated in the forward pass.

Plan:

We plan to run separate experiments using convolutional neural networks (CNN) and deep residual networks (ResNet) on MNIST and CIFAR10 datasets to compare and contrast the power of ResNets over CNNs. We hypothesise that Deep ResNets with same number of layers as that in CNNs, will outperform CNNs by far with respect to accuracy, and convergence speed on both datasets. We also believe that ResNets might perform less accurately with CIFAR10 dataset as compared to MNIST dataset due to lack of normalization factors. We also plan to test ResNets using normalization between layers as well. If time and resources permit, we plan to test out the model on CIFAR100 dataset too.

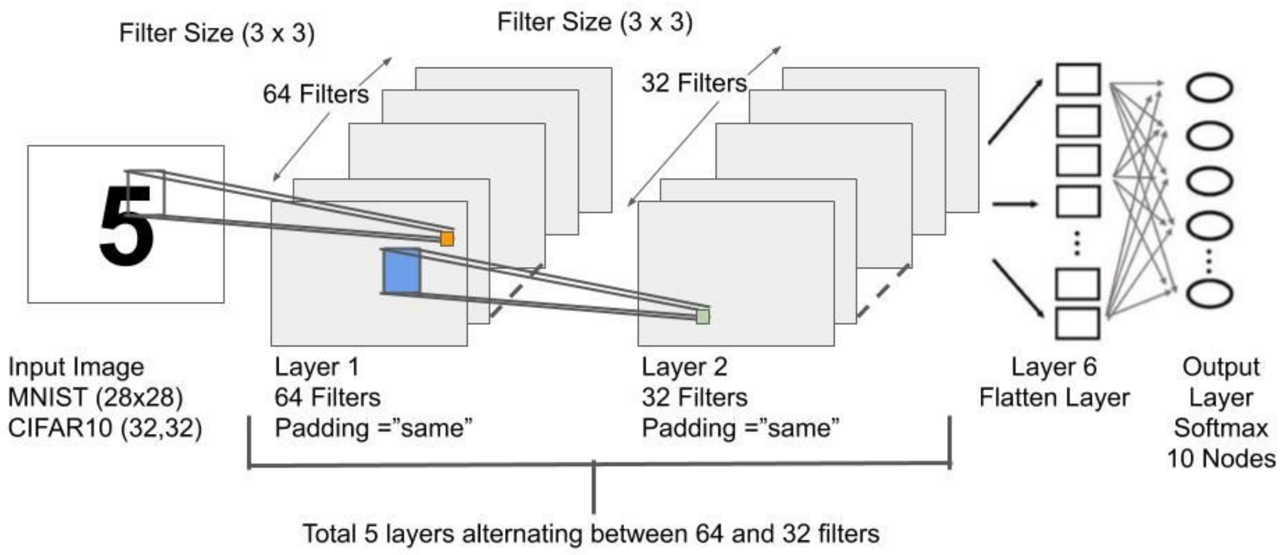

Figure 2. Network Architecture of Deep Convoluted Neural Network (CNN). The input image is a 28 x 28 (mnist) or 32 x 32 (cifar10) matrix. There are five convolution layers that alternate between 64 and 32 filters produced by kernel size 3 x 3. After the final convolution, the network is flattened and a dense layer of ten output nodes are added for the ten classes. The final layer performs a softmax to obtain the final output class.

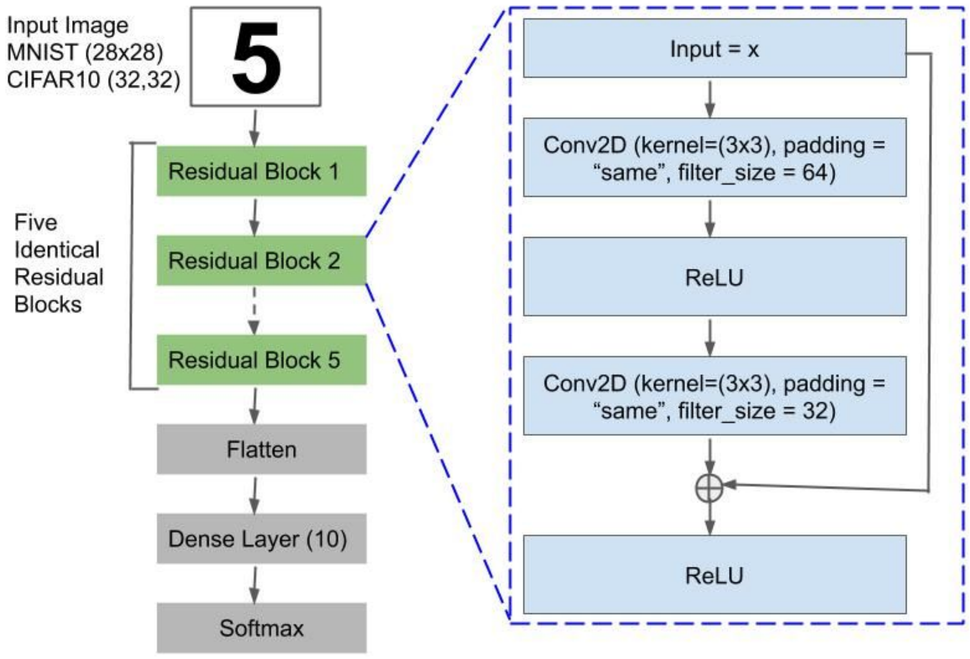
Input Image
MNIST (28x28)
CIFAR10 (32,32)
5
Five Identical Residual Blocks
Residual Block 1
Residual Block 2
Residual Block 5
Flatten
Dense Layer (10)
Softmax
Input = x
Conv2D (kernel=(3x3), padding = "same", filter_size = 64)
ReLU
Conv2D (kernel=(3x3), padding = "same", filter_size = 32)
ReLU

Figure 3. Deep Residual Architecture containing five identical Residual Blocks. The input image is a 28 x 28 (mnist) or 32 x 32 (cifar10) matrix. Within each residual block there are two convolution layers that alternate between 64 and 32 filters produced by kernel size 3 x 3. There is ReLU activation in between the convolution layers. After the final convolution, the input (x) is added to the output followed by another ReLU activation. After the final residual block, the network is flattened and a dense layer of ten output nodes are added for the ten classes. The final layer performs a softmax to obtain the final output class.

The project will be implemented in Python 3.7 and libraries like numpy, matplotlib, TensorFlow, Keras, etc. will be used. A standard Nvidia 1060 6GB GPU graphics card installation with a Ubuntu 18.04 version Linux system will be used to train our model.