Carl G. Britt III
ECE 692 – Project 2

TASK 1:

Configuration:

The base structure which will be used to compare against other variations will be described and will be referred to as vanilla. The first layer is a convolutional 32x3x3 layer with a stride of 1. The output is padded to keep the dimensions the same. This layer is followed by a max 2x2 pooling layer. The third and fourth layers are convolutional 64x3x3 layers with a stride of 1, followed by a max 2x2 pooling layer, which total to 5 layers. The last layer is a fully connected layer with 512 nodes and dropout of 0.5. All the neurons use the ReLU activation function. An Adam optimizer is used to schedule the weight changes during back progagation. This was chosen over Stochastic Gradient Descent (SGD) as it was easier to obtain better accuracies with reasonable numbers of epochs. Finally, the objective function is cross entropy. Each configuration was trained for a 100 epochs and a batch size of 100.

Data manipulations include preprocessing the data by training wide normalization, where each pixel is converted to z-scores from the means and standard deviations of all the images in the training set. Augmentations to the training set include random horizontal flipping (50%), random crops of 26x26, and random rotations to at most 30 degrees in either direction.

Combining this structure and this data regime yields the results shown in Table 1. Changing the activation function to sigmoids decreased accuracy, which is expected due to the neuron saturation. In other words, since sigmoids saturate, it can take longer before the weights associated with that neuron change significantly. This is somewhat remedied by using the cross-entropy cost function (Nielson Chapter 3) such that the derivative of the activation function is directly dependent on the error from ground truth; however, the ReLU activation function was also observed to learn faster than sigmoids as they do not saturate. Without data normalization or augmentation, the structure became prone to overfitting as the training accuracy would be near 98%, while the testing accuracy would be dramatically less. Using a different initializer, truncated_normal instead of uniform_scaling, did not significantly change the results. Using a different objective function, mean_square instead of cross entropy, seemed to decrease accuracy, which is expected due to the derivatives being a direct function to the error from ground truth. Removing dropout did not significantly change the results, as the vanilla model was not prone to overfitting once normalization and data augmentation was added. Note that the training accuracy was lower than the testing accuracy, which is an indicator that the network is doing a decent job of generalization.

Table 1: Results after 100 epochs of training

config	val acc	train acc
vanilla	0.8268	0.8038
sigmoid	0.8062	0.7683
nodatanorm	0.7555	0.9837
nodataugment	0.7553	0.9809
nodatanormoraugment	0.1	0.0981
truncated_normal	0.8268	0.8026
mean_square	0.8016	0.7602
nodropout	0.8179	0.8202

Figure 1 shows the test accuracy changes as a function of epoch number. Each epoch took around 27 seconds, leading to a total training time of around 54 minutes. Figure 2 shows the training accuracy as a function of epoch number. It’s interesting to note the step change in training accuracy around epoch number 30. To this day, this is the best performing LeNET-5 algorithm in the class, but 13% below the best performing algorithm on the CIFAR-10 set.

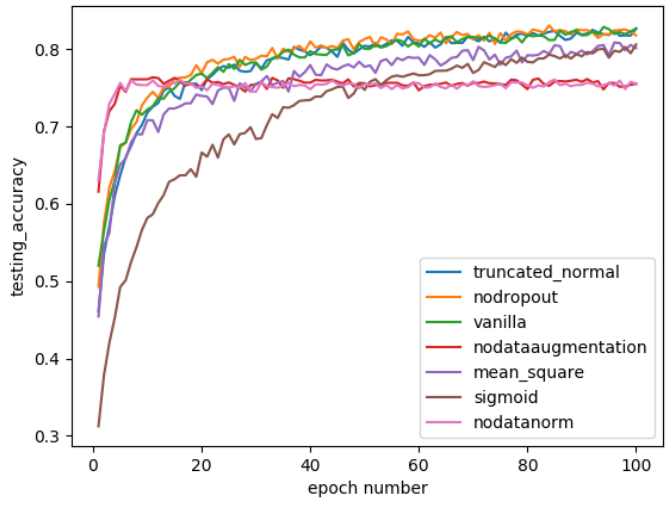

Figure 1: Testing accuracy for the LeNet5 inspired network as a function of epoch number. .

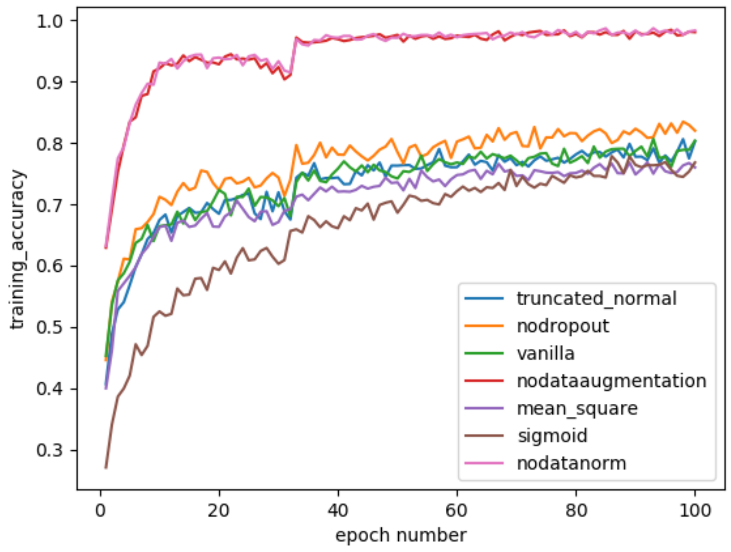

Figure 2: Training accuracy for the LeNet5 inspired network as a function of epoch number.

TASK 2

VGG16 model was imported and ran with the minimum performance attained to convince me that it was actually working. CIFAR10 was imported in two ways, the first way was to take the output from the third layer, and construct the fully connected layer for classification (Figure 3). The alternate way was to pad the 3 dimensional image array with zeros in each of the channels, then run the images through the entire network (Figure 4). Training took roughly a night, such that adjusting the hyper parameters was not explored further. The data augmentation and preprocessing procedures were the same as Task 1. The optimizer used was SGD with a learning_rate of 0.001, a learning rate decay of 1E-5, momentum of 0.9, and Nesterov acceleration was turned on. Cross entropy was used for the objective function.

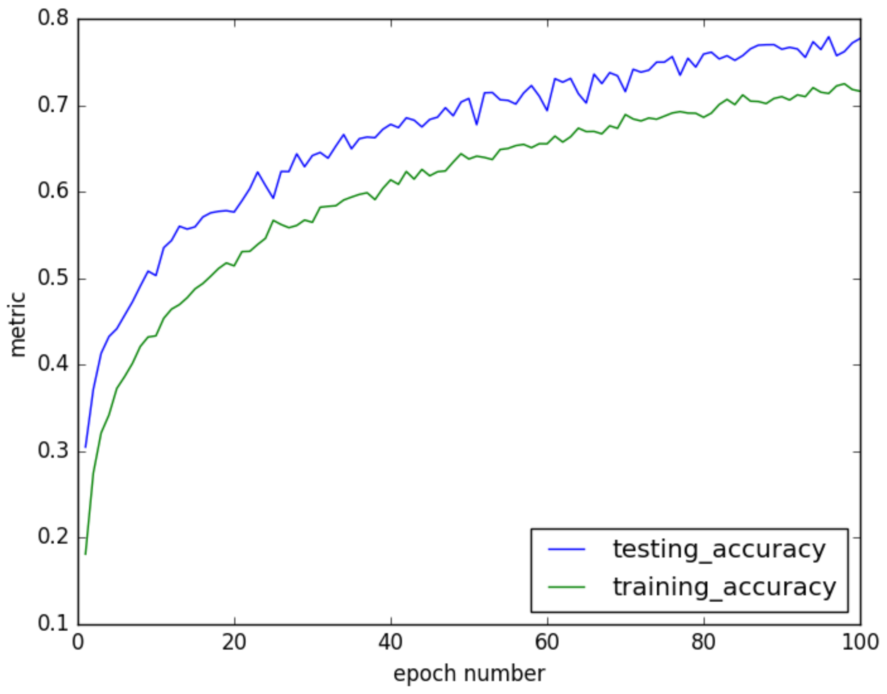

Figure 3: VGG16 by extracting the third layer and applying readout.

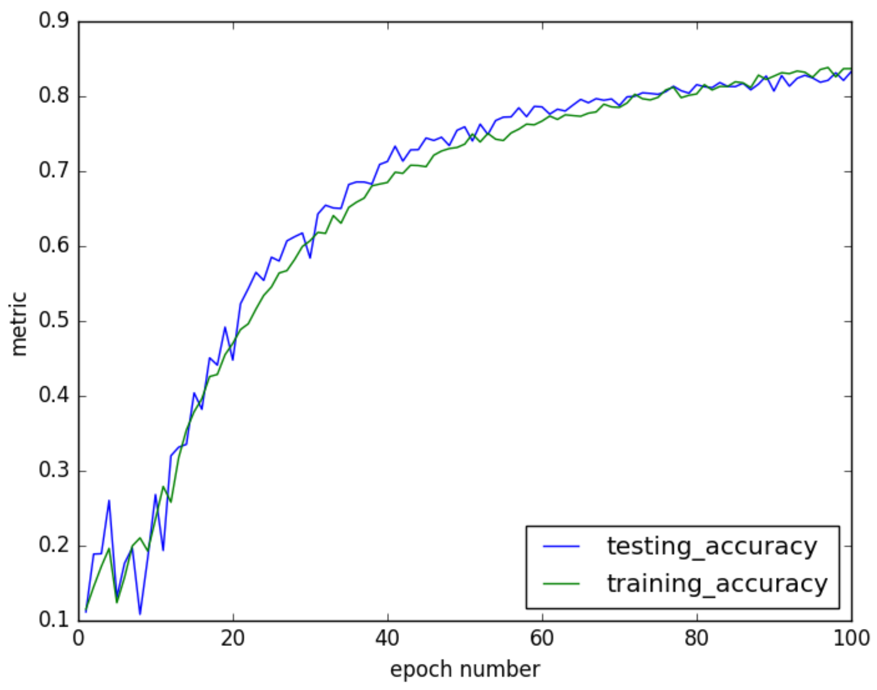

Figure 4: VGG16 Accuracy by padding with zeros and feeding through entire structure.

The best performing VGG16 network I could come up with is around 13% below the top performing algorithm on the CIFAR-10 set.

LeCUN:

LeCUN et al. pushes for the automatic learning of features instead of manual design. He uses the scenario off identifying check numbers as the case study. He also proposes gradient based learning is a very practical tool for finding the local optima of the network, which doesn't affect the practicality of the systems. CNNs work by restricting receptive fields of hidden units to be local, thereby extracting local structures and applying it to the entire input. The three main reasons CNNs work is due to local receptive fields, shared weights (weight replication), and spatial or temporal subsampling (max / average pooling). These three properties help design features that are robust to subject scaling and translation within an image. To emphasize, what's important is not the position of a feature within the image, but the relative positions of features in relation to one another. LeCUN also mentions that break in network symmetry is essential for differentiating classes, that and objective function which promotes competition between classes is important, and that the classification result from multiple networks should be better than the result from one (Boosted Lenet-4 vs. Lenet5). Global training is much more preferred over than individual module training due to self-regularization, and these networks should be trainable using a gradient descent algorithm as long as the activation neurons are differentiable. It isn’t essential that the network is entirely feedforward, as it can be more general or abstract such as a Graph Transformer Network (GTN).

AlexNET:

Introduces dropout to replicate the gains obtained from using multiple networks to classify. With dropout, different structures are sampled within the structure itself, which helps with overfitting such that overdependence on certain neurons or clusters of neurons are not necessary. Dropout replicates the result at which LeCUN was referring to, where the classification result is of different networks (such as in boosted LeNET4) instead of a single network. CNNs have much fewer connections and parameters than MLPs, therefore they are much easier to train. They use ReLU instead of sigmoids (or hyperbolic tangent) for their activation function, which increases the training speed as the neurons are less prone to saturation (deals with vanishing or exploding gradient problems). They use local response normalization to help with generalization, therefore they use overlapping convolution filters. They use data augmentation: random cropping to increase the training set by a factor of 2048, alter the color of images using PCA, and dropout.

Googlenet:

The main claim to fame is the inception modules, they use both depth and breadth in their network. They use 1x1 convolutional layers both to increase representational power (linear transformation followed by non-linear ReLU of the inputs) as well as decrease computational time. Introduce sparsity and replace fully connected layers by sparse ones, even inside convolutions. They cite Aurora et al. If the probability distribution of the dataset is representable by a large, very sparse deep neural network, then the optimal network topology can be constructed layer after layer by analyzing the correlation statistics of the preceding layer activations and clustering neurons with highly correlated outputs. Resonates well with Hebbian principle – neurons that fire together, wire together. Googlenet attempts to consider how an optimal local sparse structure of a convolutional vision network can be approximated and covered by readily available dense components. The goal is to find the optimal local construction and to repeat it spatially. As for new data augmentation scripts, they choose to use a more aggressive cropping approach with 4 scales, 144 crops per image. Softmax probabilities averaged over multiple crops to obtain the final prediction.

VGGNet:

VGGNet architecture focuses purely on the depth of the network to construct both detailed and abstract features on the input image. They focus solely on 3x3 convolution filters and 2x2 max pooling. They did not use normalization; however, they did preprocess the data by subtracting the mean RGB value obtained from the training set. They also use 1x1 convolutions often to increase nonlinearity of decision function without affecting the receptive fields of the convolutional layers. This has been inspired by the network in network approach. The final contribution is they initialized the earlier layers by training it by itself, as random initialization would not yield favorable results.

ResNET:

The biggest idea is that it is easier to optimize the residual mapping than to optimize the physical mapping. Easier to push the residual to zero than to fit an identity mapping by a stack of nonlinear layers. They also use bottleneck architectures to handle dimension reduction, convolutions, an dimension restoration. The identity shortcuts also increase training time.