Advanced Machine Learning (GR5242) ▪ Fall 2017

Building Neural Networks for CIFAR-10 Images Classification: Comparing models developed with advanced techniques

Wanhua He(wh2362); Kexin Nie (kn2403)
Yuqi Shi (ys2969); Youyang Cao (yc3232)

Columbia University, Class of 2017, United State

December 18, 2017

1. Introduction

CIFAR-10 is a data set containing 60000 32x32x3 RGB colorful images, widely for image classification project. Each image of CIFAR-10 contains a object, and there are ten object categories in total. The simply structure of this data set has enabled a large amount of deep learning project implement using various techniques. For our project, we explored many high-performance models beforehand and try to apply those advanced techniques, such as fractional max pooling, batch normalization, on our models. We also compared all our models, where we had chance to evaluate various techniques. This project helps us have a better understand of the structure of neural network.

- Brainstorming

Before we start this project, after our group members first discussion, we may start with a simple logistic regression. Next basic steps may include trying to add convolutional, max pooling, relu, dropout, fully connected layers, and some other architectures before the logistic regression step. Some current methods used include: fractional max-pooling; generalizing pooling functions in convolutional neural networks such as mixed, gated, and tree; spatially-sparse convolutional neural networks; using scalable Bayesian optimization; deep residual learning; exponential linear units; (aggressive) data augmentation while training, etc. We set our goals as decreasing the test error of our model and avoid overfitting problem.

2. Previous Works

To have a better understanding of our models, readers need to know the following basic concepts.

2.1 Fractional Max Pooling

The intuition is that the exact location of a feature is less important than its rough location relative to other features. It quickly reduces the size of the hidden layers, and it provide a degree of translation and elastic distortion invariances. This helps to control overfitting. While max-pooling act on the hidden layers of the network, reducing their size by an integer multiplicative factor alpha, for example 2x2, which is the default choice for building convolutional networks, discards 75% of your data. However, the spatial size is reduced so rapidly so stacks of back-to-back layers are need to build really deep networks, and pooling regions are non-overlapping, so generalization is limited.

To further reduce overfitting, Benjamin (2015) formulated a fractional version of max-pooling where alpha is allowed to take non-integer values. Fractional max-pooling (FMP) is stochastic as there are lots of different ways of constructing suitable pooling regions. It reduces the spatial size of the image by a factor of alpha with 1<alpha<2. The randomness introduced by FMP is related to the choice of pooling regions. The overlapping pooling regions are defined by:

$$P_{i,j} = [a_{i-1}, a_i] \times [b_{j-1}, b_j] \tag{2.1}$$

The pseudo-random sequence generate much more stable pooling regions and it takes the form:

$$A_i = ceiling(\alpha(i+u)), 1 < \alpha < 2 \tag{2.2}$$

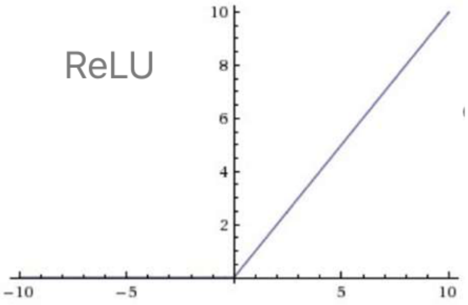
In our case, we use $\sqrt[2]{2}$ as alpha.

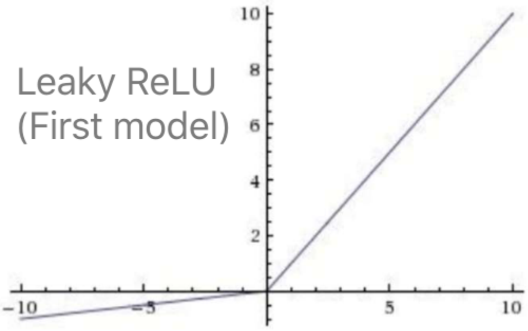
2.2 Leaky ReLu

The Rectified Linear Unit (ReLU) computes the function f(x)=max(0,x). In other words, the activation is simply thresholded at zero (see image on the left).

Left: Rectified Linear Unit (ReLU) activation function, which is zero when x < 0 and then linear with slope 1 when x > 0.

ReLUs have several advantages and disadvantages. ReLUs can significantly accelerate the convergence rate of stochastic gradient descent than the sigmoid function. And ReLUs can be achieved by simply setting the activation matrix to be zero. On the other hand, ReLUs units may become vulnerable during training and may "die." For example, large gradients flowing through ReLU neurons may result in a weight update, so that neurons will not be activated again at any data point. Which means RELU units may die irreversibly during training as they may be eliminated from the data manifold.





We can observe that the learned activations is similar to leaky rectified linear units (Leaky ReLU). Leaky ReLUs are one attempt to fix the “dying ReLU” problem. Instead of the function being zero when x < 0, a leaky ReLU will instead have a small negative slope. The function of Leaky ReLU is:

$$\begin{cases} h(x) & = x, x > 0 \\ h(x) & = kx, otherwise \end{cases} \tag{2.3}$$

Some people report success with this form of activation function, but the results are not always consistent. In our training model, we replace the ReLU by Leaky ReLU in our layers.

2.3 Batch normalization

Batch normalization aims to address the ‘internal covariate shift’ by normalizing layer inputs (Ioffe and Szegedy, 2015). Internal covariate shift refers to the problem that, because the distribution of each layer’s inputs changes during training as the parameters of the previous layers change, training slows down requiring lower learning rates and careful parameter initialization is required. Batch normalization makes normalization as a part of the model architecture and performs the normalization for each training mini-batch. Therefore, it allows us to use much higher learning rates and be less careful about initialization. It also acts as a regularize, in some cases eliminating the need for Dropout.

3. Goals

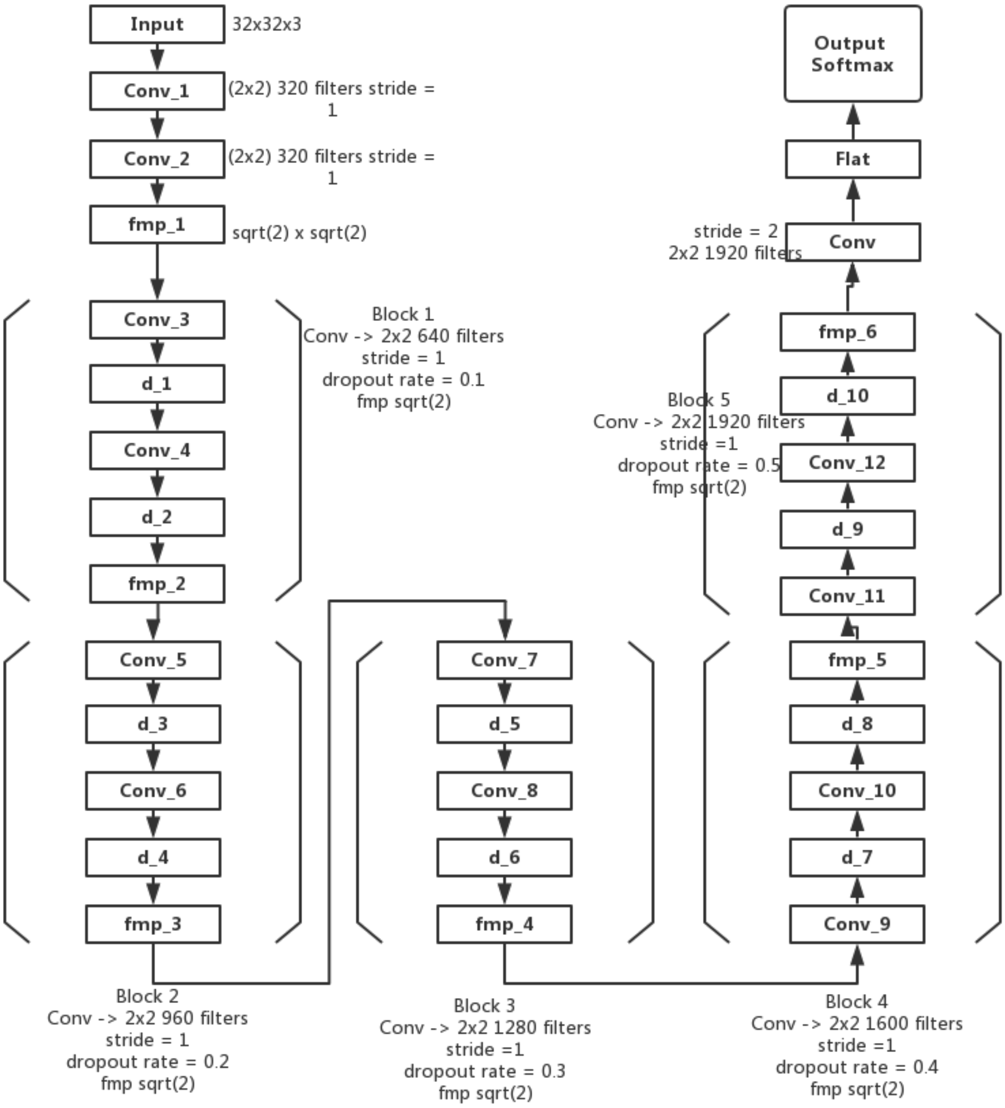
For this project, our goal is to improve the base model with some other techniques. We found our base models by reading many academic research papers and thinking through the process of building the network. For CNN FMP and Reduced AllCNN, the base model is from Striving For Simplicity: The ALL Convolutional Net (Springenberg, 2015). In this paper, the author believes that substituting max pooling the convolutional layer with stride increase the generalization for each layer think max pooling simply takes the max values.

4. Methods

4.1 CNN-FMP

In this model, we implement the All CNN model with fractional max pooling. Since in the experiment part, the All CNN model doesn’t outperform the max fractional pooling model. We have the hypothesis that maybe fractional pooling has increased the regularization of layers largely and we can use fmp instead of conv. So we changed almost all the convolutional layers with stride 2(Since they substitute the traditional pooling layers) in All CNN network.

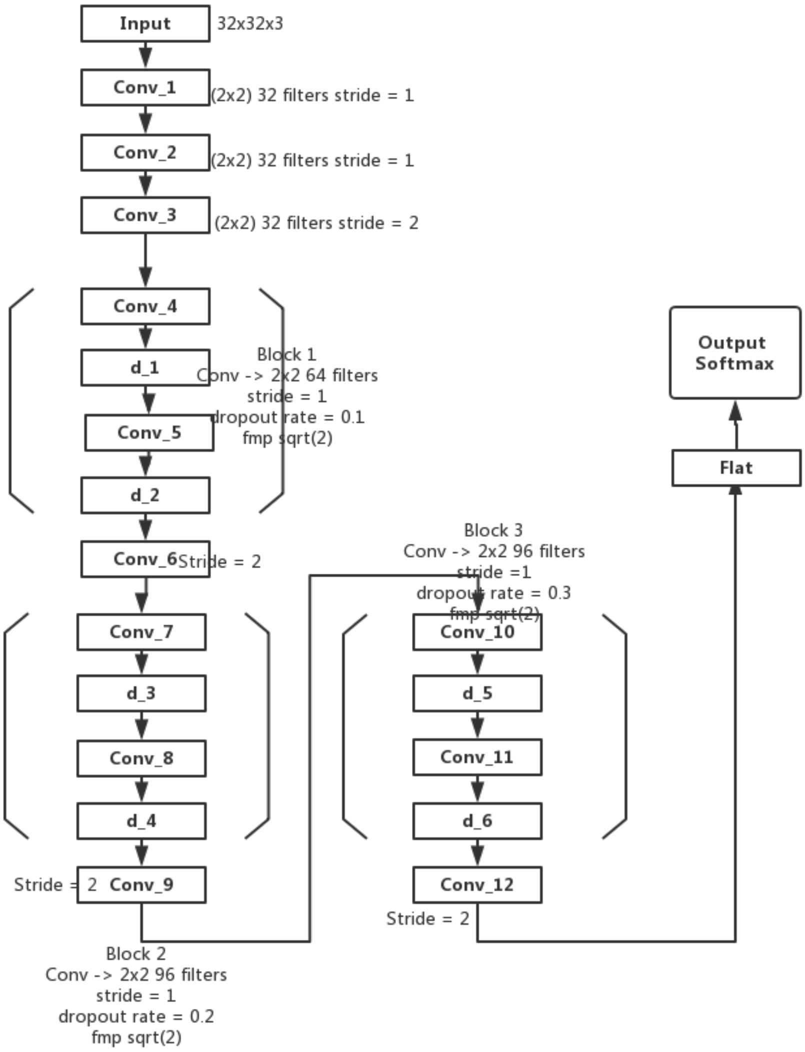
4.1.1 Graphic



4.2 Reduced AllCNN

Another way to improve AllCNN is to get rid almost half of its layers. Some people believe that deeper network performs better. However, according to the author of AllCNN, the smaller network might work bettter here, since CIFAR-10 has small dimension. To prevent overfitting too many training data, we decide to reduce the layers number.

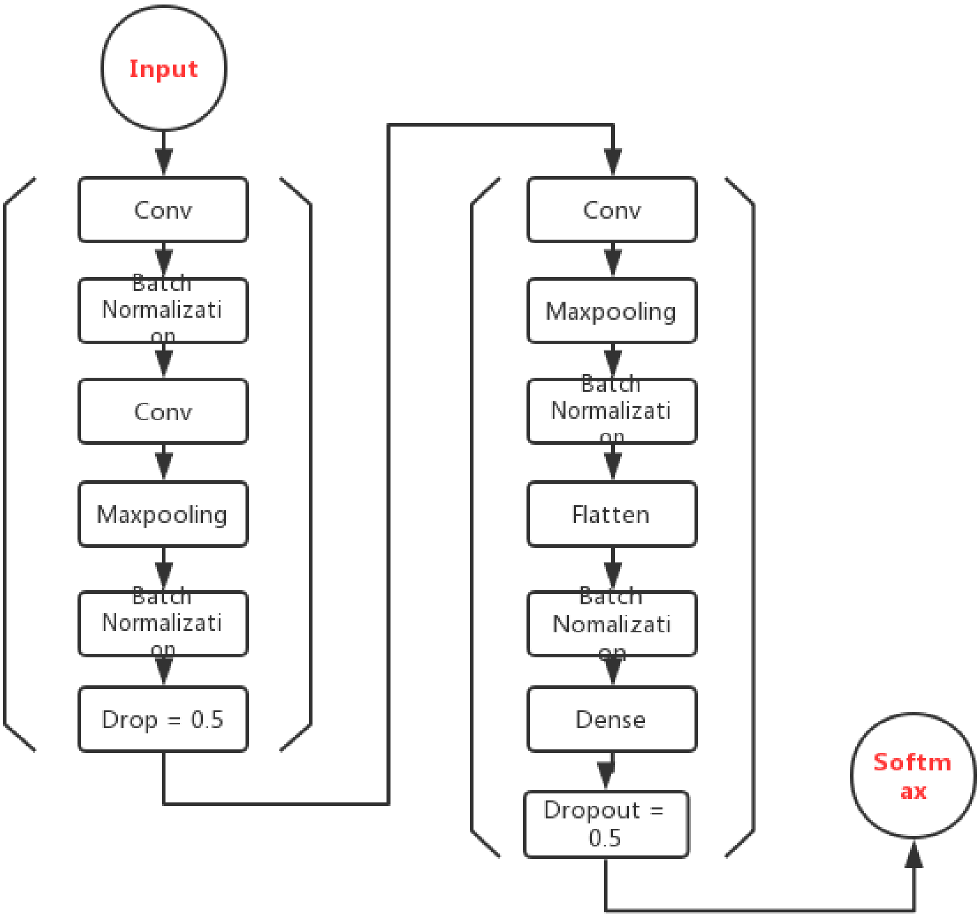
4.2.1 Graphic



4.3 CNN-BN

We hand coded our first model from scratch. Although it is a comparatively deep network with more layers, the performance is not satisfying. Therefore, we implemented our second model using keras.

We implement the first model from several aspects. (1) In our first model, We use the fractional max pooling method several times since it can lead to faster convergence rate by selecting superior invariant features. However, we neglect that it also may result in the features information of different locations in image completely lost in this step. So we use the Keras method and reduce the max pooling layers in our model. (2) We also add batch normalization layers in our model since batch normalization can easily train deeper networks and regularizes the model. This uses a mini-batch data set to center the data with zero mean and unit variance. Each layers takes a mini batch data set and stores the running average of these means and standard deviations. This helps in training a deeper neural network. In our first model, the high learning rate may result in the gradients explode or vanish. Batch Normalization helps us address these issues. (3) We reduce our blocks and also reduce the batch sizes so our computation complexity decrease significantly. The structure of our Keras model is as follows:



Finally, the test accuracy of our model significantly increased to approximate 0.68.

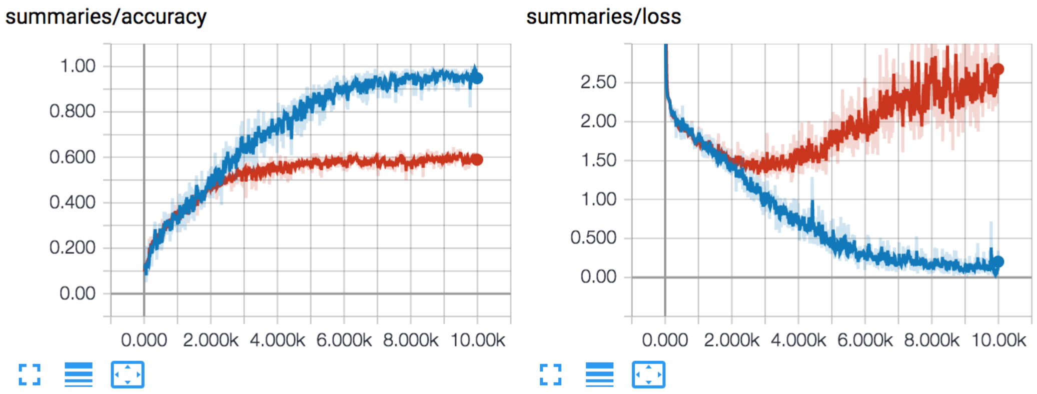
5. Experiment

5.1 Data Preprocessing

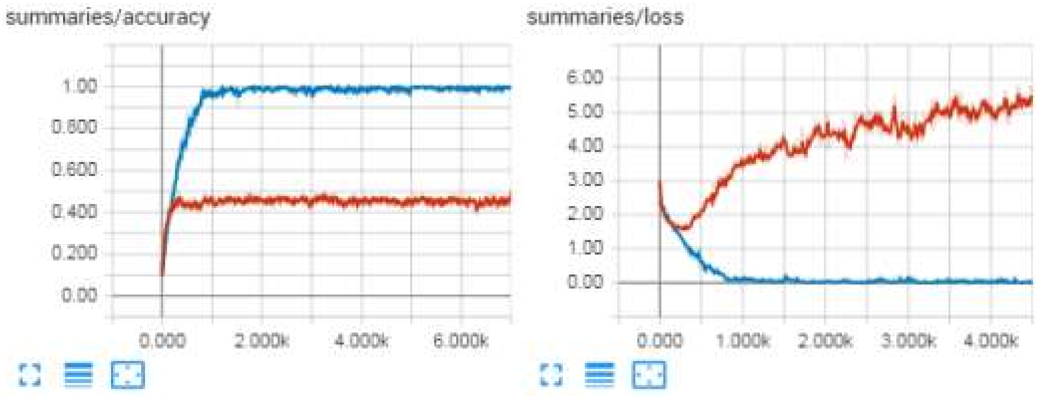
There are 60000 data in CIFAR-10 set. We firstly keep 10000 test data separately from training process, then we train each model with 40000 training and 10000 validation set on each step, returning the validation errors. We tried to increase the dimension of 36x36x3 data to 126x126x3 according to AllCNN author, since it might help us to build a deeper network with high-dimensional data. However, our Tesla K80 GPU has limited computation ability, so we have to give up this method.

5.2 Training and Validation Accuracy

- CNN FMP(Blue is train, Red is validation):



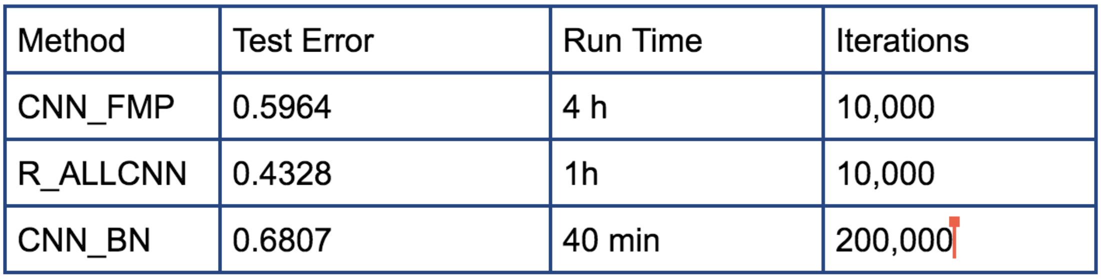
- Reduced ALLCNN:



-CNN BN

Since this method doesn't use Tensorboard, user can see the training progress inside our notebooks.

5.3 Test Results:



Method	Test Error	Run Time	Iterations
CNN_FMP	0.5964	4 h	10,000
R_ALLCNN	0.4328	1h	10,000
CNN_BN	0.6807	40 min	200,000

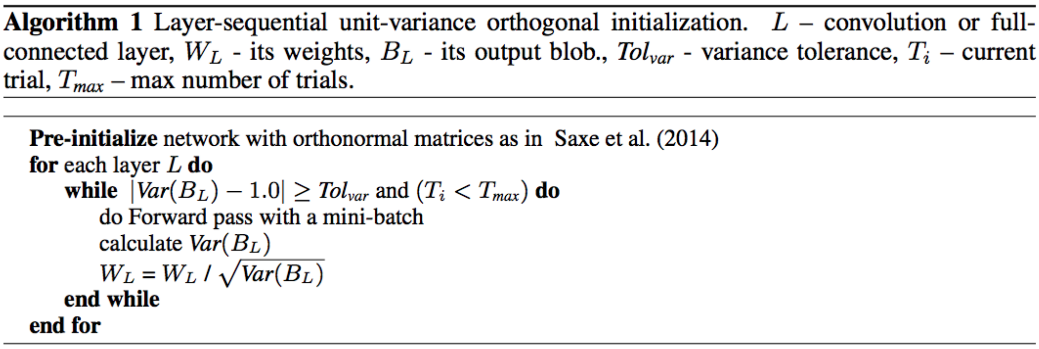
It seems that CNN BN(Convolutional Networks with Batch Normalization) has the best performance, Batch Normalization might have helped with limiting variables values, which might help with overfitting issues. It also has a lower training time due to its small batch size and less number of variables to train.

6. Discussion

According to the test results, we never outperform the ALLCNN base model, which has a test accuracy of 0.98. Since we cannot do image augmentation before training, so we have been training heavy network on a low dimension data, which cause overfitting(You might see from the training process graphs). Also, we apply leakyRelu to CNN FMP and Reduced ALLCNN since it is an activation function with good performance, but they still didn't over perform CNN BN which used Relu. From the results, it seems like low-dimensional data like CIFAR-10 really doesn't need a very deep network, but good normalization and pooling layers. If we had more time, we might try to use techniques such as building a small Recurrent Neural Network, using some other activation functions, changing batch size to smaller.

7. Possible Next Step

Initialization: Implement Layer-sequential unit-variance (LSUV) initialization for weight initialization for deep net learning (Minshkin and Matas, 2015). Currently, initialization with Gaussian noise with mean equal to zero and standard deviation set to 0.01 and adding bias equal to one for some layers is very popular. However, it has a potential problem that the output layer can be extremely large or diminishing. The empirical result shows this approach of initialization, if applied to maxout and ReLU, leads to a better generalization error, and also converges faster; though it fails to converge on sigmoid. The algorithm is as follows: First, pre-initialize weights of each convolution or inner-product layer with orthonormal matrices. Second, proceed from the first to the final layer, normalizing the variance of the output of each layer to be equal to one. Their result shows this approach of initialization, if applied to maxout and ReLU, leads to a better generalization error, and also converges faster; though it fails to converge on sigmoid.



Algorithm 1 Layer-sequential unit-variance orthogonal initialization. L – convolution or full-connected layer, W_L - its weights, B_L - its output blob., Tol_{var} - variance tolerance, T_i – current trial, T_{max} – max number of trials.

```
Pre-initialize network with orthonormal matrices as in Saxe et al. (2014)
for each layer $L$ do
    while $|Var(B_L) - 1.0| \geq Tol_{var}$ and $(T_i < T_{max})$ do
        do Forward pass with a mini-batch
        calculate $Var(B_L)$
        $W_L = W_L / \sqrt{Var(B_L)}$
    end while
end for
```

8. Reference

Graham, B. (2014). Fractional max-pooling. arXiv preprint arXiv:1412.6071. Jost Tobias Springenberg, Alexey Dosovitskiy, Tomas Brox, Martin Riedmiller. (2015, April). Striving For Simplicity: The All Convolutional Net. arXiv:1412.6806v3

Ioffe, S., Szegedy, C. (2015, June). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In International Conference on Machine Learning (pp. 448-456).

Maas, A. L., Hannun, A. Y., Ng, A. Y. (2013, June). Rectifier nonlinearities improve neural network acoustic models. In Proc. ICML (Vol. 30, No. 1). Mishkin, D., & Matas, J. (2015). All you need is a good init. arXiv preprint arXiv:1511.06422. Springenberg, J. T., Dosovitskiy, A., Brox, T., & Riedmiller, M. (2014). Striving for simplicity: The all convolutional net. arXiv preprint arXiv:1412.6806.