Machine Learning Engineer Nanodegree
Capstone Project

Convolutional Neural Networks Optimum Max-Pooling Design

Manouchehr Bagheri
June 21st, 2017

I. Definition

Project Overview

Deep learning have demonstrated impressive results on a number of computer vision and natural language processing problems. At present, state-of-the-art results in image classification (Simonyan & Zisserman (2015) [1]; Szegedy et al. (2015) [2]) and speech recognition (Sercu et al. (2015) [3]), etc., have been achieved with very deep (>16 layer) CNNs. Thin deep nets are of particular interest, since they are accurate and at the same inference-time efficient (Romero et al. (2015) [4]).

The vast majority of modern convolutional neural networks (CNNs) used for object recognition are built using the same principles: They use alternating convolution and max-pooling layers followed by a small number of fully connected layers (e.g. Jarrett et al. (2009) [5]; Krizhevsky et al. (2012) [6]; within each of these layers piecewise-linear activation functions are used. The networks are typically parameterized to be large and regularized during training using dropout. A considerable amount of research has over the last years focused on improving the performance of this basic pipeline.

This project will classify images from the CIFAR-10 dataset. The images in the dataset will be preprocessed, then different models of the neural networks will be trained. Since the spatial pooling methods like max-pooling causes limited performance due to the rapid reduction in spatial size, in this project we want to try two approaches that have been reported through recent studies. The first one is applying the fractional version of max-pooling (Graham et al. (2015) [7]). The second approach is to replace the max-pooling with a subsampling method by utilizing a convolutional layer with increased stride (Springenberg et al. (2015) [8]). Finally we evaluate the prediction results for Accuracy and Loss values for each method applied on the CIFAR-10 image classification dataset.

Problem Statement

The max-pooling act on the hidden layers of Convolutional Neural Networks, reducing their size by an integer multiplicative factor (mostly $\alpha = 2$). This will help simplifying the network and reducing the process time, but its byproduct is discarding 75% of the data that causes a degree of invariance with respect to translations and elastic distortions [7]. However, if you simply alternate convolutional layers with max-pooling layers, performance is limited due to the rapid reduction in spatial size, and the disjoint nature of the pooling regions.

On the other hand, using deep and thin networks are very hard to train by backpropagation if deeper than five layers, especially with uniform initialization [4].

It seems that there is a tradeoff between the performance and process complicity and time for CNN basic pipelines. Looking for an optimal way of applying spatial pooling is the goal of this project. Since Convolutional networks almost always incorporate some form of spatial pooling, and very often it is α x α max-pooling, we

applied following CNN pipelines on CIFAR-10 dataset and compared their accuracy results to obtain the optimum model:

1- **Model-A (Max-Pooling):** Applying CNNs by incorporating a spatial max-pooling of the $\alpha \ x \ \alpha$ form with $\alpha = \{2, 3\}$.
2- **Model-B (FMP: Fractional Max-Pooling)**: Applying CNNs by incorporating a fractional version of max-pooling [7] where is allowed to take non-integer values between $1 < \alpha < 3$.
3- **Model-C (All-CNN):** Applying All-CNNs by replacing the max-pooling layers by convolutional layers with increased stride 2 with no max-pooling [8].

Architectures of these networks are described in different tables for comparison. We will use the same number of Convolution and Max Pool layers for each model but with different α values and compare their TensorFelow accuracy levels.

Metrics

Validation accuracy:

In this project we are looking for a comparable improvement in the results of the accuracies obtained from each model. The evaluation metric for the model will be TensorFlow accuracy function on the test images in the CIFAR-10 dataset. It calculates how often predictions matches labels. The accuracy function creates two local variables, total and count that are used to compute the frequency with which predictions match the labels.

Accuracy is a common metric for binary classifiers; it takes into account both true positives and true negatives with equal weight.

$$accuracy = \frac{true\ positives + true\ negatives}{dataset\ size}$$

Since CIFAR-10 dataset contains 60,000 32x32 RGB images, which are divided into 10 classes it is a class balanced classifier. Accuracy is a good metric for class balanced classifiers.

Loss:

The usual method for training a network to perform N-way classification is multinomial logistic regression, aka. Softmax regression. Softmax regression applies a softmax nonlinearity to the output of the network and calculates the cross-entropy between the normalized predictions and a 1-hot encoding of the label. For regularization, we also apply the usual weight decay losses to all learned variables.

II. Analysis

Data Exploration

This project will use the well-known object recognition dataset, CIFAR-10. It consists of 60000 32x32 RGB color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images [9].

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining

images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

Here are the classes in the dataset, as well as 10 random images from each (Figure 1):

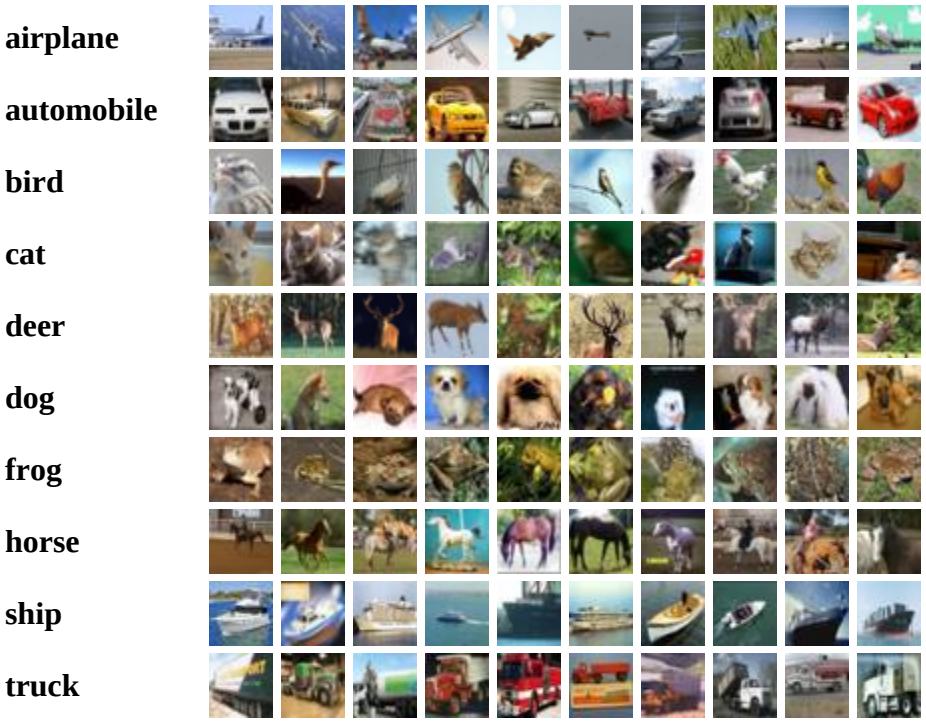

Figure 1: CIFAR-10 classes

The classes are completely mutually exclusive. There is no overlap between automobiles and trucks. "Automobile" includes sedans, SUVs, things of that sort. "Truck" includes only big trucks. Neither includes pickup trucks.

Exploratory Visualization

In this study, I used my completed image classification project for Machine Learning Nanodegree as baseline. Following files are used for that project [14]:

- image_classification.ipynb: The main file where used as baseline on the project.
- Two helper files: problem_unittests.py and helper.py

For exploring the Cifar-10 dataset, I applied different values for the batch_id and sample_id in “Explore the Data” section. The batch_id is the id for a batch in this dataset, they are in the range of 1 thru 5. The sample_id is the id for an image and label pair in the batch, so they will be in the range of 0 to 9999 for each batch (total of 5 training sets with 10000 images). Below is an example of code output when batch_id = 1 and sample_id =9999:

```
Stats of batch 1:
Samples: 10000
Label Counts: {0: 1005, 1: 974, 2: 1032, 3: 1016, 4: 999, 5: 937, 6: 1030, 7: 1001, 8: 1025,
9: 981}
First 20 Labels: [6, 9, 9, 4, 1, 1, 2, 7, 8, 3, 4, 7, 7, 2, 9, 9, 9, 3, 2, 6]
```

```
Example of Image 9999:
Image - Min Value: 4 Max Value: 252
Image - Shape: (32, 32, 3)
Label - Label Id: 5 Name: dog
```

It shows that batch 1 contains 10000 samples with about 1000 image per class (label). The label id for batch-id 1, sample-1 is Label Id: 5 Name: dog. By keeping same batch-id and different sample ids, I realized the different labels and classes will be shown as output. Same experiment repeated when I kept same sample-id but changed the batch-ids.

```
Stats of batch 1:
Samples: 10000
Label Counts: {0: 1005, 1: 974, 2: 1032, 3: 1016, 4: 999, 5: 937, 6: 1030, 7: 1001, 8: 1025,
9: 981}
First 20 Labels: [6, 9, 9, 4, 1, 1, 2, 7, 8, 3, 4, 7, 7, 2, 9, 9, 9, 3, 2, 6]

Example of Image 99:
Image - Min Value: 0 Max Value: 255
Image - Shape: (32, 32, 3)
Label - Label Id: 1 Name: automobile
```

```
Stats of batch 2:
Samples: 10000
Label Counts: {0: 984, 1: 1007, 2: 1010, 3: 995, 4: 1010, 5: 988, 6: 1008, 7: 1026, 8: 987, 9
: 985}
First 20 Labels: [1, 6, 6, 8, 8, 3, 4, 6, 0, 6, 0, 3, 6, 6, 5, 4, 8, 3, 2, 6]

Example of Image 99:
Image - Min Value: 2 Max Value: 236
Image - Shape: (32, 32, 3)
Label - Label Id: 8 Name: ship
```

This shows that the order of the samples in Cifar-10 dataset are randomized.

Algorithms and Techniques

The classifier model I used in this experiment is Convolutional Neural Network applied for image recognition process on CIFAR-10 dataset. Three CNN models in this study are built in a common way by combining two types of layers:

- Layers of convolutional filters.
- Max-pooling with variable fractional kernel size, α, in the range of $1 \leq \alpha \leq 3$.

Following algorithms are used in building different three different models and the algorithm outputs for them are Accuracy and Loss values for each classified classes:

- **Model-A (Max-Pooling):** When $\alpha = 2$ or 3, the standard max-pooling CNN model with stride two is being implemented.
- **Model-B (Fractional Max-Pooling):** When $1 < \alpha < 2$ and $2 < \alpha < 3$, the fractional max-pooling layers are being utilized for further examinations.
- **Model-C (All-CNN):** When $\alpha = 1$, the CNN model contains only convolutional layers in which the max-pooling layers are replaced with standard convolutional layers with same output number and stride two.

The following parameters can be tuned to optimize the classifier:

- Preprocessing parameters:
    - Normalizing(Normalize a list of sample image data in range of 0 to 1)
    - One-hot encode(One hot encode the list of sample labels)

- Training parameters:
    - Training length (number of epochs)
    - Batch size (how many images to look at once during a single training step)
    - Solver type (what algorithm to use for learning)
    - Learning rate (how fast to learn; this can be dynamic)
    - Weight decay (prevents the model being dominated by a few “neurons”)
    - Momentum (takes the previous learning step into account when calculating the next one)

- Neural network architecture:
    - Number of layers
    - Layer type (convolutional, full or fractional max-pooling, all CNN, dropout, fully-connected)
    - Layer parameters

After normalizing the data and one-hot encoding the list of sample labels, all the CIFAR-10 data will be preprocessed as Training, Validation, and Testing Data categories and saved to file in disk.
During training, both the training and the validation sets are loaded into the RAM. After that, random batches are selected to be loaded into the GPU memory for processing. The training is done using the Mini-batch gradient descent algorithm.

Benchmark

Because of limitations on accessing powerful computers and research tools, I wanted to explore the characteristic and features of CIFAR-10 dataset through recent similar studies and examine their behavior around the conditions that I would be able to apply with my resources. For example I used 10 epochs in my studies and

wanted to compare my results with those studies around same number of epochs and see if there are significant differences between the results.

Similar results could be seen when applying different activation functions (Mishkin et al. 2016 [12]) (Figure 2):

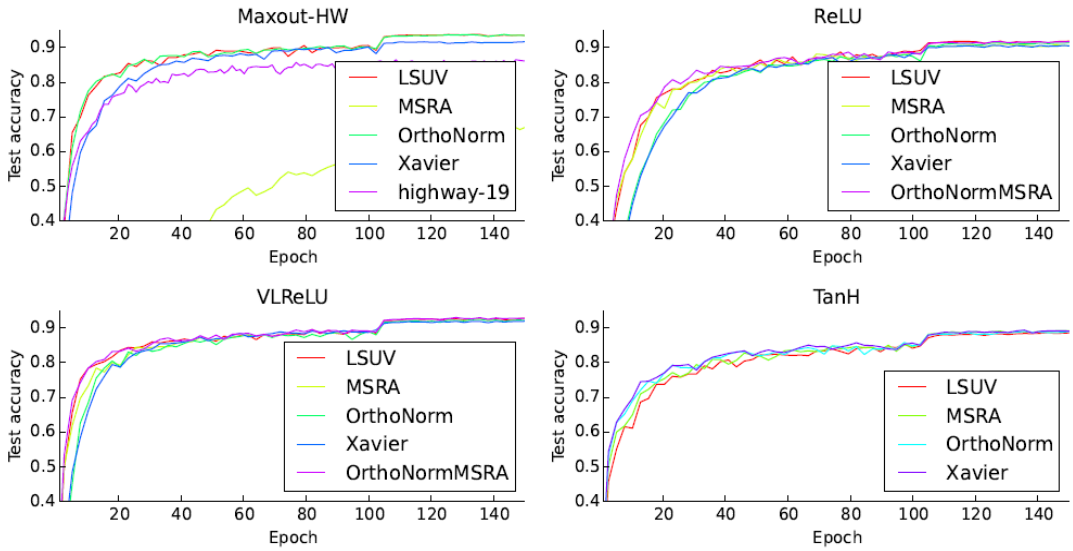

Figure 2: CIFAR-10 accuracy of FitNet-4 with different activation functions.

According to the recent studies on implementing a variant of the VGG net [10], and applying in in different conditions [11], the accuracy level is around 60% for 10 epochs which is close to my experiments (Figure 3).

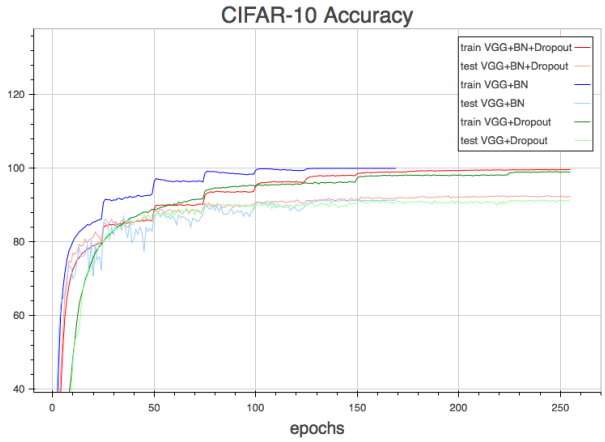

Figure 3: CIFAR-10 dataset experiment: test accuracy vs. epoch number

I also explored the dataset behavior for Loss based on different epoch values [13]. Since there's noise in training data, we should choose the right epochs that converge to a good model without overfitting (Figure 4).

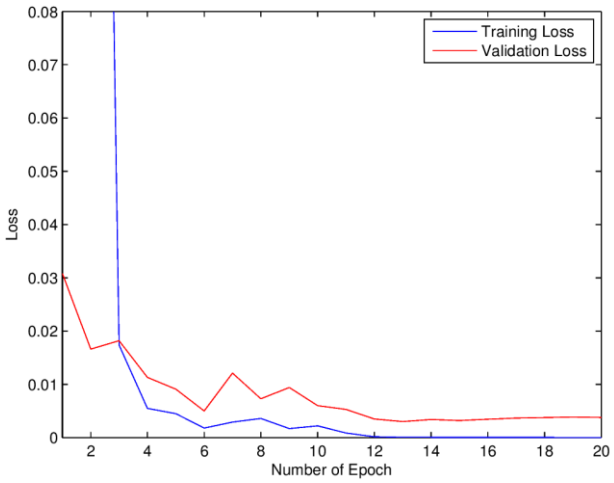

Figure 4: CIFAR-10 dataset experiment: train/validation Loss vs. epoch number

III. Methodology

Data Preprocessing

In this project we will first download the CIFAR-10 dataset for python and then implementing the following preprocess functions by normalizing the image date and one-hot encoding the labels.

Normalize:

Each image in the CIFAR-10 dataset is a 32x32 RGB color image that can be considered as a list, x, with the image shape of (32, 32, 3). For each layers of RGB colors, we can consider as 256 grayscale from 0 to 255. By dividing each list of image data by 255, (x / 255), the output would be a Numpy array of Normalize data in the range from 0 to 1 with the same shape (32, 32, 3).

One-hot encode:

For the one hot encode function, we used LabelBinarizer in the preprocessing module of Sklearn. By implementing the function the list of the labels urn the list of labels as One-Hot encoded Numpy array. The possible values for labels are 0 to 9. The one-hot encoding function should return the same encoding for each value between each call to one_hot_encode:

```
array ([[1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 1]])
```

Randomize Data:

As explained in the exploring the data above, the order of the samples are randomized. So we didn't need to for this dataset.

Implementation

As outlined in the Algorithms and Techniques section above, I used three models for this project and below I will describe the common layers used in each model. Each layer is built into a function that will be used to create the Convolutional Models.

Later on the metrics, algorithms, and techniques that implemented for each Convolutional Model will be described separately.

Input:

This model is a standard max-pooling CNN model that needs to read the image data, one-hot encoded labels, and dropout keep probability as input:

- Implemented neural_net_image_input function:
    - Return a TF Placeholder
    - Set the shape using image_shape with batch size set to None.
    - Name the TensorFlow placeholder "x" using the TensorFlow name parameter in the TF Placeholder.
- Implemented neural_net_label_input
    - Return a TF Placeholder
    - Set the shape using n_classes with batch size set to None.
    - Name the TensorFlow placeholder "y" using the TensorFlow name parameter in the TF Placeholder.
- Implemented neural_net_keep_prob_input
    - Return a TF Placeholder for dropout keep probability.
    - Name the TensorFlow placeholder "keep_prob" using the TensorFlow name parameter in the TF Placeholder.

We built each one of following layers into a function: Convolution and Max Pooling Layer, Flatten Layer, Fully-Connected Layer, and Output Layer.

Convolution and Max Pooling Layer

For this layer we created a function called "conv2d_maxpool" to apply convolution and then max pooling:

Parameters:

x_tensor: TensorFlow Tensor
conv_num_outputs: Number of outputs for the convolutional layer
conv_ksize: kernel size 2-D Tuple for the convolutional layer
conv_strides: Stride 2-D Tuple for convolution
pool_ksize: kernel size 2-D Tuple for pool
pool_strides: Stride 2-D Tuple for pool

Steps:

- Created the weight and bias using conv_ksize, conv_num_outputs and the shape of x_tensor by using TensorFelow Neural Network version of conv2d (tf.nn.conv2d).

- Applied a convolution to x_tensor using weight and conv_strides with ‘same’ padding and std = sqrt (2/n) with n being the number of inputs of the layer (n = kernel_width*kernel_height*depth_previous_layer).
- Added bias
- Added ReLU as nonlinear activation function to the convolution.
- Applied Max Pooling using pool_ksize and pool_strides with ‘same’ padding.

Flatten Layer and Fully-Connected Layer

Created the “flatten” function to change the dimension of x_tensor from a 4-D tensor to a 2-D tensor. The output is the shape (Batch Size, Flattened Image Size). Then created the “fully_conn” function to apply a fully connected layer to x_tensor with the shape (Batch Size, num_outputs).

Output Layer

Implement the “output” function to apply a fully connected layer to x_tensor with the shape (Batch Size, num_outputs). In this layer the softmax activation, and cross entropy is pre coded and there is no need to add them to the code.

Create Convolutional Models

The function “conv_net” is implemented to create a convolutional neural network Model-A (Max-Pooling). It takes in a batch of images, x, and outputs logits. The layers functions that was created above were uses to create this model:

- Applied 3 or 4 Convolution layers with stride 1, each one followed by max-pooling layers with stride 2
- Applied a Flatten Layer
- Applied 3 Fully Connected Layers
- Applied TensorFlow's Dropout to the first 2 Fully Connected layers in the model using keep_prob
- Applied an Output Layer
- Returned the output

Train the Neural Network

Implemented the function “train_neural_network” to do a single optimization. The optimization uses the tf.train.AdamOptimizer() optimizer to optimize in session with minimizing the cost. Cost is being calculated by tf.nn.softmax_cross_entropy_with_logits where logits is the output of the previously defined function “conv_net”. The optimization should use optimizer to optimize in session with a feed_dict of the following:

- x for image input
- y for labels
- keep_prob for keep probability for dropout

This function will be called for each batch and uses following inputs and is only optimizing the neural network.

Show Stats

The print_stats function prints loss and validation accuracy. The global variables valid_features and valid_labels are used to calculate validation accuracy. Keep-probability of 1.0 is being used to calculate the loss and validation accuracy.

Hyperparameters

Following parameters were tuned in order to get the better results:

- I tried different epochs to set the number of iterations until the network stops learning or start overfitting. The epochs = 10 is the value I chose as it was giving same trend of training progress for all models.
- I also tried different batch_size values to the different numbers that my machine has memory for. Best results with reasonable time for my machine was for the batch_size = 64 and I used it for all models to establish a baseline for comparison.
- The keep_probability was set to the probability of keeping a node using dropout. The best value that worked for my machine with best and quickest results was .8 or dropping 20% of data after the first two of three fully connected layers.

Train on a Single CIFAR-10 Batch

Instead of training the neural network on all the CIFAR-10 batches of data, we used a single batch. This saved time while we iterate on the model to get a better accuracy. Once the final validation accuracy was in a reasonable range, I used the setup for parameters value to run the model on all the data in the next section.

Fully Train and Test the Model

After getting a good accuracy with a single CIFAR-10 batch, I tried it with all five batches. Finally I tested the model against the test dataset. This will be your final accuracy. In this study I just kept the models with an accuracy around or greater than 50%.

CNN Models Implementation

In this section, the above mentioned implementation procedure will be described for each of the three different Convolutional Models and the metrics, algorithms, and techniques used for them will be explained.

- **Model-A (Max-Pooling):**
  - Applied 3 Convolution layers with stride 1, each followed by a max-pooling layers with stride 2
    I. With $\alpha = 2$ and stride = 2, the pooling regions are disjoint
    II. With $\alpha = 3$ and stride = 2, the pooling regions are overlapping

- **Model-B (Fractional Max-Pooling):**
  - Applied 3 Convolution layers with stride 1, each followed by a max-pooling layers with stride 2
    I. With $1 < \alpha < 2$ and stride = 2, the pooling regions are disjoint
    II. With $2 < \alpha < 3$ and stride = 2, the pooling regions are overlapping

- **Model-C (All-CNN):**
  In this model I set the α value to 1 ($\alpha = 1$) which means no max-pooling is taking place after each convolution layer. So the model will be all CNN with no spatial pooling.
  - Applied 3 Convolution layers with stride 1
    I. With $\alpha = 1$ and stride = 2, no pooling and all CNN layers
    II. With $\alpha = 1$ and stride = 2, the max-pooling layers replaced with standard convolutional layers with same output number and stride=2

Table 1 shows the configuration of the three base neural network models used for classification on CIFAR-10 dataset. The fractional max-pooling, Model-B (FMP), has been applied to different kernel sizes in the range $1 < \alpha < 2$ with increasing steps of 0.2 and only the best result is shown in the table to avoid cluttering the paper.

Model		
A(Max-Pooling)	**B (Fractional Max-Pooling)**	**C(All-CNN)**
Input 32 x 32 RGB image		
4 x 4 conv.; 128 ReLU	4 x 4 conv.; 128 ReLU	4 x 4 conv.; 128 ReLU
2 x 2 max-pooling stride 2	1.8 x 1.8 max-pooling stride 2	
3 x 3 conv.; 256 ReLU	3 x 3 conv.; 256 ReLU	3 x 3 conv.; 256 ReLU
2 x 2 max-pooling stride 2	1.8 x 1.8 max-pooling stride 2	
3 x 3 conv.; 512 ReLU	3 x 3 conv.; 512 ReLU	3 x 3 conv.; 512 ReLU
2 x 2 max-pooling stride 2	1.8 x 1.8 max-pooling stride 2	
Flatten Layer(4D to 2D x-tensor (Batch Size, Flattened Image Size))		
Fully Connected Layer: fully_conn(x, 128)		
Dropout: (keep_prob = .8)		
Fully Connected Layer: fully_conn(x, 64)		
Dropout: (keep_prob = .8)		
Fully Connected Layer: fully_conn(x, 32)		
Output Layer: output(x, 10)		
Loss and Optimizer (softmax_cross_entropy and Adam Optimizer().minimize(cost))		
Accuracy		
Accuracy: 0.42 Loss: 34	Accuracy: 0.46 Loss: 19	Accuracy: 0.39 Loss: 25

Table 1: The three base networks used for classification on CIFAR-10

Although the results are not close to the state of the art results indicated in the benchmark section, but they are close to their results around 10 epochs as it was mentioned in that section. Among the three models the Model-B obtained the best results for Testing Accuracy and Loss compare to the other two models.

Refinement

In order to improve the performance of our training models, I added one more convolution and max-pooling layer with higher number of outputs. I have also noticed that when the pooling regions are disjoint, we might lose accuracy due to dropping data that were not evaluated during max-pooling. So by applying overlapping pooling regions, we may improve the accuracy for standard and fractional max-pooling.

Each convolutional filter of a CNN produces a matrix of hidden variables. The size of this matrix is often reduced using some form of pooling. For regular 2 x 2 max-pooling and stride 2, $N_{in} = 2N_{out}$. With $2 < \alpha < 3$ and stride $= 2$, the pooling regions are overlapping and $N_{in}/N_{out} \approx 2$. So the spatial size of any interesting features in the input image halve in size with each pooling layer [7]. Since the speed was an important in this study, we applied the FMP with $N_{in}/N_{out} \in (2, 3)$.

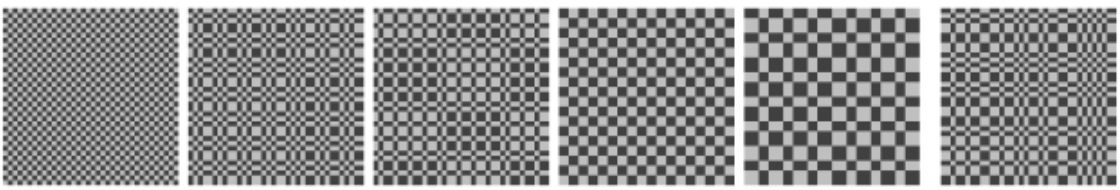

Figure 5: Left to right: A 36 x 36 square grid [7]; disjoint pseudorandom FMP regions with $\alpha \in \{\sqrt[3]{2}, \sqrt{2}, 2, \sqrt{5}\}$; and disjoint random FMP regions for $\alpha = \sqrt{2}$. For $\alpha \in (1, 2)$ the

rectangles have sides of length 1 or 2. For α∈ (2, 3) the rectangles have sides of length 2 or 3.

For Model-C, ALL CNN, I added one more convolutional layer with same output number but increased stride to 2 to replace the max-pooling layer on other models. The replacement CNN layers can be chosen with higher strides than preceding layer in order to simulate the max-pooling subsampling and speeding up the process

IV. Results

Table-2 below shows the configuration of the same three base neural network models after refinement on CIFAR-10 dataset. For each model I added one more pair of convolution and max-pooling layers which makes it the total of 8 filter layers. For the fractional max-pooling, Model-B, I applied different kernel sizes in the range $2 < \alpha < 3$ with 0.2 incremental steps. All models have three Fully Connected Layers after convolution layers with 128, 64, and 32 output numbers. The first two fully connected layers are followed by a dropout layer with 20% data drop.

conv2d_maxpool configuration	Max-Pooling Kernel Size	Loss	Accuracy
(128, (3, 3), (1, 1), (1, 1), (2, 2)) (256, (3, 3), (1, 1), (1, 1), (2, 2)) (512, (3, 3), (1, 1), (1, 1), (2, 2)) (1024, (3, 3), (1, 1), (1, 1), (2, 2))	1	13.5	0.4254
(128, (3, 3), (1, 1), (1.4, 1.4), (2, 2)) (256, (3, 3), (1, 1), (1.4, 1.4), (2, 2)) (512, (3, 3), (1, 1), (1.4, 1.4), (2, 2)) (1024, (3, 3), (1, 1), (1.4, 1.4), (2, 2))	1.4	26.18	0.4591
(128, (4, 4), (1, 1), (1.6, 1.6), (2, 2)) (256, (3, 3), (1, 1), (1.6, 1.6), (2, 2)) (512, (3, 3), (1, 1), (1.6, 1.6), (2, 2)) (1024, (3, 3), (1, 1), (1.6, 1.6), (2, 2))	1.6	17.31	0.4668
Same as above	1.8	17.95	0.4779
Same as above	2	51.6	0.4819
Same as above	2.2	39.59	0.4819
Same as above	2.4	26.53	0.5358
Same as above	2.6	36.32	0.4945
Same as above	2.8	51.27	0.5236
Same as above	3	49.41	0.5184
Same as above	1	23.2	0.4337

(128, (4, 4), (1, 1), (1, 1), (2, 2)) (128, (4, 4), (2, 2), (1, 1), (2, 2)) (256, (3, 3), (1, 1), (1, 1), (2, 2)) (256, (3, 3), (2, 2), (1, 1), (2, 2)) (512, (3, 3), (1, 1), (1, 1), (2, 2)) (512, (3, 3), (2, 2), (1, 1), (2, 2)) (1024, (3, 3), (1, 1), (1, 1), (2, 2)) (1024, (3, 3), (2, 2), (1, 1), (2, 2))	1	0.8994	0.4885

Table 2: The three base networks used for classification on CIFAR-10 after Refinement

The refinement attempts shows some progress on the Loss and Accuracy values. The highest accuracy level, 0.5358, was achieved with fractional max-pooling Model-B with the kernel size of α = 2.4. But the best achieved Loss value was 0.8994 that was for All CNN Model-C. The accuracy value for this model was around 0.5 that is in acceptable range.

Model Evaluation and Validation

I applied all the trained models on the test dataset of CIFAR-10 with different parameter tuning attempts and selected the best results for each models for evaluation (table 3):

Model		
A(Max-Pooling)	**B (FMP)**	**C(All-CNN)**
Input 32 x 32 RGB image		
4 x 4 conv.; 128 ReLU	4 x 4 conv.; 128 ReLU	4 x 4 conv. stride 1; 128 ReLU
3 x 3 max-pooling stride 2	2.4 x 2.4 max-pooling stride 2	4 x 4 conv. stride 2; 128 ReLU
3 x 3 conv.; 256 ReLU	3 x 3 conv.; 256 ReLU	3 x 3 conv. stride 1; 256 ReLU
3 x 3 max-pooling stride 2	2.4 x 2.4 max-pooling stride 2	3 x 3 conv. stride 2; 256 ReLU
3 x 3 conv.; 512 ReLU	3 x 3 conv.; 512 ReLU	3 x 3 conv. stride 1; 512 ReLU
3 x 3 max-pooling stride 2	2.4 x 2.4 max-pooling stride 2	3 x 3 conv. stride 2; 512 ReLU
3 x 3 conv.; 1024 ReLU	3 x 3 conv.; 1024 ReLU	3 x 3 conv. stride 2; 1024 ReLU
3 x 3 max-pooling stride 2	2.4 x 2.4 max-pooling stride 2	3 x 3 conv. stride 2; 1024 ReLU
Flatten Layer(4D to 2D x-tensor (Batch Size, Flattened Image Size))		
Fully Connected Layer: fully_conn(x, 128)		
Dropout: (keep_prob = .8)		
Fully Connected Layer: fully_conn(x, 64)		
Dropout: (keep_prob = .8)		
Fully Connected Layer: fully_conn(x, 32)		
Output Layer: output(x, 10)		
Loss and Optimizer (softmax_cross_entropy and Adam Optimizer().minimize(cost))		
Accuracy		
Accuracy: 0.52 Loss: 49	Accuracy: 0.54 Loss: 26	Accuracy: 0.4885 Loss: .8994

Table 3: Best performed base models used for classification on CIFAR-10 after Refinement

The refinements described in previous section improved the results for all three models.

For Model-A the additional convolution and max-pooling layer increased the accuracy for 23% from 0.42 to .52 and increased the loss for 44% from 34 to 49.

The Model-B had highest level of accuracy among all models and increased about 17% from its previous value before the refinement. It seems applying the overlapping pooling region had major role in the improvement together with additional convolutional layer. The loss value increased after refinement from 19 to 26.

The Model-C had the most significant improvement on both accuracy and loss values, but it took a very long time to complete the classification process on the test data in my machine. The accuracy increased 25% from .39 to .49 and massive decrease in loss from 25 to .8995(about 27 times smaller).

By comparing above results, I select Model-C as final model since it had highest improvement on both accuracy and loss. Although it was taking a very long time on my machine for completion, I believe it would work within reasonable time on more powerful computers even for bigger datasets.

Justification

Comparing to the benchmarks and state of the art results, the Model-C (All CNN), performs very close to them for epoch values around 10 (Figures 6 & 7).

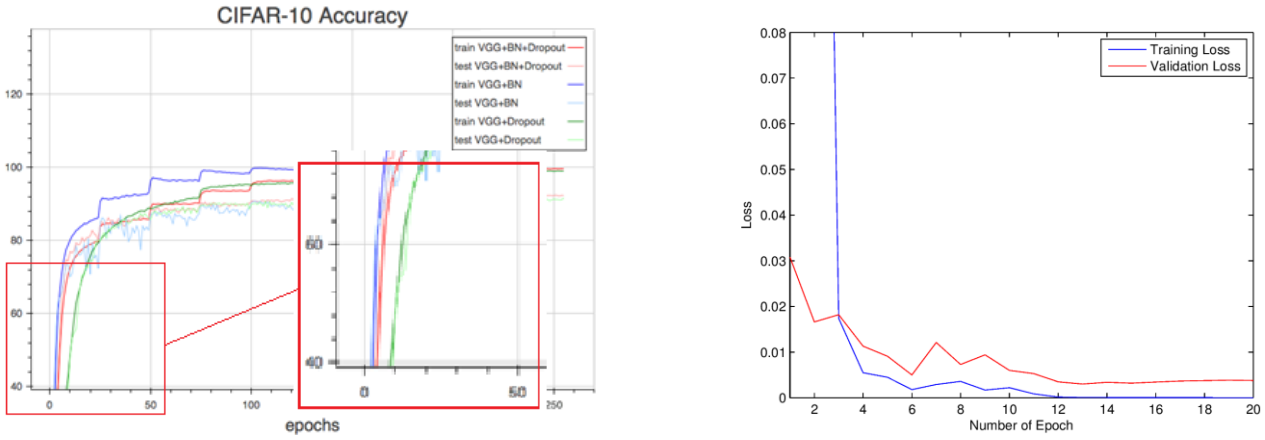

Figure 6: CIFAR-10 accuracy and loss around epochs=10 for benchmarks

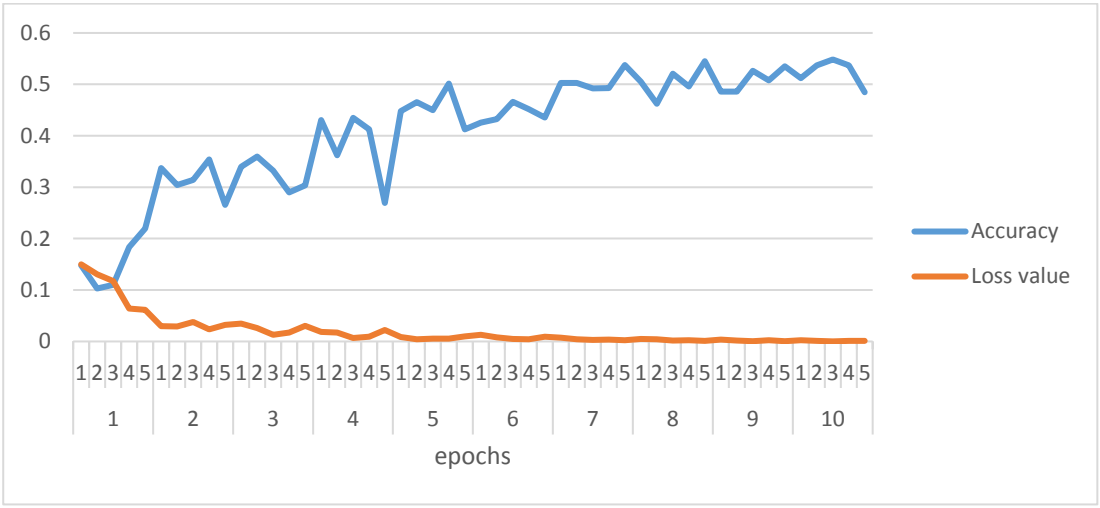

Figure 7: CIFAR-10 accuracy and loss around epochs=10 for Model-C

V. Conclusion

Free-Form Visualization

We have applied the methodologies used in two recent studies on utilizing the Fractional Max-Pooling (FMP) [7] and All Convolutional Neural Network (All-CNN) [8]. Both methods showed better results than the standard CNN approach of image classification on CIFAR-10 dataset. Below is the summary list of advantages and disadvantages of each methods:

Fractional Max-Pooling (FMP):

- Should be tried for several different FMP kernel sizes with small epochs to obtain the optimum size
- Improves the testing accuracy but doesn't perform significantly well for loss
- Takes less time compare to All-CNN method but longer time than standard CNN
- Is sensitive to disjoint pooling region and needs careful stride selection [7]
- Performs better on overlapping pooling region

All-CNN:

- Improves the classifier's performance on Loss extremely (more than 30 times better than FMP and CNN)
- Improves Testing Accuracy almost similar to FMP and standard CNN
- Takes longer process time than FMP and standard CNN
- The replacement CNN layers can be chosen with same output number but higher strides than preceding layer in order to simulate the max-pooling subsampling and speeding up the process [8]

Reflection

Following steps took place in this project:

- Searching for existing studies and solutions to address the precision and process time problems for CNN image recognition methods
- Selected two recent studies that was suggested to improve CNN methods: Fractional Max-Pooling (FMP) [7] & All-CNN [8]
- Applied three methods, Standard CNN, FMP, and All-CNN on a simple baseline configuration of 3 pairs of CNN and max-pooling layers with 10 epochs due to the power of accessible machine
- Studied the results and found improvement ways for refinement
- Added one more pair of CNN & max-pooling to the standard and FMP models. Added one CNN layer in the All-CNN model that replaces the max-pooling layer in other models. Each replacement layer has same output number as preceding layer but higher stride in order to fulfill the subsampling effect of max-pooling
- Compared the performance of each model and suggested the optimum model based on the condition and resource availability for each research or study

Below is the visualized results of our studies in this project:

Figure 8 shows the results of all above mentioned experiments we made in this project for the Loss value. Experiment 23 is the Model-C All CNN where the max-pooling layer was replaces with convolution neural network (kernel size=1) and stride=2 where the Loss value decreased to almost zero from twenties in all other model configurations:

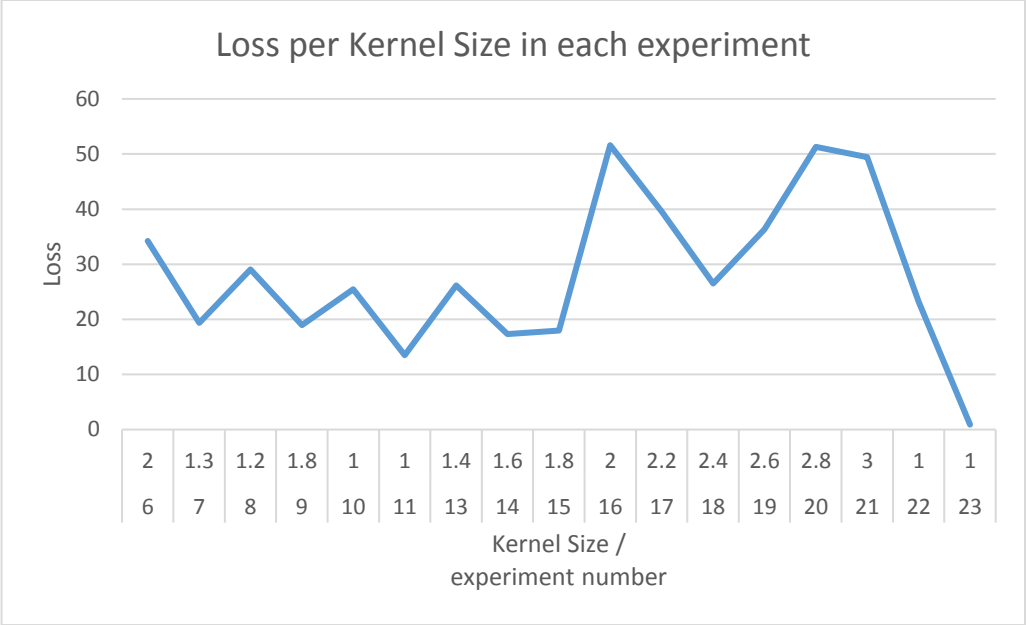

Figure 8: CIFAR-10 Loss value around epochs=10 for all Models

Figure 9 shows the results of all experiments we made in this project for the Validation Accuracy. Experiment 23 is the Model-C All CNN shows reasonable and comparable value to other models and their different configurations:

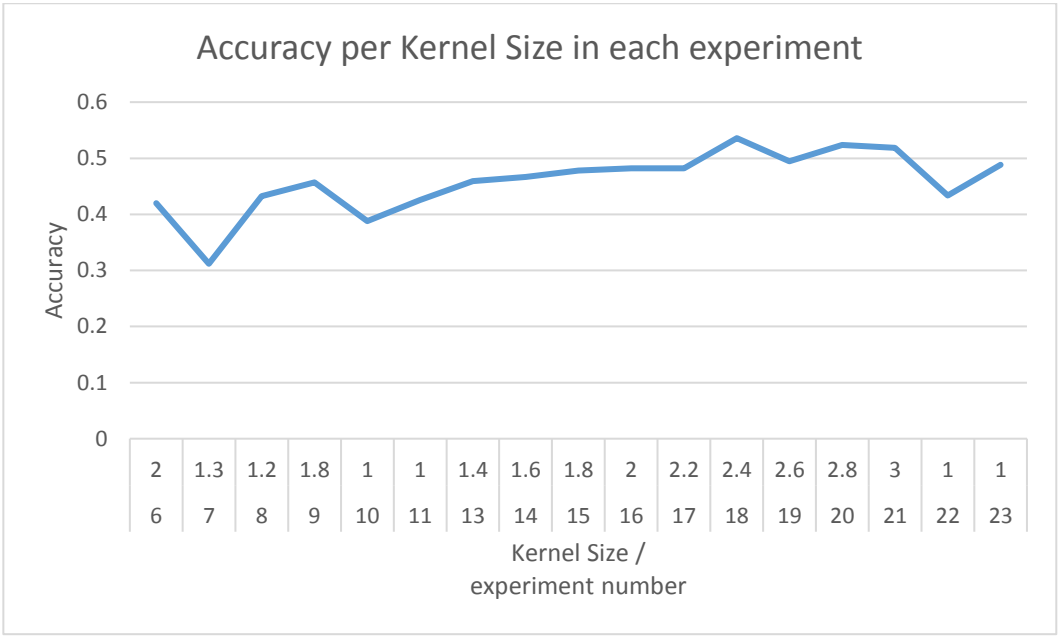

Figure 9: CIFAR-10 Accuracy value around epochs=10 for all Models

Below is the experiment 23 result output for CIFAR-10 image recognition by applying All-CNN Model-C:

```
Testing Accuracy: 0.4885549363057325
```

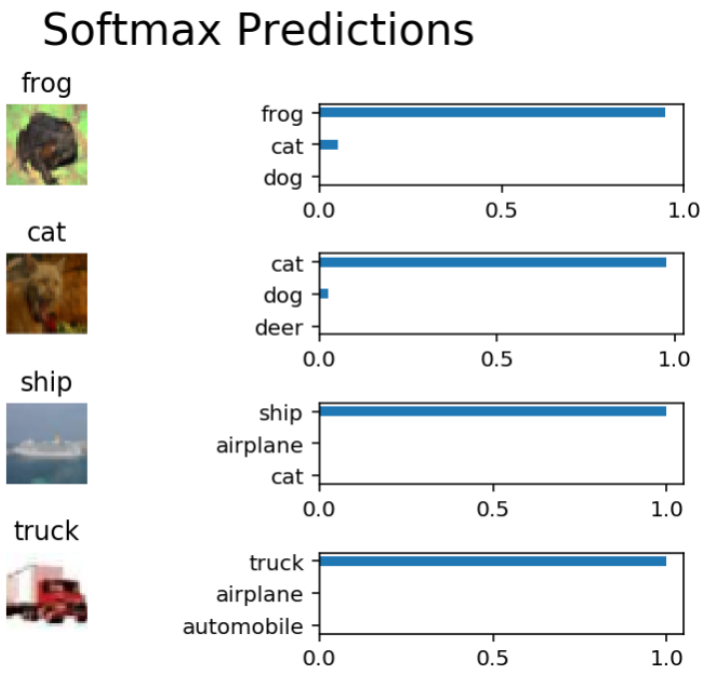

Figure 10: ALL CNN model output for CIFAR-10 test dataset

Improvement

Considering above observations, I would select ALL-CNN model for image recognition for state of the art powerful machines with strong GPU units and apply it for deeper convolutional neural networks and much higher epochs.

In case of time and machine power concerns, I would choose FMP method with concentrating on kernel size and overlapping max-pooling regions to achieve the optimal results.

Model-A, the standard CNN, is the quickest model if we have restriction on time and machine process power.

REFERENCES

[1] Simonyan, Karen and Zisserman, Andrew. Very deep convolutional networks for large-scale visual recognition. In Proceedings of ICLR, May 2015. URL http://arxiv.org/abs/1409.1556*.*

[2] Szegedy, Christian, Liu, Wei, Jia, Yangqing, Sermanet, Pierre, Reed, Scott, Anguelov, Dragomir, Erhan, Dumitru, Vanhoucke, Vincent, and Rabinovich, Andrew. Going deeper with convolutions. In CVPR 2015, 2015. URL http://arxiv.org/abs/1409.4842*.*

[3] Sercu, T., Puhrsch, C., Kingsbury, B., and LeCun, Y. Very Deep Multilingual Convolutional Neural Networks for LVCSR. ArXiv e-prints, September 2015. URL http://arxiv.org/abs/1509/08967*.*

[4] Romero, Adriana, Ballas, Nicolas, Kahou, Samira Ebrahimi, Chassang, Antoine, Gatta, Carlo, and Bengio, Yoshua. Fitnets: Hints for thin deep nets. In Proceedings of ICLR, May 2015. URL http://arxiv.org/abs/1412.6550*.*

[5] Jarrett, Kevin, Kavukcuoglu, Koray, Ranzato, Marc'Aurelio, and LeCun, Yann. What is the best multi-stage architecture for object recognition? In ICCV, 2009.

[6] Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In NIPS, pp. 1106–1114, 2012.

[7] Graham, Benjamin: Fractional Max-Pooling, May 2015. URL https://arxiv.org/abs/1412.6071

[8] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller: STRIVING FOR SIMPLICITY: THE ALL CONVOLUTIONAL NET, April 2015. URL https://arxiv.org/abs/1412.6806

[9] "The CIFAR-10 and CIFAR-100 are labeled subsets of the 80 million tiny images dataset collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton: https://www.cs.toronto.edu/~kriz/cifar.html

*[10] Uri Shaham et al: Understanding Adversarial Training: Increasing Local Stability of Neural Nets through Robust Optimization:*https://www.researchgate.net/publication/284219659_Understanding_Adversarial_Training_Increasing_Local_Stability_of_Neural_Nets_through_Robust_Optimization#pfc

[11] Sergey Zagoruyko: 92.45% on CIFAR-10 in Torch: http://torch.ch/blog/2015/07/30/cifar.html

[12] Dmytro Mishkin, Jiri Matas: ALL YOU NEED IS A GOOD INIT https://arxiv.org/abs/1511.06422

[13] Jiao Dong: Simple LB 0.23800 solution (Keras + VGG_16 pre trained) https://www.kaggle.com/c/state-farm-distracted-driver-detection/discussion/20747

[14] Udacity/ machine-learning / projects / image-classification /: https://github.com/udacity/machine-learning