Capstone Project Report

Machine Learning Engineer Nanodegree

Tracy Xu

Jan 31th, 2017

Definition

- **Project Overview**

  This project is taken from one of the favorite for-fun Kaggle competitions in 2013: Dogs vs. Cats Redux. The goal of this fun project is very straightforward, training a model to classify a given image, and returns the result of whether this image contains a dog or a cat. This problem is a classic image classification task, and although this problem might sound naïve, there could be many ways to tackle the challenge, including the recent powerful technique, deep learning. In this project, I will first use the existing famous Convolution Neural Network (CNN), LeNet-5, which is an excellent architecture for handwritten digit recognition [1] as the benchmark. Later on, I will build a deep learning model with similar ideas as LeNet-5 and train it with the given dataset obtained from Kaggle. [2]

Fig. 1 Dogs vs. Cats Redux Cover

- ## Problem Statement

  The goal is to create a model that takes images containing a dog or a cat, and the model should return whether the image contains a dog or a cat. This is a classification task, to be specifically, a binary classification. Thus, Convolutional Neural Network seems a great solution to the task. Details on model selection would be given in the "Algorithm and Techniques" section.

  The tasks to tackle the challenge break down to the following:

  1. Download and extract the image dataset
  2. Analyze the image size
  3. Preprocess images on size and normalization
  4. Feed images to train CNN model
  5. Fine tune the model and iterate

- ## Metrics

  Since the task is a classic classification problem, accuracy would be a good metric choice. The formula looks as: $Accuracy = \frac{\text{number of correctly classified images}}{\text{number of total images}}$. Although F1 score is widely used for binary classification, precision and recall are relatively not important in this project. Thus a simple accuracy metric is used. During model training, there would be 3 types of accuracy, training accuracy, validation accuracy and test accuracy. The training accuracy refers to the classification accuracy when the model is under training. The training accuracy would increase over the training trials, since the model is learning to adjust the weight parameters to correct and improve itself. The validation accuracy refers to the classification accuracy on the validation dataset, which is part of the training dataset but with the label hidden to the model. The validation set is used when there might not be sufficient dataset available for model training. Finally, the test accuracy refers to the classification accuracy on the dataset, which are never seen by the model, and it is a good metric to see if the model is generalized well. In this project, validation accuracy would be used as the metric.

Analysis

- ## Data Exploration

  The training dataset obtained from Kaggle includes 12,500 dogs and cats images respectively. Total size of training folder is 571.9 MB. Each image in the training folder has the label as part of the filename. For example, a file named “cat.855.ipg” indicates this image is labeled as “cat”; similarly, a file named “dog.481.jpg” indicated this image is labeled as “dog”. Labels of images would be preprocessed to generate the “y_train” dataset. Notice the “cat.1317.jpg” contains not only two cats, but also two persons. In the actual model training, this image is not used. Details will be given in the “Data Preprocessing” section.

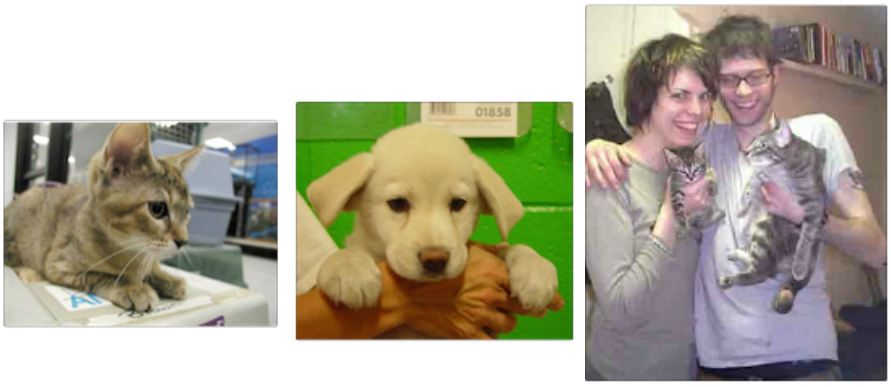

Fig. 2 Cat and Dog Filename

- ## Exploratory Visualization

  Also, each image comes with different height pixels and width pixels, which would be a problem for model training. Let’s take a look at the range of the pixel values among all the 25,000 images. The mean width is 404 pixels and the mean height is 360 pixels. Width and height vary a lot and there are 2 outliers that are in the scale of 1000 x 700 pixels. Also, since each image is in RGB format, it has 3 channels, the size of width and

height need to be reduced for processing efficiency. Specifically for the LeNet-5 model, the input requires 32 x 32.

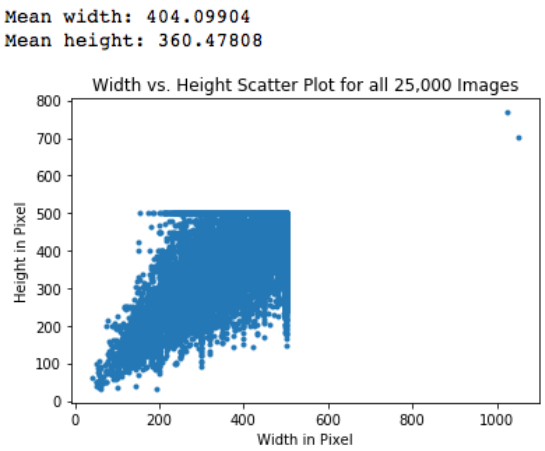

Fig. 3 Plot of Width And Height Pixel Value

- **Algorithms and Techniques**

Till now, the first two questions you might want to ask are: what the earth is neural network and what the earth is convolution? To answer the first question, the neural network is a model used in computers to simulate the neuronal structure of the mammalian cerebral cortex in brains, in order to learn languages or recognize images like humans. [3] To answer the second question, convolution is defined as "an integral that expresses the amount of overlap of one function g as it is shifted over another function f." [4]

Here is an example, think of the 3 x 3 yellow block shown in fig. 4 as function g and the 5 x 5 green block as function f. The yellow block slide over the green block from left to right, from top to bottom starting at the top left corner, notice the x1 and x0 written in red inside the yellow block, these are the weights. The left image in fig.4 shows the initial position of yellow block, and the result is 1*1 + 0*1 + 1*1 + 0*0 + 1*1 + 1*0 + 0*1 + 0*0 + 1*1 = 4, which is shown in the convolved feature. The right image in fig.4 shows the final position of the yellow block and the final result of the convolved feature matrix. The green block could be looked as the original image and the yellow block as some kind

of filter. Different filter has different weights inside the yellow block generating different convolved feature. In fig.5, the input image would be the green block and the kernel would be the yellow block and the feature map would be the pink block. We could notice the feature map is slightly blurred and this is because in the kernel, the weights in outer layer are one half of that in inner layer, which soothes out the edges. This kernel is called Gaussian blur. [5]

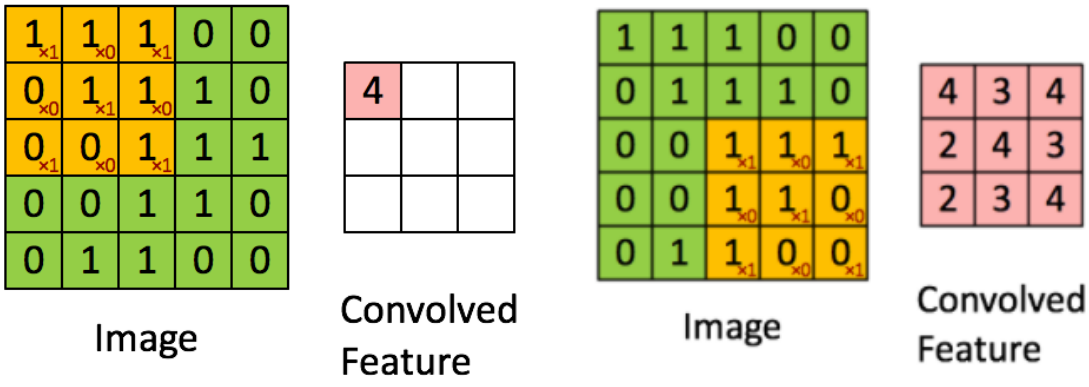

Fig. 4 Convolution Process Image source [6]

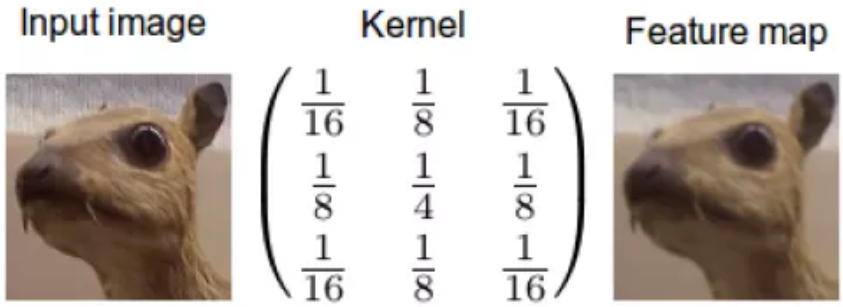

Fig. 5 Convolution Example Image source [5]

Convolutional Neural Network (CNN), which consists of a number of convolutional and subsampling layers followed by fully connected layers. [7] The convolutional layer is to extract features from the input image and preserving the spatial information and subsampling layer is followed to down sample the feature map, thus reducing the dimensionality. For example, subsampling techniques include max pooling, sum pooling, average pooling and so on. In this project, we will be using the max pooling layer

because it often works well in practice. In fig.6, the original input is a 4 x 4 block, consisting of four 2 x 2 blocks, namely red, green, yellow and blue blocks. The max pooling process is to take a look at these four 2 x 2 blocks and pick the maximum value out of 4 numbers of each block and put them together as a new 2 x 2 block. The max pooling process is a great way to reduce the dimension of the input while keeping track of the index max activation. [8]

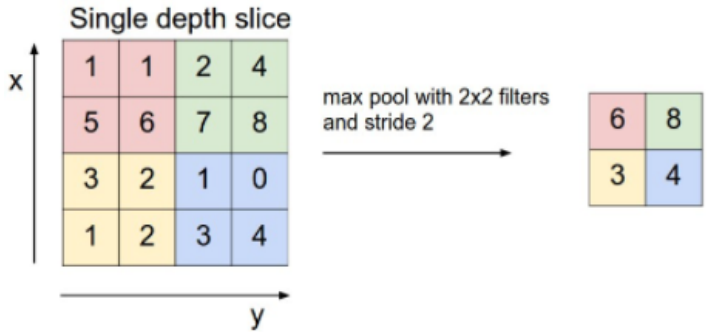

Fig. 6 Convolution Process Image source [8]

There are several factors to consider when tuning the CNN model:

1. Types of optimizer: Stochastic Gradient Descent, Adam Optimizer, etc.
2. Learning rate for optimizer: determines how quickly the gradient updates
3. Batch size: number of dataset to train at one shot
4. Epochs to train: number of total training iterations

Similarly, there are several factors to consider when designing the CNN model:

1. Number of hidden units: determines the width of the neural network
2. Weight decay: regularization on the network weights to reduce overfitting
3. Activation function: to introduce non-linearity and squash the output to desired range
4. Weight initialization: the initialization could have a huge impact on avoid training algorithm stuck on local minima
5. Preprocessing input data: simply resize the input images or use PCA to reduce dimensionality for efficient model training

Keeping the above hyper parameters in mind when designing and tuning the neural network model, helps to achieve optimal result within a reasonable amount of time.

- **Benchmark**

LeNet-5 architecture is used as benchmark. Notice that the input takes 32 x 32 images, and the LeNet-5 model contains 2 convolutional layers and each convolutional layer followed by one max-pooling layer. There are 2 full connection layers before the output layer.

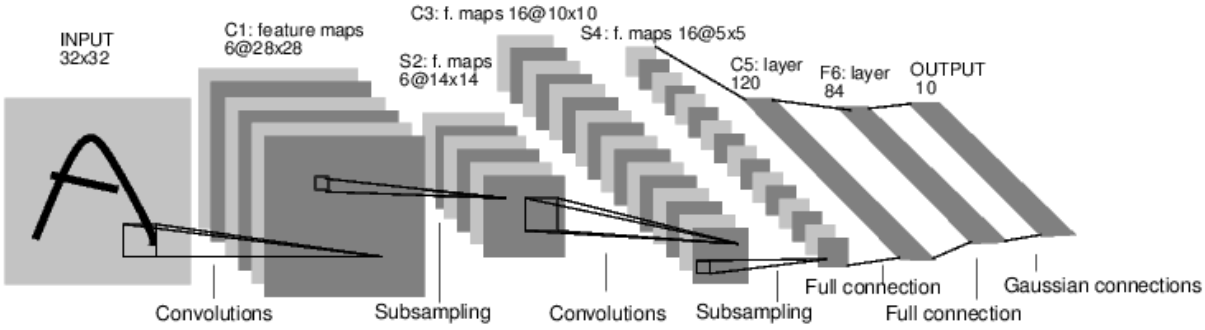

Fig. 7 LeNet-5 Architecture Image source [9]

Using batch size 128, learning rate 3e-4, 2,000 training images and 500 validation images running in 100 epochs, the validation accuracy is upper bounded by 67.6%. We will use 67.6% as our benchmark.

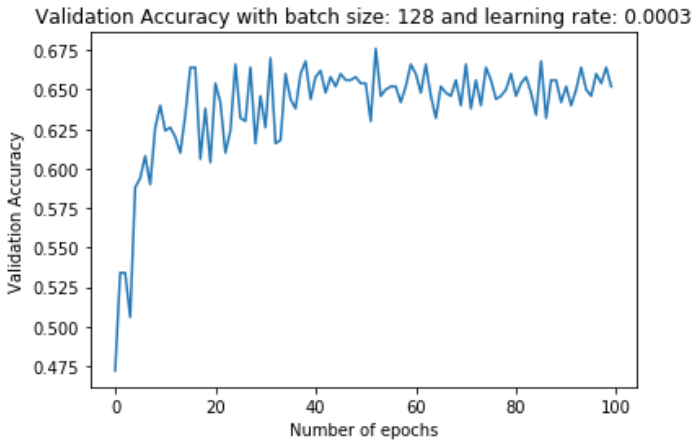

Fig. 8 Benchmark Validation Accuracy Plot

Methodology

- **Data Preprocessing**

  After knowing about the mean height and width pixel values for all 25,000 images, the first step is to resize the image. The second step is to normalize each image on each channel. Since GPU is not available for fast model training, only 2500 images (10% of whole dataset) are used. Within these 2500 images, 2000 images (80%) are used for training and 500 images (20%) are used for validation. 1250 dog images are picked from dog.0.jpg to dog.1249.jpg and cat images are picked in the same fashion. For the benchmark model, the input takes 32 x 32 images, so the images are resized accordingly; besides the 32 x 32 format, 128 x 128 format is also used for further model exploration and experiment. After images are resized, a max-min scaling is implemented to ensure each RGB channel is in a normalized scale. Next, all the data are one-hot encoded, split and shuffled into training and validation dataset. Then, the shuffled and split data are saved as pickle format, which is a powerful algorithm for serializing and de-serializing Python object structure. During training, dataset could be loaded directly from pickle format without the need to regenerate training dataset from scratch.

- **Implementation**

  The CNN model design has 3 convolutional layers and each layer is followed by a max pooling layer. The first fully connected layer takes the output from the third max pooling layer, which has a number of 64*3*3 = 576 parameters. A dropout layer with probability 0.5 of keeping the weights is added to reduce overfitting, since only 10% of the whole dataset are used. The last layer has a sigmoid activation function, since the output is binary and sigmoid is suitable for binary classification.

Fig. 9 CNN Model Architecture

ReLu (Rectified Linear Unit) activation function is used after each convolutional layer, which maps negative value to 0 and maps non-negative value to a linear function with slope 1 starting at origin. The ReLu is a simple threshold comparison function with less expensive operations like sigmoid or tanh. [10]

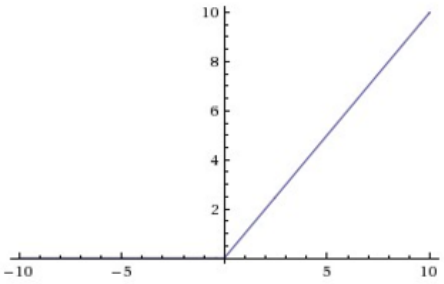

Fig. 10 ReLu activation function [10]

The Adam Optimizer is used to optimize the loss during training, because Adam Optimizer is computationally efficient and suitable for large dataset. Epoch is initialized as 100, and each epoch training time is around 15 second for training 2000 images that are 128 x 128 x 3. Learning rate is initially set as 3e-4 and batch size is set as 128 at first run. The result is shown below, the model is trained for 100 epochs and the validation accuracy reaches around 71.0% after 20 epochs. The curve fluctuates a little bit, so a smaller learning rate should be used for refinement. When implementing the weight initialization in Tensorflow for convolutional and max pooling layers, I found it a little bit challenging, because the stride and filter size must be correctly calculated for code to run without prompting the error message. Fortunately, Tensorflow API is well documented.

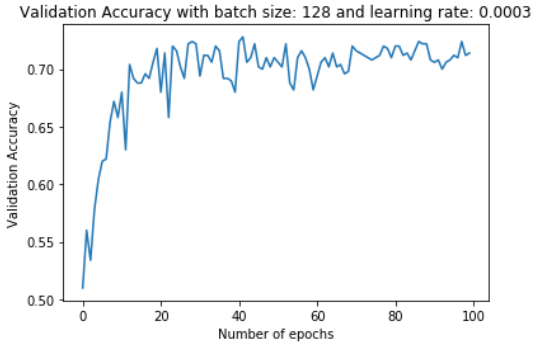

Fig. 11 Validation Accuracy Plot with Learning Rate 3e-4

- **Refinement**

The learning rate is lower to 0.0001 and the result is shown in fig. 12 below for the 50-epoch training trials, the validation accuracy has an increasing trend, thus a 100-epoch trial is conduct to further investigate the lower learning rate performance.

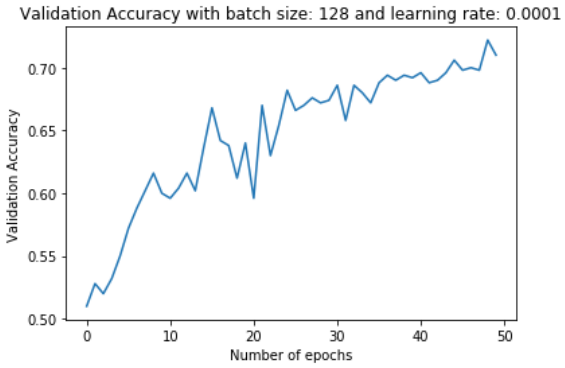

Fig. 12 Validation Accuracy Plot for 50 epochs

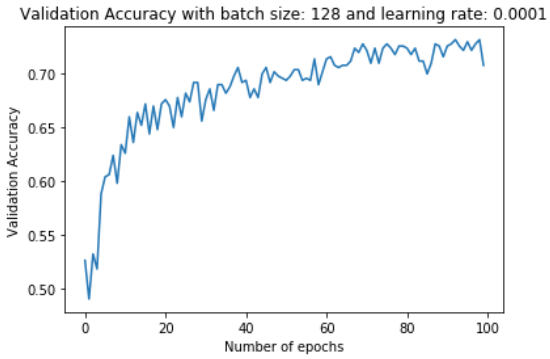

Fig. 13 Validation Accuracy Plot for 100 Epochs

In fig. 13, the validation accuracy approaches 72.8% after 80 epochs, has a 1.8% improvement compared to the performance when learning rate is 0.0003.

For curiosity, learning rate is set as 0.0005 and the validation accuracy approaches 72.3% after 20 epochs. Overall, the model with learning rate 0.0001 yields the best performance.

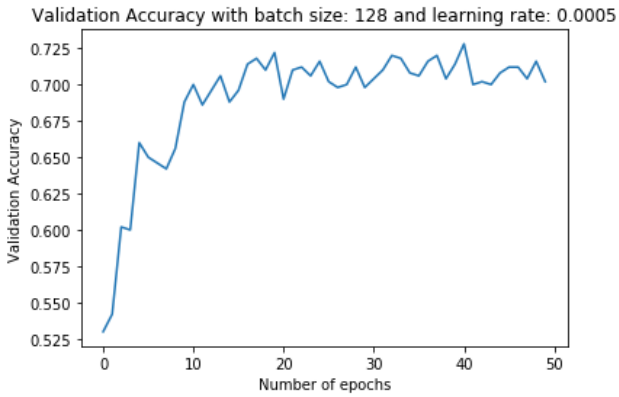

Fig. 14 Validation Accuracy Plot with Learning Rate 5e-4

Results

- ## Model Evaluation and Validation

  As discussed in refinement section, the CNN model with learning rate 1e-4, batch size 128, and epochs 100 achieves 72.8% validation accuracy. The input and model architecture is listed below. The first two convolutional layers have filter depth of 32 and the third convolutional layer has a filter depth of 64. The shape of the filters of the convolutional layers is 3 x 3 and the shape of the max pooling layer is 2 x 2 with strides of 2.

Input

This architecture accepts a 128x128xC image as input, where C is the number of color channels. C is 3 in this case.

Architecture

Layer 1: Convolutional. The output shape should be 63x63x32.

Activation. Relu activation function.

Pooling. The output shape should be 31x31x32.

Layer 2: Convolutional. The output shape should be 29x29x32.

Activation. Relu activation function.

Pooling. The output shape should be 14x14x32.

Layer 3: Convolutional. The output shape should be 6x6x64.

Activation. Relu activation function.

Pooling. The output shape should be 3x3x64.

Flatten. Flatten the output shape of the final pooling layer such that it's 1D instead of 3D.

Layer 4: Fully Connected. This should have 288 outputs.

Activation. Relu

Layer 5: Drotput. This should have 144 outputs.

Layer 6: Fully Connected (Logits). This should have 2 outputs.

Activation. Sigmoid

Fig. 15 Input and Model Architecture

- **Justification**

  Compared to the 67.6% validation accuracy benchmark, the CNN model achieves 72.8% validation accuracy, obtaining 5.2% improvement. This improvement could come from several factors. First of all, the input size for benchmark model is 32 x 32 whereas the size for final model is 128 x 128. Pixel information could be missing during the image resizing stage. Secondly, the benchmark model has two convolutional layers and the first convolutional layer learns 6 filters and the second learns 16 filters. The number of filters learned by convolutional layers in benchmark is less than that of final model. In other words, the feature extraction is deeper in the final model.

Conclusion

- **Free-Form Visualization**

  After the image loading and preprocessing stage, image visualization is performed on one random training sample to show the image has correct shape and label. In one-hot encoding stage, cat is coded as "1" and dog is coded as "0". In the example below, a cat image is shown with label "1" and shape 128 x 128 x 3, which are our desired parameters for training and validation images.

Visualize Image

```
print('Updated Image Shape: {}'.format(X_train[0].shape))
index = random.randint(0, len(X_train))
image = X_train[index].squeeze()

plt.figure(figsize=(2,2))
plt.imshow(image)
print(y_train[index])
```

```
Updated Image Shape: (128, 128, 3)
1
```

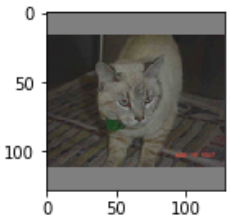

Fig. 16 Image Visualization in Data Preprocessing

- **Reflection**

  This project involves data analysis, data preprocessing, model design, hyperparemeter tuning and patience. I am quite proud of the entire end-to-end pipeline, from image resize, normalization, one-hot encoding, data packaging for loading easiness and efficiency all the way to model training, hyperparameter tuning and achieving decent validation accuracy. However, I found hyperparameter tuning process very challenging, because each training epoch took about 15 seconds and to have a better understanding on the validation accuracy, at least 50 epochs (around 13 mins) were needed. Overall, I am pretty satisfied with the model design and performance, given the fact that the model is only trained with 10% of the whole dataset.

- **Improvement**

  To achieve higher validation accuracy, GPU could be used to assist model training with larger size of dataset. In this project, only 2500 images that are 10% of the total dataset are used. Furthermore, image augmentation could be used to generate random rotated, shifted or flipped images for better feature extraction. Also, transfer learning could be used to re-fine tune the existing famous models, namely VGG, AlexNet, GoogLeNet and ResNet, etc

Reference

[1] http://yann.lecun.com/exdb/publis/pdf/lecun-95b.pdf

[2] https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition/data

[3] http://pages.cs.wisc.edu/~bolo/shipyard/neural/local.html

[4] http://mathworld.wolfram.com/Convolution.html

[5] http://timdettmers.com/2015/03/26/convolution-deep-learning/

[6] http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution

[7] http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/

[8] http://cs231n.github.io/convolutional-networks/

[9] http://eblearn.sourceforge.net/lib/exe/lenet5.png

[10] http://cs231n.github.io/neural-networks-1/