Convolutional Networks for Localization

1 Introduction:

Image recognition has gained a lot of interest more recently which is driven by the demand for more sophisticated algorithms and advances in processing capacity of the computation devices. These algorithms have been integrated in our daily life from cell phone passwords to car-plate recognition. Figure 1 illustrates a general flow of an image recognition algorithm. Basically, these algorithms gets the image file as input and generates probability distribution of different classes in their database. For instance, in this figure, image recognition algorithm receives an image of a car and generates probability distribution as illustrated on the right side of the figure where car category has the highest probability while bicycle or dog has lower probabilities. Main goal of machine learning society is to improve the probability of a car in this scenario (this is mainly referred as true positive) while reducing the probability of bicycle, dog, non-car classes etc (this is mainly referred as false positive) by improving the quality of the image recognition algorithm.

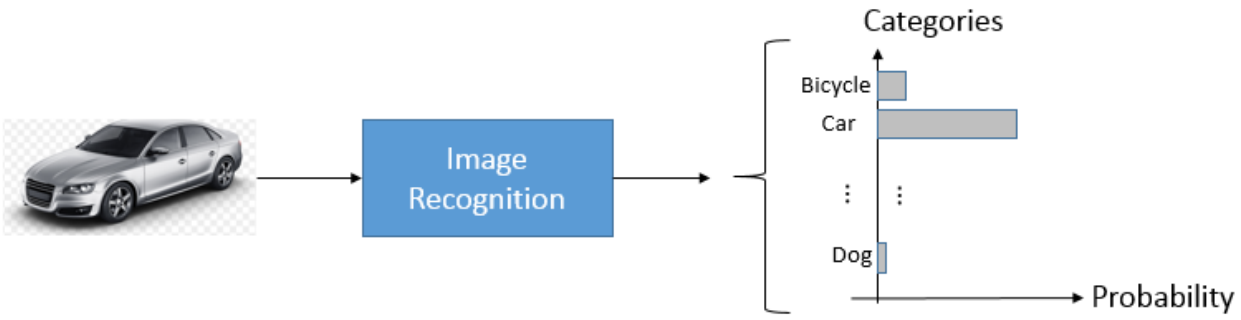

Figure 1 Image Recognition Flow

Image recognition algorithms can be classified into two major categories: Pre-defined image descriptor based and descriptor learning based. Algorithms in pre-defined category uses image processing techniques to generate important observations (or more technically features) and use these observations to detect/recognize an object. They mainly uses the key image points such as small regions around corners of the image to find a corresponding image category in their database that matches these important (or key) points. SIFT [1], SURF[2] and Amazon Image Recognition- GDC [3] are three popular image recognition algorithms based on pre-defined descriptor. On the other hand, descriptor learning based algorithms generate these important observations (or features) while building (or more technically training) the algorithm. They use a large dataset of images and their categories to find these features. With the advancement of processing speed of GPUs and computational elements and availability of large image datasets with their usage become widely accepted by the machine learning community. For instance, Alexnet has shown great improvement on image recognition performance by using Convolution Neural Network (CNN) based image recognition algorithm in [4]. Their success lies in the use of GPUs and large image datasets.

In this study, we look into CNN based image recognition algorithm and their usage in object localization. We have developed a simple image recognition algorithm using CNN and Cifar-10 dataset and use the trained CNN during localization (finding) of objects in larger images. It is worth noting that, we only localize

objects from the category trained in simple CNN. We have applied the convolutional layer of simple CNN to full image. The resulting convolved features are fed into fully connected layer. We use a windowing (2x2 windows) approach at the FC layer to find the location of object. In order to test the approach, we stitch smaller images to construct larger images.

In Seciton 2, we describe the image dataset, in Section 3 we provide a general flow of CNN and our architecture. In Section 4, we briefly explain the preprocessing steps of the input data. In Section 5, we provide the results of our simple CNN and localization algorithm.

2 Dataset

In this study, we have used the publicly available image dataset of Cifar-10 [5]. There are 60K total images in this dataset where 50K is categorized for training and remaining 10K is categorized for test phases. These images are 32x32x3 RGB format (width=height=32 and one dimension for each color component R, G, B). Each image is labeled with their corresponding category such as Ship, Cat, Automobile etc. Figure 2 illustrates some of the images and their labels from the Cifar-10 dataset.

We have converted the category information (labels) to one-hot encoding that is used by the final classification. For instance, we convert label=3 to a vector as [0 0 0 1 0 0 0 0 0 0] (labels starts from 0 to 9). We use this vector format in the classification to calculate training error at every step of the training.

(a) (b) (c) (d)

Figure 2 (a) Automobile – Label=1 (b) Horse-Label=7 (c) Ship-Label=8 (d) Cat-Label=3

2.1 Data Exploration

We have count the number of occurrence of each labels in the training set (50K set). Figure 3 illustrates the distribution of these labels. It is obvious from the figure that the distribution is uniform.

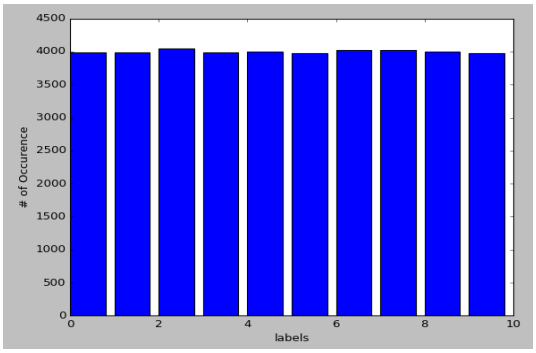

Figure 3 Number of Occurrence for each Category in the Training Set

2.2 Metrics

We have used mean square error (MSE) to observe the performance of the simple CNN estimator during training. At each step of the training we have calculated MSE as follows: $MSE = \frac{1}{N}\sum_{i=0}^{N-1}(y_i - \hat{y}_i)^2$ where N is the number of categories (which is 10), y_i represents true vector (assuming one-hot encoding, it is a vector of zeros except one position which is the true category), $\hat{y}_i$ represents our estimation at the current step.

We have used true-positive and false-positive values to estimate the strength of our localization approach. True positive and false positive is calculated as follows: $TP = \frac{\#\ of\ True\ Classification}{\#\ of\ Position\ with\ object}$, $FP = \frac{False\ Classification}{\#\ Positions\ that\ has\ not\ object}$

3 CNN Architecture

It has been shown that CNN outperforms many image recognition algorithms in ImageNet dataset [4]. CNN algorithm consists of several convolution (CNV) operations followed of the image sequentially which is followed by pooling operation (PL) to generate the neurons feed into fully connected (FC) layer.

Input: of CNV is typically 2D image data with either RGB (red, green, blue) components, or YUV (Luminance and Chrominance) or single Y (Luminance) components. These are typically unprocessed RAW input data, and our dataset is described in Section 2.

Convolutional Layer (CNV Layer): Convolution, in general, is an operation of filtering the input data. In image recognition context, 2D filters are used to process data at different layer to generate the features. These filters are pre-defined and their coefficients are computed during training (which is described later). Basically, training is done through backpropagation algorithm with gradient descent [6]. Figure 4 illustrates a simple 2D convolution operation where the filter is 3x3 (where the filter taps are w1 to w9) that is applied to 2D data (data can be input image or intermediate feature maps).

We have multiple of CNV layers in the CNN image recognition and input to these CNV layers are called Input feature map and output of the CNV layers are called output feature map. At the very first CNV layer, input image with its each component becomes input feature maps with 3 2D data. Figure 5 illustrates high-level input output feature map structure with LxL CNV filters. K convolutional filter output is combined to generate one output feature map.

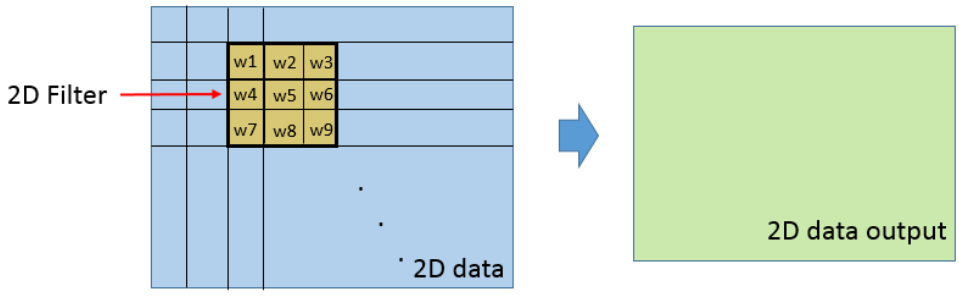

Figure 4 Convolution Operation

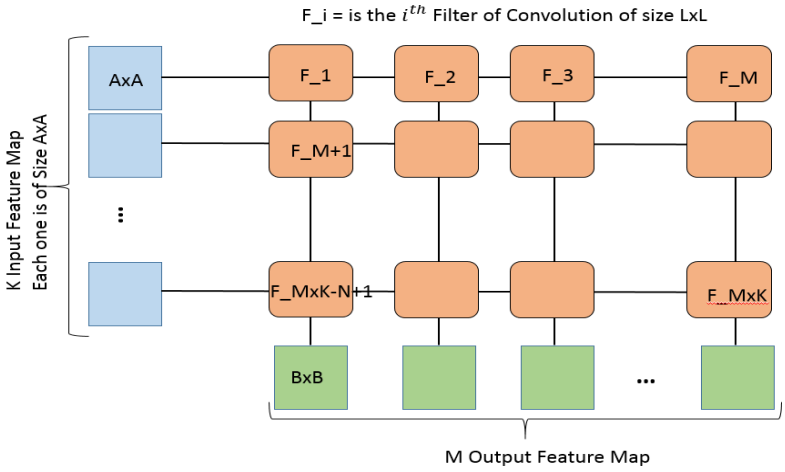

Figure 5 Input and Output Feature Map with K input Feature Map and M Output Feature Map and LxL Convolutional Filter

Pooling Layer (PL): This layer contains both the activation operation that is applied to each elements after convolution and subsampling of the data after activation. Activation operation is typically a nonlinear operation that is applied to each elements of convolution output such as $\max(0, x)$(which is also called Rectifier Linear Unit) or $\frac{1}{1+e^x}$ etc where x is the input data. Activation operation does not change the size of the input. Subsampling is applied after activation that reduced the size of the input to typically ½ at each dimension. Window size that is used during subsampling is also 2D such as 2x2 or 3x3. If the input data size is (W,H) then the size after pooling will be (W/2,H/2) if ½ subsampling is used.

Fully Connected (FC) Layer: This layer will reduce the size of input data to the size of classes that the CNN is trained for by combining output of CNV layer with different weights. Each neuron at the output of the CNV layer will be connected to all other neurons after weighted properly, Similar to CNV layer, weight of these taps in FC layer is found though backpropagation algorithm.

Classification Layer (CL): This is the final layer of the CNN that converts the output of FC to probability of each object being in a certain class. Typically soft-max type of algorithms are used in this layer.

Convolutional Networks are similar to Neural Networks where neurons are replaced by the convolutional operation at the initial layers. We can do such a replacement if we consider focus on a similar features at different positions of the image which is achieved by convolutional filters. At the later stages, neurons in the Neural Network are also used in Fully Connected layer of Convolutional Network.

3.1 Simple CNN Architecture

In our simple CNN, we have used 2 CNV layers followed by 2 PL layers which is followed by 3 FC layers as illustrated in Figure 6.

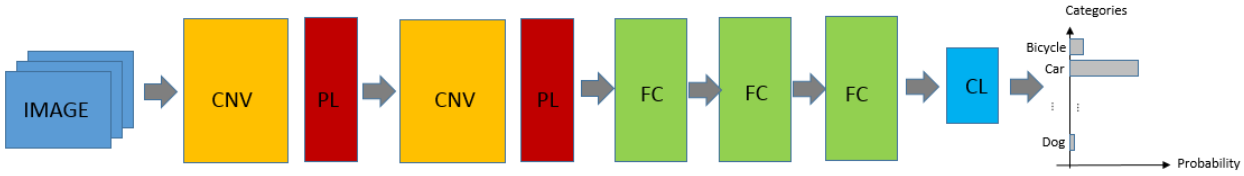

Figure 6 simple CNN architecture with 2CNV and 3 FC layers

We receive 32x32 inputs from Cifar-10 dataset. We have used 5x5x96 filters at the first CNV layers and 5x5x192 filters in the second CNV layers. The first CNV has 3 input feature maps and 96 outputs feature maps. In the Pooling Layers, we apply max(0,x) (Rectifier Linear Unit) after each CNV which is also followed by 2x2 maximum subsampling (For every 2x2 window, maximum value within the window is passed to next layer). Similarly, in the second CNV layer, CNV has 96 input feature maps and generates 192 output feature maps that is followed by the same Polling layer. The change of size in feature maps ca be summarized as follows: 32x32->28x28 (cropping input image@preprocessing) -> 12x12 (after first CNV and pooling) -> 4x4 (after second CNV and pooling). At the input of fully connected layer, we have 4x4x192 feature map that is generated by CNV and Pooling layers which is 3072 samples. In first FC layer we reduce the size to 384 samples, in the second 192 and the last FC reduce the size to 10 samples which is the number of classification categories we have in Cifar-10. The CL layer is a simple SoftMax Classifier.

4 Training CNV and FC Filters

In the training phase of the CNN where we estimate the CNV and FC layer filter taps, we use stochastic gradient descent algorithm [7]. Training is done offline through backward propagation algorithms [8]. We have built the CNV using Tensor Flow [9] as provided part of this submission.

Gradient descent algorithms are very popular in solving quadratic equations though dynamic programming. Stochastic gradient descent algorithm is a variant of gradient descent where we use a small random sample of data at every iteration in order to find optimum weights of the CNN.

One of the important parameters of gradient descent is the learning rate which is typically represented by α. Basically, large α helps gradient descent to converge faster but with decent quality of weights, while small α has lower converge speed but achieves better weight. Thus, we have changed the α parameter along the training with certain decay factor. We start with large α value and reduce it as we iterate over the dataset. This has improved the accuracy by couple of percent compared to a fix α value.

Figure 7 illustrates mean square error (MSE) of the predicted probability and actual labels of the Cifar-10 dataset over 250K many iteration. MSE value reduces as we iterate over the data longer but saturates after certain point.

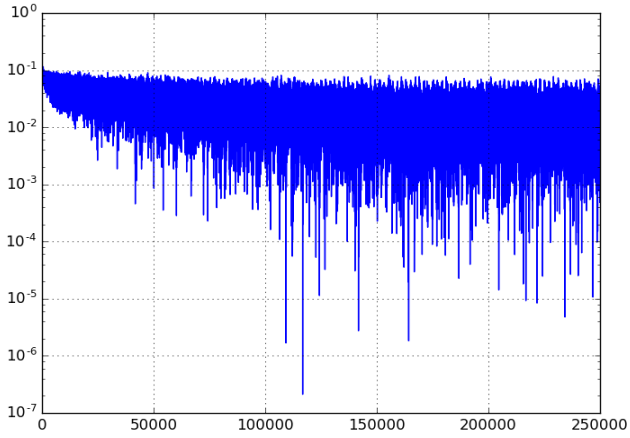

Figure 7 MSE during Training

We have also used regularization of the FC weights to prevent them becoming very large and break the convergence of gradient descent algorithm. We have used L2 regularization provided by Tensor Flow. In addition, we have used drop-out of connections randomly to increase the randomization of each stream from one layer to another. In the drop-out scheme, each data going from one layer t another is randomly dropped with certain probability provided. We have used 60% dropout rate in our architecture.

4.1 Preprocessing Steps

We have preprocessed the RGB data to improve the quality of the CNN algorithm:

(1) Data Normalization: We have removed the mean of the input color components and set their variance to 1. This has improved the results by approximately 1.2%.
(2) We have randomly cropped the 32x32 RGB input images into 28x28 RGB images. We have increased our data set and introduced more randomization to input data pixel. This has improved the simple CNN performance by 0.9%.
(3) For half of the time, we have applied horizontal reflection of the image. This is also for randomization of pixel for the same object class. We have observed another 0.8%
(4) We have used one-hot encoding for the categories to compare in the classification stage. For instance, we convert label=3 to a vector as [0 0 0 1 0 0 0 0 0 0] (labels starts from 0 to 9)

5 CNN and Localization Results

Using the simple CNN architecture described in Section 3, we have achieved 80% true positive classification in Cifar-10 set after 8 hours of training. Next we have applied this CNV to larger images for localization. Figure XX illustrates the flow of the localization architecture. CNV layers of the simple CNN and localization CNN are same which uses 5x5x96 and 5x5x192 filters trained for simple CNN. Similarly, same pooling layers are used. The main difference in the localization CNN is the way FC layer is applied. The size of the convolution layer output is a function of input image size which is larger than 4x4x192. For instance, if the input image size is 64x64x3, then the output of the two CNV layers become 13x13x192. Here, we have applied 4x4x192 tap FC layer to every 4x4x192 blocks in 13x13x192. This is illustrated in both Figure 8and Figure 9.

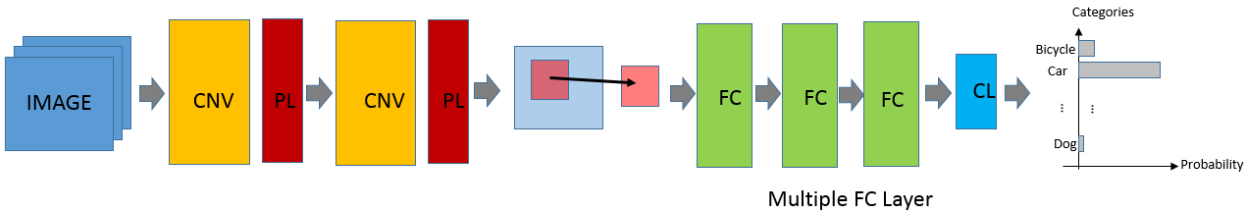

Figure 8 Flow Diagram of Localization

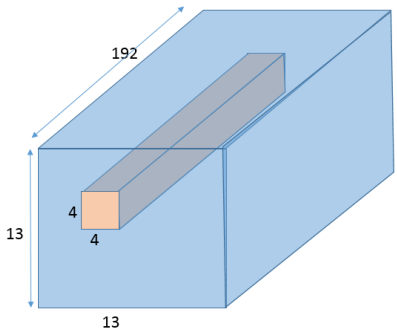

Figure 9 FC in Localization CNN

By applying this search window approach, we detect whether there is an object in the region of interest. We have used a threshold value of 0.98 to decide whether there is a valid object at the location or not. We compare the maximum probability value generated by SoftMax classifier with the threshold value. If the probability is higher than the threshold, we decide that there is an object at that location.

We have stitched images from the test image set of Cifar-10 and created test images for localization as illustrated in Figure 10. Then we have applied the localization CNN to these images and highlight the locations where we decide that there is an object. We also provide category of the object with the symbols used in the right of each image.

We have realized that the true positive rate of the localization is quite high which is approximately 70% on the images we tested; however, the false positive rate is also quite high that is approximately 20%.

We have calculated true positives as follows: If an image at one of the corners is detected correctly within the region, we call this prediction as true positive. For instance, there is a frog image on the top right corner of Figure 10-a. While our localization CNN predict this frog correctly at 6 position within the region it occupies, we consider this as single true positive prediction. Similarly, the ship on the bottom left of the same image is classified correctly 2 times which are considered as true positives. False positive is calculated as the number of cases we decide there is an object at a certain position while there is none at that class. For instance, in Figure 10-a, we see that our localization CNN decides that there are several horses in the middle of the image while there is none; so we consider these as false positive.

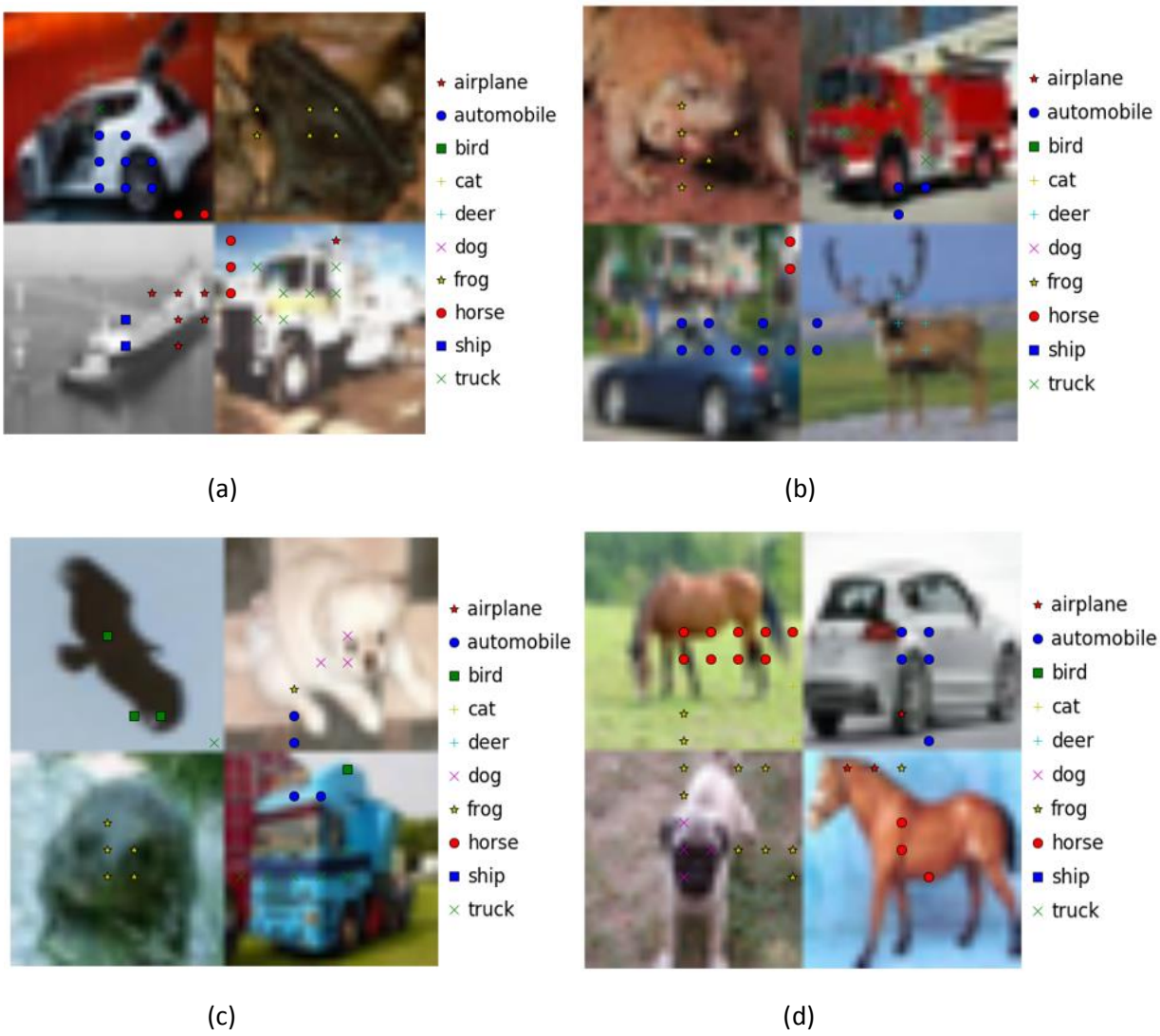

Figure 10 Examples of Object Localization

6 Conclusion

We have trained simple CNN algorithm of size 32x32 using Cifar-10 image set and applied it to larger image sizes for localization. We have shown that localization of the objects with decent quality is possible with the architecture we have used. Our localization algorithm shows 70% true positive and approximately 20% false positive results. We may need to improve the false positive by changing the threshold or clustering the decision points.

Even though the results are promising. One of the next step of this project is to improve the false positive rate for the objects that actually does not exist. This can be improved by using longer training sequence and more randomization. This is a time taking process and could be improved with the help of GPU technology (however, I have run most of these on regular CPU).

7 References

[1] https://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf

[2] http://www.vision.ee.ethz.ch/en/publications/papers/articles/eth_biwi_00517.pdf

[3] https://computervisionblog.wordpress.com/2012/01/29/how-things-work-amazon-flow-app-algorithm/

[4] http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[5] https://www.cs.toronto.edu/~kriz/cifar.html

[6] https://en.wikipedia.org/wiki/Gradient_descent

[7] https://en.wikipedia.org/wiki/Stochastic_gradient_descent

[8] https://en.wikipedia.org/wiki/Backpropagation

[9] https://www.tensorflow.org/