**Neural Network**

陳柏任 著

2015年9月

**Content**

[**1 Introduction 1**](#_Toc433539618)

[**2 What is the Neural Network? 2**](#_Toc433539619)

[**2.1 Neural Networks 2**](#_Toc433539620)

[**2.2 Training for Neural Networks – Back-propagation 5**](#_Toc433539621)

[**3 What is Deep Learning in Neural Networks? 10**](#_Toc433539622)

[**3.1 Introduction 10**](#_Toc433539623)

[**3.2 Types of DNNs 11**](#_Toc433539624)

[**4 What is the Convolutional Neural Network (CNN)? 12**](#_Toc433539625)

[**4.1 Overview 12**](#_Toc433539626)

[**4.2 Convolutional Layer 13**](#_Toc433539627)

[**4.3 Pooling Layer 17**](#_Toc433539628)

[**4.4 Fully-connected Layer 18**](#_Toc433539629)

[**4.5 Overfitting 18**](#_Toc433539630)

[**4.6 Some famous CNNs 19**](#_Toc433539631)

[**5 Toolkit 21**](#_Toc433539632)

[**5.1 The layer 22**](#_Toc433539633)

[**5.2 Use a pre-trained model 24**](#_Toc433539634)

[**6 Applications 25**](#_Toc433539635)

[**7 Conclusion 25**](#_Toc433539636)

[**8 Reference 25**](#_Toc433539637)

1 Introduction

 Nowadays, the neural network is widely used in many fields, such as classification, detection, and so on. As the support vector machine (SVM), neural network is a machine learning technique. It can learn more non-linear features so the accuracy is often very high. But it is very time-consuming for training. In this tutorial, we will introduce the neural network and one of the neural network frameworks in image processing – the convolutional neural network. Also, we will introduce some applications of the convolutional neural network.

2 What is the Neural Network?



Figure 1: An example of the traditional neural network [1]

**2.1 Neural Networks**

Neural networks are the techniques of machine learning. They are just like the neural networks in biology. There are many neurons and many connections between neurons. Figure 1 is an example of the neural network. The white circles represent neurons and the arrows represent the connections between neurons. Note that the connections are directed, therefore we use arrows to represent it. In this section, we will introduce what the neural network is.

First, we need to know what the neuron is. In biology, neurons have inputs, thresholds, and output. If the input voltage is larger than threshold, the neuron will be activated and a signal is transmitted to output. Note that the neuron might have many inputs but there is only one output signal. The operation model of the neuron in machine learning is very like the one in biology. They also have the inputs and outputs. Despite the neuron’s output is connected to many neurons in Figure 1, the value of the outputs are the same. Of course, there are some differences of them. Instead of the threshold, the “neuron” in machine learning use a function to transfer the inputs to the output. There are many choices of the activation function. We often choose it as the sigmoid function *σ*(*x*).

 

The sigmoid function is very similar to the step function, which acts similar to thresholding. When *x* is a large positive number, the output of the sigmoid function is near to 1. When *x* is much smaller than 0, the output is near to zero. We can see these factst in Figure 2. Another good property is that the sigmoid function is continuous and differentiable. So we can apply some mathematics on it.



Figure 2: The sigmoid function [7]

Another difference is the weight. The weights describe that how much each input affects the neuron. That is, we will not just put every inputs into the activation function. The value of activation function’s input is the linear combination of the inputs. The mathematical representation is as follows:

 

where *N* is the amount of the inputs,  are weights of , and *σ*( ) is the activation function. However, there is a problem of it! We reduce the amount of inputs to 1 and change the weight to observe how weights influence on the output. The result is shown in Figure 3(a). One can see that 0 can be viewed as the threshold to determine whether the output is near to 0 nor near to 1. However, how do we modify the model if we want to change the threshold to a value other than 0? In this case, we add a bias *θ* to achieve that so that we can shift the sigmoid function. The result of the sigmoid function with bias is shown in Figure 3(b). So the new relation is revised as follows:

 

The parameter *θ* is the bias and other notations are the same as above. And  are parameters that are needed to be learned. That is how neuron works.





Figure 3: (a) The result of the sigmoid function with different weights of input but without bias. (b) The result of the sigmoid function with different weights of bias.[8]

If we connect many neurons, the neural networks appear. Let’s look back to Figure 1. The colored rectangles are consisted of many neurons and we call these rectangle layers. The layer contains one or many neurons and these neurons will not connect to each other. We often call the first layer the input layer and we call the last layer the output layer. The layers between the input layer and output layer are called the hidden layers. We often connect every neuron in previous layer to every neurons in the next layer. We call it full-connected. Figure 1 is a good example to show that.

There are many neurons in the neural networks. Each neuron has many weights. Therefore, the goal is that we should find the proper weights to fit the data. That is training the networks so that the outputs are close to the desired outputs. In the next section, we will introduce the method of training – backpropagation.

**2.2 Training for Neural Networks – Back-propagation**

**2.2.1 The Sketch of the Back-propagation Algorithm**

The backpropagation algorithm is briefly described as follows:

Phase 1: Propagation —

 This phase contain two steps, forward propagation and back propagation. The forward propagation step is to input the training data to the neural networks and calculate the output. Then we will get the error of this output from the groundtruth of the training data. We can back propagate the error to each neuron in each layer. That is the back propagation step.

Phase 2: Update the weight —

We update the values of weights of the neuron according to the error.

Repeat the phases 1 and 2 until the error is minimum. Then we finish the training. The mathematical detail will be described as follows.

**2.2.2 How to Back-propagate?**

Before introducing back-propagation, we define some notation for convenience. We use  to represent the input to node *j* of layer *l* and  for the weight from node *i* of layer *l*-1 to node *j* of layer *l*. is represented to bias of node j of layer *l*. Similarly,  represents the output of node *j* of layer *l* and  represents the desired output, that is the ground truth of the training data. is an activation function. We use the sigmoid function here.

In order to get the minimum error, we define the cost function:

 

where *x* is the training data input and *dj* is the desired output. *L* is the total number of the layers, and  is the output of the neural network corresponding to the input *x*. In order to make the derivative easier, we multiply the summation in (4) by a constant 1/2.

Our goal is to find the minimum. We first compute partial derivatives of the cost function with respect to any weight. That is,

 

Now, we consider two cases: The node is an output node or it is in a hidden layer. In Output layer, we first compute the derivative of the difference of the ground truth and the output. That is,

 

The last equation is based on the chain rule. The node *k* is the only one with weight  so other terms will get zero after applying the differentiation. And  is the output of activation function (sigmoid function here). So, the equation becomes:

 

where  is the linear combination of all inputs of the node *j* in the layer *L* with the weights. As mentioned above, the sigmoid function is derivative. The derivative of sigmoid function also has a very special form:

 

Therefore, the partial derivative function becomes:

 

The last term is based on chain rule. Remember that . Thus, (9) becomes:

 

Note that  is related to  and not related to  where *i* ≠ *j*. By this equation, we find the relation between *j* node of *L*-1 layer and the *k* node of *L* layer. We define the new notation

 

to represent the *k* node of the *L* layer term. So the equation becomes:

 

Then, we consider the *l* hidden layer node. We first consider the layer *L*-1 which is just previous to the output layer. Similarly, we need to apply partial derivative over weights on the cost function. But the weights are for hidden layer nodes this time.

 

Note that there is a summation over *k* in *L* layer. It is because that the varying of the weights  for the hidden layer node will affect the neural network output . Again, we apply chain rule and get:

 

Then, we modify the last derivative term by chain rule:

 

The 2nd line of (14) comes from the fact that the input of  is a linear combination of the outputs of the node of the previous layer with the weight. Now, we find that the derivative term is not related to *k* node of the *L* layer. Again, we simplify the derivative term based on the chain rule:

 

Again, we can define all terms besides the  to be . Therefore the equation becomes:

 

Now, we put the result of these two cases together:

 

Then, we apply the similar process to the bias term. For example we calculate the partial derivation on the bias of the *k* node in the last layer *L*  and get:

 

Because of , the last term is 1. The equation can be update to:

 

no matter which output it is. So the gradient of the cost function over bias is:

 

This relation holds for any layer *l* we are concerned with.

Now, we have done all mathematical derivation. Then, we start to describe the backpropagation algorithm:

**The Backpropagation Algorithm**

1. Run the network forward with your input data to get the network output
2. For each output node, compute



1. For each hidden node, compute



1. Update the weights and biases as follows:
Given



Apply



The parameter *η* in the algorithm is called the learning rate. We will repeat this algorithm until the error is minimum or below some threshold. Then, we finish the training process.

3 What is Deep Learning in Neural Networks?

**3.1 Introduction**

Because the computation efficiency is rapidly improved, deep learning in neural network becomes more and more popular recently. Briefly speaking, deep learning in neural network (or deep neural networks, DNNs) is the neural network that has many hidden layers. But the deeper neural network makes training more difficult. We will briefly introduce two different type of DNNs here.

**3.2 Types of DNNs**

There are two type of DNNs, the feedforward DNN and the recurrent DNN. The feedforward DNN is like Figure 1. The neurons in the hidden layer *n*+1 are all connected to the neurons in the hidden layer *n*. Because of containing many hidden layers, we say that feedforward neural networks are deep in space, which means that there is no any cyclic path in the neural networks. This type is very common in the applications of neural networks.

Another type is recurrent neural networks (RNNs). Unlike the feedforward neural networks, RNNs have at least one cyclic path. We can see it in the right of the Figure 4.



Figure 4: The feedforward neural networks(left) and the recurrent neural networks(right) [4]

Until now, we may have a question: How do we use the RNNs? Because of the cyclic path, we cannot treat the neural networks as the feedforward neural networks. It may cause the infinity loop. An important idea is that we unfold the RNNs to different time stages, as in Figure 5. For example, on the left of the figure 5, there is a node A connects to node B and a cyclic to node A itself. We do not handle the cyclic path and the connections at the same time. We assume that the output of the node A in the time n as the input of the node B and the node A in the time *n*+1. It is shown on the right of the figure 5. So, in addition to deep in space property in feedforward neural networks, RNNs are also deep in time. So the RNNs can model the dynamical systems. For example, the RNNs often used in voice identification or capturing the text from the image. The famous way to train the RNNs is backpropagation though time (BPTT). There are many RNNs structures, traditional RNNs, Bidirectional RNNs [5] and Long-short term memory (LSTM) RNNs [6]. The detail of these structures was described in [4~6].



B

B

A

A

A

B

B

A

Figure 5: The model about how we train the RNNs [4]



Figure 6: The layers in CNNs are 3-dimension [1]

4 What is the Convolutional Neural Network (CNN)?

**4.1 Overview**

One of the special feedforward neural networks is the convolutional neural network. In the traditional neural network, the neurons of every layer are one-dimensional. In the convolutional neural network, we often use it in the image processing so we can assume that the layers are 3-dimension, which are height, width and depth. We show this in Figure 6. The CNN has two important concepts, locally connected and parameters sharing. These concepts reduced the amount of parameters which should be trained.

There are three main types of layers to build CNN architectures: (1) the convolutional layer, (2) the pooling layer, and (3) the fully-connected layer. The **fully-connected** layer is just like the **regular** neural networks. And the **convolutional** layer can be considered as **performing convolution many times** on the previous layer. The **pooling** layer can be though as **downsampling** by the maximum of each  block of the previous layer. We stack these three layers to construct the full CNN architecture.



Figure 7: An example of the structure of the CNNs – LeNet-5 [3]

**4.2 Convolutional Layer**

**4.2.1 Locally Connected Network**

In image processing, the information of an image is the pixel. But if we use the full connected network like before, we will get too many parameters. For example, a  RGB image will have  parameters per neuron. So if we use the neural network architecture in Figure 1, we need over 3 million parameters. The large number of the parameters makes the whole process very slow and would lead to overfitting.

After some investigation of images and optical systems, we know that the features in an image are usually local and one just notice the low-level features first in the optical system. So we can reduce the full connected network to the locally connected network. It is one of the main ideas in the CNN.

 

5

Figure 8: An example of the convolutional layer [1]

Just like the mostly image processing do, we can locally connect a square block to a neuron. The block size can be  or  for instance. The physical meaning of the block is like a feature window in some image processing tasks. By doing so, the number of parameters can be reduced to very small but it will not lower the performance. In order to extract more features, we can connect the same block to another neuron. The **depth** in the layers is how many times we connect the same area to different neuron. For example, we connect the same area to 5 different neurons. So, the depth is five in the new layer in the Figure 8 above.

Note that the connectivity is local in space and full in depth. That is, we connect all depth information (for example, RGB 3 channels) to next neuron but we just connect local information in height and width. So there might be  parameters in the Figure 8 for the neuron after the blue layer if we use the  window. The first and second variables are height and width of window size and the third variable is depth of the layer.

We will move the window inside the image and make the next layer also have height and width and be a two-dimensional one. For example if we move the window 1 pixel each time, or stride 1, in a  image and the window size is  there are  neurons in the next layer. We might find that the size is decreased (from 32 to 28). So in order to preserve the size, we add zero pad to the border in general. Back to the example above, if we pad with 2 pixels, there are  neurons in the next layer which keep the size in height and width. We can discuss the stride 1 case. If we use window size *w*, we need to zero-pad with  pixels. Therefore, we do not need to figure out whether the size is still available in another layer. Also, we find that the neural networks with zero-pad work better than the ones without zero-pad. The border information will not affect so much because those values are only used once.

In the next part, we will discuss the “stride”. The stride means the shifting distance of the window each time. For example, suppose that the stride is 2 and the fist window covers the region of *x* ∈ [1, *m*]. Then the second window covers the region of *x* ∈ [3, *m*+2] and the 3rd window covers the region of *x* ∈ [5, *m*+4].

Let us consider an example, if we use stride 1 and window size  in  image without zero-pad, there are  neurons in the next layer. If we change the stride 1 to stride 2 and others remain the same, there are  neurons in the next layer. We can conclude that if we use stride s, window size  in  image, there are  neurons in the next layer. What if we use stride 3 and others remain the same? We will get , which is not the integer, in width. So stride 3 is not available because we cannot get a complete block in some neurons.

**4.2.2 Parameters Sharing**

Let us look back to the example in the Figure 8. In that example, there are  neurons in the next layer with stride 1, window size  and with zero-pad, and the depth is 5. Each neuron has  parameters (or weights). So there are  parameters in the next layer. The idea is that we can share the parameters in each depth! That is  neurons in each depth use the same parameters. So there are only  parameters in each depth and  parameters in total. It greatly decreases the amount of parameters. By doing so, the neurons in each depth in the next layer is just like applying convolution to the image. And the learning process is like learning the convolution kernel. This is why this neural networks is called ‘Convolutional’ neural networks.

**4.2.3 Activation Function**

In the traditional neuron model, we often use the sigmoid function for the activation function. Some people propose other choices for the activation function. One of them is Rectified Linear Units (ReLUs). The function is . Krizhevsky et al. [2] compared the performances of using the ReLUs function and the sigmoid function as the activation function in CNNs. They found that the model with ReLUs needs less iteration time while reaching the same training error rate. We can see the result in Figure 9, the solid line is the model using ReLUs and the dashed line is to use the sigmoid function. So more and more CNNs models use ReLUs for the activation function in the neuron model recently.



Figure 9: The comparison of the ReLU and the sigmoid function [2].

**4.3 Pooling Layer**

Although we use locally connected networks and parameter sharing, there are still many parameters in the neural networks. Compared with a relatively small dataset, it might cause overfitting. So we often insert the pooling layers to the networks. It can progressively reduce the amount of parameters and hence the computation time in the networks. The pooling layer applies **downsampling** to the previous layer by using **the max function**. It operates independently on each depth of the previous layer. It means that the depth of the next layer is the same as that of the previous layer. Also, we can set the amount of pixels when we move the window, or stride, as the convolutional layer. For example, in Figure 10, the window size of  and the stride of 2 are used. At each window, we get the maximum to represent the value of the next layer.



Figure 10: A simple example of the pooling layer [1]

Note that there are two type of pooling layers. If the window size equal to stride, it is **traditional pooling**. If the window size is larger than the stride, we call it **overlapping pooling**. In practice, we often use the window size  and the stride size 2 in the traditional pooling and use the window size  and the stride size 2 in the overlapping pooling because the bigger parameters will be very destructive.

In additional to max pooling, we can use other functions. For example, we can calculate the average of the window to represent the value of the next layer, which is called average pooling, and use *L*2-norm, which is called *L*2-norm pooling.

**4.4 Fully-connected Layer**

The third layer is the fully-connected layer. This layer is just like the traditional neural network. We connect all the neurons in the previous layer to a neuron in next layer and the final layer is the output. In Figure 7, the F6 layer is a fully-connected layer and there are ten neurons in the output layer.

**4.5 Overfitting**

Now, we know that the structures of the CNNs are very huge. They have many neurons and connections. Of course, they have many weights needed to train. But the amount of training data are not often enough to train the huge network. It may cause some overfitting problem so that the performance might be worse. We need some technique to prevent this problem. There are many ways to do this.

One type of them is reducing the weights in training. Dropout is a famous technique to achieve it. Dropout sets the output of each hidden neuron to zero with probability 0.5. So these neurons will not contribute to the feedforward step and will not participate in backpropagation. For different inputs, the neural network is sampled a different structure. But in test step, we use all the neurons but multiply their outputs by 0.5. This technique reduces the amount of neurons in training.

Another type of them is data augmentation. We can mirror the images, upside-down the images, sample the images and so on. These ways will increase the number of training data. So it can prevent the overfitting.

**4.6 Some famous CNNs**

There are some famous CNN architectures. Some experiments show that they have better performance. So we sometimes use them instead of design by ourselves. We will introduce some of it.

**4.6.1 AlexNet**



Figure 11: The structure of AlexNet [2]

Alex et al developed this network in 2012. And it is widely used nowadays. The structure is shown in Figure 11. There are five convolutional and three fully-connected layers. We may find that the structure in AlexNet is divided into two blocks. That is because the authors use two GPUs to train the data in parallel. This network is used in large-scale object classification. The last layer has 1000 neurons. That is because the architecture was originally designed for identifying 1000 objects. People will replace the last layer depends on their work. The authors did many experiments to get the best result. So the performance of this structure is very stable and this net is widely used in many applications.

**4.6.2 VGGNet**

VGGNet [10] was developed in 2014 and it won the ILSVRC-2014 competition. It is more powerful but very deep. It has 16~19 layers. The structure is described in Figure 12. They designed five structures. After some experiments, the D and E are the best structure. The performance of E is a little bit better than B. But the parameters in E are larger than D. So we can choose one of them based on what we need. The characteristic of VGGNet is that it applied multiple convolutional layers with small window sizes instead of a convolutional layer with large window size followed by pooling layer. It makes the network more flexible.



Figure 12: The structure of VGGNet [10]

5 Toolkit

We often use the toolkit of Caffe [9] developed by University of California for the CNN. It works on Linux (the author has tested on Ubuntu and Red Hat) and OS X. And the code environment is Python. Readers can install the environment at [15]. It supports the CUDA GPU machine. It is very time-consuming in CNNs training. Therefore the speed issue is very important. According to the author, Caffe can process over 60M images per day with a single NVIDIA K40 GPU. It is very fast! (But you need NVIDIA display card.) So Caffe is widely used in the CNN for vision. We will not introduce this tool very detail. We will just introduce how to set the layers in Caffe and how to use a pre-trained model.

**5.1 The layer**

The layer setting is written in the .prototxt file. We will show how to use in the following. Readers can see <http://caffe.berkeleyvision.org/tutorial/layers.html> for more detail.

**5.1.1 Basic information**

name: "Places205-CNN" // the name of this network

input: "data" // the name of input

input\_dim: 64 // batch size ( how many images will input in one time)

input\_dim: 3 // depth of the image

input\_dim: 227 // width of the image

input\_dim: 227 // height of the image

**5.1.2 Convolutional layer setting**

layers {

 layer {

 name: "conv1" # Layer name

 type: "conv" # tell Caffe which type it is. Can be "conv" "pool" "relu" ...

 num\_output: 96 # number of output, that is depth of the next layer

 kernelsize: 11 # convolutional filters size 11\*11

 stride: 4 # how many pixels does this filter shift

 weight\_filler {

 type: "gaussian" # Initialize the filters

 std: 0.01 # Initialize the standard deviation (default mean is 0)

 }

 bias\_filler {

 type: "constant" # Initialize the biases to 0

 value: 0. }

 blobs\_lr: 1. # learning rate for the filters

 blobs\_lr: 2. # learning rate for the biases

 weight\_decay: 1. # decay multipliers for the filters

 weight\_decay: 0. # decay multipliers for the biases

 }

 bottom: "data" # previous layer name

 top: "conv1"

}

**5.1.2 Pooling layer setting**

layer {

 name: "pool1"

 type: "Pooling"

 bottom: "conv1"

 top: "pool1"

 pooling\_param {

 pool: MAX # the type is max. Can be MAX, AVE, or STOCHASTIC

 kernel\_size: 3 # pool over a 3x3 region

 stride: 2 # shift two pixels (in the bottom blob) between pooling regions

 }

}

**5.1.3 Activation function**

layers {

 layer {

 name: "relu1"

 type: "relu" # use Re-Lu function

 }

 bottom: "conv1"

 top: "conv1"

}

**5.1.4 Dropout setting**

layers {

 layer {

 name: "drop6"

 type: "dropout"

 dropout\_ratio: 0.5 # the probability of dropout

 }

 bottom: "fc6"

 top: "fc6"

}

**5.1.5 Fully-connected layer**

layers {

 layer {

 name: "fc7"

 type: "innerproduct" # fully-connected layer

 num\_output: 4096

 weight\_filler {

 type: "gaussian"

 std: 0.005

 }

 bias\_filler {

 type: "constant"

 value: 1.

 }

 blobs\_lr: 1.

 blobs\_lr: 2.

 weight\_decay: 1.

 weight\_decay: 0.

 }

 bottom: "fc6"

 top: "fc7"

}

**5.2 Use a pre-trained model**

Training the CNN is very time-consuming. Fortunately, there are many works using the Caffe toolkit. They might publish their pre-trained models in the Internet. We can write a python code to use these pre-trained models, instead of training the models again.

caffe.set\_mode\_gpu() # use gpu mode to speed up

net = caffe.Classifier(MODEL\_FILE, # the structure file (.prototxt)

 PRETRAINED, # the pretrained parameter file(.caffemodel)

 MEAN, # the mean of the pretrained data

 raw\_scale=255,

 image\_dims=(256,256)) # image size

prediction = net.predict([input\_image]) # feedforward to get the prediction

print 'predicted class:', prediction[0].argmax() # get the maximum probability class

6 Applications

The CNNs are used in many areas. For example, the object classification [2][10], object detection [11], speech recognition [12], action recognition [13][14] and so on. And there are more and more works using the CNNs. Readers can read these reference paper for more detail.

7 Conclusion

We introduce the CNNs. Based on parameter sharing and locally connected layers, CNNs are successfully used in many image process work and also get good performance. But the CNNs training is very time-consuming. Also we need to notice the overfitting problem because there are a large number of parameters. So we often used in large scale tasks. Then, we introduce the Caffe, a powerful toolkit for the CNNs. To get more information, readers can go to the Caffe’s website. There are some sample codes, tutorial for training and the pre-trained model we can use. In the end, we should also know that the structures of the CNNs are not theoretical. Most of them are based on many experiments to get the best result.

8 Reference

[1] Fei-Fei Li & Andrej Karpathy, “Stanford class: Convolutional Neural Networks for Visual Recognition” Jan. 2015. [Online]. Available: http://cs231n.stanford.edu/syllabus.html [Accessed: Sept. 28, 2015]

[2] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).

[3] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

[4] Jaeger, Herbert. *Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the" echo state network" approach*. GMD-Forschungszentrum Informationstechnik, 2002.

[5] Schuster, Mike, and Kuldip K. Paliwal. "Bidirectional recurrent neural networks." *Signal Processing, IEEE Transactions on* 45.11 (1997): 2673-2681.

[6] Gers, Felix. "Long short-term memory in recurrent neural networks."*Unpublished PhD dissertation, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland* (2001).

[7] David Poole, “Artificial intelligence – Foundations of computational agents”. 2010 [Online]. Available: http://artint.info/html/ArtInt\_180.html [Accessed: Sept. 28, 2015].

[8] Nate Kohl, “Stackoverflow topic: Role of Bias in Neural Networks” Mar. 23, 2010. [Online]. Available: http://stackoverflow.com/questions/2480650/role-of-bias-in-neural-networks [Accessed: Sept. 28, 2015]

[9] BLVC, “The toolkit for the CNN – Caffe” 2014. [Online]. Available: http://caffe.berkeleyvision.org/ [Accessed: Sept. 28, 2015]

[10] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." *arXiv preprint arXiv:1409.1556* (2014).

[11] Ross Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik. "Rich feature hierarchies for accurate object detection and semantic segmentation." *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*. IEEE, 2014.

[12] Abdel-Hamid, O., Mohamed, A. R., Jiang, H., & Penn, G. "Applying convolutional neural networks concepts to hybrid NN-HMM model for speech recognition." *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*. IEEE, 2012.

[13] Ji, Shuiwang, Wei Xu, Ming Yang, Kai Yu. "3D convolutional neural networks for human action recognition." *Pattern Analysis and Machine Intelligence, IEEE Transactions on*35.1 (2013): 221-231.

[14] Simonyan, Karen, and Andrew Zisserman. "Two-stream convolutional networks for action recognition in videos." *Advances in Neural Information Processing Systems*. 2014.

[15] Python, Mar. 23, 2011 [Online]. Available: https://www.python.org. [Accessed: Sept. 28, 2015].