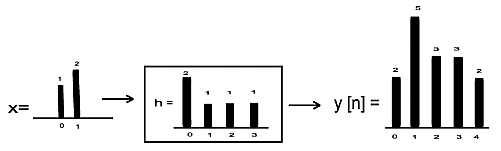
**Convolution Neural Network**

Convolution layers are used to extract the features from input training samples. Each convolution layer has a set of filters that helps in feature extraction. In general, as the depth of CNN model increases, complexity of features learnt by convolution layers increases. For example, first convolution layer captures simple features while the last convolution layer captures complex features of training samples.

1 line 🡪 2 line 🡪 Square 🡪 Combination of Squares 🡪 Combinations of squares and other shapes

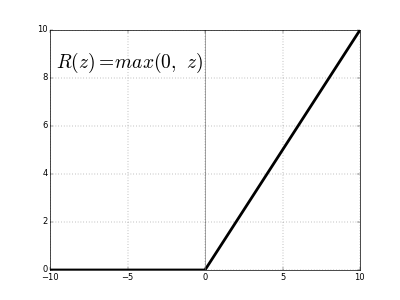
Features are extracted by taking the convolution of portion of data sample under consideration. The amount of data portion that the filter traverses each time is proportional to the stride length and padding value. Data samples may/ may not be subjected to zero padding before convolution.



Convolution of a Signal

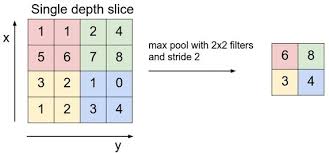
The output of a filter is then passed through an activation unit called ReLU (Rectified Linear Unit). This unit converts the data into it’s non-linear form. The output of ReLU is clipped to zero only if the result is negative else the output of convolution layer is retained.

Sigmoid units are not preferred as activation unit because of vanishing gradient problem. If the depth of CNN is large, then by the time the gradient found at the input layer traverses to the output layer, it’s value would diminish largely. This results in the overall output of the network varying marginally. This, in turn, results in slow/no convergence. To avoid such a situation, ReLU is preferred.



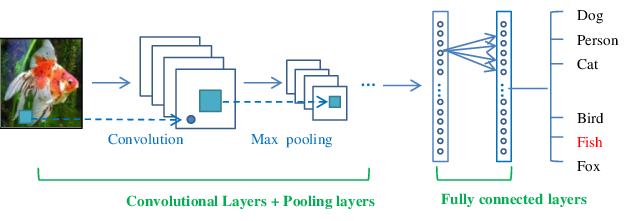
Output of ReLU

The output of ReLU is then passed through a pooling layer. Pooling layer remove any redundant features that’s captured during convolution. Thus, this layer reduces the size of data sample. The principle behind pooling is that it assumes that adjacent values of image pixels are nearly identical. The average/minimum/maximum of four adjacent pixel values are used to carry out pooling. In general, size of input image is reduced by half with help of a 2\*2 filter. The input data may/ may not be subjected to zero padding before pooling.



Max Pooling

This process of passing data through convolution and pooling layer successively is repeated per the design of CNN model. For learning purpose, this process is repeated 2-4 times. The output from successive convolution and pooling layer is then passed through a fully connected neural network layer. It’s functioning is like that of a multi-layer neural network. Here, each neuron unit acts as feature map that carries information about a unit.

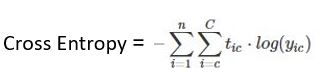


Design of Convolution Neural Network

Dropout layer is used to make the CNN model robust against noise. These layers are generally introduced between 2 fully connected neural network layers. They temporarily cut a portion of data flowing between two fully connected layers. This is equivalent to making the model learn to classify accurately in presence of noise. Thus, chances of model classifying inaccurately because of overfitting is reduced.

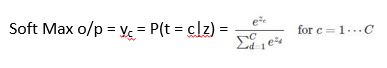
The output of CNN model is calculated using SoftMax function. SoftMax is preferred as it gives the probability of outputs for different classes rather than just >= 0.5 in the case of sigmoid output. The usage of SoftMax function to find output results based on the highest probability of class results in an increase in accuracy the of output.

Cross entropy is used to measure the performance of the system. They are calculated with help of a SoftMax function. The advantage here is that the SoftMax output is the trace of the elements corresponding to the class that we know that the output belongs too. This, in general, saves the computation time.



Here,

* Tic = Target Output
* yic = Soft Max Output
* C= = Number of Classes
* N = Number of Data Samples



Here,

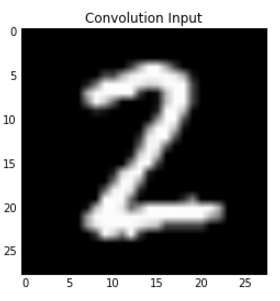
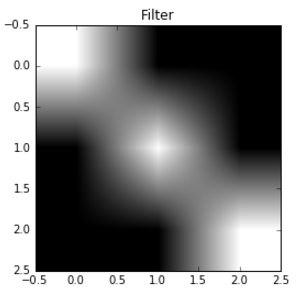
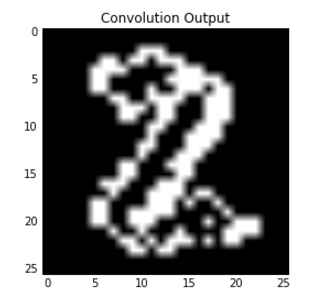
* yc = probability of current output belonging to class c
* Numerator = exponential of weighted sum o/p of class c
* Denominator = sum of exponential of weighted sum o/p of classes 1 to C

**Overview of CNN Layers**

|  |  |  |  |
| --- | --- | --- | --- |
| Layer | Purpose | Implementation | Reason for Working |
| Convolution | Extract features from i/p data sample | y(t) = x(t)\*h(t-T) | Small size filters trace the entire data sample at a faster rate. Hence, they learn from the mapping with pictures efficiently. |
| ReLU | Convert data to non-linear form | y = y, when i/p > 0  0, otherwise | Removes problem of vanishing gradient descent |
| Pooling | Remove redundant features  Make model invariant to translation, rotation and scaling | y = max of 4 adjacent  feature values | Features next to each other have similar values |
| Fully Connected Neural Network | Classification of i/p data | Back Propagation Algorithm | Each neuron acts as a feature map |
| Dropout | Make model robust to noise | Temporarily cut the flow of small portion of data b/w 2 fully connected layers | Reduces overfitting by making the model less complex |
| Regularization | Make model robust to noise | Penalizes cost function and weight updates for every wrong prediction | Reduces overfitting by making the model less complex |
| Multiple Convolution Layer | Extract high level/more complex features | y(t) = x(t)\*h(t-T) | Lesser number of filters enable faster extraction of features from data set |

**MNIST**

Convolution Process

The above figures depict the output of filter and convoluted images after being subjected to convolution. The output of convolution says that the filter is good at detecting diagonal edges. This is shown by distinct edges in image of picture along the diagonal. The flat edges of image are not so distinct as the filter is designed to detect diagonal edges.

In our experiment, we have assumed threshold of filter as +2. Black pixel is represented as 0 and white pixel is represented as 1. The 3\*3 filter values used are [[1,0,0], [0,1,0], [0,0,1]]. Thus, when convolution output is greater than threshold of 2 defined, we consider output of that convolution as 1 else 0.

*Features*

* Filter Pixel Size: 3 \* 3
* Input Pixel Size: 32 \* 32
* Output Pixel Size: 28 \* 28

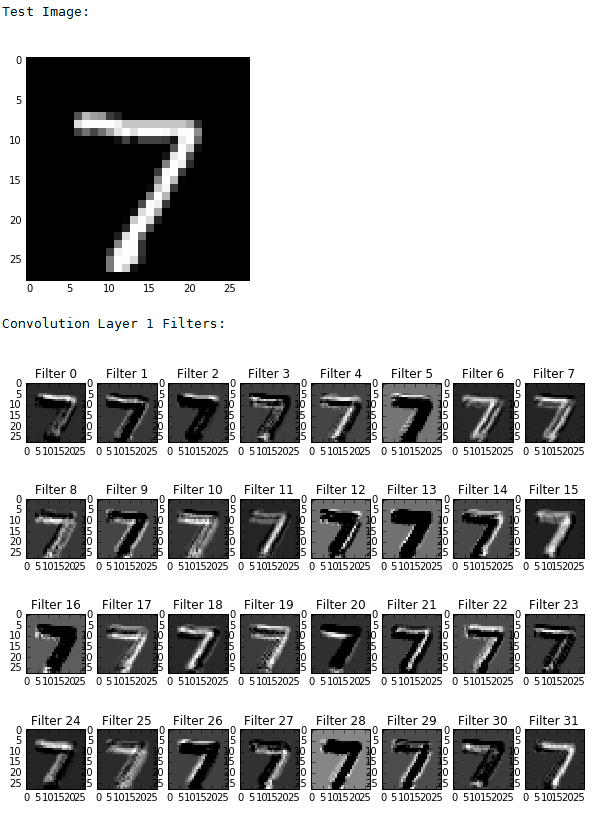


Figure Convolution Layer 1 Output

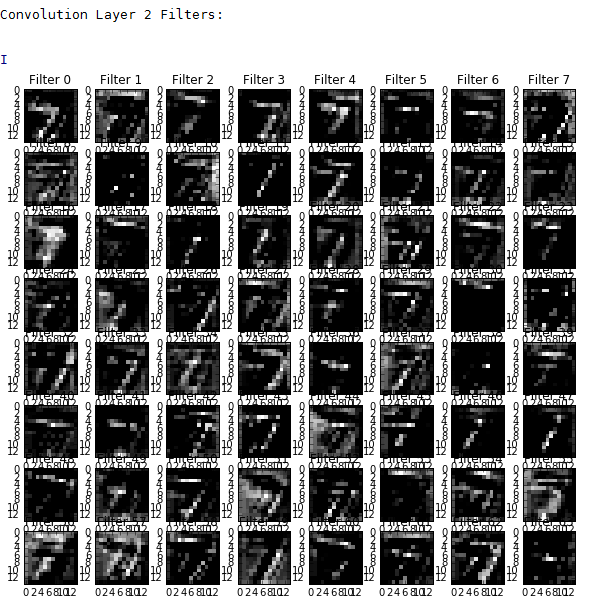


Figure Convolution Layer 2 Output

**Note:**

The convolution outputs represent how that image activates the neurons of the convolutional layers. Each filter learns to activate optimally for different features of the image.

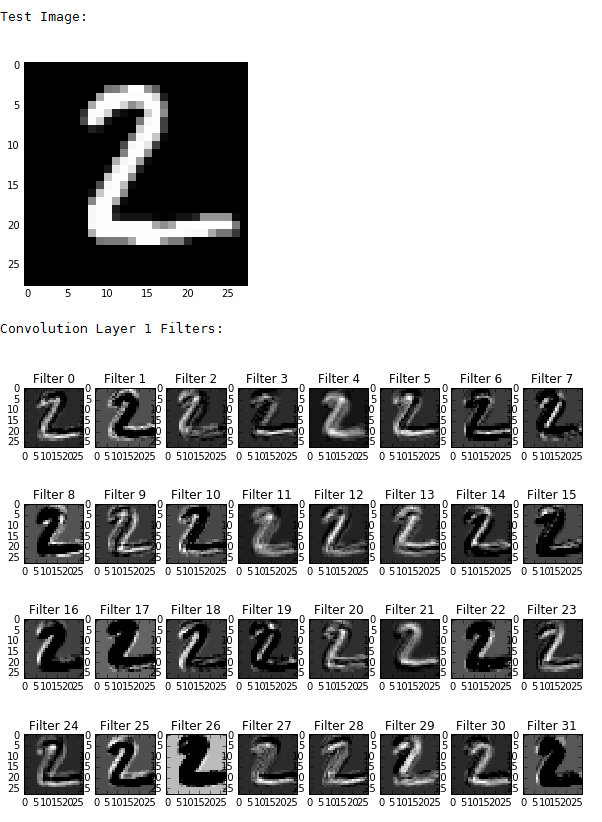


Figure Convolution Layer 1 Output

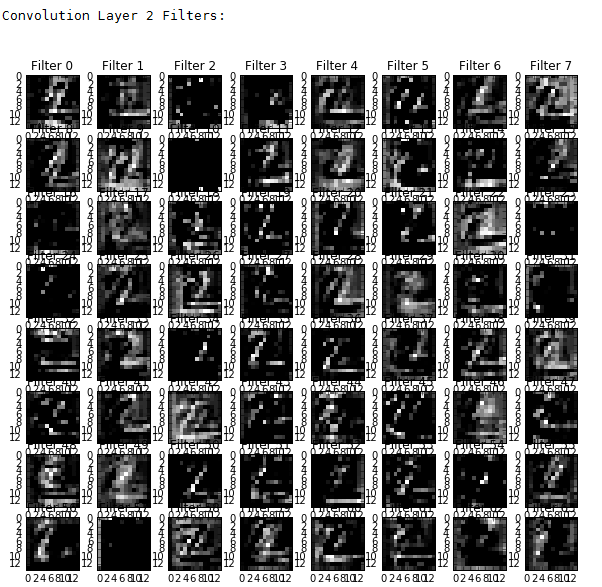


Figure Convolution Layer 2 Output

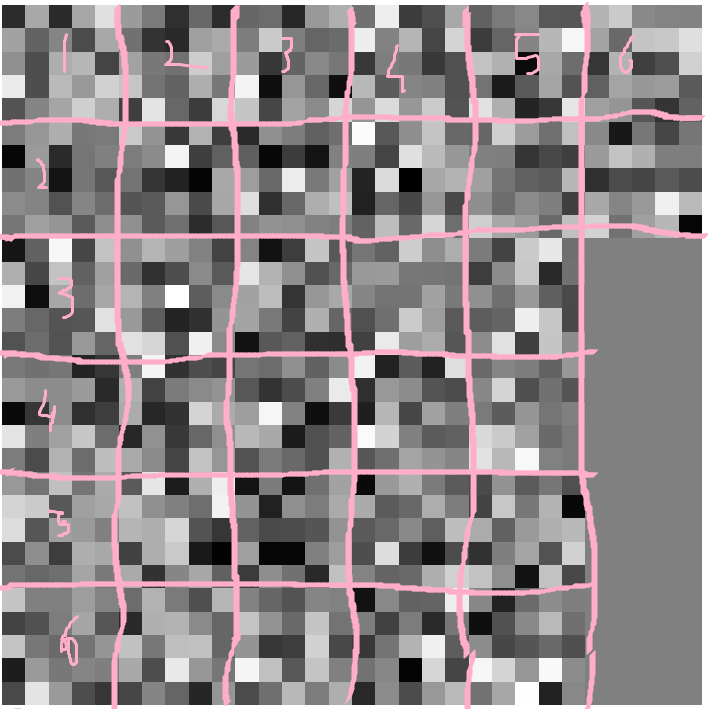


Figure Convolution Layer 1 Filter Value

The figure depicts that 32 filters help in convolution of the input image. Each of the 32 filters helps in extraction of 32 distinct features from the input

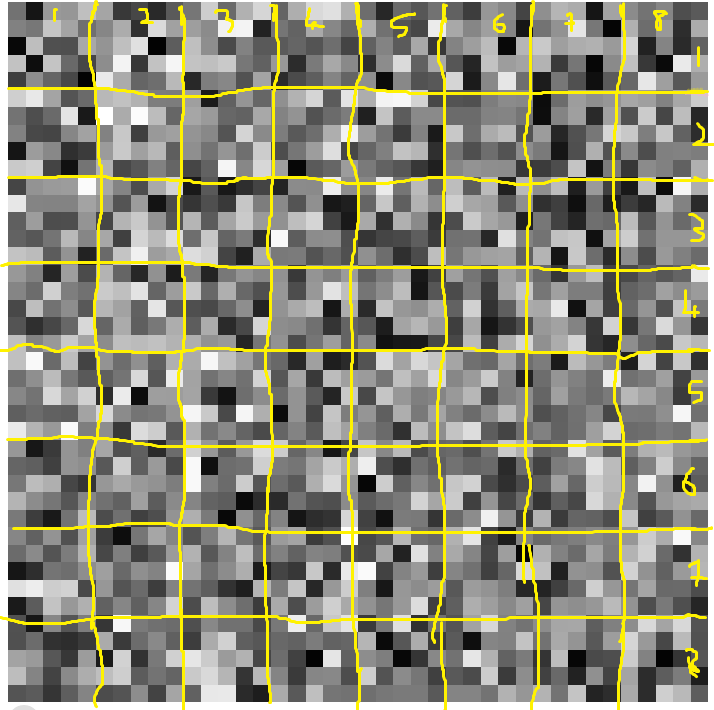
****

Figure Convolution Layer 2 Filter Value

The figure depicts that 64 filters help in the convolution of layer 1 output. Each of these 64 filters helps in extraction of 64 distinct features from the input

**Architecture Design**

1. Default

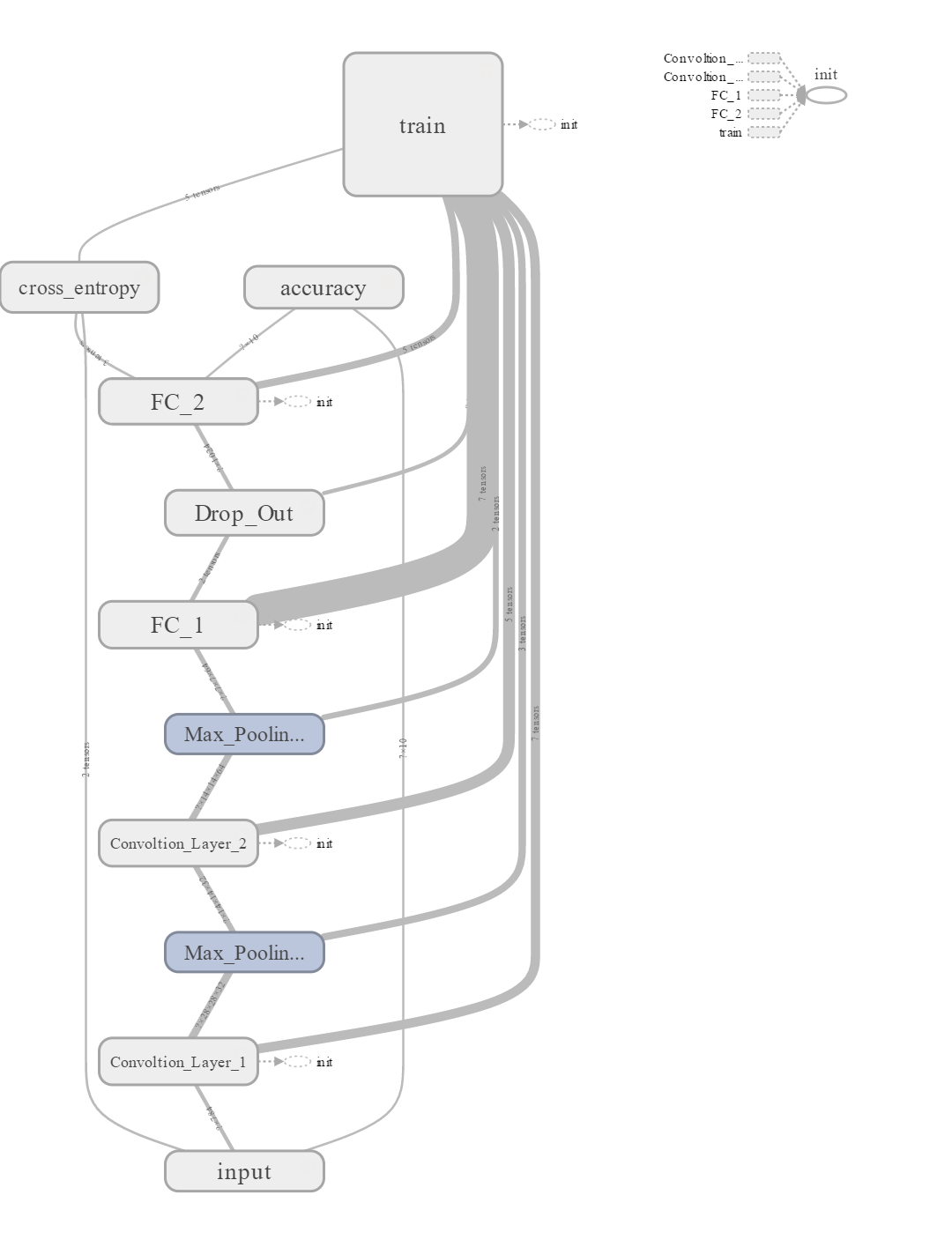
|  |  |  |  |
| --- | --- | --- | --- |
| Layer | Input | Output | Filter |
| Conv\_1 | (?, 32, 32, 1) | (?, 28, 28, 32) | 5\*5 |
| Pool\_1 | (?, 28, 28, 32) | (?, 14, 14, 32) | 2\*2 |
| Conv\_2 | (?, 14, 14, 32) | (?, 14, 14, 64) | 5\*5 |
| Pool\_2 | (?, 14, 14, 64) | (?, 7, 7, 64) | 2\*2 |
| Flatten | (?, 7, 7, 64) | (?, 3136) | N/A |
| FA\_1 | (?, 3136) | (?, 1024) | N/A |
| FA\_1 | (?, 1024) | (?, 10) | N/A |

1. LeNet

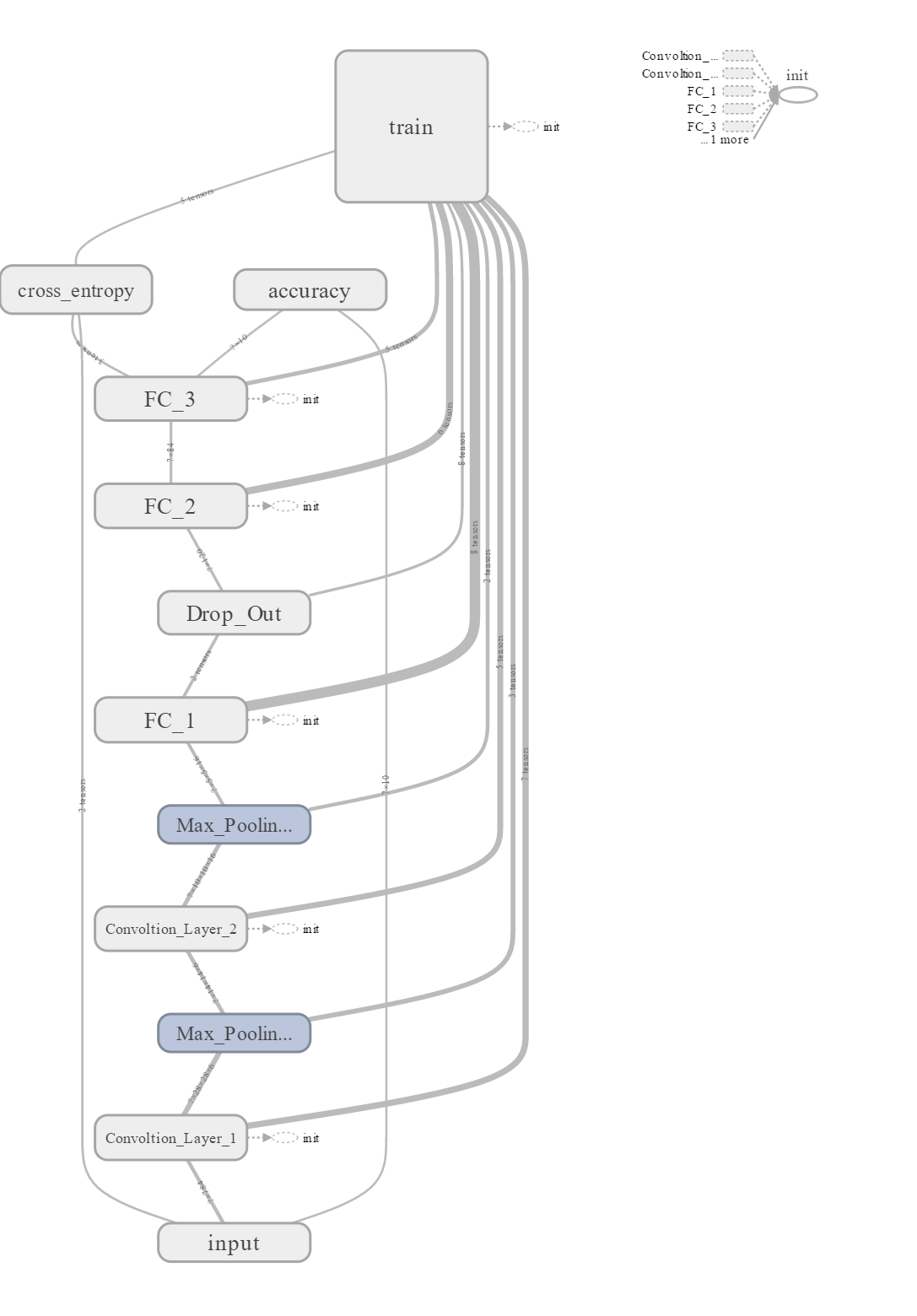
|  |  |  |  |
| --- | --- | --- | --- |
| Layer | Input | Output | Filter |
| Conv\_1 | (?, 32, 32, 1) | (?, 28, 28, 6) | 5\*5 |
| Pool\_1 | (?, 28, 28, 6) | (?, 14, 14, 6) | 2\*2 |
| Conv\_2 | (?, 14, 14, 6) | (?, 10, 10, 16) | 5\*5 |
| Pool\_2 | (?, 10, 10, 16) | (?, 5, 5, 16) | 2\*2 |
| Flatten | (?, 5, 5, 16) | (?, 400) | N/A |
| FA\_1 | (?, 400) | (?, 120) | N/A |
| FA\_2 | (?, 120) | (?, 84) | N/A |
| FA\_2 | (?, 84) | (?, 10) | N/A |

1. Modified

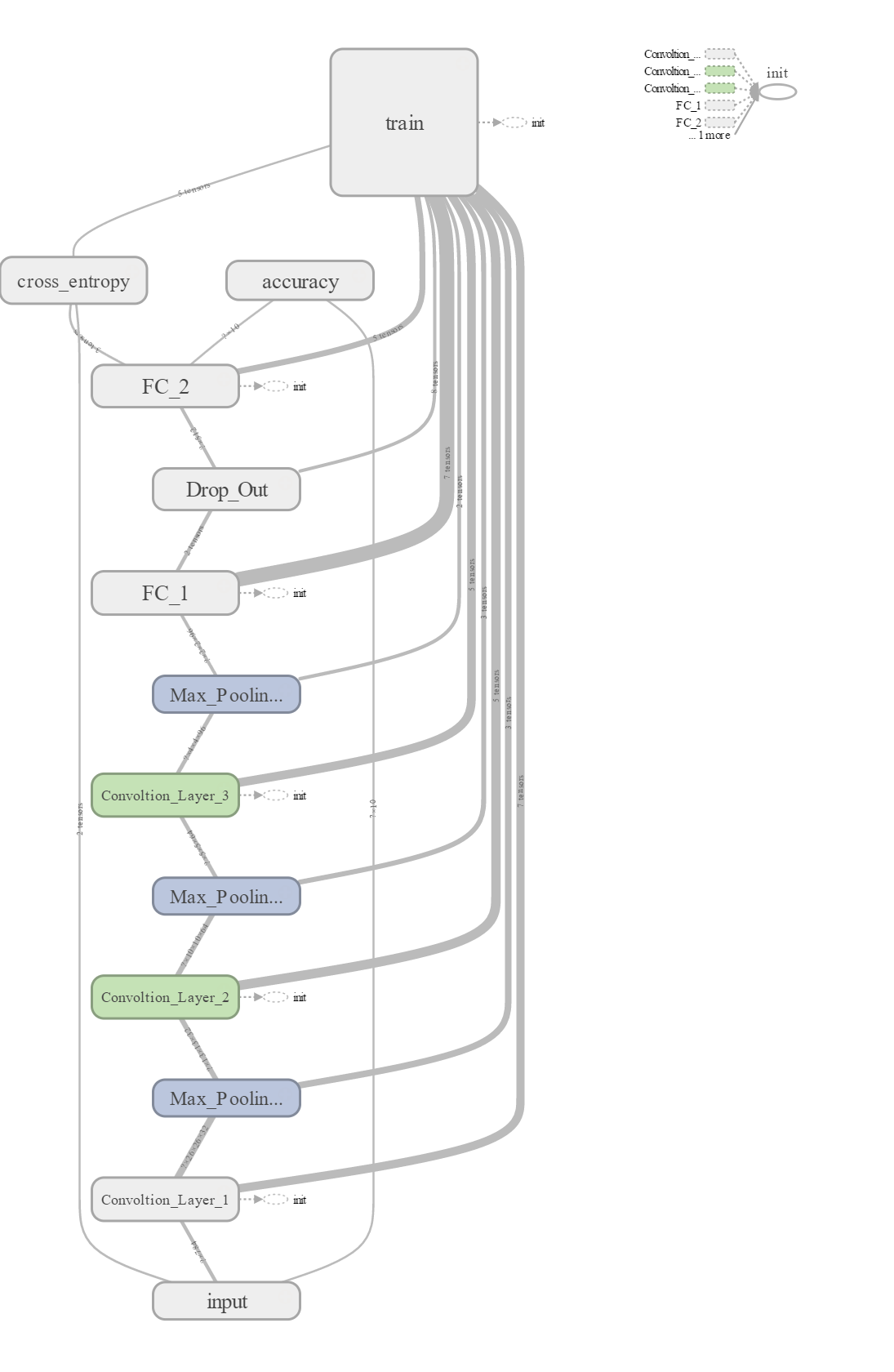
|  |  |  |  |
| --- | --- | --- | --- |
| Layer | Input | Output | Filter |
| Conv\_1 | (?, 28, 28, 1) | (?, 26, 26, 32) | 3\*3 |
| Pool\_1 | (?, 26, 26, 32) | (?, 13, 13, 32) | 2\*2 |
| Conv\_2 | (?, 13, 13, 32) | (?, 10, 10, 64) | 4\*4 |
| Pool\_2 | (?, 10, 10, 64) | (?, 5, 5, 64) | 5\*5 |
| Conv\_3 | (?, 5, 5, 64) | (?, 4, 4, 96) | 2\*2 |
| Pool\_3 | (?, 4, 4, 96) | (?, 2, 2, 96) | 2\*2 |
| Flatten | (?, 2, 2, 96) | (?, 384) | N/A |
| FA\_1 | (?, 384) | (?, 512) | N/A |
| FA\_2 | (?, 512) | (?, 10) | N/A |

****

**Default**

****

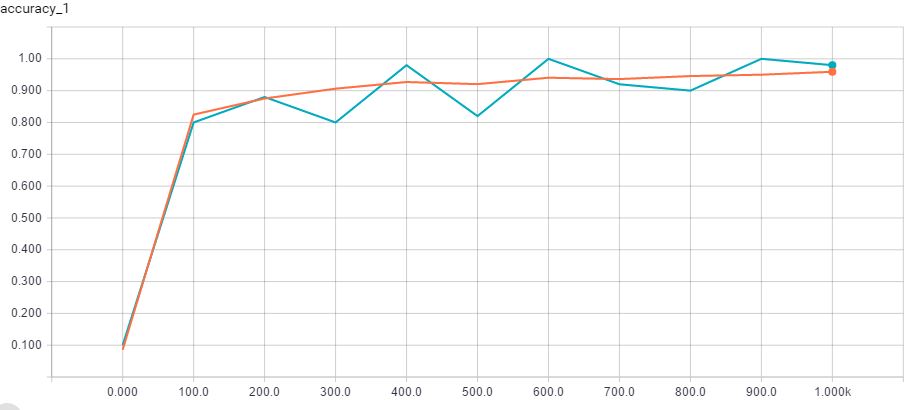
**LeNet**

****

**Modified**

**Accuracy**

1. Default

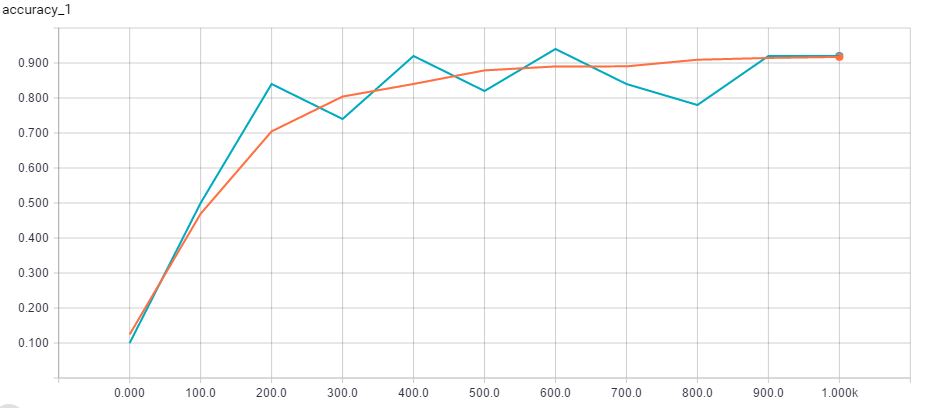
****

Test

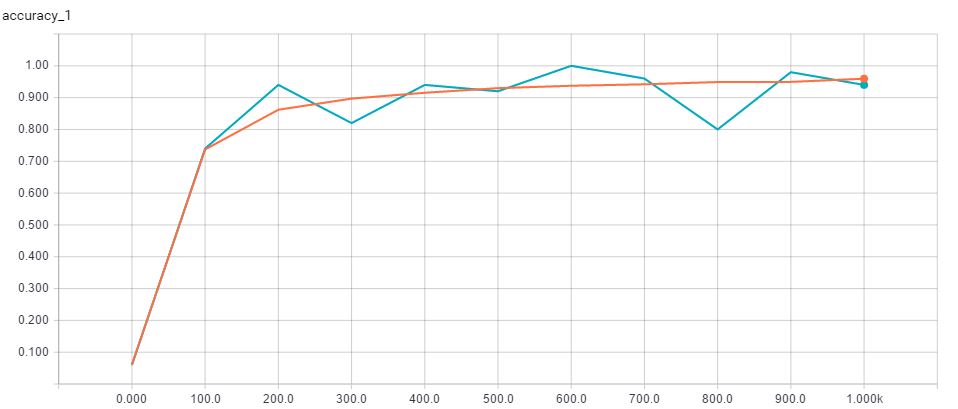
Training

Dropout = 0.1

1. **LeNet**

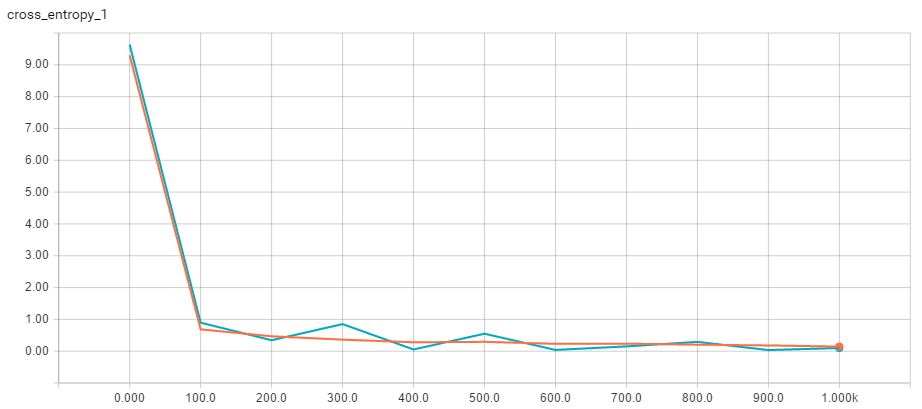
****

1. **Modified**

****

**Cross Entropy**

1. Default

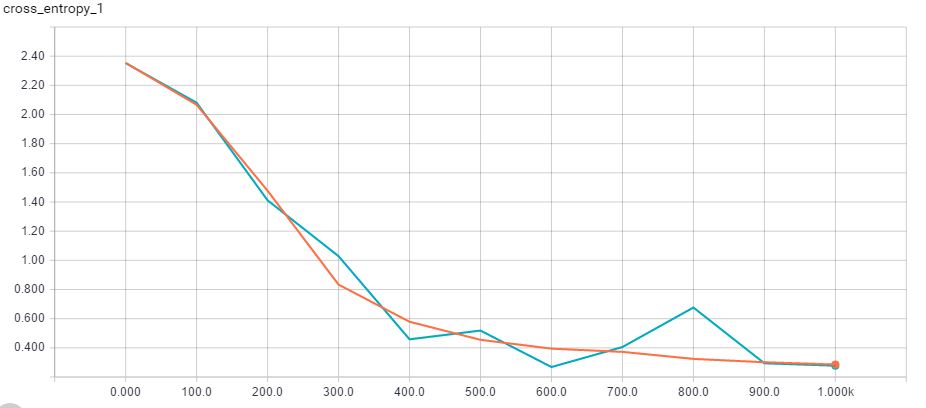
****

Test

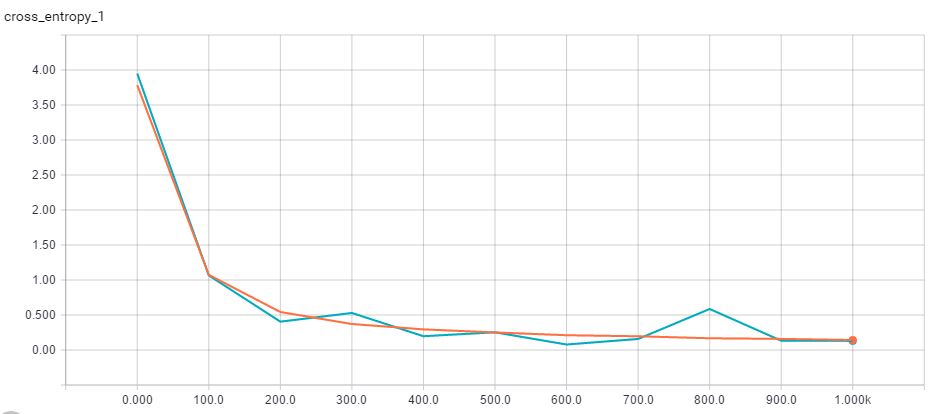
Training

Dropout = 0.1

1. LeNet

****

1. Modified



**Results**

1. Dropout = 0.0

## Accuracy

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Iteration | Default | | LeNet | | Modified | |
| Train | Test | Train | Test | Train | Test |
| 0 | 0.1 | 0.1136 | 0.14 | 0.0958 | 0.12 | 0.0855 |
| 100 | 0.88 | 0.8118 | 0.62 | 0.5424 | 0.72 | 0.777 |
| 200 | 0.88 | 0.8735 | 0.84 | 0.7531 | 0.94 | 0.8926 |
| 300 | 0.8 | 0.8984 | 0.82 | 0.8288 | 0.92 | 0.9127 |
| 400 | 0.96 | 0.9136 | 0.82 | 0.851 | 0.96 | 0.9315 |
| 500 | 0.88 | 0.9186 | 0.86 | 0.8617 | 0.9 | 0.9375 |
| 600 | 1 | 0.9269 | 0.96 | 0.8862 | 1 | 0.9407 |
| 700 | 0.96 | 0.9298 | 0.88 | 0.8904 | 0.92 | 0.9448 |
| 800 | 0.84 | 0.9404 | 0.76 | 0.8985 | 0.84 | 0.9512 |
| 900 | 0.98 | 0.9456 | 0.86 | 0.9065 | 0.94 | 0.9467 |
| 1000 | 0.92 | 0.9498 | 0.92 | 0.9109 | 0.94 | 0.9615 |

1. Dropout = 0.1

## Accuracy

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Iteration | Default | | LeNet | | Modified | |
| Train | Test | Train | Test | Train | Test |
| 0 | 0.1 | 0.0857 | 0.1 | 0.1245 | 0.06 | 0.0612 |
| 100 | 0.8 | 0.8247 | 0.5 | 0.4693 | 0.74 | 0.7372 |
| 200 | 0.88 | 0.8752 | 0.84 | 0.7047 | 0.94 | 0.862 |
| 300 | 0.8 | 0.9062 | 0.74 | 0.8041 | 0.82 | 0.8969 |
| 400 | 0.98 | 0.9271 | 0.92 | 0.8401 | 0.94 | 0.9153 |
| 500 | 0.82 | 0.9204 | 0.82 | 0.8789 | 0.92 | 0.9298 |
| 600 | 1 | 0.9406 | 0.94 | 0.8902 | 1 | 0.9373 |
| 700 | 0.92 | 0.9363 | 0.84 | 0.8904 | 0.96 | 0.9419 |
| 800 | 0.9 | 0.9458 | 0.78 | 0.9093 | 0.8 | 0.949 |
| 900 | 1 | 0.9501 | 0.92 | 0.9144 | 0.98 | 0.9495 |
| 1000 | 0.98 | 0.9593 | 0.92 | 0.9174 | 0.94 | 0.9595 |

1. Dropout = 0.5

## Accuracy

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Iteration | Default | | LeNet | | Modified | |
| Training | Test | Training | Test | Training | Test |
| 0 | 0.12 | 0.0866 | 0.16 | 0.1374 | 0.1 | 0.119 |
| 100 | 0.84 | 0.8175 | 0.6 | 0.5079 | 0.68 | 0.7068 |
| 200 | 0.88 | 0.8853 | 0.78 | 0.7023 | 0.94 | 0.8438 |
| 300 | 0.82 | 0.9024 | 0.7865 | 0.7865 | 0.78 | 0.8873 |
| 400 | 0.96 | 0.9181 | 0.821 | 0.821 | 0.92 | 0.9084 |
| 500 | 0.88 | 0.9221 | 0.8518 | 0.8518 | 0.84 | 0.9234 |
| 600 | 1 | 0.9315 | 0.873 | 0.873 | 0.98 | 0.9316 |
| 700 | 0.98 | 0.934 | 0.882 | 0.882 | 0.94 | 0.9353 |
| 800 | 0.86 | 0.9433 | 0.8915 | 0.8915 | 0.86 | 0.943 |
| 900 | 1 | 0.9471 | 0.8992 | 0.8992 | 0.98 | 0.9463 |
| 1000 | 0.94 | 0.9519 | 0.9035 | 0.9035 | 0.94 | 0.9522 |

**Observations**

* The time needed to train the network increases with increase in depth of the model.
* The lower value of dropout probabilities results in increased accuracy of the model. While extreme values of dropout probability, decrease the accuracy of the model.
* The accuracy of the model is highest for the architecture found on TensorFlow tutorial (Default), while its lowest for the one that is inspired by LeNet model. This is because of the filter mappings between LeNet layers is not same as that of the original LeNet model. The accuracy of the Modified version of Default CNN model is considerably better. The reason for this is the presence of an additional convolution and sampling layer that captures additional information of the image to be classified.

**Cats and Dogs**

*Architecture Design*

|  |  |  |  |
| --- | --- | --- | --- |
| Layer | Input | Output | Filter |
| Conv\_1 | (?, 64, 64, 3) | (?, 64, 64, 32) | 3\*3 |
| Pool\_1 | (?, 64, 64, 32) | (?, 32, 32, 32) | 2\*2 |
| Conv\_2 | (?, 32, 32, 32) | (?, 32, 32, 64) | 3\*3 |
| Conv\_3 | (?, 32, 32, 64) | (?, 32, 32, 64) | 3\*3 |
| Pool\_2 | (?, 32, 32, 64) | (?, 16, 16, 64) | 2\*2 |
| FA\_1 | (?, 16, 16, 64) | (?, 512) | N/A |
| Dropout = 0.5 | | | |
| FA\_2 | (?, 512) | (?, 2) | N/A |

test\_size = 600 n\_epoch = 25

train\_size = 1400 batch\_size = 10

total\_size = 2000 learning\_rate = 0.0005

Iteration Count = 10 | Loss = 0.69 | Accuracy = 100 %

*Data/Image augmentation*techniques are used to artificially expand the dataset. Few of the popular augmentations techniques people use are grayscales, horizontal flips, vertical flips, random crops, color jitters, translations, rotations, and much more. By applying a couple of these transformations to training data, we can easily double or triple the number of training examples.

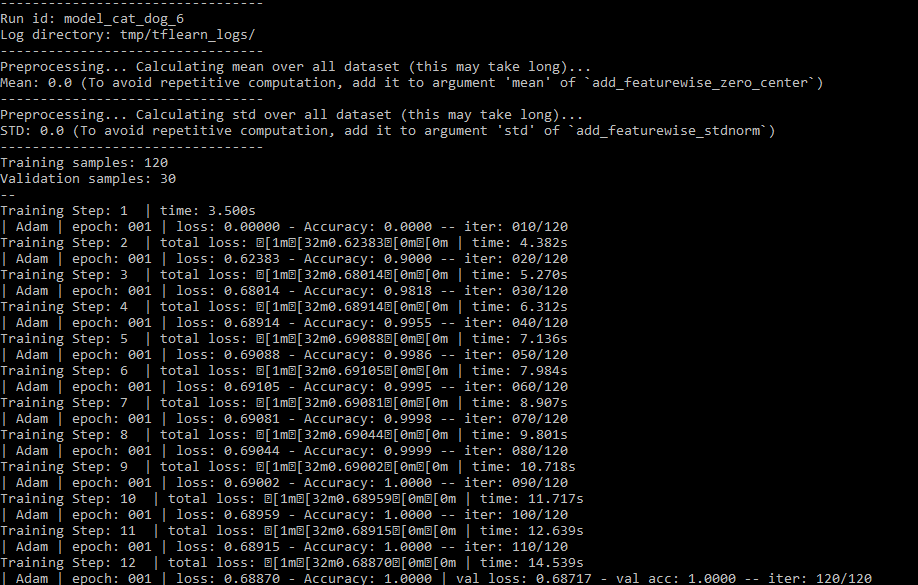
*Image Preprocessing*

* Used as an argument of input\_data.
* Defined pre-processing methods is applied both at training and testing time.
* Functions
  + add\_featurewise\_zero\_center()
    - Zero center every sample with specified mean
  + add\_featurewise\_stdnorm()
    - Scale each sample by the specified standard deviation.
    - If no std specified, std is evaluated over all samples data.

*Image Augmentation*

* Applied only at training time
* Similar to Image Preprocessing
* Functions
  + add\_random\_flip\_leftright
    - Randomly flip an image
  + add\_random\_rotation
    - Randomly rotate an image by a random angle within a range

*Results*



**Word2Vec**

Word2vec is a computationally efficient predictive model for learning word embedding from raw text. Word embedding helps to place similar words near each other while irrelevant words away from each other. For example, it yields a vector approximating the representation for **vec(‘Rome’)** because of the vector operation:  **vec(‘Paris’**) – **vec(‘France’) + vec(‘Italy’).**

Word2vec comes in many flavors. We will focus on, **Continuous Bag-of-Words** (CBOW) model and the **Skip-Gram** model.

**Continuous Bag-of-Words**:

The input to the model could be **wi − 2, wi − 1, wi + 1, wi + 2,** the preceding and following words of the current word we are at. The output of the neural network will be **wi**. Hence, we can think of the task as "predicting the word given its context". The number of words we use depends on setting for the context window size. It runs several times faster to train in comparison with the skip-gram, slightly better accuracy for the frequent words.

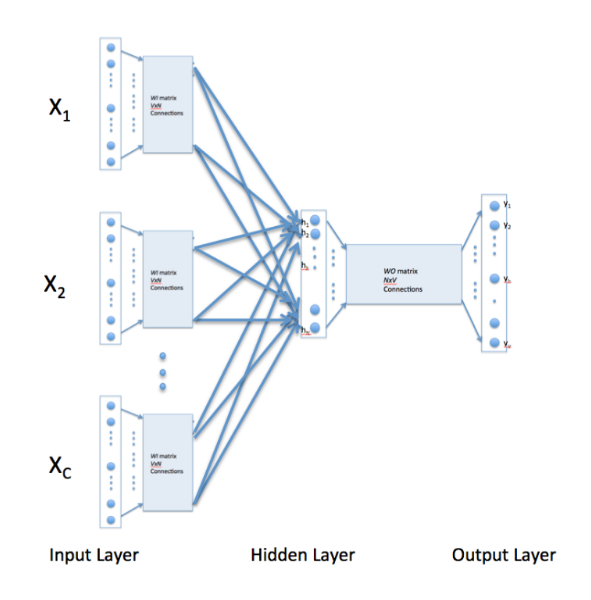


Figure 1 Continuous Bag of Words

**Skip-gram**:

The input to the model is **wi**, and the output could be **wi − 1, wi − 2, wi + 1, wi + 2**. Therefore, the task here is "**predicting the context given a word**". In addition, words that are more distant is given less weight by randomly sampling them. The contextwindow size is randomly chosen between 1 and max size for each training sample, resulting in words with the maximum distance being observed with a probability of 1/c while words directly next to the given word are always observed. It works well with small amount of the training data, represents well even rare words or phrases.

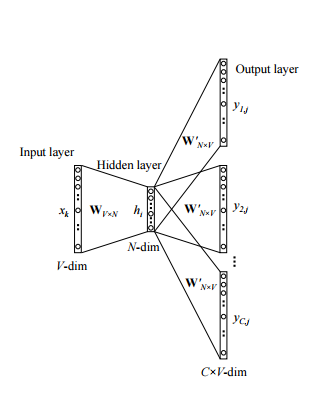


Figure 2 Skip Gram Model

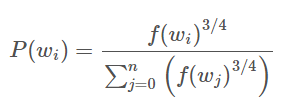
**Negative Sampling**

The size of our word vocabulary means that our Word2vec neural network has a tremendous number of weights, all of which would be updated slightly by every one of our billions of training samples. This results in need of lot of computation time. Negative sampling addresses this by having each training sample only modify a small percentage of the weights, rather than all of them.

With negative sampling, we are instead going to randomly select just a small number of “negative” words to update the weights for. (In this context, a “negative” word is one for which we want the network to output a 0 for). We will also still update the weights for our “positive” word, which are the positive input samples at hidden layer (context words in our skip gram model).

In general, we select negative samples in the range of 5-20 words works well for smaller datasets, and with 2-5 words for large datasets.

The probability for selecting a word as a negative sample is related to its frequency, with more frequent words being more likely to be selected as negative samples. Each word is given a weight equal to its frequency (word count) raised to the 3/4 power. The probability for a selecting a word is just its weight divided by the sum of weights for all words.



**Example**

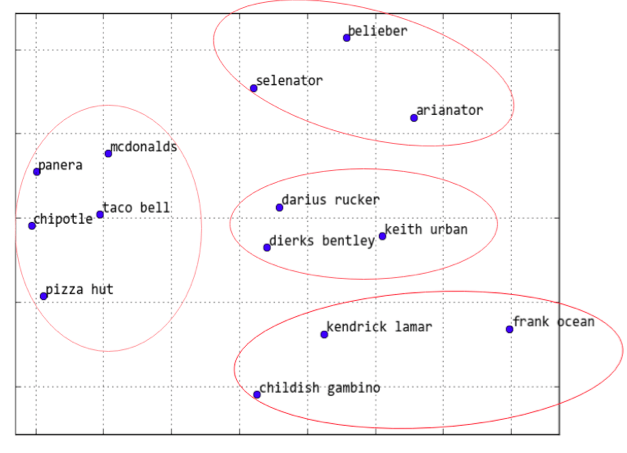


Figure Distance between Words

The above figure depicts the concept of word2vec wherein similar words are close to each other while dissimilar words are away from each other

**Recurrent Neural Network**

Recurrent Neural Network (RNN) are used when a current input depends on previous sequence of inputs. These are generally used for text generation, sentiment analysis, machine translation, and chat box creation.

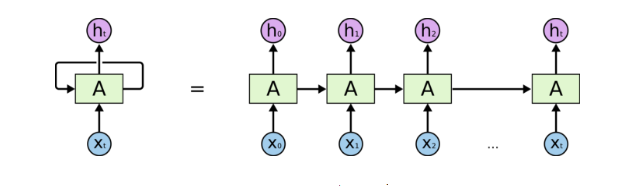


Figure Multi-Layer RNN

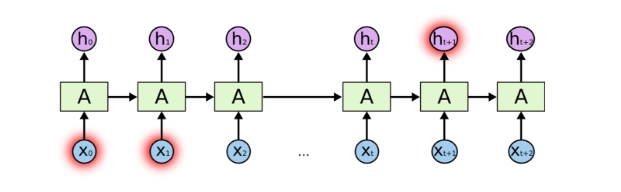


Figure Problems Arising Due To Long Term Input Dependency in Basic RNN

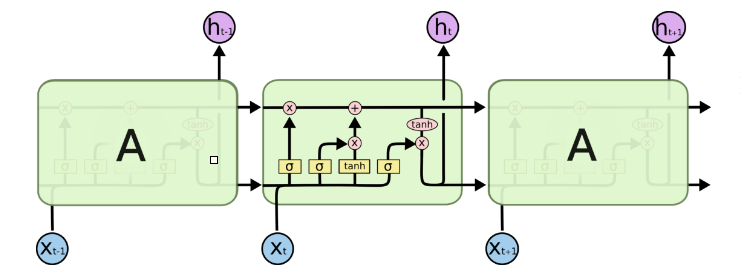
003

Figure RNN Using LSTM

ht

ct-1

ct

Figure Inside LSTM Network

tanh

it

ft

ot

Ct

ht-1

ht

s

tanh

s

s

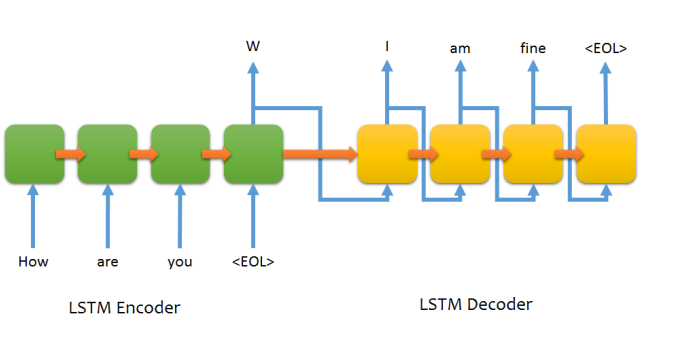
xt

**Overview of LSTM Layers**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Layer | Purpose | | Implementation | Reason for Working |
| Word Embedding | Represent words in form of vector |  | | Each sentence is represented in form of 1 hot encoded vectors containing weight matrix corresponding to it |
| Forget Gate | Forget data stored in previous state | ft = s (Wf. [ht-1,xt] + bf ) | | When we see a new subject, we want to forget the old subject |
| Input Gate | Decide what new information we’re going to store in the cell state | it = s(Wi .[ht-1,xt] + bi )  Ct = tanh ( Wc.[ht-1,xt] + bc ) | | *Sigmoid* layer decides which values we’ll update  *tanh* layer creates a vector of new candidate values, Ct, with which cell state can be updated |
| Cell State | Update the old cell state, Ct−1, into the new cell state Ct | Ct = ft\*Ct-1+it\*Ct | | *Sigmoid* layer which decides what parts of the cell state we’re going to output |
| 8Output Gate | Decide what will be the next hidden state | ot = s(Wo.[ht-1,xt] + bo)  ht = ot\* tanh(Ct) | | Put the cell state through *tanh* (to push the values to be between −1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decide to. |

**Sequence to Sequence Modelling**

Sequence to Sequence Modelling is used in application like text classification, machine translation, and chat bot and so on. They are constructed by joining two sets of RNN, called as an encoder and decoder. Output of encoder is stored in a context vector. This vector acts as hidden state input values to the decoder.



Input is fed to the encoder and decoder with the help of a word embedding matrix. This embedding matrix is built based on Skip Gram/ Continuous Gram model. Input is fed to the decoder only during training phase. During the test phase, accuracy of model is tested based on only the output of context vector. A technique called as attention mechanism is used to decide how much of the input affects the output, i.e. input that doesn’t contribute to o/p is ignored.

Accuracy of model can be increased by using stacks of LSTMS on top of each other. Though this increases training time, accuracy of o/p is guaranteed. A typical Sequence to Sequence to has a very large training dataset, general in order of 100,000.

***Pseudocode:***

* Build word2index and index2word word vector of both encoder and decoder inputs
* Word embedding vector can be based upon Skip Gram or any similar word2vec model
* Initialize the weights of RNN’s i/p unit, hidden unit and o/p unit for both encoder and decoder
* Feed input to encoder sequentially and store its output in context vector
* Output of context vector acts as hidden unit input to the decoder
* True decoder i/p is used as i/p to decoder units during training, while o/p of previous decoder unit is used as i/p to current decoder unit during testing
* At decoder, o/p is displayed after passing through every decoder input unit. While in encoder, o/p is collected only at end of encoder unit.
* Here, programming an encoder is like building an RNN for sentiment analysis purpose while programming an encoder is like building an RNN for text generation purpose

**Parameters**

* num\_layers = 3
* xseq\_len, yseq\_len = 20
* num\_sentences = 91295s
* xvocab\_size, yvocab\_size = 8002
* batch\_size = 16, emb\_dim = 1024

**Chat Bot Results Using Twitter Dataset**

q : [go see them]; a : [i was thinking of that]

q : [a unk of the before we say goodnight from last at unk state park in maine]; a : [this is a great idea to be a great day for a while]

q : [unless you want this ]; a : [what is he]

q : [my fav moment from the debate last night]; a : [wait for the first time]

q : [ok i dont see any catholic terrorist]; a : [not even to be a threat to the point]

q : [thats your sister]; a : [i dont know what i was talking about]

q : [radical islam unk a threat to the united states and our leaders need to realize this before its too late]; a : [is that a joke]

q : [literally the same thing happened to me ]; a : [thats what i said]

q : [last chance to win vip passes to new york comic con amp meet the walking dead cast go]; a : [if you were in the same place i dont have to see it]

q : [4 correction that video is from a few years ago]; a : [its been a fan of my life but i dont know what to do]

q : [we are doing band there are many unk in the band]; a : [i have to be a good job]

q : [im sure it does but i doubt ill see a unk amount of money from it]; a : [i think its a good idea]

q : [which report did this graph come from the unk unk data is really interesting and hard to find]; a : [is this a lot of times from the world of the world of the year]

q : [thats pretty cool but now i am dying to find out what s first tweet will be]; a : [it was a good time for the first time]

q : [great debate poll numbers i will be on at 700 to discuss enjoy]; a : [dont have to be a president]

q : [yooo me too]; a : [ahh i know i was just thinking about this]

q : [the ladies of could not be more idiotic or blind they should invite on for a talk]; a : [they are not sarcastic]

q : [clothes that dont fit me no more]; a : [ur so cute]

q : [good evening unk]; a : [good morning all]

q : [so pretty much she didnt want him so u were is 2nd choice ]; a : [lol i dont know what to say about it ]

q : [another reminder of my favorite quote people sell what they need and buy what they dont need thanks]; a : [this is a joke]