Capstone Project

Machine Learning Engineer Nanodegree

Vladislav Prokhorov
November 16st, 2016

Definition

Project overview:

My project refers to convolutional neural network image recognition.
More concrete in this project we will work on recognition arbitrary multi-digit numbers obtained from The Street View House Numbers (SVHN) Dataset.

Recognizing sequence of digits from photographs is important component of modern-day map making. If we would have model which can recognize address number from image with high degree of accuracy then we could associate this number with a known street address and it would help determine with a high degree of accuracy, the location of the building it represents. So our model can help to create more accurate maps

Problem statement:

Problem: the recognizing arbitrary multi-digit numbers from Street View imagery.
Solutions strategy: 1) First strategy can be typically separate out the localization, segmentation, and recognition steps it means that we need to work with each digit from multi-digit number separately.
2) In the second strategy we can use deep convolutional neural network and combine 3 steps together so we don't need to work with each digit separately instead we can feed image with sequence of digits into neural network and it gives us prediction.
Conclusion: obviously the second strategy is more convenience, so we will create a deep convolutional neural network

Metrics:

I will use standard accuracy function, by that I mean the number of correct predictions made divided by the total number of predictions made
But I need to mention one very important thing: by correct prediction I mean when the whole sequence of digits is correct. For example, if prediction is 123 and true value is 123 then we can say that neural network gives a correct prediction,another example,if prediction is 124 and true value is 123 then the neural network gives incorrect prediction in spite of the fact that two digits match.

Analysis

Data Exploration:

In SVHN dataset we have 73257 digits for training, 26032 digits for testing, and 531131 additional, somewhat less difficult samples, to use as extra training data.
Also before preprocessing step we have 10 classes, 1 for each digit. Digit '1' has label 1, '9' has label 9 and '0' has label 10..
I want to notice that I will not use extra training data because of computational power of my computer so my data set will contain only 73257 digits for training and 26032 digits for testing or 33402 multi-digit numbers for training and 13068 multi-digit numbers for testing

Figure 1: A sample image

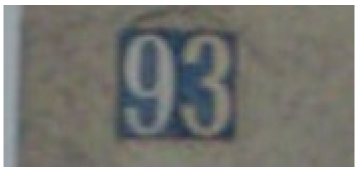


Figure 2: A sample image

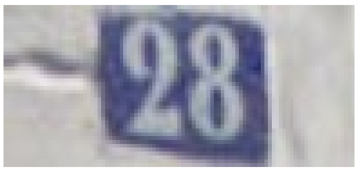
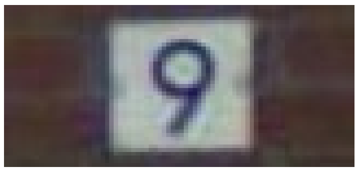


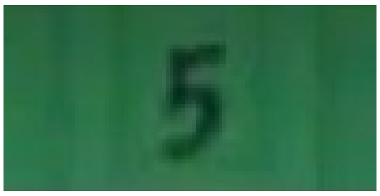
Figure 3: A sample image



When I explored the images, I noticed that there are certain characteristics in the data and I want to point some of them and show some examples that match certain characteristics:

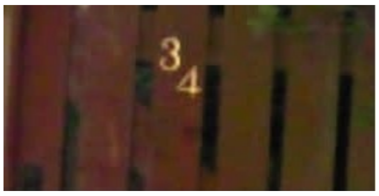
1) First characteristic is 'understandable format'. For neural network it shouldn't be a problem to recognize these numbers.

Figure 4: Understandable image



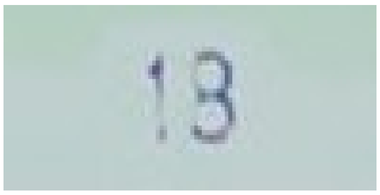
2) Second characteristic is a non-standard baseline. Here neural network can have problems with recognition of these numbers.

Figure 5: Image with non-standard baseline



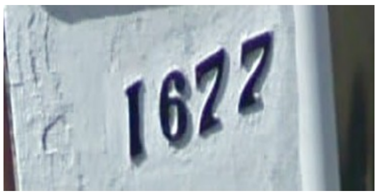
3) Third characteristic is broken outlines. It is hard task for neural network to recognize these numbers, if we look at the Figure 6 we can say that even for human it is hard to say this is 18 or 13.

Figure 6: Image with broken outlines



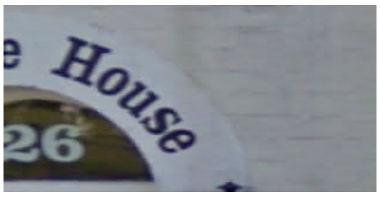
4) Fourth characteristic is non-standard fonts. If we look at the Figure 7, the last two digits(seven and seven) may resemble two and two because of non-standard font, so neural network can make a mistakes in recognition of these numbers.

Figure 7: Image with non-standard fonts



5) Fifth characteristic is a bad localization. The problem in this case is to localize where the digits on the image, fortunately in the SVHN data set we have bounding boxes which show us where the digits is, so it is not a hard problem to recognize these numbers, but in real world examples we need to clearly identify where digits on the image to recognize them.

Figure 8: Image with bad localization

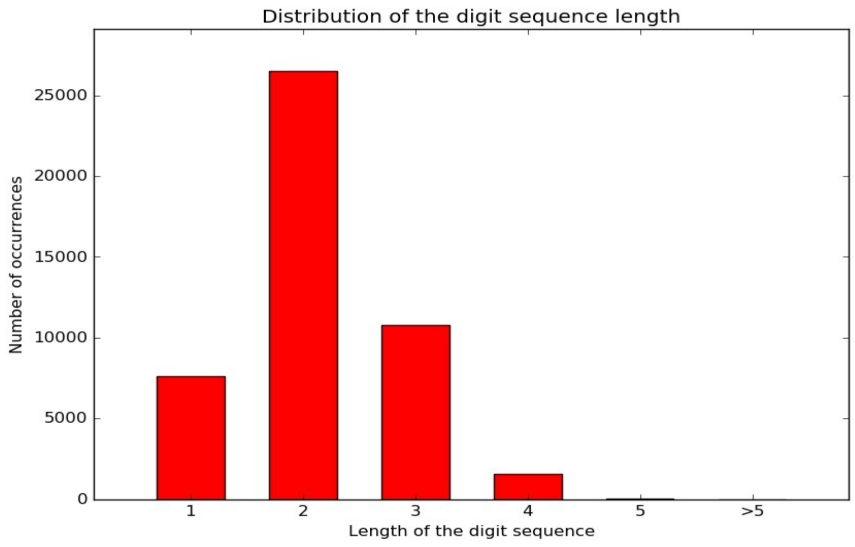


So I pointed some characteristics of the images we have to deal with. Many images from our data set fit to these characteristics, of course there are also other but I pointed the main characteristics. Also I want to mention that most images from data set fit to the first characteristic('understandable images').

Exploratory Visualization:

There are outliers in our data set, to identify them we can plot the distribution of the digit sequence length.

Figure 9: Graph



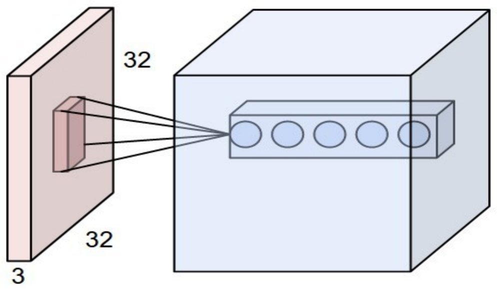
From this graph we can see that there are very few images with five-digit number and six-digit number, in particular our data set contains 11 five-digit numbers and only one six-digit number
We can conclude that for our model will be very difficult to predict numbers with 5 and more than 5 digits due to the lack of similar examples in our data set.

Algorithms and Techniques:

As I mentioned above to solve this problem I will use deep convolutional neural network. Before I tell you how convolutional neural network works I would like to point out why I choose this algorithm rather than any other.
1) Firstly, Convolutional neural net takes advantage of the fact that the input consists of images so CNN considers information about the input in 3d structure: weight,height,depth (by depth I mean RGB channels of the image) while for example regular neural nets request input space in 1d dimension, so CNN saves the structure of the image and we don't need to flatten image pixels in 1d dimension, so with CNN we can learn more about the images.
2) Secondly, CNN uses less number of weights than for example a regular neural network due to the fact that CNN has property of shared weights and neurons in CNN connected to a small region of the layer before it(I will say more about it later).
3) Thirdly, worth noting that empirically CNN gives better accuracy for image recognition than any other algorithm.

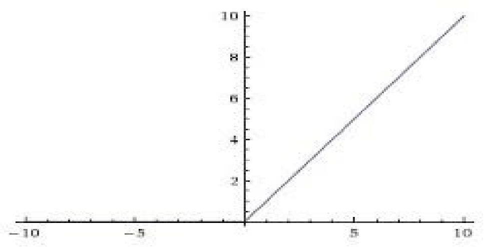
Now we can start to talk about how convolutional neural net works:
1) Input:
Input for CNN is raw pixel values of the image but as I pointed above these pixels CNN gets in 2d or 3d dimension, 2d or 3d depends on the input images, for example greyscale image will be 2d input whereas 3d will be a color image, in our case CNN will use color images as input.
2) Convolutional layer:
This is the main layer of CNN which determines its name. This layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. You can look at the Figure 10 for visualization.

Figure 10: A visualization of how convolutional layer works [2]



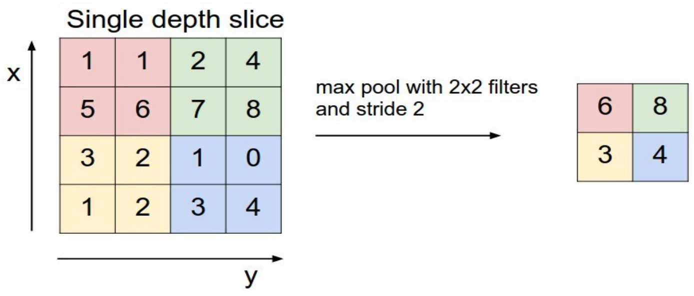
Output of CONV layer will be in 3d dimension where depth is hyperparameter, it depends of how many filters we use, for example, if we decide to use 12 filters then depth for CONV layer will be also 12, width and height CONV layer depend on the other hyperparameters but before start to talk about it we need to understand what is filter.
The main motivation behind using convolutional layers is that it is typically true of images that pixels in close proximity are more related with each other than with pixels that are a greater distance away. Thus, compared to fully connected layers, convolutional layers give a better indication of general features that appear in an image by taking advantage of this spatial structure of images. Moreover, Conv layers more computational efficient than fully-connected layers due to the sharing parameters.
3) Filter:
People sometimes use this term in slightly different ways, I think the most appropriate definition call this shared weights, why shared? imagine CONV layer with 12 filters(depth 12, or 12 depth slices) each filter from 12 has the same size, for example, if neurons connected to local regions in the input with size 5x5x3 then filter has size 5x5x3 so in this case we have only 12x5x5x3(+ 12 biases parameter) unique weights, so for each neuron in one depth slice we have the same vector of weights.
4) Strides,receptive field and zero-padding:
As I say above width and height CONV layer depend on the hyperparameters and now I must explain them:
Receptive field – imagine example with neurons that connected to local regions in the input with size 5x5x3 now then receptive field has size 5x5, important to note that the extent of the connectivity along the depth axis is always equal to the depth of the input volume.
Stride – stride is slide the receptive field. When the stride is 1 then we move the receptive field one pixel at a time
Zero-padding - pad the input with zeros around the border.
5) Rectified linear unit layer(Relu):
In Relu layer we apply an activation function to each neuron from convolutional layer. Activation function is max(0,x) .
We will use this activation rather than others(tanh,sigmoid) because it does not saturate, very computational efficient and converges much faster than others in practice.

Figure 11: Relu activation function



6) Pooling layer:
This layer will perform a downsampling operation along the spatial dimensions (width, height), it can reduce the number of parameters and computation in CNN, this can help us to control overfitting. Usually people use this layer with receptive field 2x2 and strides = 2(in our case we will also use these values), so if the input size is 12x12x16 then output of pooling layer will be 6x6x16, more concrete, pooling layer will choose one neuron from four with highest activation value(max pooling), there is also a general pooling but we don't use this

Figure 12: A visualization of how Pooling layer works [2]



7)Dropout:
This is a beautiful and simple regularization technique, while training, dropout is implemented by only keeping a neuron active with some probability p (a hyperparameter), or setting it to zero otherwise. This technique can help us to control overfitting, in our neural network we will use it.
8)Softmax:
At the end of our CNN(final layer) we will use five softmax classifiers due to the fact that we want to predict five digits, softmax assign probabilities to an object being one of several different things, it gives us a list of values between 0 and 1 that add up to 1, each softmax classifier connected to all neurons in the previous layer
9)Fully connected layer:
Before the final layer(softmax classifier) we will use fully connected layer,each neuron in FC layer connected to all the neurons in the previous layer.
10)L2 regularization:
The idea is to penalize high values of weights, this gives CNN better generalization, hence prevents from overfitting, we will use it.
11) Stochastic Gradient Descent and Adagrad Optimizer
The procedure of repeatedly evaluating the gradient and then performing a parameter update is called Stochastic Gradient Descent, using this method, we can minimize any function but in machine learning we use this for minimizing cost function and therefore train model or in our case convolutional neural net.
Adagrad Optimizer is more advanced optimization method, it adapts learning rate: Adagrad reduces learning rate to weights that have high gradients and increases learning rate to weights that have small or infrequent gradients.
We will use this optimization method in our neural net because it faster converges

Benchmark:

I will try to achieve 90% accuracy on the test data, the reason why I have chosen this threshold is that these guys from Google achieved 96% accuracy but they used more training data than me(remember that I don't use extra data) and they used more powerful computers so with my computer it will be nice to reach even 90%

Methodology

Data Preprocessing:

As I pointed above we have 33402 images for training and 13068 images for testing, each image in png format and each digit at the image has its own bounding box which shows where the digit is located, firstly, I want to say what I do with x_data(images):
As each image has different size we should make all images the same size, hence we should resize them to 32x32x3, I chose 32x32x3 because of computational power of my computer, I tried to resize them to 64x64x3 as Google guys suggested, but my computer was too slow with this input size.
Before resizing we need to crop image by rectangular bounding box that will contain individual digit bounding boxes and then resize this crop. It's worth noting that before resizing I tried to expand bounding boxes by 30% and then crop images by these expanded bounding boxes but I got bad results, on 20% accuracy less than without expansion so I throw out this step from my preprocessing.

Now we can start to talk about labels(y):
Initially the correct labels were for every digit, this is not convenient for our neural network because we want to feed all image(not digits separately) to neural network and then get prediction.
So we combined correct labels so that now we have images and corresponding correct labels,for example image with number 123 has label 123 or in python environment [1,2,3,10,10], also very important to note that we will try to predict maximum 5 digits at the image, the reason why we do this is that our data set has only one six-digit number as I have already mentioned in the Exploratory Visualization section. If image with such number is found then we write only five first digits as correct label for this image, for example the correct label for image with number 123456 will be [1,2,3,4,5], so we change this example to more appropriate one, because of it is outlier. 'Blank' digit we write as 10 as you can see in this example [1,2,3,10,10],

Implementation:

At the beginning I ran into a problem with definition of the cost function for our neural net due to the fact that we should use 5 softmax classifiers(or 6, depending on the implementation) at the end of a neural net. Each classifier has to predict the one digit, for example, if the prediction is [1,2,10,10,10] then the first two classifiers predicted 1 and 2 correspondingly and the rest predicted 10 . In first time as Google guys suggested I tried to define neural net with 6 classifiers at the end, one additional classifier was added to predict how many digits at the image but my loss went down to particular values and then it stopped at these values, these values were big and I got around 1,5% accuracy on the test data it was really far from what I wanted to get :)I understood that it was bug in my implementation of the cost function but instead of trying to find this bug I decided to try another implementation of the neural net.
Instead of using 6 classifiers at the end I tried to use only five, therefore my neural net didn't predict directly how many digits at the image. As I mentioned above in the correct labels I started to use 10 for blank digits and it started to work! Loss started to converge, with initially implementation I got 40% accuracy on the test data it was still far from what I wanted to get but really better than 1.5% and I realized that I was moving in the right direction.

Code of correct implementation of the cost function:

```
loss = (tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits(
                logits_1, y[:, 1])) + \
        tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits(
                logits_2, y[:, 2])) + \
        tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits(
                logits_3, y[:, 3])) + \
        tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits(
                logits_4, y[:, 4])) + \
        tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits(
                logits_5, y[:, 5])))
```

In the cost function we summarize 5 losses of each classifier, when we minimize cost function we train our neural net.
In the initially implementation I used such architecture of neural net:
INPUT -> 1CONV -> RELU -> POOL -> 2CONV -> RELU -> POOL -> FC -> RELU -> 5X SOFTMAX

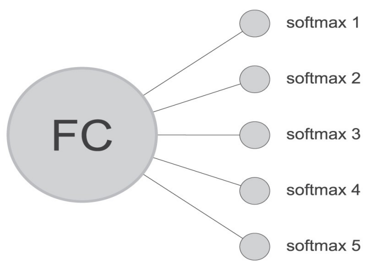
Where all Conv layers had receptive field with size 5x5, “same” padding and strides = 1.
1Conv had 16 filters whereas 2Conv had 32 filters.
As I pointed above two Polling layers had receptive field 2x2 and strides = 2.
Fc layer had 64 neurons.
In the Conv neural nets we need to feed data by batches, so I used batch = 64.
I used 150000 iterations for training neural network, it took me 5-6 hours.
For minimization cost function I used Adagrad Optimizer with such parameters: Starting learning rate == 0.05, decay_steps == 10000, decay_rate == 0.96. I chose this optimizer because it converges faster than standard stochastic gradient descent

Code of implementation of optimizer:

```
global_step = tf.Variable(0)
learning_rate = tf.train.exponential_decay(
         0.05, global_step, 10000, 0.96)
optimizer = tf.train.AdagradOptimizer(learning_rate).minimize(
        loss, global_step=global_step)
```

Also as I pointed above in the initial implementation I expanded bounding boxes by 30% in the preprocessing step.
Also It's worth noting that for compute accuracy on the test data I fed test data by batches because of computational power of my computer(my GPU has low memory) six times by 2178 examples and then combine results.

Figure 13: A visualization of the end of our neural net



Refinement:

As I pointed above with my first implementation I achieved 40% accuracy on the test data. It was the bad result and now I want to say what I did to improve this result, remember that threshold that I want to achieve is 90%:
1) I changed architecture of my neural net, I added additional convolutional layer, so in the final implementation I used such architecture:

INPUT -> 1CONV -> RELU -> 2CONV -> RELU -> POOL -> 3CONV -> POOL -> FC -> RELU -> 5X SOFTMAX

We can see that I stacked first two convolutional layers together and I didn't put pooling layer between them because multiple stack CONV layers can develop more complex features of the input before the destructive pooling operation.
After changing we have 1conv layer with 16 filters, 2conv layer with 32 filters and 3conv layer with 64 filters, I didn't change parameters of receptive fields and strides
2) I increased the size of the fully connected layer from 64 to 1024
3) As I pointed above I stopped the using of expansion of bounding boxes by 30% I threw out this method from preprocessing step, it was the main change that I did.
My test accuracy became around 75% after these changes were implemented, better but not good enough, I noticed that when neural network had trained long enough batch accuracy on the training data became around 98-100%. Comparing this value with test accuracy, it is very likely that it is overfitting so we need to add some regularization methods to avoid overfitting
4) I added dropout to each Conv layer and to fully connected layer. While training only 50% neurons were active
5) I added L2 regularization to cost function to penalize high values of weights with parameter of lambda = 0.001. This parameter determines how strong we penalize the high values of weights

Code of implementation of regularization:

```
regularizer = (0.001*tf.nn.l2_loss(W_conv1) + 0.001*tf.nn.l2_loss(W_conv2) + \
               0.001*tf.nn.l2_loss(W_conv3) + 0.001*tf.nn.l2_loss(W_fc1) + \
               0.001*tf.nn.l2_loss(W1) + 0.001*tf.nn.l2_loss(W2) + \
               0.001*tf.nn.l2_loss(W3) + 0.001*tf.nn.l2_loss(W4) + \
               0.001*tf.nn.l2_loss(W5))
```

In the final implementation we have the cost function with the regularization term:

```
loss = (tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits(
                self.logits_1, self.y[:, 1])) + \
            tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits(
                self.logits_2, self.y[:, 2])) + \
            tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits(
                self.logits_3, self.y[:, 3])) + \
            tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits(
                self.logits_4, self.y[:, 4])) + \
            tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits(
                self.logits_5, self.y[:, 5])) + regularizer)
```

After all of these operations we get 82% test accuracy much better than in the first implementation.

Results

Model Evaluation and Validation:

With the final implementation our model reached 82% test accuracy after 150000 training iterations, I chose this model and hence such model parameters because they gave best test accuracy that I had achieved in comparison with any other combination of parameters that I had tried to use.
I think this result is maximum that I can achieve with my computer. I tried to use more convolutional layers in my neural network but my GPU was too slow with complicated neural networks.
It worth noting that convolutional neural network has a lot of parameters which we can tune, but I couldn't play with parameters a lot, again because of computational power of my computer.
Also for training model I used only the part of the training data(remember that I don't use extra data) so from my opinion of view we need to add more training data before we can use our model on the real data.
But what I want to say is that we have done good basis for more complicated neural net. We can easy expand our solution if we will have more powerful computer.

Justification:

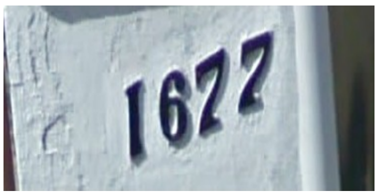
My benchmark is 90%, while I have got only 82% so unfortunately I haven't achieved the benchmark.
As I mentioned above I think that this result is maximum that I can achieve with my computer.
But I think that my model can partially solve the problem that I specified. But in order to completely solve the problem we need to get around 98% accuracy because for the purpose of building a map, it is extremely important to have at least human level accuracy.

Conclusion

Free-From Visualization:

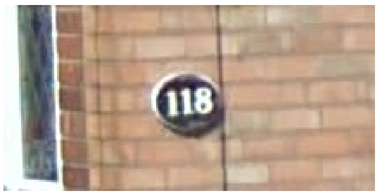
In this section I want to provide a few examples of predictions of our convolutional neural net.
Firstly, I want to consider one interesting example that I provided before at the Data exploration section

Figure 14: incorrect prediction of 1622



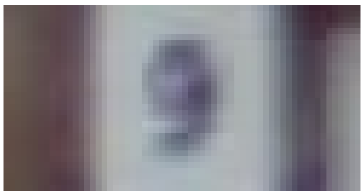
At the Figure 14 we can see that neural net has made a mistake in this prediction. Remember, that at the Data exploration section I referred this example to non-standard fonts characteristic and I mentioned that the last two digits(seven and seven) might resemble two and two, so because of the presence of the non-standard font at this image, neural net gave inaccurate prediction for this example.

Figure 15: correct prediction of 118



At the Figure 15 we can see that neural net has made a correct prediction. I think that this example fit to the 'understandable format' characteristic(from Data exploration section) and for neural net it hasn't been a problem to recognize this number.

Figure 16: correct prediction of 9



At the Figure 16 we can also see the correct prediction, but it worth noting that despite of the fact that it is very blurry image, our neural net has made the correct prediction.

Reflection:

Ok,now I want to summarize the entire process that I used for my project:
At the beginning, we explored data to emphasize the interesting characteristics in them. Then we started to work on the data preprocessing, we prepared data to right format to feed its into neural network. After that we worked on the neural network architecture and then changed parameters to improve the performance of our model.
The main interesting aspect that I met in my project is that when I threw out expansion of the bounding boxes from my preprocessing step I got much better results, that was very surprising for me.

The main difficult part of the project was to define the cost function, I tried the different approaches and spent a lot of time on this
My final model didn't reach my expectations because it didn't achieve the benchmark. And as I mentioned above I think that my model can only partially solve the problem.

Improvement:

There are methods that can improve our model:
Firstly, the main improvement that we can make is to add more training data for training our model. As we remember we didn't use extra data for training
Secondly, due to the computational power of computer we couldn't try a lot of hyperparameters. So we need to spend more time on tuning them.
Thirdly, to improve our model we also need to use more complex convolutional neural net, we need to add more convolutional layers.
Fourthly, we were training our model only for 6-8 hours, but to improve our model we should train more.
With these improvements I believe that we can achieve the benchmark and even to overcome it.

References:

(1)http://static.googleusercontent.com/media/research.google.com/en//pubs/archive/42241.pdf

(2)http://cs231n.github.io/convolutional-networks/

(3)http://neuralnetworksanddeeplearning.com/chap6.html

(4)https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf

(5)http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

(6)http://www.deeplearningbook.org

(7)http://deeplearning.stanford.edu/tutorial/