Image Classification Using Convolutional Neural Networks

Kaushik Krishnan[*], **Chukwuemeka Okoro**[†] and **Chenchao Shou**[‡]

Abstract

In this project, we trained a convolutional neural network (CNN) that was able to classify a photo into one of three labels: *blurred, crisp-scenic or crisp-human* with high accuracy (91%). In order to achieve the optimal performance, we designed 4 CNN architectures and tuned 4 CNN parameters: (1) initial learning rate (2) input image size (3) size of convolutional filter (4) number of training steps. Through extensive experiments, we found that adjusting learning rate is crucial to achieving the optimal CNN performance.

1 Introduction

Image classification and organization plays an important role in everyday life; from remote sensing and medical diagnosis to art collation and personal album management. An automatic photo classifier becomes a time-saving and efficient tool for these situations. As a concrete example, suppose you go on a trip, take hundreds of photos and would like to share with your friends or on an online social media. Manually browsing through the pictures in order to select good ones can be a laborious process. With the automatic classifier in place, you can select photos from a much smaller portion of photos so that the efficiency of selecting photos is improved.

The use of machine learning tools provides a good platform for the automation of the image classification process. Particularly, convolutional neural networks (CNNs) have been shown to be best suited for this task, achieving best performance (lowest testing error) for competitions and in practice [1, 2].

In this project, we perform an experimental study of image classification using CNN. Specifically, we train a convolutional neural network on thousands of images, with each having one of three labels: *blurred*, *crisp-scenic* and *crisp-human*. Here a crisp-scenic photo is defined as a crisp image that does not contain any human in the picture whereas a crisp-human photo refers to a crisp image that have human in it. In this way, the three labels are mutually exclusive. This classification problem is a standard multi-class classification problem.

The goal of the project is to design and train a convolutional neural network to achieve high classification accuracy on testing samples. The CNN algorithm is implemented in Python, using TensorFlow software libraries. To achieve optimal classification performance, we have tested 4 CNN architectures and tuned 4 CNN parameters. The best accuracy that we were able to achieve was about 91%.

2 Related work

Convolutional Neural Networks are not the only means that have been used for large scale image classification purposes. Researchers have found that traditional multiclass classification approaches did not perform well for image classification purposes due to the extreme high dimensionality of the feature space [3]. In that paper, a linear SVM based method that used a re-mapped feature space was then proposed, and it was shown to outperform the Gaussian Radial Basis Functions (RBF).

One of the challenges we faced in this project was

[*] kkrishn3@illinois.edu
[†] koro2@illinois.edu
[‡] cshou3@illinois.edu

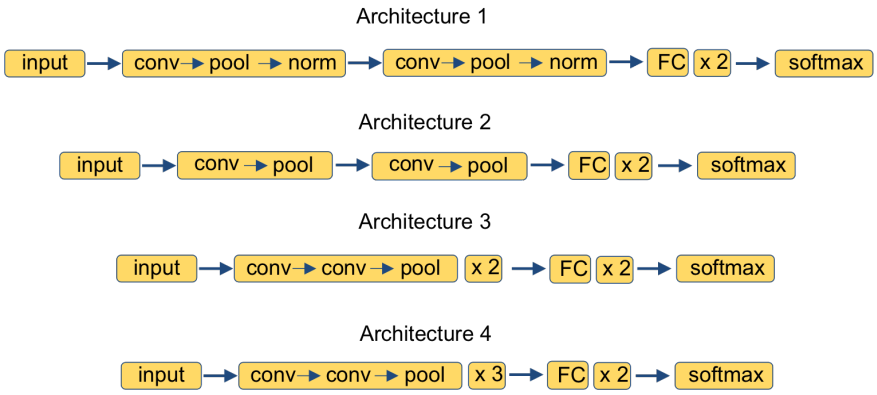

Figure 1: Four CNN architectures. For architecture 1 and 2, the depth of convolution layer is 64. For architecture 3 and 4, the depth of convolution layer is 32. "FC" stands for the fully connected neural network layer. The output is the softmax regression.

deciding on an optimal image resolution since high image resolutions would blow up the feature space. In [1], a deep convolutional neural network was trained on image resolutions of 256×256 and classified 1.2 million images using deep CNNs, where the number of classes were 1000. With a CNN that had 60 million parameters and 650,000 neurons they were still able to secure a top 5 error rate of 15.3%. This justifies our use of convolutional neural networks for the purpose of classifying 4477 images into three labels. On the implementation side, furthermore, they used a very efficient GPU implementation as such large neural networks cannot be efficiently ran on a single CPU.

One of the three labels that we classify images into is *blurred*. Yan [2] used a two-stage Deep Belief Networks to not only classify whether an image is blurred or not, but also classify the type of blur. For example, blur due to camera disturbance (*motion blur*), blur due to atmospheric disturbance (*gaussian blur*) or lens aberrations (*out of focus blur*) are some of the labels they are using. In their work they use a number of different techniques like Support Vector Machines, Neural Networks, Convolutional Neural Networks and Deep Belief Networks. The testing accuracies they reported were 80.8% for support vector machines, 91.7% for neural networks, 96.8% for CNNs and 98.2% for deep belief networks.

These good results obtained by [2] on CNNs combined with the results reported by [1] for extremely high dimensional datasets convinced us to use the CNN procedure for our image classification task.

3 Experiment

3.1 Dataset

The blurred images and the crisp scenic images were downloaded from CERTH Image Blur Dataset[1]. The crisp human images were downloaded from Leeds Sports Pose Dataset Repository[2]. Fig. 2 shows some examples of blurred, crisp-scenic and crisp-human photos. The entire dataset was splitted into training (3732 images) and testing (745 images) so that the number of training samples was roughly 5 times the number of the testing samples. All the images were resized to a uniform size of 32 by 32 and were saved as row-majoring binary files before feeding into CNN algorithms.

3.2 CNN architecture design

Designing a good CNN architecture is crucial to the success of the CNN classifiers. In this project, we designed four different CNN architectures: one from TensorFlow CNN Tutorial [4] and the other three based on the recommendations in the online Stanford CNN notes [5]. Figure 1 shows the four architectures considered in this project. Each architecture comprises of a stack of convolution layers, max-pooling layers (or normalization layers) and fully connected layers.

[1] `http://mklab.iti.gr/project/imageblur`. When constructing crisp scenic dataset, 3 images that contained human were manually removed from the dataset.

[2] `http://www.comp.leeds.ac.uk/mat4saj/lsp.html`

Figure 2: Photo samples from the dataset. (a) a crisp-scenic photo (b) a crisp-human photo (c) a blurred photo.

3.3 CNN parameters

a **Initial learning rate:** The convolutional neural network is trained using stochastic gradient descent (SGD) algorithm with mini-batches. To achieve the best possible performance of SGD, the learning rate is often set to decay exponentially according to Eq. 1

$$R = R_0 \gamma^N, \tag{1}$$

where R is the learning rate of SGD. R_0 is the initial learning rate. N is the number of training steps. γ is the learning rate decay factor, which was set to be 0.1 as suggested by the TensorFlow CNN Tutorial [4].

b **Input image size:** When the images are inputted to CNN algorithm, they are first processed before being fed into a convolutional layer. This image processing includes randomly flipping images from left to right, image whitening, randomly cropping the images to a specified size, etc. (see [4] for details). So the image that the first convolutional layer sees may be smaller than the image initially inputted to CNN algorithm, which is 32 by 32.

The input image size parameter here refers to the size of the image after cropping (i.e. the size of the image that is actually seen by the first convolutional layer). Note that if the input image size is 32, it is essentially using the entire image (i.e. without cropping).

c **Size of convolutional filter:** This parameter determines the size of the receptive field of the convolutional layer. In this work, we consider two filter sizes that are most commonly used by people [5]: 3×3 and 5×5.

d **Number of training steps:** This is the number of mini-batches used in the stochastic gradient descent algorithm, determining how long a CNN classifier is going to be trained.

3.4 Implementation

The CNN algorithm was implemented using TensorFlow software package. The training and evaluation of CNN classifiers were primarily done using Amazon Web Service (AWS) c4.2xlarge instances[3]. All the source codes of this project are publicly available[4].

3.5 CNN optimization

In order to find the optimal CNN classifier, we need to jointly optimize over 4 CNN architectures and 4 CNN parameters. Since the training of CNN is a time consuming process, it's not realistic to perform full-factorial experiments to find the best CNN.

Instead, we propose a greedy optimization approach, where we first conduct full-factorial experiments on 4 CNN architectures and 2 CNN parameters (size of convolution filter and input image size) with fixed number of training steps (= 1000) and initial learning rate (= 0.1). This is equivalent to testing $4 \times 2 \times 2 = 16$ different conditions. The number of training steps in this first step is relatively small so it can be viewed as a preliminary step for optimization.

[3] https://aws.amazon.com/ec2/instance-types/#burst

[4] https://github.com/gradcshou/CS446-PhotoKlass

Next, we choose the top 2 configurations from the 16 experiments as described previously and optimize the initial learning rate with a larger number of training steps (= 5000) for each of the two configurations. Lastly, we select the better of the top performers of the two configurations as our best classifier.

4 Results

Table 1 shows the prediction accuracy on the test dataset for different CNN architectures, filter sizes and input image sizes. As we can see, with all the other parameters fixed, the performance of input image size = 32 is always better than that of input image size = 24. This shows that using the entire image is better than using cropped image. This is interesting because image cropping is suggested in the TensorFlow CNN Tutorial [4] while our experiment shows that this may not be a good strategy. Intuitively, this result is not so surprising because generally speaking, some information of the image is lost as a result of cropping. The top two configurations (marked bold in the table) that we found during the preliminary optimization step are:

C1. CNN architecture 1, convolutional filter size = 3×3, input image size = 32×32

C2. CNN architecture 3, convolutional filter size = 3×3, input image size = 32×32

With the top two configurations determined, we set the number of training steps = 5000 and varied the initial learning rate for fine-tuning.

Figure 3 shows the training loss curve for CNN configuration 2. We observe that when learning rate is 0.05, the loss function decreases initially but overshoots to large values towards the end. This shows that the stochastic gradient descent algorithm can become unstable when the learning rate is large. Indeed, obtaining a good training loss curve, similar to the blue curve shown in Fig. 3, is the first step towards achieving the optimality of a CNN classifier.

Figure 4 shows the training and testing accuracies for CNN configuration 1 when the initial learning rate = 0.05. We see that the training accuracy is significantly larger than the testing accuracy. In fact, the training accuracy reaches 100% after about 2000 training steps. The large discrepancy in the training and testing accuracy suggests a strong overfitting.

Table 1: Preliminary results of optimizing CNN

Case	CNN Arch.	Filter size	Input image size	Accuracy
1	1	3	24	0.871
2	**1**	**3**	**32**	**0.889**
3	1	5	24	0.858
4	1	5	32	0.888
5	2	3	24	0.867
6	2	3	32	0.874
7	2	5	24	0.861
8	2	5	32	0.876
9	3	3	24	0.824
10	**3**	**3**	**32**	**0.892**
11	3	5	24	0.814
12	3	5	32	0.880
13	4	3	24	0.699
14	4	3	32	0.885
15	4	5	24	0.750
16	4	5	32	0.883

The overfitting is much alleviated when the learning rate is reduced to 0.01, as illustrated in Fig. 5. Here we observe that the training accuracy is slightly above the testing accuracy, indicating that initial learning rate = 0.01 is close to optimal.

Table 2 shows the accuracy on the test dataset for the two selected configurations with different learning rates (0.005, 0.01, 0.05). Here the number of training steps is fixed to be 5000. As we can see, for both configurations, initial learning rate = 0.01 gives the highest accuracy. This shows that finding the right learning rate is critical to the success of the optimization of CNN. The performance of CNN can suffer from undertraining (i.e. learning rate being too small) or overtraining (i.e. learning rate being too large). The best CNN classifier we've obtained has an accuracy of about 91%, as marked bold in Table 2.

5 Conclusion and future work

In this project, we trained a convolutional neural network (CNN) that was able to classify a photo into one of three labels: *blurred, crisp-scenic or crisp-human* with high accuracy (91%). We designed 4 CNN architectures and tuned 4 CNN parameters: (1)

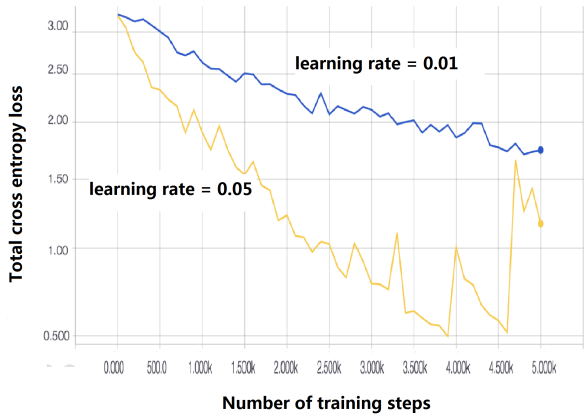

Figure 3: Training loss curves for different initial learning rates for CNN configuration 2. Total cross entropy loss is the objective to be minimized in the stochastic gradient descent (SGD) algorithm. The number of training steps is the number of mini-batches in SGD.

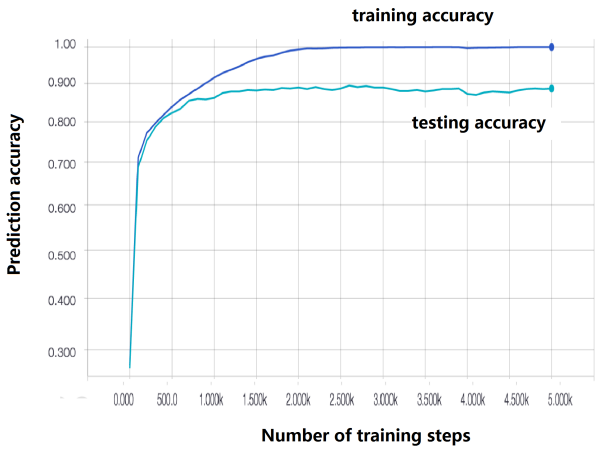

Figure 4: Accuracy curves for CNN configuration 1 when initial learning rate = 0.05. Training accuracy is significantly higher than testing accuracy, suggesting a strong overfitting.

Table 2: Final optimization results of the CNN classifier

Case	Config.	Initial learning rate	Accuracy
1	1	0.005	0.874
2	**1**	**0.01**	**0.905**
3	1	0.05	0.885
4	2	0.005	0.858
5	2	0.01	0.878
6	2	0.05	0.760

initial learning rate (2) input image size (3) filter size (4) number of training steps, in order to achieve the optimal classification performance. Through extensive experiments, we learned that

1. Optimizing CNN can be a quite lengthy process due to the long training time and the numerous parameters to tune.

2. Finding the right learning rate is crucial to achieving the optimal CNN.

A major hindrance to this work was limited computing resources. This restricted us from performing cross validations to obtain average accuracy as well as the confidence interval of the accuracy. It also limited the number of configurations/parameters we could experiment with. Having access to more powerful computing resources would yield faster experiment cycle, hence increasing the chances of obtaining a better model.

Due to the long training time of CNN, we were not able to train on a very large number of samples. This could affect generalization, especially for the samples outside the distribution of the dataset. With richer and more diverse samples, we believe the CNN classifier can be more powerful in classifying an arbitrary random image (for instance, say, an image randomly pulled from the internet).

Acknowledgments

We would like to thank Professor Matthew West for providing the access to Amazon Web Service (AWS) Illinois account so that we were able to use AWS instances for training and optimizing CNN classifiers.

References

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *Advances In Neural Information Processing Systems*, pages 1–9, 2012. ISSN 10495258. doi: http://dx.doi.org/10.1016/j.protcy.2014.09.007.

[2] Ruomei Yan. Image Blur Classification and Parameter Identification using Two-stage Deep

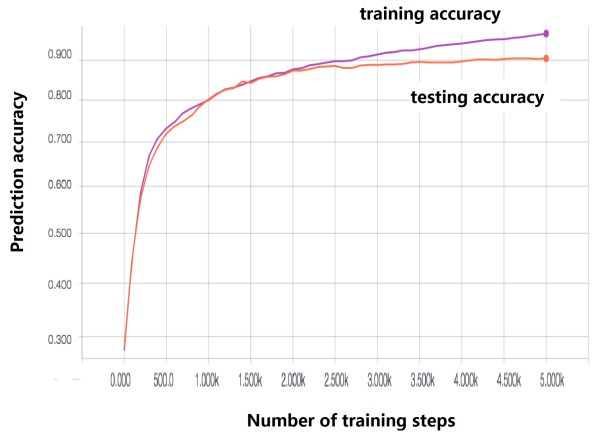

Figure 5: Accuracy curves for CNN configuration 1 when initial learning rate = 0.01. Training accuracy is slightly above testing accuracy, suggesting a slight overfitting. Initial learning rate = 0.01 is close to optimal.

Belief Networks. *Bmvc2013*, pages 1–11, 2013. doi: 10.5244/C.27.70.

[3] O Chapelle, P Haffner, and V N Vapnik. Support vector machines for histogram-based image classification. *Neural Networks, IEEE Transactions on*, 10(5):1055–1064, 1999. ISSN 1045-9227. doi: 10.1109/72.788646.

[4] Tensorflow convolutional neural networks tutorial. `https://www.tensorflow.org/versions/r0.12/tutorials/deep_cnn/index.html`.

[5] Stanford convolutional neural networks notes. `http://cs231n.github.io/convolutional-networks/`.