Machine Learning Engineer Nanodegree

Capstone Project

Prateek Gupta
July 7th, 2018

1. Definition

Domain Background

In 1995's Clueless, you may recall Cher Horowitz using cutting-edge software to select her plaid ensemble. Cher's machine could identify chic head-to-toe looks, adding a small dose of sci-fi to the rom-com classic. Twenty-two years later, the 90's fiction movie is closer than ever to reality: Artificial intelligence in fashion is here.

Online fashion will be transformed by a tool that understands taste. Because if you understand taste, you can delight people with relevant content and a meaningful experience. Outfits are the asset that would allow taste to be understood, to learn what people wear or have in their closet, and what style each of us like.

Problem Statement

Customers can purchase wide variety of fashion products like T-shirts from an e-commerce platform. One of the major fashion attributes of a T-shirt is the type of graphic present on it. Graphic type plays a major role in identifying T-shirt categories and determining its saleability. Some example graphic types are:


Biker

Varsity


Music

The challenge is to identify the graphic type of a T-shirt. Take in an image as an input and give graphic type of the T-shirt as the output. The model can be used during cataloguing new T-shirts on any e-commerce platform. With graphic type present as an attribute, customers will be able to search and filter for different type of T-shirts.

Evaluation Metrics

The metric used for this project is multi-class logarithmic loss (also known as **categorical cross entropy**). Here each image has been labelled with one true class and for each image a set of predicted probabilities should be submitted. **N** is the

number of images in the test set, **M** is the number of image class labels, $\mathbf{y_{ij}}$ is 1 if observation **i** belongs to class **j** and **0** otherwise, and $\mathbf{p_{ij}}$ is the predicted probability that observation **i** belongs to class **j**. A perfect classifier will have the log-loss of 0.

Multiclass log-loss punishes the classifiers which are confident about an incorrect prediction. If the class label is 1 (the instance is from that class) and the predicted probability is near to 1 (classifier predictions are correct), then the loss is really low, so this instance contributes a small amount of loss to the total loss and if this occurs for every single instance (the classifiers is accurate) then the total loss will also approach 0. On the other hand, if the class label is 1 (the instance is from that class) and the predicted probability is close to 0 (the classifier is confident in its mistake), as *log(0)* is undefined it approaches infinity, so theoretically the loss can approach infinity.

2. Analysis

Data Exploration

Each image correlated with a link is in the *jpeg* format. The resolution of each image is **1080 x 1440**, with the DPI of **72 pixels/inch**. The dataset features **23** different classes of graphic types for T-shirts. Training set includes **8265** labelled images (**train**=6675; **valid**=1580) and the testing set includes **3562** rows. Images are guaranteed to be of fixed dimensions. The dataset was labelled by identifying the graphic type for each T-shirt captured from various angles.

Features

The features I'm going to use are represented by each individual pixel in the images. The CNN classifier works with coloured image, so there is a third dimension that has to be added to each pixel. This third dimension is an array with 3 elements, each one representing the Red, Blue and Green channel of the pixel.

Labels

Each image in the dataset is associated with a label, which is an integer ranging from 0 to 22. This label is actually an index of the class description in a list that contains the 23 classes.

Label Distribution

The dataset is imperfectly distributed among the 23 labels. The no. of images ranges from *tens* for a few labels to *thousands* for others.

Algorithms and Techniques

The main goal of this project is to build a Convolutional Neural Network (CNN) classifier to predict image labels. CNN algorithm consists of several convolution (CNV) operations followed of the image sequentially which is followed by pooling operation (PL) to generate the neurons feed into fully connected (FC) layer. I'll be

using the Keras library with a Tensor Flow backend to achieve our goal.

Convolutional Layer (CNV Layer): Convolution, in general, is an operation of filtering the input data. In image recognition context, 2D filters are used to process data at different layer to generate the features. These filters are pre-defined and their coefficients are computed during training (which is described later). Figure 1 illustrates a simple 2D convolution operation where the filter is 3x3 (where the filter taps are w1 to w9) that is applied to 2D data (data can be input image or intermediate feature maps).

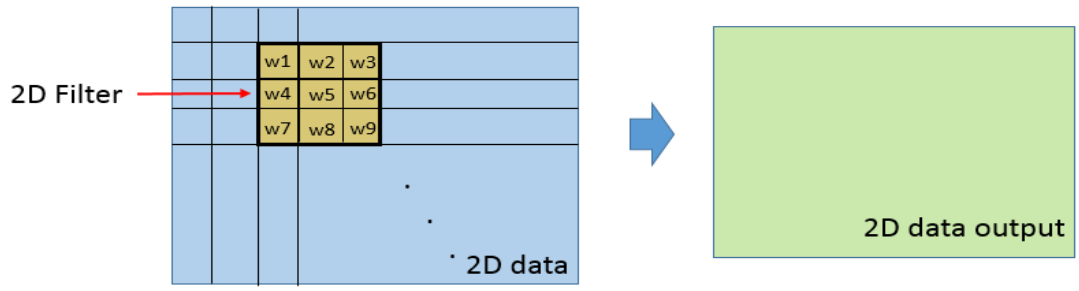


Figure 1: Convolution Operation

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

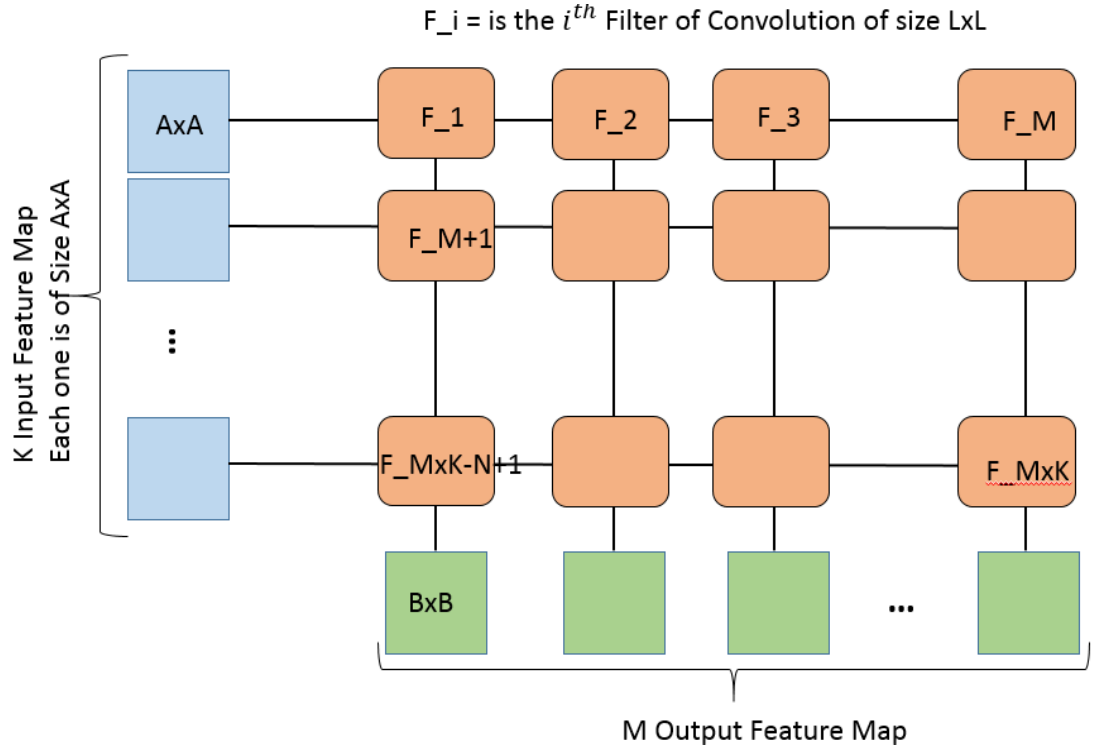
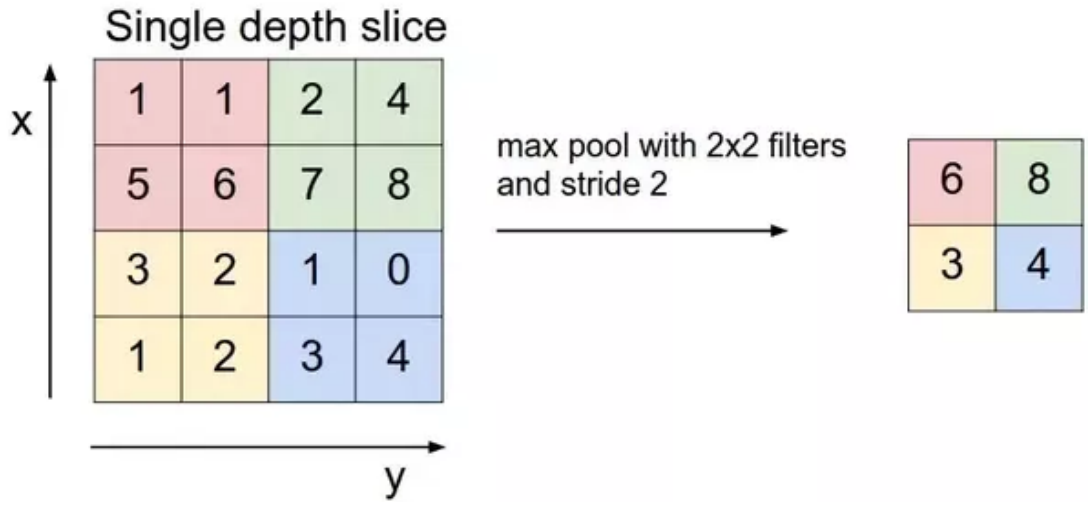


Figure 2: Input and Output Feature Map with K input Feature Map and M Output Feature Map and LxL Convolutional Filter

After the Convolution layer is applied, I can execute Pooling. This layer will reduce the image a little bit, by creating smaller, aggregated sections of the stacks. Pooling will walk through the zones of the image and pick the highest scored pixel on that zone. This process helps the detection process to assimilate variances on the selected feature, like small differences in positioning and tilting. Window size that is used during subsampling is also 2D such as 2x2 or 3x3. If the input data size is (W,H) then the size after pooling will be (W/2,H/2) if ½ subsampling is used. For reference, it is common to set the window size and stride of the pooling to the same value.



I can use a special layer, provided by Keras, to reduce the risk of over fitting. The Dropout layer will randomly set a fraction of the inputs to 0. This will force the algorithm to apply the learning process again for those inputs.

There are other types of layers that can be applied to the network, and the same layer type can be at different parts of the network design. The choice of which layers to use, and their order, how many neurons to add to each layer are part of the art of building a CNN. More skilled engineers can even build custom layers, tailored specifically to the problem they're dealing.

The output of each layer becomes the input for the next. At first, each neuron from a given layer is connect to all the neurons from the next layer. This creates a complex network of connections, where each connecting line has a specific weight that controls the significance of that particular output to the next layer. One of the main jobs of the CNN algorithm, implemented by Keras, is to define those weights.

The training process applies a somewhat simple technique to find the optimal weights for each neuron, called **backpropagation**. The algorithm will work with already labelled images (**train** set) and calculate the error rates for each prediction. Based on the error rate, new weights that reduce that rate will be define. After the training is complete, some neurons might have been given weights equals to 0. The output connection for those neurons are ignored, resulting on a simpler network.

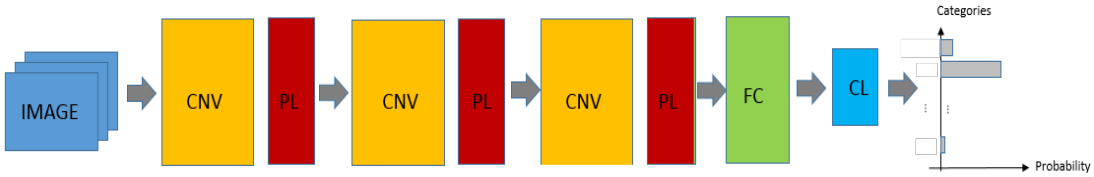


Figure 3: CNN architecture with 3CNV and 1 FC layers

Benchmark Model

Random choice: We predict equal probability for a T-shirt to belong to any class of the 23 classes for the naive benchmark.

A well-designed convolutional neural network should be able to beat the random choice baseline model easily. However, due to computational costs, it may not be possible to run the transfer learning model with VGG-16 or Resnet-50 architecture for sufficient number of epochs so that it may be able to converge.

3. Methodology

CNN Classifier

Data Preprocessing

Before feeding the dataset images into our Sequential Model, there is a minor preparation. I've normalized the data, dividing the RGB values by the maximum value, 255. This results on decimal numbers from 0 to 1. I'll also need to perform some resizing and scaling during the prediction phase. That preparation is needed, since the training set was composed by 1080x1440 images.

Implementation

I've used Keras' Sequential Model. This model allows us to use many processing layers that run sequentially and improve the accuracy of the classifier on each iteration, or epoch. In my case I've implemented a pattern using the following steps:

1) Convolution2D and MaxPooling2D (repeated 3 times), increasing the number of feature maps from 16 to 32 and, finally, to 64.
2) Dropout layer
3) In order to find the parameters that make the value of the loss function as low as possible, it is important to find the optimal value of the parameter. Solving this problem is called optimization. I use RMSprop optimizer (divide the gradient by a running average of its recent magnitude) for this purpose.

The execution of layers is applied **5** times.

Training Time

The training took too long (40 min per epoch) to complete. I couldn't run the algorithm on a more powerful machine due to *personal* constraints.
In the end, I had to cut down the dataset I initially started with (as mentioned in the proposal submitted earlier against the project). The result, however, was a

classifier with 18.90% accuracy, which is still not up to production standards, but suitable for this project.

Refinement

As a final model, this project was carried out using the transfer learning method. The following is a detailed structure of the ResNet50 model selected for transfer learning. ResNet50 introduces is residual learning. In residual learning, instead of trying to learn some features, we try to learn some residual. Residual can be simply understood as subtraction of feature learned from input of that layer. ResNet50 does this using shortcut connections (directly connecting input of nth layer to some (n + x)th layer. It has proved that training this form of networks is easier than training simple deep convolutional neural networks and also the problem of degrading accuracy is resolved.

1. Benchmark Model (Random): 4.35%
2. CNN Model: 18.90%
3. Transfer Learning with ResNet50 Model: 84.75%

4. Results

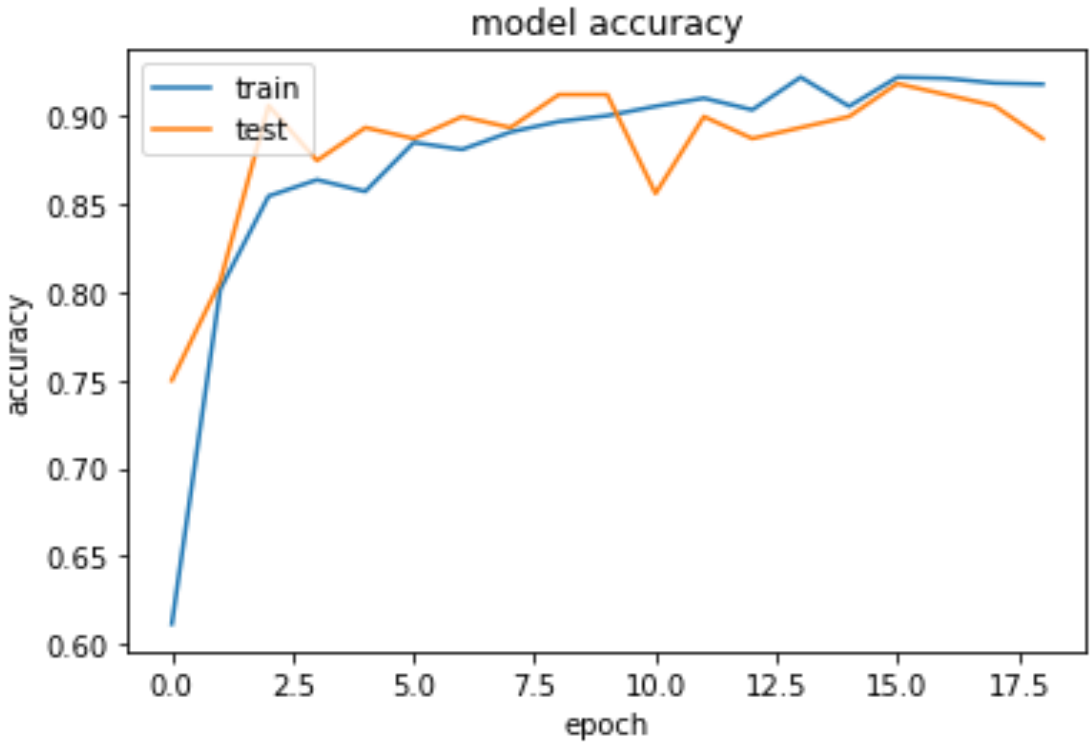
Model Evaluation and Validation

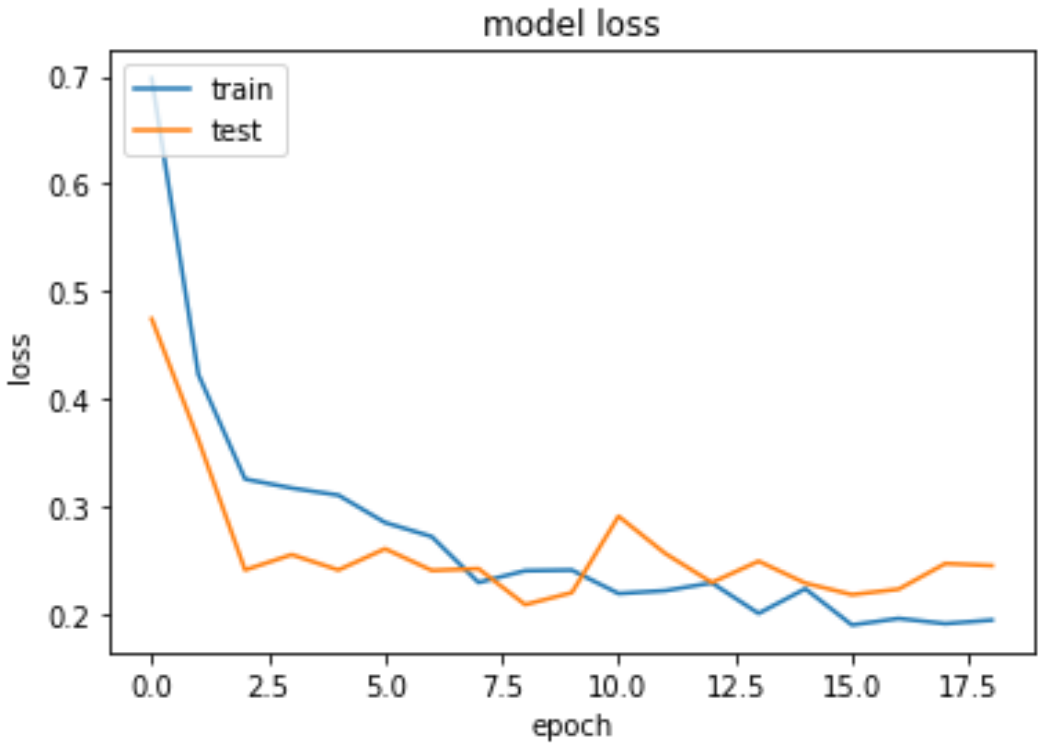
In transfer learning, we take a pre-trained model (network + weights), and then remove the FC network, and construct our own in place of it. Doing so, it will remove the pre-trained weights at those layers. Now the original network before the FC network is frozen so that, when the training is started on the new data set, only the newly added FC network is trained. Here the input to the FC network is called the **bottleneck features**. They represent the activation map from the last convolutional layer in the network.

Hence with frozen weights in the main network, we will get to use the pre-trained weights to get important feature activations as bottleneck features into our newly added FC network, and now this new FC network (trained on our data) gives us the required inference as per training.

Justification

The final results are found stronger than the benchmark result reported earlier. The accuracy of the benchmark model was 4.35%, but it improved to 84.75% accuracy with two models' development.
The following two graphs show the performance of the final model. Accuracy *almost* always increases with each epoch. The loss value indicates how good or bad a particular model is each time the optimization is repeated. Ideally, the loss value is expected to decrease after each or several repetitions.


model accuracy
train
test
accuracy
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
epoch

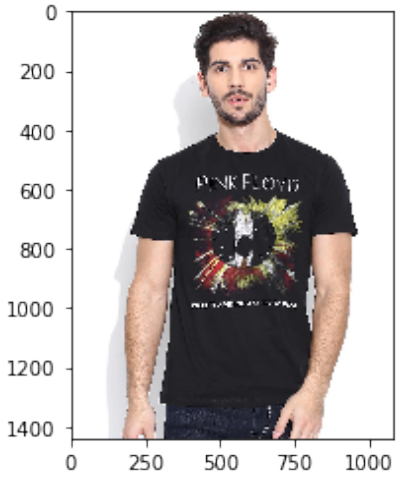

model loss
train
test
loss
0.7
0.6
0.5
0.4
0.3
0.2
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
epoch

5. Conclusion

Free-Form Visualization

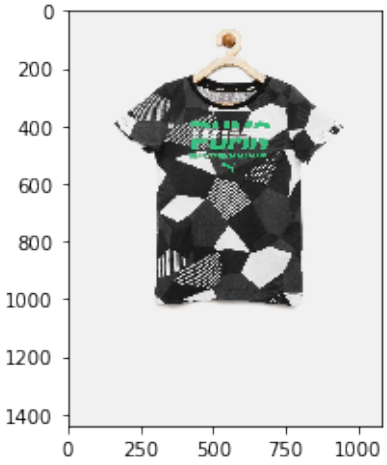
The predicted results of the final model were compared with the actual label values. Let's look at the predictions for the four image data below. It can be seen that *graphic type* is **different** from the actual label value. In other words, **not** all of the following results are successful predictions.

```
I'm sorry, Dave. I'm afraid I can't do that!
(Correct: Shirt)
```



```
That's a t-shirt. Graphic type: Solid
(Incorrect: Music)
```

```
I'm sorry, Dave. I'm afraid I can't do that!
(Incorrect: T-Shirt with a Collar)
```



```
I'm sorry, Dave. I'm afraid I can't do that!
(Incorrect: T-Shirt on a Hanger)
```

Reflection

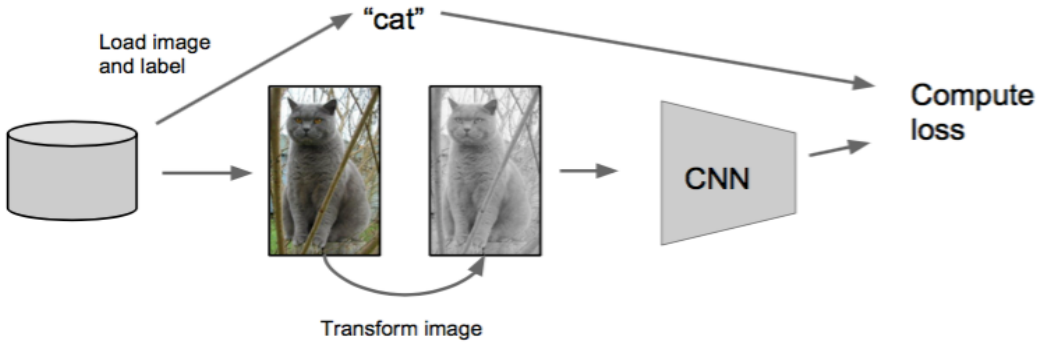
Given that I work in a fashion e-commerce company, I found an interesting problem (with the data set) in the organisation itself. This project is a *classical* image classification problem. So, I decided to use a CNN model. I then navigated through the data set, which turned into a cumbersome task as I could not pre-process and train the on my machine or on an AWS instance. I had to cut down on the training, validation, and testing sets to get going. ☹

Visualization gave me a deeper understanding of the data. To improve upon, I tried to change the various parameter values, but decided to use transfer learning using the ResNet50 model in a more innovative way. Comparing the results of this model with previous models, we can see that the performance is better. A key part of this project is finding ways to produce classifiers with good accuracy through insufficient training data. I think this has been solved through transfer learning.

Improvement

Image Augmentation

Image augmentation, or data augmentation, is a method used to improve CNN performance. If you move all the pixels in the cat image one space to the right, it looks the same in the human eye. However, because the computer expresses and recognizes the image as a pixel vector, the cat image moved by one pixel is recognized as different from the original image. To solve this problem, we use data augmentation.



To summarize again, data augmentation is a method of changing a pixel without changing the label of the image, and proceeds with learning using the modified data. Also, almost all network models to date have used data augmentation, which is a very common method.

Change Optimizer

I didn't get much time and space to experiment with different optimizers such as SGD (Stochastic Gradient Descent), or Adam (Adaptive Moment Estimation). RMSprop is an unpublished adaptive learning rate, but yet widely used across neural networks. I still have to form a base, and a better understanding for myself to learn, and experiment more with the available optimizers.