**DEEP LEARNING NOTEBOOK**

**Shreenidhi Sudhakar**

Contents

[Convolution Neural Network 3](#_Toc489380651)

[Introduction 3](#_Toc489380652)

[Overview of CNN Layers 6](#_Toc489380653)

[Extraction of Feature – MNIST 7](#_Toc489380654)

[Hyper-Parameter Grid Search 9](#_Toc489380655)

[Custom Image Augmentation 11](#_Toc489380656)

[Image Augmentation 11](#_Toc489380657)

[Custom Image Augmentation 12](#_Toc489380658)

[Histogram Equalization 16](#_Toc489380659)

[Histogram 16](#_Toc489380660)

[Histogram Equalization 17](#_Toc489380661)

[Adaptive Histogram Equalization 17](#_Toc489380662)

[Contrastive Limited Adaptive Equalization 18](#_Toc489380663)

[Learning Rate Scheduler 19](#_Toc489380664)

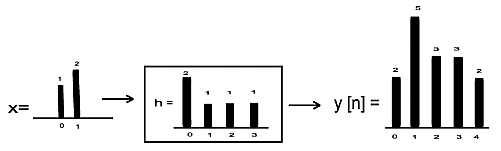
# Convolution Neural Network

## Introduction

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 lines 🡪 Square 🡪 Combination of Squares 🡪 Combinations of squares and other shapes

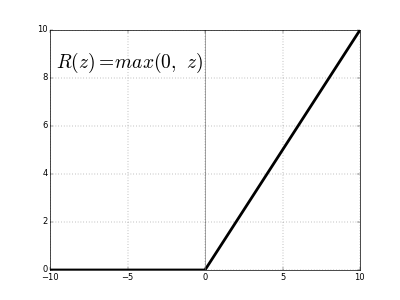
Features are extracted by taking the convolution of portion of data sample under consideration. The amount of data portion that the filter traverses each time is proportional to the stride length and padding value. Data samples may/ may not be subjected to zero padding before convolution.



Convolution of a Signal

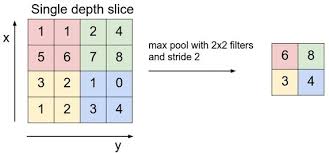
The convolution output is then passed through an activation unit called ReLU (Rectified Linear Unit). This unit converts the data into its non-linear form. The output of ReLU is clipped to zero only if convolution output is negative.

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 have diminished 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.



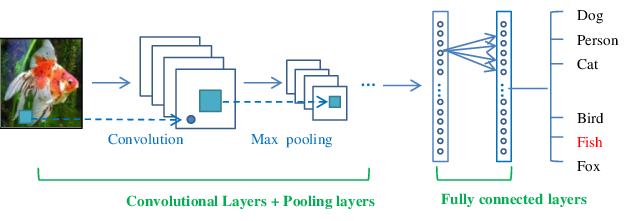
Output of ReLU

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 as 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 multi-layer neural network. Here, each neuron unit acts as feature map that carries information about a unit.

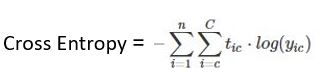


Design of Convolution Neural Network

Dropout layer is used to reduce over-fitting by making the CNN model robust to 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.

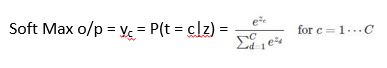
The output of CNN model is calculated using SoftMax function. SoftMax is preferred as it gives the probability of outputs for different classes rather than just >= 0.5 in the case of sigmoid output. The usage of SoftMax function to find output results based on the highest probability of class results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

More info regarding CNN can be found at:

* [CS231n Convolutional Neural Networks for Visual Recognition](https://cs231n.github.io/convolutional-networks/)
* [A Beginner's Guide To Understanding Convolutional Neural Networks](https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/)

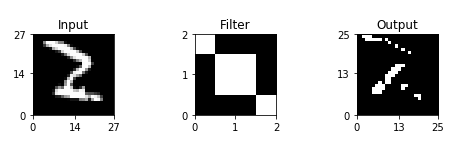
## Overview of CNN Layers

|  |  |  |  |
| --- | --- | --- | --- |
| 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 |

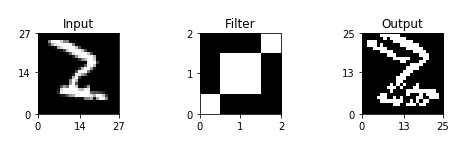
## Extraction of Feature – MNIST

This section gives the reader an intuitive idea of how convolving an image with a kernel will result in extraction of features from the input image. We consider an image of digit 2 that is being convolved with the same 3\*3 filter flipped by 90 degrees in each example.

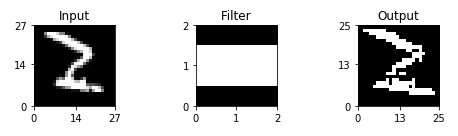
In our experiment, we have assumed threshold of filter as +2. Black pixel is represented as 0 and white pixel is represented as 1. Results of convolution are as seen below.



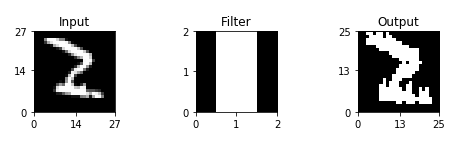
The output of convolution says that filter is good at detecting inner edges. This is shown by distinct edges in image of picture along the diagonal. The 3\*3 filter values used are [[2,-1,-1], [-1,2,-1], [-1,-1,2]]. Thus, when convolution output is greater than threshold of 2 defined, we consider output of that convolution as 1 else 0.



The output of convolution says that the filter is bad at diagonal edges. This is shown by distinct edges in image of picture along the diagonal. The 3\*3 filter values used are [[-1,-1,2], [-1,2,-1], [2,-1,-1]]. Thus, when convolution output is greater than threshold of 2 defined, we consider output of that convolution as 1 else 0.



The output of convolution says that the filter is good at detecting horizontal edges. This is shown by distinct edges in image of picture along the horizontal plane. The 3\*3 filter values used are [[-1,2,-1], [-1,2,-1], [-1,2,-1]]. Thus, when convolution output is greater than threshold of 2 defined, we consider output of that convolution as 1 else 0.



The output of convolution says that the filter is good at detecting vertical edges. This is shown by distinct edges in image of picture along the vertical plane. The 3\*3 filter values used are [[-1,2,-1], [-1,2,-1], [-1,2,-1]]. Thus, when convolution output is greater than threshold of 2 defined, we consider output of that convolution as 1 else 0.

Source Code

* [Feature Extraction](https://github.com/shree6791/Deep-Learning/blob/master/CNN/MNIST/keras/src/Feature%20Extraction.ipynb)

## Hyper-Parameter Grid Search

Hyper-Parameter Grid Search is used to find the best combination of hyper-parameters that help convnet to achieve highest accuracy. This approach is preferred as it’s difficult to configure a convnet accurately as it depends on a lot of hyper-parameters. Few examples of convnet hyper-parameters are, size of kernel, dropout, number of convolution layers, batch size, type of activation function, stride of filters, and so on.

Different combination of convnet hyper-parameters yield different accuracy of inputs. In our experiment, we have assumed dropout and kernel size to be our hyper-parameters that we are going to vary. It’s found that kernel size of 5 and dropout of 0.1 yields the best convnet model. Architecture design of the this best convnet model is as below:

|  |  |  |  |
| --- | --- | --- | --- |
| Layer | Input | Output | Filter |
| Conv\_1 | (?, 28, 28, 1) | (?, 24, 24, 8) | 5\*5 |
| Pool\_1 | (?, 24, 24, 8) | (?, 12, 12, 8) | 2\*2 |
| Conv\_2 | (?, 12, 12, 8) | (?, 8, 8, 32) | 5\*5 |
| Pool\_2 | (?, 8, 8, 32) | (?, 4, 4, 32) | 2\*2 |
| Dropout\_1 | (?, 4, 4, 32) | (?, 4, 4, 32) | N/A |
| Flatten | (?, 4, 4, 32) | (?, 512) | N/A |
| FC\_1 | (?, 512) | (?, 64) | N/A |
| Dropout\_2 | (?, 64) | (?, 64) | N/A |
| Output\_Layer | (?, 64) | (?, 2) | N/A |

Table 1 Architecture Design of Best Model Found Using Hyper-Parameter Grid Search

Note:

* Pool = Pooling
* Conv = Convolution
* FC = Fully Connected

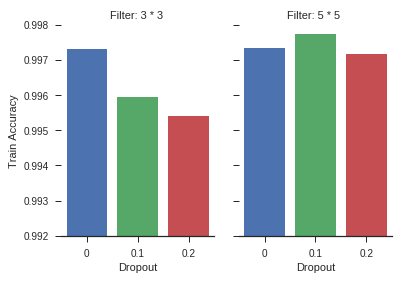
We infer from our experiment that hyperparameters play a pivotal role in choosing a model that enhance its accuracy. In our case, increase in dropout rate and reduction in filter size has been found to reduce accuracy of model. These results can’t be argued to hold good for other convnet models designed for different dataset.

Source Code

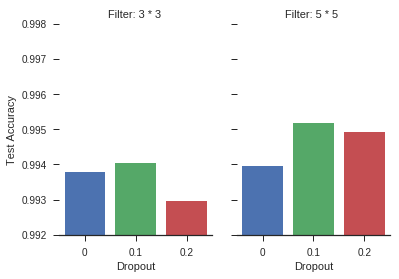
* [Intern Task 1](https://github.com/shree6791/Deep-Learning/blob/master/CNN/MNIST/keras/src/Intern%20Task%201.ipynb)

Accuracy

* Train



* Test



*\* Training accuracy is always greater than or equal to test accuracy.*

More info on Hyper-Parameter grid search can be found at:

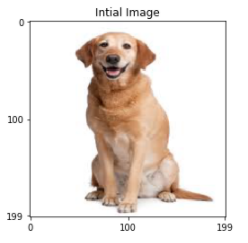
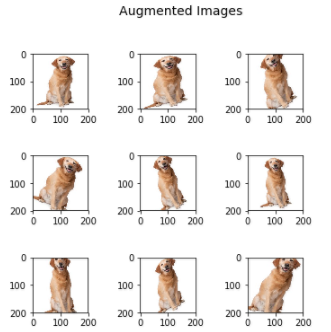
* [How to Grid Search Hyperparameters for Deep Learning Models in Python With Keras](http://machinelearningmastery.com/grid-search-hyperparameters-deep-learning-models-python-keras/)

## Custom Image Augmentation

### Image Augmentation

Image augmentationis atechnique that is used to artificially expand the data-set. This is helpful when we are given a data-set with very few data samples. In case of Data Science, this situation is bad as the model tends to over-fit when we train it on limited number of data samples.

Image augmentation parameters that are generally used to increase the data sample count are zoom, shear, rotation, preprocessing\_function and so on. Usage of these parameters results in generation of images having these attributes during image augmentation. Image samples generated using image augmentation, in general results in increase of existing data sample set by nearly 3x to 4x times.

In Keras, we achieve Image augmentation with help of a function called as ImageDataGenerator. Basic outline of the function definition is as below:

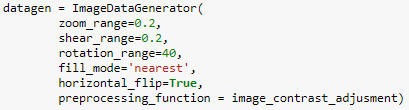
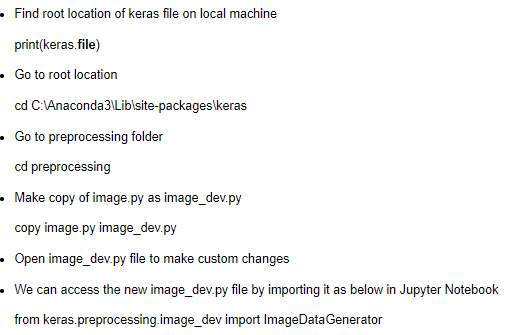


Figure 1 Function to Initialize Data Augmentation Parameters

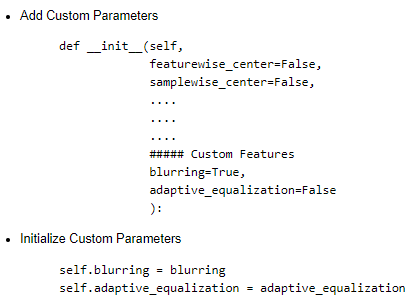
### Custom Image Augmentation

We may want to define our own preprocessing parameters for ImageDataGenerator in Keras in-order to make it a more powerful Image Generation API. We can achieve this by making changes in the Keras image.py file.

For understanding, it’s always good to create copy of image.py and do the changes in the duplicate copy. This is achieved on a Windows machine operating on Anaconda environment by following the below steps:

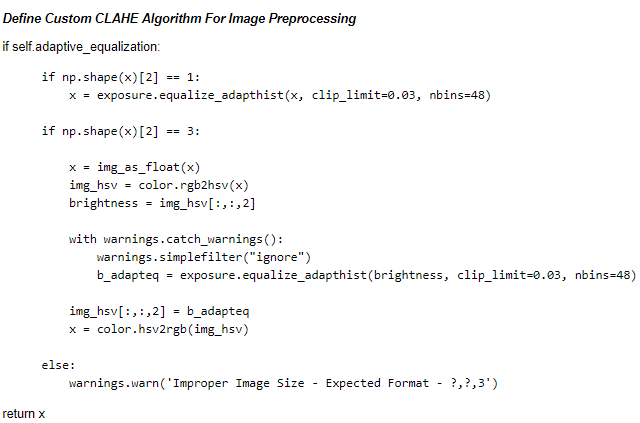


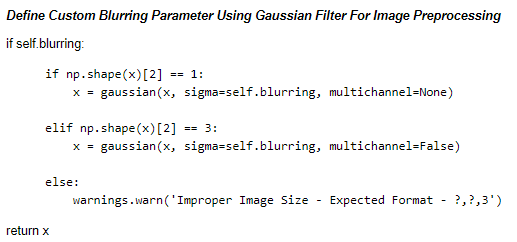
Now add the custom parameters that you will like to see in ImageDataGenerator by following the below steps:



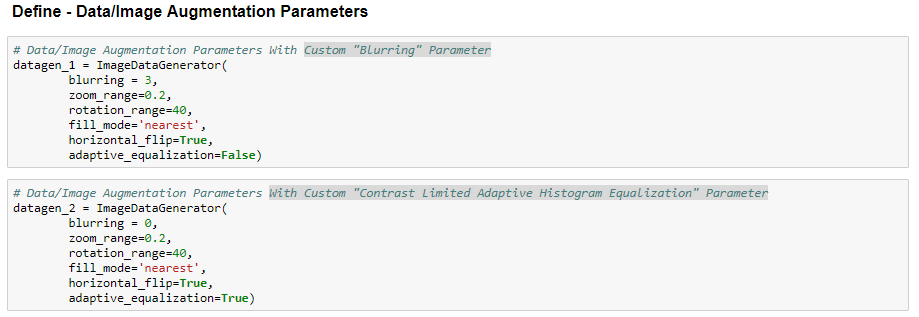
**Define Code To Preprocess Image Using Custom Parameters**

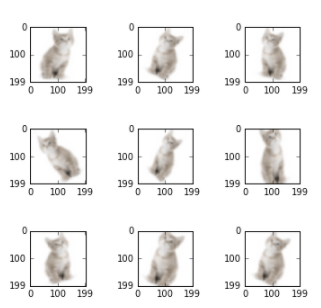
* We can define these code either in def standardize(self, x) or def random\_transform(self, x, seed=None) method.
* The difference b/w the 2 is that, standardize(self, x) is used to return preprocessed batch of images, while random\_transform(self, x, seed=None) is used to return a single preprocessed image
* Let's define the preprocessing code for Contrast Limited Adaptive Histogram Equalization in standardize(self, x) while that for Blurring using Gaussian filter in random\_transform(self, x, seed=None)



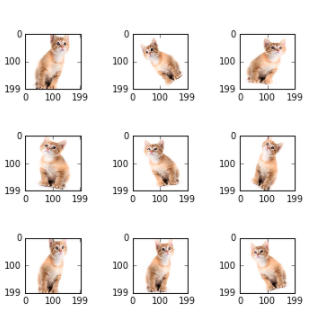


Experimental Results





Augmented Images Obtained Using - datagen\_1



Augmented Images Obtained Using - datagen\_2

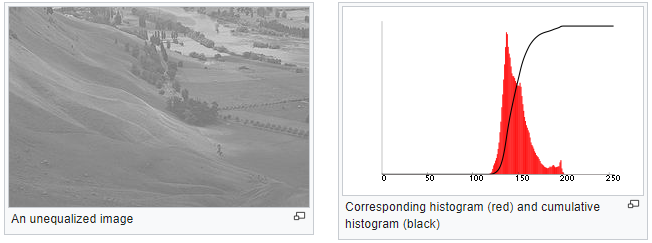
Source Code

* [Image Augmentation](https://github.com/shree6791/Deep-Learning/tree/master/CNN/Cats%20and%20Dogs/keras/src)

## Histogram Equalization

### Histogram

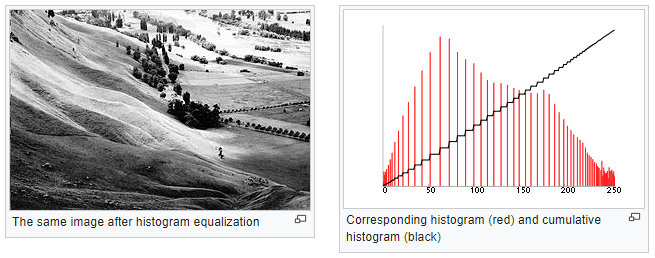
Histogram is a graphical representation of the intensity distribution of an image. In simple terms, it represents the number of pixels for each intensity value considered.



In the above figure, X-axis represents the tonal scale (black at the left and white at the right), and Y-axis represents the number of pixels in an image. Here, the histogram shows the number of pixels for each brightness level (from black to white), and when there are more pixels, the peak at the certain brightness level is higher.

### Histogram Equalization

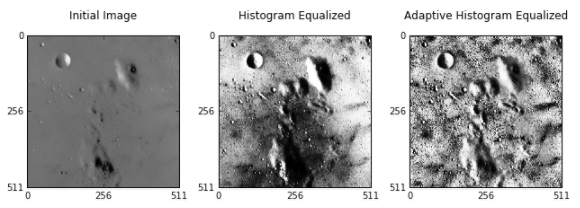
Histogram Equalization is a computer image processing technique used to improve contrast in images. It accomplishes this by effectively spreading out the most frequent intensity values, i.e. stretching out the intensity range of the image. This method usually increases the global contrast of images when its usable data is represented by close contrast values. This allows for areas of lower local contrast to gain a higher contrast.



A color histogram of an image represents the number of pixels in each type of color component. Histogram Equalization cannot be applied separately to the Red, Green and Blue components of the image as it leads to dramatic changes in the image’s color balance. However, if the image is first converted to another color space, like HSL/HSV color space, then the algorithm can be applied to the luminance or value channel without resulting in changes to the hue and saturation of the image.

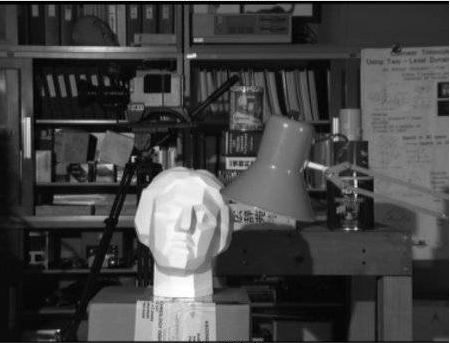
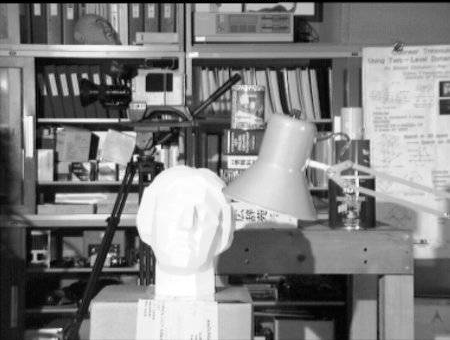
### Adaptive Histogram Equalization

Adaptive Histogram Equalization differs from ordinary histogram equalization in the respect that the adaptive method computes several histograms, each corresponding to a distinct section of the image, and uses them to redistribute the lightness values of the image. It is therefore suitable for improving the local contrast and enhancing the definitions of edges in each region of an image.

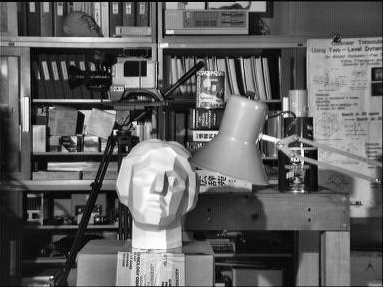


### Contrastive Limited Adaptive Equalization

Contrast Limited AHE (CLAHE) differs from adaptive histogram equalization in its contrast limiting. In the case of CLAHE, the contrast limiting procedure is applied to each neighborhood from which a transformation function is derived. CLAHE was developed to prevent the over amplification of noise that adaptive histogram equalization can give rise to.

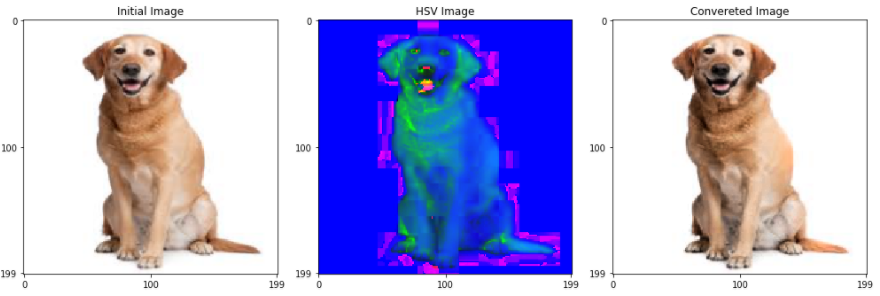
 

Initial Image Histogram Equalized Image

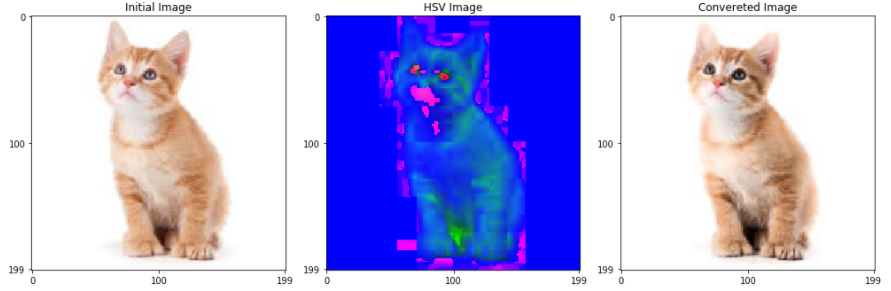


Contrastive Limited Adaptive Equalized Image

Experimental Results



Example 1 CLAHE applied to color image in HSV space and later transformed back to RGB color space



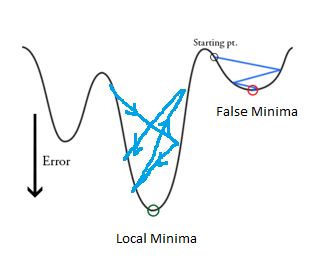
Example 2 CLAHE applied to color image in HSV space and later transformed back to RGB color space

Source Code

* [Histogram Equalization](https://github.com/shree6791/Deep-Learning/blob/master/CNN/MNIST/keras/src/Intern%20Task%201.ipynb)

## Learning Rate Scheduler

In training deep networks, it is helpful to reduce the learning rate as the number of training epochs increases. This is based on the intuition that with a high learning rate, the deep learning model would possess high kinetic energy. As a result, it’s parameter vector bounces around chaotically. Thus, it’s unable to settle down into deeper and narrower parts of the loss function (local minima). If the learning rate on the other hand was very small, the system then would have low kinetic energy. Thus, it would settle down into shallow and narrower parts of the loss function (false minima).



Learning Rate vs Loss Function

The above figure depicts that a high learning rate will lead to random to and fro moment of the vector around local minima while a slow learning rate results in getting stuck into false minima. Thus, knowing when to decay the learning rate can be hard to find out.

We base our experiment on the principle of **step decay.** Here, we reduce the learning rate by a constant factor every few epochs. Typical values might be reducing the learning rate by half every 5 epochs, or by 0.1 every 20 epochs. These numbers depend heavily on the type of problem and the model. One heuristic you may see in practice is to watch the validation error while training with a fixed learning rate, and reduce the learning rate by a constant (e.g. 0.5) whenever the validation error stops improving.

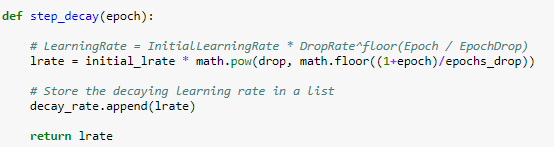
In practice, **step decay** is preferred as it’s easier to interpret hyperparameters like fraction of decay and the step timings in units of epochs. Also, it’s found to provide stabilization to the value of learning rate which in turn helps the stochastic gradient descent to exhibit fast convergence and a high rate of success.

Steps:

* Initialize
* Initial learning rate
* Drop - factor by which learning rate reduces after every epoch
* Epochs Drop – Number of iterations after which the learning rate should reduce



* Write a decay function to compute drop in learning rate after a *time interval = epochs drop*

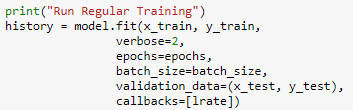


* Initialize **Learning Rate Scheduler** Callback

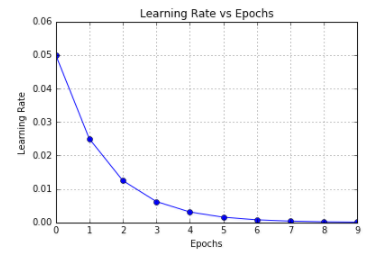
It is a callback that allows to define a function to invoke during program execution. The defined function takes an epoch index as input and returns a new learning rate to use in stochastic gradient descent. When this callback is used, the learning rate specified by stochastic gradient descent is ignored.

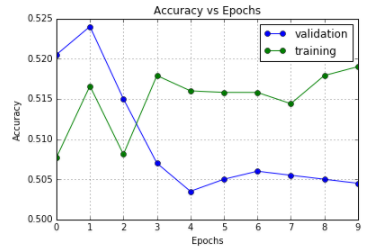
C:\Users\shree\AppData\Local\Microsoft\Windows\INetCache\Content.Word\DLR_5.png

* Train CNN model



Experimental Results





Source Code

* [Adaptive Learning Rate](https://github.com/shree6791/Deep-Learning/blob/master/CNN/MNIST/keras/src/Intern%20Task%201.ipynb)