Using Convolutional Neural Networks to demystify aesthetic works of art

Pujun Bhatnagar
Stanford University
353 Serra Mall, Stanford, CA, 94305
pujun@cs.stanford.edu

Abstract

When one visits a museum and looks at some piece of art, she/he not only understands the subject matter that is being depicted, but is also able to analyze other 'subjective' qualities such as the aesthetic quality, emotional intensity, creativity etc. associated with the piece. And after 1-2 short glimpses, the person is able to come up with a description that aptly describes it. All this is possible because humans have a notion of aesthetic appeal of art works. In this paper, these abilities are transferred to a computer. Different types of Convolutional Neural Networks architectures and Siamese Networks are used to train models that hierarchically understand and intelligently predict the degree of aesthetics of an image. I visualize the learned weights in order to understand the underlying features that makes some art works so appealing and am able to perform better than the state of the art techniques that have been previously used to tackle this problem.

1. Introduction

The field of Computer Vision often deals with problem related to classification or localization of objects in images. However, when humans look at images, they not only just see objects, but are able to analyze and make comments about the subjective quality of an image, talk about its aesthetic quality, its emotional intensity and comment about artist's creativity. Unlike computers, which with the advent of Deep Learning architectures, often treat images as digital dark matter that can be mined to learn hidden structure and representations and recognize, classify and localize images and objects, humans go further and are able to easily describe these images with words.

1.1. Subjective Qualities of Images

Many psychological case studies have been conducted to provide insight into the human brain to find how humans evaluate a piece of art. These show that for most humans, looking at a photo or a piece of art results in psychological responses in various parts of the brain. [12]. These have been found to be correlated to 'quality' of an image, which can be split into objective and subjective quality of images.

The objective qualities of images can be related to the photo properties like exposure, color, contrast, background and sharpness. For instance, case studies show that a photo is fully appreciated when it is well exposed and not-blurry. Blurry shots and over or under-exposed photos are generally regarded as bad shots [8].

The remaining subjective qualities of the images, as the name suggests are harder to quantify and these are the attributes which make this regression problem challenging. For instance, some case studies show that people, when asked about any image, often attribute aesthetics to questions like 'Is the composition balanced?' or 'How is the color distribution in the image?' More often than not, there are no nice mathematical ways to capture these subjective qualities as they vary from person to person [9].

Therefore, it is easy to see that even though the objective qualities can be easily quantified in order to predict which photos are aesthetically pleasing, we still haven't found ways to capture the subjective qualities of images. There are some heuristics, like ***The Rule of the Third*** and ***The Golden Angle***, that attempt to capture these notions with varying degree of success[12]. Finally, photos are also associated to people's memory, which may contribute to their appeal. Both extrinsic and intrinsic image features for making an image memorable have been studied [9].

1.2. Problem Statement

This paper aims at providing insights to the understanding of human being's appreciation of art and beauty; aims at answering the pivotal question ***'What is it that makes an image/art a magical mystical work that blows people away when they look at it?'*** Using the hierarchical learning capabilities of a Convolutional Neural Networks, the models learn these unique set of features that contribute to the works' popularity and, when given an input image, are able to assign a score to the image based on its aesthetic appeal. Towards the end, the developed model is used to predict the

scores of renaissance artworks to see how do these learned features do against world renowned artworks.

1.3. Challenges

One of the biggest challenges in estimating image aesthetics is that most current techniques only utilize low level features, that are unable to capture the higher level representations that may contribute to aesthetics. Also, traditional techniques, being very imperative and non-hierarchical in nature, are unable to figure out the best ways to combine and integrate the lower level features to assign correct score. Finally, before 2012, when AVA dataset was released, there was no proper dataset, with enough detailed meta-data about aesthetics, that could be used to train such models.

2. Related Work

Some studies, aiming to quantify the aesthetic quality of images, have been done in the past. Most of the early approaches tried to approach these problems by using handcrafted features based on photographic intuitions [8]. For instance, the use of low level features such as spatial distribution of edges, color frequency distributions, contrast in addition to using some high level features like object placement, orientation templates depth of view etc. have been used to train classifiers to solve this problem. Most of these approaches work well with their own dataset but fail to deliver similar results on more generalized data. [12]

More recent works in this domain have started using Convolutional Neural Networks. For instance, Lu et al. showed state of the art aesthetic classification performance using convolutional Neural Network [9]. In 2012, Murray et al. introduced the AVA which a large scale dataset of images with aesthetic and style rating. In their original work, they formulated a binary classification problem and trained SVMs with Fisher Vector signatures to achieve 67% accuracy [11]. Lu et al. also used the same AVA dataset and same experimental settings as [11] and were able to obtain 71% accuracy on this dataset by using single column deep convolutional networks. The convolutational layer contained 64 kernels, size of 11, 5,3 and 3 respectively and the fully connected layer contained 1000, and 256 neurons respectively. In 2014, the RAPIDNet, was able to outperform existing models and techniques in this domain[10]. Their most accurate architecture contained 4 convolutional layers and 2 fully connected layers. They achieved this by using a specialized dual column network where one column is the same as their original aesthetics classification network and the second column is used to classify image style which then combine into a aesthetic prediction [9].

3. Technical Approach

3.1. Problem formulation

Based on the AVA dataset, the problem statement turns itself into a regression problem where each of the input image would be given a score from 1 to 10 based on its aesthetic evaluation. However, since CNNs rarely work well on regression problems, I decided to discretize the scores into 10 buckets, 1...10, thus transforming this into a classification problem. Additionally, in order to compare my results with past results, these scores were also transformed into two labels: "HIGH" or "LOW" (i.e. as a binary classification problem) based on a threshold δ, where, any $score \leq \delta$ was labeled as LOW while $score > \delta$ was treated as HIGH. The initial value of δ was chosen to be 5.

3.2. Dataset

For this project, AVA dataset [11], containing 250 thousand labeled images was used. This data is an amalgamation of photos that were submitted to a worldwide photography competition, where each contestant submitted entries in various competitions and the members of the community voted on each of the entries. Each image is labeled on a scale of 1 to 10 based on overall appeal for the given contest theme by an average of 200 site members [11].

It was decided to use this dataset for a variety of reasons. First, it has enough images that I did not need to worry about having too little data. Second, various research studies have already been done using this dataset which can serve as good baselines and comparisons for the project. Finally, while starting to work on this problem, this was the only dataset that offered so much meta-data in a well-defined format.

The dataset was obtained by scraping from **DPchallenge.com** using a python script. Here is the distribution of the resulting data and the associated statistics:

For more information about the dataset, one can refer to [11]. It contains a detail comparison of AVA with other datasets and documents good reasons why AVA is a suitable choice when training a model for image/art aesthetics.

4. Experiment

4.1. Data Pre-processing

As highlighted in [12][8], there are objective and subjective characteristics that lead to an overall aesthetic effect. Since the input is not of any default size or aspect ratio, data pre-processing was done for the model to properly pick up these features. Firstly based on 231N lectures, the mean was subtracted from all the samples. Also, Based on [9] suggestions, the images were re-sized in two ways: first, by warping the images to be 256 X 256 without considering the aspect ratio (i.e. simply down-sampling to 256

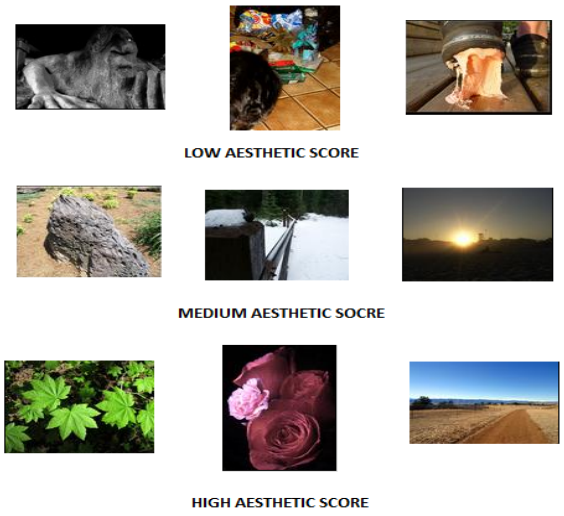

Figure 1: Preview of AVA data set

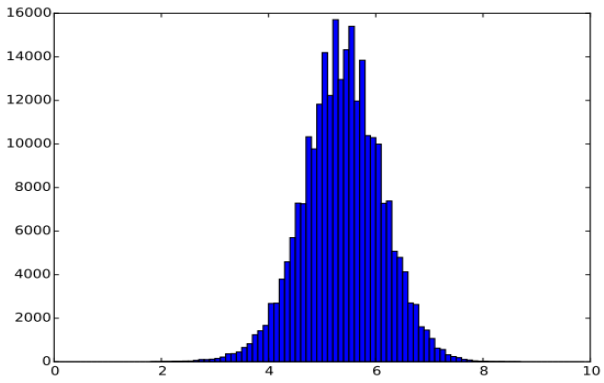

Figure 2: Histogram showing rating meta-data of AVA dataset

X 256). Second: by doing data augmentation. 10 random crops of size 256 X 256 from Top-Left, Top-Right, Bottom-Left, Bottom-Right and Center of the image and taking a mirror image of these while preserving the associated label were also taken as an input.

4.2. Models

4.2.1 Naive Baseline: A Non-CNN model

Using WEKA data mining and machine learning software [4], a 3 layer Neural net model and a support vector regressor were trained to perform regression task on the aesthetics of images. These models served as baselines for the more sophisticated CNN models. PCA was performed on the input images to do away with dimensions with little variation and train these models faster.

Figure 3: Data Augmentation used to increase performance

4.2.2 Better Baseline- Machine Learning applied on specific feature

Studies in the past have used different properties of images to come up with heuristics for image aesthetics [3]. Therefore, a stronger baseline was developed using these features, which included image sharpness (calculated using Gradient and Gradient Entrophy methods described in [8]), color distribution, SIFT BOW descriptor and salient maps.

4.2.3 CaffeNet

CaffeNet model was the first CNN model that was used to do the classification task. It is available in Caffe modelzoo [7] and is similar to the AlexNet architecture. It contains 5 convolutional layers and 3 fully connected layers. To perform this task, the last fully connected layer was replaced with a FC layer, initialized with random weights, and fine tuned on the AVA dataset.

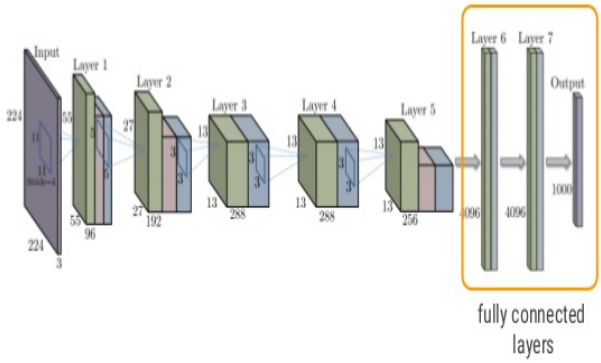

Figure 4: CaffeNet

Additionally, to compare my experiments with the previously published results, the scores were converted into HIGH and LOW labels, i.e. binary classification task. The

deep network was initialized with ImageNet weights to and was trained over 120k epochs with batch size of 128.

4.2.4 VGGNet

After playing with CaffeNet, it was decided to move to a deeper architecture in hope of seeing performance gains. Therefore, I downloaded VGGNet and repeated the same steps as CaffeNet. VGGNet is a 16-layer (13 convolutions and 3 fully connected layers) network from Simonyan et al. [13]. Even with this, the last fully connected layer was replaced with a Soft max loss layer, initialized with random weights, and fine tuned on AVA dataset. Again, as done with the CaffeNet, a separate copy of VGGNet was trained to do classification task.

Training this model was more cumbersome than imagined and the first couple of trails failed in achieving any accuracy (when compared to CaffeNet). After investigation, it was found that this network architecture was much more sensitive to learning rate and the initial learning rate was too high which made the loss blow up. Therefore, I went with a learning rate of 10^{-6}. Also, since this model is more computationally intensive than CaffeNet, I trained it for 100k epochs.

NOTE: I also tried using PReLU instead of ReLU for both the CaffeNet and VGGNet, while keeping all other hyper-parameters constant to study its affect on accuracy.

4.3. Convolutional Neural Network with 'Deeper' channels

This was inspired [3] and by the intuition that the current industry standard is to use features like color distribution, composition, edge distribution. Therefore, what if, for every image, these features were computed and augmented to make them more deeper. After looking into various features and based on [5], the following features were augmented as additional channels to every image:

1. Salient Map as channel 4
2. Color distribution Map as channel 5
3. sharpness Map as channel 6

I decided to make a convolutional network with deeper 8 conv layer architecture followed by 2 fully connected layer. Also, since I was using 256 * 256 warped images with 6 channels (augmented information), I came up with the following architecture:

This was trained in Python using numPy (see the end of the paper for the source code).

4.3.1 Siemese Networks

This was inspired by [9]. The generic structure of Convolutional Neural Network, containing M Conv-ReLU-Pool followed by N Affine Layers + regression layer, remained the same but two CNNs were used instead of one. The intuition

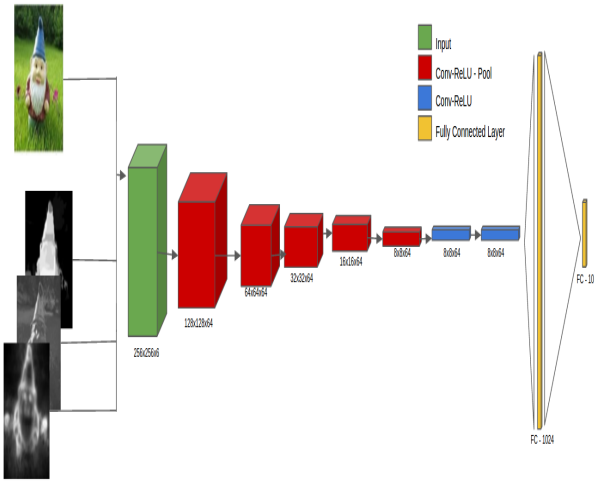

Figure 5: Integrating global and local information into deeper channels and running the CNN

Layer	Parameters	Channels	ReLU	POOL	Layer*
Data	256x256	6			
CONV1	3x3x6	64	Yes	Yes	128x128x64
CONV2	3x3x64	64	Yes	Yes	64x64x64
CONV3	3x3x64	64	Yes	Yes	32x32x64
CONV4	3x3x64	64	Yes	Yes	16x16x64
CONV5	3x3x64	64	Yes	Yes	8x8x64
CONV6	3x3x64	64	-	-	8x8x64
CONV7	3x3x64	64	-	-	8x8x64
CONV8	3x3x64	64	-	-	8x8x64
FC-1	FCx1024				
FC-2	FCx10				

Table 1: Custom CNN network trained using numPy and Python

was that since the convolutional network was used to solve the task of predicting image aesthetics, which was modeled as returning a number between 1-10, a L-2 loss was used in the last layer while partially training (only a few layers weights' were allowed to be updated) and fine-tuning the model.

Research and techniques described in these areas, especially those covered in CS231N lectures and by Lu.et al [10], were significantly used while playing with data augmentation and to fine tune the parameters for different CNNs. For instance, inspired by [10], I used the idea of using two CNNs together. The first was a normal CNN that was taking the entire image as the input while the second CNN tried to integrate the global and the local view, while simultaneously accounting for style information.

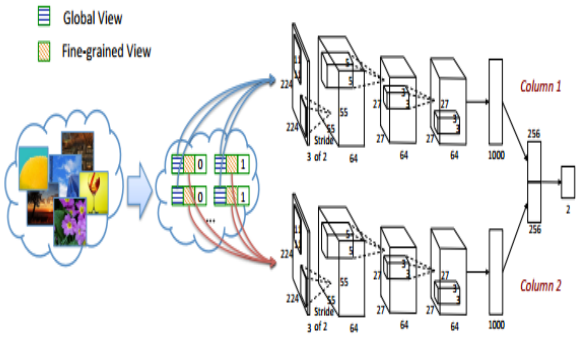

Figure 6: RAPIDNet [9]

5. Results

5.1. Baseline Results

In [9], Lu et al. achieved 64% baseline accuracy. [11], Murray et al. were able to achieve a 67% accuracy without using CNNs. The following were my baseline results (* means that in addition to image values, custom features were also used):

SVR	NN	SVM*	NN*	[9]	[11]
66.7%	67.6%	68.5%	69.01%	64%	67 %

Table 2: Comparing baseline accuracies for predicting image aesthetics

Therefore, I decided to choose the highest value, 69%, as the baseline for further experiments.

5.2. CaffeNet experiment

The first experiment was training the CaffeNet with randomly cropped images. After training it for 100k epochs, I got 74.3% accuracy. After reading [14], I decided to try warping the images. The intuition was that CNNs initialized with ImageNet weights, which are trained on down sampled whole images, do not inherently work well with non-down sampled random crops [3]. I was able to achieve 77.02% accuracy. This was counter intuitive, since I was initially assuming that random crops would perform better since the power of CNN comes from the local features.

My results		Results from [14]	
Random Crops	Warps	Random Crops	Warps
74.3%	77.02%	74.1%	76.82%

Table 3: Comparing CaffeNet performance

My results were better than previous studies. This could be attributed to having more data, more computing power and more number of epochs. Also, my results were consistent with previous studies which concluded that warps are better than random crops for CaffeNet because CaffeNet, originally was trained on downsampled whole trained.

5.3. VGGNet experiment

After training CaffeNet, I moved to VGG16 in hope to see more improvements. Here is how my results compare with the experiments done in [14] and [9]:

My VGGNet	[14] VGGNet	[9] ConvNet
78.6%	77.07%	74.46%

Table 4: Comparison of VGGNet experiments

This shows that the fine-tuned version of VGGNet was a little better than [14] and much better than [9]. I believe that this increase in accuracy happened because of training for more epochs while keeping low learning_rate.

5.4. Results of Custom CNN with deeper channels

This was inspired by [3] and I implemented this by using Python and numPy. It took longer than CaffeNet and VGGNet to train and the model was able to achieve 77.8% accuracy.

[3] ConvNet1	[3] ConvNet2	ConvNet w/ deeper channels
70.6%	75.2%	77.8%

Table 5: Comparison of CNN with deeper channels with others

5.5. Comparing all one-column CNNs tried so far

After training CaffeNet, VGGNet and custom CNN, I compared my experiment results with previous studies [14] [9] [11] as a midpoint check. I see that overall my model achieves better results than past studies. This could be attributed to two main reasons: I initialized the CNNs with ImageNet weights whereas past studies had trained things from scratch. Also, my CNN models were deeper which may have allowed the network to have more expressive power.

When experimenting with ReLU and PReLU, it turned out that since the original CNN weights had used ReLU during training, replacing with PReLU led to a lower performance. Also, while doing this, I had to add another parameter for PReLU and this lead to the exploding loss

phenomena in some cases. After some failed attempts, the learning rate was set to 10^-6 to make sure that the weights did not blow up.

5.6. Weight Visualizations

Here are the weight visualizations of the convolutional network with deeper channels.

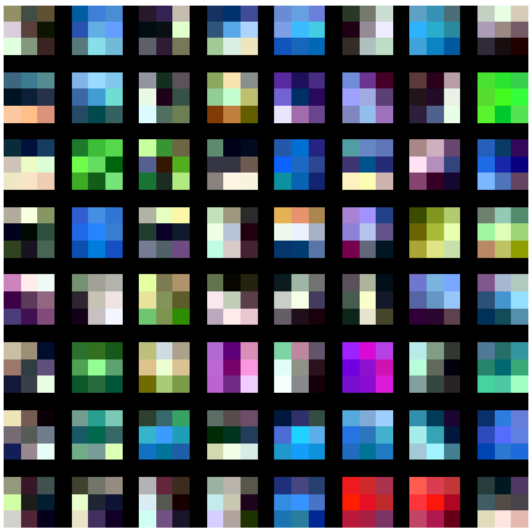

Figure 7: weight visualizations of trained CNN with deeper channels: 1st convolutional layer

In the first layer, most of them are lower level features that capture a lot of the edges and color information. As seen in the figure, there is a lot of blue and green and some red. After seeing these visualizations, I revisited the data to confirm if AVA indeed had a lot more blue and green colors. It turns out that most of the highly aesthetic images of landscape either contain an ocean, sky and/or a tree in the background. This explains why the first layer had abundance of those colors.

When I looked at the weights of the fully connected layer, most of them captured the higher level features. However, as seen in the figure above, the particular set of weights looked most visually interesting. The neuron in the 1st row 1st column and 2nd row, 2nd col (zero indexed) were activated for most of ***Carvaggio's work***, which are most famous for *chiaroscuro*: contrasting treatment given to light and shade in drawings and paintings. This shows that the network may be able to hierarchically combine features into more abstract and broad concepts like shades.

Therefore, after doing these experiments, it was confirmed that pre-trained and fine-tuned CNNs could indeed be used for predicting picture aesthetics.

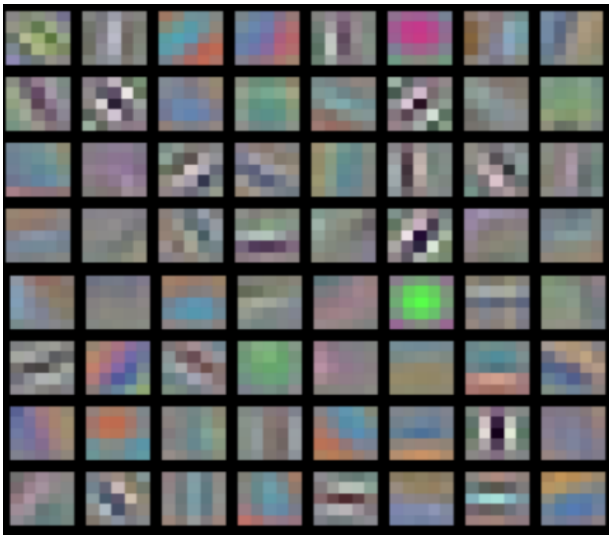

Figure 8: Weight Visualizations of FC-1024

5.7. Results of using Siamese Networks

Inspired by [9], I decided that the following would be my baseline while trying Siamese networks. Caffe has a

δ	[24]	SCNN	AVG_SCNN	DCNN	RDCNN
0	66.7%	71.20%	69.91%	73.25%	74.46%
1	67%	68.63%	71.26%	73.05%	73.70%

Figure 9: RAPIDNet results from [9]

Siamese model, [1], which was used to replicate RAPIDNet. Like [3], I decided to use a L-2 loss for the last layer and train the model. I was able to achieve 71.45% accuracy which is lower than the original paper. I tried playing with the parameters but was not able to achieve a much better result.

5.8. Choosing the best network to classify popular art forms from Renaissance Era

6. Classification and mis-classifications

6.1. From the AVADataset

Most of the image that correctly labeled had very explicit attributes that allowed the model to assign the correct label. For instance, most of the HIGH aesthetics images had good contrast, great placement of objects and overall sense of aesthetics. Similarly, most of the images that were correctly labeled as LOW had attributes, like most of them were dull and had random colors, spots and no homogeneity, that made it easy for the models to classify them correctly.

I noticed an interesting trend with the images that were mis-classified. First, most of them, from a broader perspec-

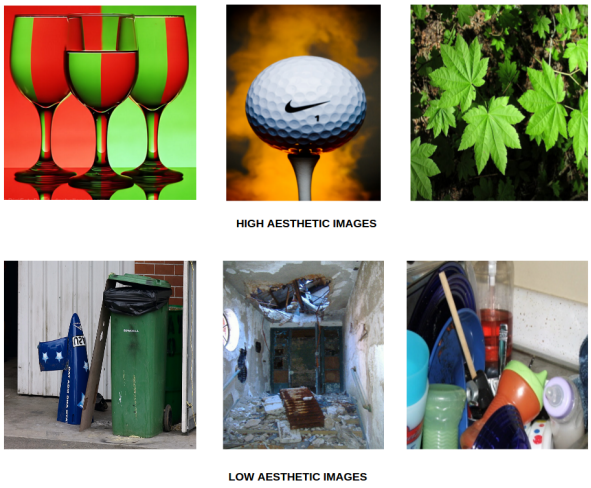

Figure 10: Correctly classified examples from AVA dataset

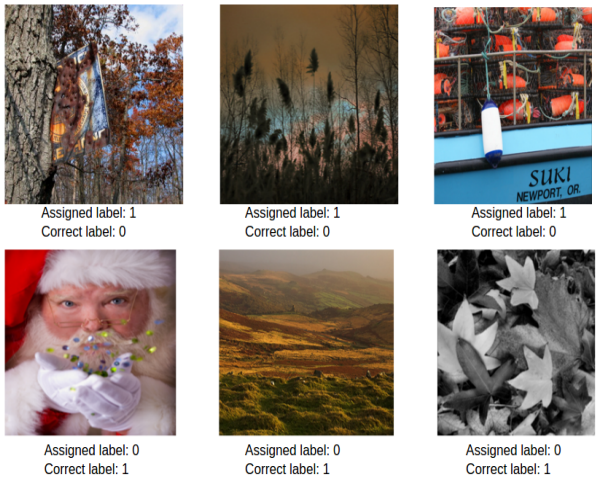

Figure 11: Incorrectly classified examples from AVA dataset

tive, looked aesthetically appealing, but if one were to pay attention to the detail, one could point of elements of the image that was responsible for the label. For instance, in the above figure, the image on the top left, looks relatively aesthetic if one looks at it. There are trees in the background and some part of the sky is visible. However, if one were to pay a closer attention, one would notice that the flag hanging from the tree is covered with mud, which is the reason why, even though the model thinks that this has HIGH aesthetics, it ought to be classified as LOW (in accordance to the comments on DPchallenge). Also, in some cases like the middle column, it was really hard to come to a decision on whether a particular image should be classified as having HIGH or LOW aesthetics. In such cases, I tried using different δ values to get some performance boost.

Finally, it is worthy to note that hand sketches, most of which had HIGH ratings, were mis-classified as images with LOW aesthetics.

6.2. Images from the Renaissance Era

After training and fine-tuning the model on the AVA dataset, I tried to test my model with renaissance art images. The idea was to test if the model was able to predict the correct label of these world famous pieces of art. Here are the results: Most of the portraits, specially with the dark

Figure 12: Best model classifying renaissance images

background were classified correctly by the model. For instance, works of ***Carvaggio***, who is world renowned for this mastery of experimenting with light and shadows, were correctly classified. Also, works of ***Leonardo Da Vinci***, were classified correctly to a great degree.

However, for most of the images of artworks that were sketches or with really light colored background, the model had real difficulty in classifying correctly. As seen in the above figure, one of the greatest works, ***The Renaissance Man***, was mis-classified as LOW. After seeing this, I proposed that this was due to the speckles that were part of the original artwork. I found a replica of the same image, which did not have the speckles, and the model was able to classify it correctly. I concluded that some of the works, were mislabeled because of the granular nature of the art.

7. Failed experiments and things learned

7.1. During the Training Process

Initially, while playing with different architectures, I found the accuracy of my models to be low. After digging deeper, I realized that I had to resolve a potential issue with my loss function. Since, I had modeled the original regression problem as a classification problem, I had to introduce

a L-2 loss function which accounted for mis-classification appropriately. For instance, if my model predicted an image as a 8 when the true score was a 7, the penalty ought to be smaller than when the true score was a 7 and the predicted score was a 5. Therefore, I ended up using L-2 loss:

$$\frac{\sum_{n=1}^{n}(x_i - \bar{x})^2}{2n}$$

and got better results.

After training a bunch of models in Caffe and getting low accuracy, I realized that monitoring the effect of learning rate was quintessential to getting good accuracies, especially in the case of VGGNet whose weights would easily explode with high learning rates.

After getting a good model, I spent some time looking at what were the areas where the model was not performing well. The following figure shows different types of mis-classifications: For the AVA dataset, the model had most

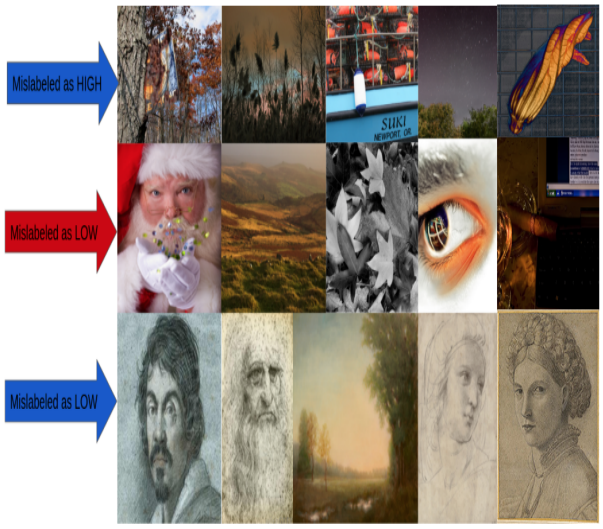

Figure 13: Mis-classifications by the best model

issues with images that broadly looked aesthetically appealing but had certain strong components that made them less appealing. In those cases, the model was not able to pick up those attributes and ended up mis-classifying them. Also, some of the images, like the one with Santa Claus, that had inherent graininess in the depicted subject, led to mis-classification. Finally for most of the renaissance images that were sketches were classified incorrectly.

There are a couple of reasons why I think this happened. Most of the images that were rated as LOW in the AVA dataset, either were of construction site or had bad exposure. In both these cases, the associated images were really grainy and in some of them, one could see dust like particles. I believe that the model learned to associate the graininess with less aesthetic beauty and thus mis-classified these images.

Also, AVA dataset did not have any sketch images. Thus, I hypothesized when the model encountered all the renaissance era sketches, it mis-classified them because of the inherent graininess associated with sketching. In order to test out this, I found a clear image of the ***Renaissance Man*** and fed it to the network and model gave it a higher score.

8. Conclusion

In this paper, I experimented with predefined Convolutional Network architectures: CaffeNet, VGGNet and Siamese Net and showed that CNNs can be successfully trained to predict the aesthetics of an image. Then, by augmenting additional information into extra channel of images, I was able implement a deeper channel architecture which was able to perform better than the existing state of the art models trained on AVA dataset. I compared my best model with other existing models and finally, I evaluated it on renaissance art to measure how well it generalized on predicting image aesthetics for these world renowned pieces.

9. Future Work

From the experiments covered in this and other related papers (see references), there should be little doubt in CNNs potential to be used for the task of rating images based on their aesthetics. This is an interesting problem domain where one can spend more time to achieve better results. I would love to look further into different loss functions, better data augmentation and regularized and transfer learning to improve accuracy.

While starting to work on this problem, I spent some time looking for appropriate data sets that could be used for training and fine tuning. However, I could only find AVA that fit this criteria. I believe that, with a more specialized dataset, for instance specific to art in Renaissance Era, I could build better more specific models and see the features that make those world class paintings so amazing.

Finally, while testing the model, I tried gathering human data on how the model was doing. In the limited time I had, I created a small user interface which showed all the images, along with their bucket (score) labels, that were classified as HIGH by the model. I noticed, though the model captures the broad notion of art, it could do even better if I were able to incorporate people's personal preferences. Given more time, I would like to explore this domain. I hope that using similar trained models, one could use a computer to evaluate the aesthetics of any image and provide editing recommendations for artists, designers and photographers.

References

[1] C. community. Siamese Networks in caffe.

[2] C. cumminity. CaffeNet and VGGnet in caffe.

[3] E. Gong. Estimating photo aesthetic rating using convolutional neural networks. 2015.

[4] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. ; the WEKA data mining software: An update; SIGKDD explorations. 11, 2009.

[5] M.-M. C. N. J. M. X. H. P. H. S. T. S.-M. Hu. Global contrast based salient region detection. 2011.

[6] Y. Jia. Caffe: An open source convolutional architecture for fast feature embedding. 2013.

[7] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408*, 5093, 2014.

[8] Y. Ke, X. Tang, and F. Jing. The design of high-level features for photo quality assessment. In *in IEEE Computer Society Conference on Computer Vision and Pattern Recognition,2006,vol.1*, page 419426. 2006.

[9] X. Lu, Z. Lin, H. Jin, J. Yang, and J. Z. Wang. Rapid: Rating pictorial aesthetics using deep learning. In *Proceedings of the ACM International Conference on Multimedia, MM 14*, page 457466. 2014.

[10] X. Lu, Z. Lin, H. Jin, J. Yang, and J. Z. Wang. Rapid: Rating pictorial aesthetics using deep learning. In *Proceedings of the ACM International Conference on Multimedia*, page 457466. 2014.

[11] N. Murray, L. Marchesotti, and F. Perronnin. Ava: A largescale database for aesthetic visual analysis. In *Computer Vision and Pattern Recognition (CVPR)*, page 24082415. June 2012.

[12] V. O. S. Dhar and T. Berg. High level describable attributes for predicting aesthetics and interestingness. *Computer Vision and Pattern Recognition (CVPR)*, pages 1657–1664, 2011.

[13] K. Simonyan and A. Zisserman. *Very deep convolutional networks for large-scale image recognition*. CoRR, abs/1409.1556, 2014.

[14] P. Veerina. Learning good taste: Classifying aesthetic images. 2015.

10. Supplementary Material

Figure 14: Correctly classified HIGH aesthetic images

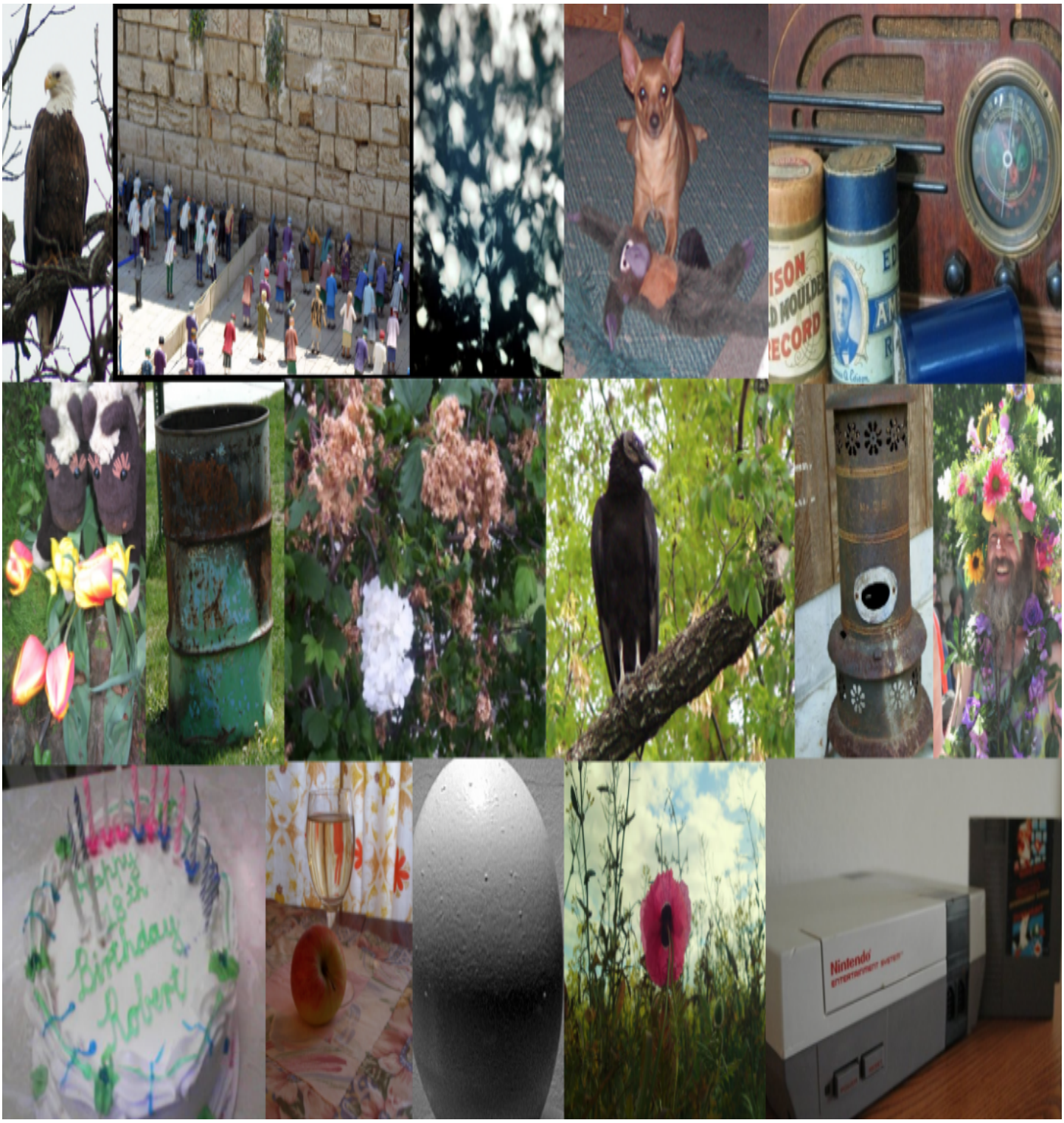

Figure 15: Correctly classified LOW aesthetic images

Figure 16: Correctly classified Renaissance images

10.1. Source Code

10.1.1 Check if AVA dataset links are valid

```
#
# While downloading AVA images, you may need to check the un-downloaded images.
# This script will display information of missing images in console.
#
#
# Note: This script should be placed under AVA_dataset/script/ and images are saved at
#    AVA_dataset/image/ folder.

import os
import sys

savePath = r'../image'
AVAtxt = r'../AVA.txt'

# arg check
if len(sys.argv) < 3:
    print 'arg failer! # python checkWholeness.py beginIndex stopIndex'
    exit()

# must >= 1
beginIndex = int(sys.argv[1])
# must >= 255530
stopIndex = int(sys.argv[2])

f = open(AVAtxt)
for line in f:
    line = line.strip().split(' ')
    imageIndex = line[0]
    imageID = line[1]

    # use begin and stop index constrain
    if int(imageIndex) < beginIndex:
  continue
    elif int(imageIndex) >= stopIndex:
  break

    # if the image does not exist, display its 'index ID' pair
    if os.path.isfile(os.path.join(savePath, imageIndex + '.jpg')) == False:
  print imageIndex, imageID
```

10.1.2 Download AVA data-set images

```
#
# Use 'AVA.txt' provided by database author to download all images from 'dpchallenge.com'.
#    This script should be placed under AVA_dataset/script/ and images are saved at
#    AVA_dataset/image/ folder.
#
#
# Usage:
#    python downloadImage.py beginIndex stopIndex
# So you can download in multi-process.
#
# Note: few images were deleted from the website, please use checkWholeness.py to check
#    missed images and manually delete their items from AVA.txt

import time
import os
import urllib
import sys
from HTMLParser import HTMLParser

##
# Heritate from HTMLParser to parse main image's url
#
class dpchallengeImageParser(HTMLParser):
    def __init__(self):
  HTMLParser.__init__(self)
  self.name = None

    # fetch url and loop out when both image's width and height exceed 200 pix.
    def handle_starttag(self, tag, attrs):
  if self.name is not None:
      return
  if tag == 'img':
            tmpWidth = 0
      tmpHeight = 0
      for key,value in attrs:
    if key == 'src':
        tmpName = value
        tmpWidth = 0
        tmpHeight = 0
    elif key == 'width':
        tmpWidth = float(value)
    elif key == 'height':
        tmpHeight = float(value)

    # extract main (big) image only
    if (tmpWidth > 150) and (tmpHeight > 150):
        self.name = tmpName
        break

savePath = r'../image'
URLprefix = r'http://www.dpchallenge.com/image.php?IMAGE_ID='
AVAtxt = r'../AVA.txt'

# arg check
if len(sys.argv) < 3:
    print 'arg failer! # python downloadImage.py beginIndex stopIndex'
    exit()

# must >= 1
beginIndex = int(sys.argv[1])
# must >= 255530
stopIndex = int(sys.argv[2])

f = open(AVAtxt)
for line in f:
    line = line.strip().split(' ')
    imageIndex = line[0]

    # use begin and stop index constrain
    if int(imageIndex) < beginIndex:
  continue
```

```
    elif int(imageIndex) >= stopIndex:
  break

    # skip exiting images
    if os.path.isfile(os.path.join(savePath,
    imageIndex + '.jpg')) == True:
  continue;

    # get display webpage url
    imageID = line[1]
    URL = URLprefix + imageID

    # url request and write image
    tic = time.time()
    # get 'dpchallenge' display page
    urlopen=urllib.URLopener()
    fp = urlopen.open(URL)
    data = fp.read()
    fp.close()
    # parse out image url in 'self.name'
    urlParser = dpchallengeImageParser()
    # ignore illegal characters (almost
    impossible appear in image's url)
    data = data.decode('ascii', 'ignore')
    urlParser.feed(data)

    # some main images are even smaller than
    advertisements
    #    or have been deleted from website
    #    they need manual downloading
    if urlParser.name is None:
  continue

    # get iamge
    fimg = urlopen.open(urlParser.name)
    data = fimg.read()
    fimg.close()
    # write image
    fout = open(os.path.join(savePath, imageIndex
     + '.jpg'), 'w+b')
    fout.write(data)
    fout.close()
    toc = time.time()

    # display progress
    print 'Processing image: {}\ttime elapse: {}\
    tstop at: {}'.format(int(imageIndex), toc -
    tic, stopIndex)
```

10.1.3 transform original AVA dataset to make it work with Caffe

```
import numpy as np

def get_avg_ratings(ratings):
  ratings = [float(x) for x in ratings]
  assert len(ratings) == 10, "something fishy yo!
    "
  count = 0.0
  for indx, r in enumerate(ratings):
    count += ( indx + 1 ) * float(r)

  return count / np.sum(ratings)

def generate_AVA_transformed():
  delta = 5.0 # threshold value for
    classification
  f = open("AVA_dataset/AVA_67140.txt")
  classification = open("AVA_dataset/
    AVA_classification.txt", 'w')
  regression = open("AVA_dataset/AVA_regression.
    txt", 'w')
  for i in f:
    arr = i.split(" ")
    file_name = "%s.jpg" % arr[0]
    ratings = arr[2:12]
    avg_rating = get_avg_ratings(ratings)
    classification_rating = 1 if avg_rating >
    delta else 0
    regression.write("%s %s\n" %
          (file_name, avg_rating))
    classification.write("%s %s\n" %
          (file_name, classification_rating))


if __name__ == "__main__":
  print "Your task has started..."
  generate_AVA_transformed()
  print "Your task is complete!"
```

10.1.4 Transform images into Caffe supported LMDB format

```
#!/usr/bin/env sh
# Create the imagenet lmdb inputs
# N.B. set the path to the imagenet train + val
    data dirs

EXAMPLE=/mnt/AVA_dataset
DATA=/mnt/AVA_dataset
TOOLS=build/tools

TRAIN_DATA_ROOT=/mnt/AVA_dataset/image/
VAL_DATA_ROOT=/mnt/AVA_dataset/image/

# Set RESIZE=true to resize the images to 256x256
    . Leave as false if images have
# already been resized using another tool.
RESIZE=true
if $RESIZE; then
  RESIZE_HEIGHT=256
  RESIZE_WIDTH=256
else
  RESIZE_HEIGHT=0
  RESIZE_WIDTH=0
fi

if [ ! -d "$TRAIN_DATA_ROOT" ]; then
  echo "Error: TRAIN_DATA_ROOT is not a path to a
     directory: $TRAIN_DATA_ROOT"
  echo "Set the TRAIN_DATA_ROOT variable in
    create_imagenet.sh to the path" \
       "where the ImageNet training data is
    stored."
  exit 1
fi

if [ ! -d "$VAL_DATA_ROOT" ]; then
  echo "Error: VAL_DATA_ROOT is not a path to a
    directory: $VAL_DATA_ROOT"
  echo "Set the VAL_DATA_ROOT variable in
    create_imagenet.sh to the path" \
       "where the ImageNet validation data is
    stored."
```

```
  exit 1
fi

echo "Creating train lmdb..."

GLOG_logtostderr=1 $TOOLS/convert_imageset \
    --resize_height=$RESIZE_HEIGHT \
    --resize_width=$RESIZE_WIDTH \
    $TRAIN_DATA_ROOT \
    $DATA/train.txt \
    $EXAMPLE/AVANet_train_lmdb

echo "Creating val lmdb..."

GLOG_logtostderr=1 $TOOLS/convert_imageset \
    --resize_height=$RESIZE_HEIGHT \
    --resize_width=$RESIZE_WIDTH \
    $VAL_DATA_ROOT \
    $DATA/val.txt \
    $EXAMPLE/AVANet_val_lmdb

echo "Done."
```

10.1.5 Custom convolutional layer for images with Deeper channels

```
# NOTE: the following file could be written by using loops, thus decreasing the number of lines of code.
#  however the author decided to write this in the unrolled fashion for debugging ease

import numpy as np

from cs231n.layers import *
from cs231n.fast_layers import *
from cs231n.layer_utils import *


class DeepChannelNet(object):
  # conv - relu - 2x2 max pool
  # conv - relu - 2x2 max pool
  # conv - relu - 2x2 max pool
  # conv - relu - 2x2 max pool
  # conv - relu - 2x2 max pool
  # conv - relu - 2x2 max pool
  # conv - relu - 2x2 max pool
  # - affine - relu
  # - affine
  # - softmax


  def __init__(self, input_dim=(3, 32, 32), num_filters=32, filter_size=7,
               hidden_dim=100, num_classes=10, weight_scale=1e-3, reg=0.0,
               dtype=np.float32):
    self.params = {}
    self.reg = reg
    self.dtype = dtype
    # extracting various parameters from the input_dimensions
    C, H, W = input_dim
    print "C: %s H: %s W: %s" % input_dim

#  all my convolutional layers
    self.params['W1'] = weight_scale * np.random.randn(num_filters, C, filter_size, filter_size)
    self.params['b1'] = np.zeros(num_filters)

    self.params['W2'] = weight_scale * np.random.randn(num_filters, num_filters, filter_size, filter_size)
    self.params['b2'] = np.zeros(num_filters)

    self.params['W3'] = weight_scale * np.random.randn(num_filters, num_filters, filter_size, filter_size)
    self.params['b3'] = np.zeros(num_filters)

    self.params['W4'] = weight_scale * np.random.randn(num_filters, num_filters, filter_size, filter_size)
    self.params['b4'] = np.zeros(num_filters)

    self.params['W5'] = weight_scale * np.random.randn(num_filters, num_filters, filter_size, filter_size)
    self.params['b5'] = np.zeros(num_filters)

    self.params['W6'] = weight_scale * np.random.randn(num_filters, num_filters, filter_size, filter_size)
    self.params['b6'] = np.zeros(num_filters)

    self.params['W7'] = weight_scale * np.random.randn(num_filters, num_filters, filter_size, filter_size)
    self.params['b7'] = np.zeros(num_filters)

# end of the convolutional era
    self.params['W8'] = weight_scale * np.random.randn(H * W * num_filters/pow(4, 2), H * W * num_filters/pow(4, 3))
    self.params['b8'] = np.zeros(H * W * num_filters/pow(4, 3))

    # for the output affine layer
    self.params['W9'] = weight_scale * np.random.randn(H * W * num_filters/pow(4,3), num_classes)
    self.params['b9'] = np.zeros(num_classes)

    for k, v in self.params.iteritems():
      self.params[k] = v.astype(dtype)


  def loss(self, X, y=None):
    W1, b1 = self.params['W1'], self.params['b1']
    W2, b2 = self.params['W2'], self.params['b2']
    W3, b3 = self.params['W3'], self.params['b3']
    W4, b4 = self.params['W4'], self.params['b4']
    W5, b5 = self.params['W5'], self.params['b5']
    W6, b6 = self.params['W6'], self.params['b6']
    W7, b7 = self.params['W7'], self.params['b7']
    W8, b8 = self.params['W8'], self.params['b8']
    W9, b9 = self.params['W9'], self.params['b9']

    # pass conv_param to the forward pass for the convolutional layer
    filter_size = W1.shape[2]
    conv_param = {'stride': 1, 'pad': (filter_size - 1) / 2}
```

```
    # pass pool_param to the forward pass for the max-pooling layer
    pool_param = {'pool_height': 2, 'pool_width': 2, 'stride': 2}

    scores = None
    # print "statistics for X, W1", X.shape, W1.shape
    a1, c1 = conv_relu_pool_forward(X, W1, b1, conv_param, pool_param)
    a2, c2 = conv_relu_pool_forward(a1, W2, b2, conv_param, pool_param)
    a3, c3 = conv_relu_pool_forward(a2, W3, b3, conv_param, pool_param)
    a4, c4 = conv_relu_pool_forward(a3, W4, b4, conv_param, pool_param)
    a5, c5 = conv_relu_pool_forward(a4, W5, b5, conv_param, pool_param)
    a6, c6 = conv_relu_forward(a5, W6, b6, conv_param)
    a7, c7 = conv_relu_forward(a6, W7, b7, conv_param)

    a8, c8 = affine_relu_forward(a7, W8, b8)
    scores, c9 = affine_forward(a8, W9, b9)

    # for i, a in enumerate([a1, a2, a3, a4, a5, a6, a7, a8]):
    #     print "A%s shape is %s" % ((i+1), a.shape)

    # print "W8 shape and bias", W8.shape, b8.shape
    # print "W9 shape and bias", W9.shape, b9.shape
    if y is None:
      return scores

    loss, grads = 0, {}
    dloss, dscores = softmax_loss(scores, y)

    da8, dw9, db9 = affine_backward(dscores, c9)
    da7, dw8, db8 = affine_relu_backward(da8, c8)
    da6, dw7, db7 = conv_relu_backward(da7, c7)
    da5, dw6, db6 = conv_relu_backward(da6, c6)
    da4, dw5, db5 = conv_relu_pool_backward(da5, c5)
    da3, dw4, db4 = conv_relu_pool_backward(da4, c4)
    da2, dw3, db3 = conv_relu_pool_backward(da3, c3)
    da1, dw2, db2 = conv_relu_pool_backward(da2, c2)
    dX, dw1, db1 = conv_relu_pool_backward(da1, c1)

    grads['W1'] = dw1
    grads['W2'] = dw2
    grads['W3'] = dw3
    grads['W4'] = dw4
    grads['W5'] = dw5
    grads['W6'] = dw6
    grads['W7'] = dw7
    grads['W8'] = dw8
    grads['W9'] = dw9

    grads['b1'] = db1
    grads['b2'] = db2
    grads['b3'] = db3
    grads['b4'] = db4
    grads['b5'] = db5
    grads['b6'] = db6
    grads['b7'] = db7
    grads['b8'] = db8
    grads['b9'] = db9

    reg_loss = 0.0
    for i in range(1,10):
        layer_name = str(i)
        # print "working on layer %s" % layer_name
        reg_loss += 0.5 * self.reg * np.sum(self.params['W' + layer_name] * self.params['W' + layer_name])
        grads['W' + layer_name] += self.reg * self.params['W' + layer_name]

    loss = dloss + reg_loss
    return loss, grads

pass
```

10.1.6 CaffeNet, VGGNet, Siamese Network implementation

For training CaffeNet, VGGNet and Siamese network, Caffe[6], along with native scripts available in the model zoo[2] and [1] were used.