CharLeNet

TRAINING A DEEP CONVOLUTIONAL NEURAL NETWORK FOR AESTHETIC EVALUATION OF DIGITAL PHOTOGRAPHY

CHARLES MACKAY

Purpose

- Ever wondered if your photography is any good?
- Are you an amateur or professional photographer who would like instant feedback on your shots?
- Are you a web designer looking for that perfect stock photo?
- The solution is here!

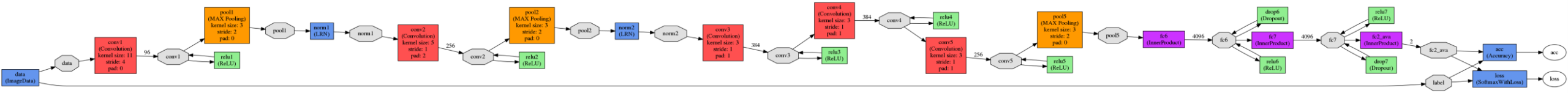
CharLeNet

- A software tool for photographers using deep neural networks trained for aesthetic evaluation of digital photography
  - Allows you to get a confidence prediction whether your photo is aesthetically pleasing or not

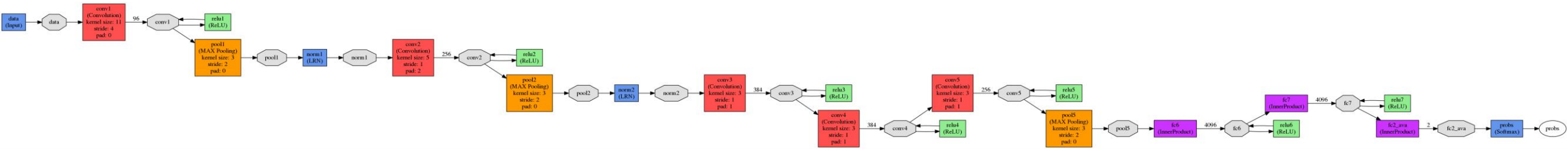
Product Description

- Deep convolutional neural net with CaffeNet's architecture and weights.
- CaffeNet's architecture is based on AlexNet, a deep CNN used for the **ILSVRC** image classification contest.
  - Was trained with 310,000 iterations on 1.3 Million photos.
  - This makes the pre-trained network highly exposed to digital photographs already, making the layer weights well suited to extract visual features.
- Trained using supervised learning on 50K photos, rated by the dpchallenge.com community.
- Used Caffe's Python module to create a Python notebook for training and deployment.

My Model

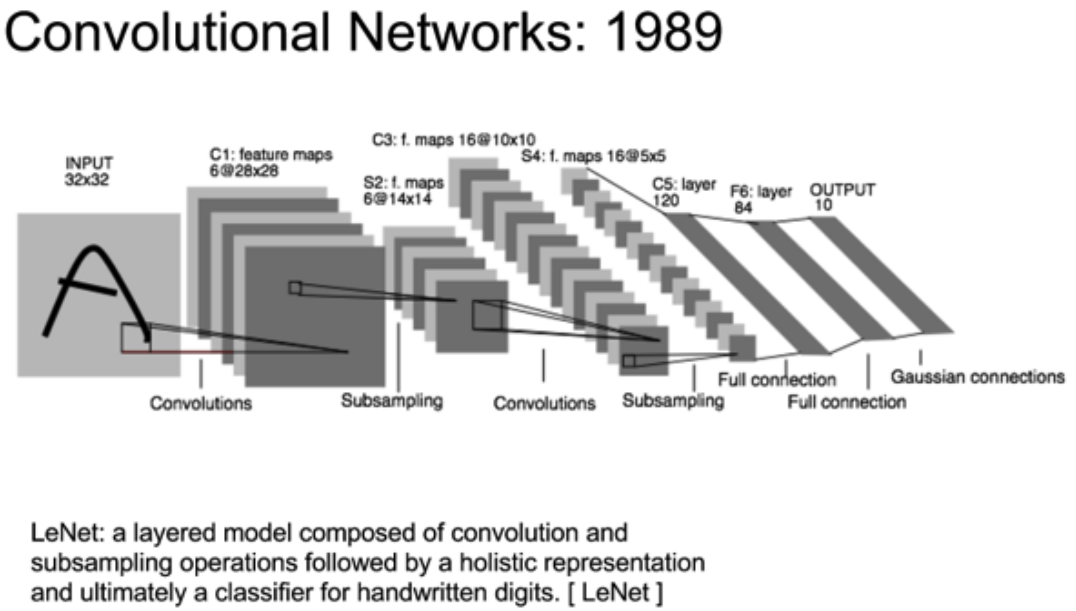


- Consists of five convolutional layers, some of which are followed by max-pooling layers, and two globally connected layers with a final 2-way SoftMax.
- Weights transferred from CaffeNet
- See layer definitions here



About the name CharLeNet (Charlie Net)

- Named after Yann LeCun's **LeNet**, a convolutional neural net invented in early 1990s.
- This was the first successful application of Convolutional Networks and was suited to read zip codes and numeric digits.
- Alexnet has a similar structure to LeNet, and most CNN's are based off of LeCun's work.
- GoogLeNet is another example.



Steps to building CharLeNet

1. Research prior work – I read a lot of academic papers and literature about CNN classification.
2. Choose a deep learning framework – I used Caffe since I discovered it when I was doing research on Deep Dream. Developed by Yangqing Jia at BVLC.
3. Acquire the dataset – the data is the most important part. Without high quality and quantity, we cannot train our net adequately.
4. Build the network – understand the layers, weights, and learning rates
5. Train the network – babysit our training until we are satisfied the network is learning, the error rate should decrease and accuracy increases. Run it over night to achieve minimum loss.
6. Test the network – try the network out on photos that are and aren't in the dataset

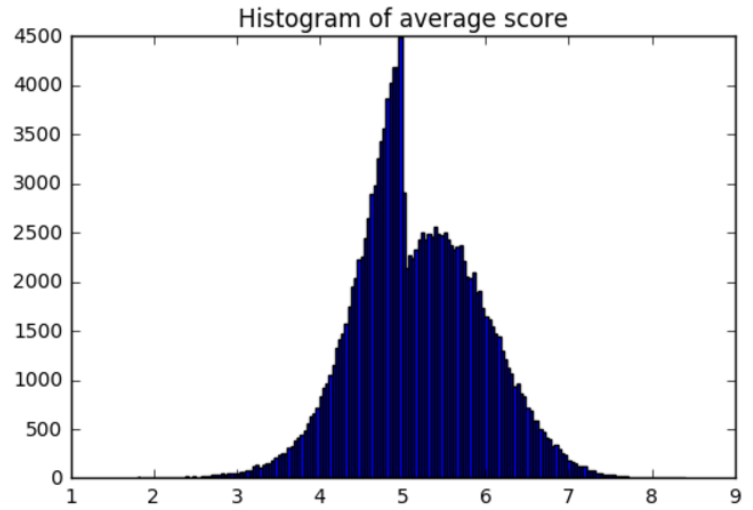
Key words

- CNN – convolutional neural networks. Made up of neurons that have learnable weights and biases.
- Weights – or synaptic weight refers to the strength or amplitude of a connection between two nodes
- Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity
- Non-linearity – activation function of a node that defines the output of that node given an input or set of inputs
- So when I say, copying weights, I'm transferring the learned parameters of a trained network's neurons to mine. This saves time and resources.

AVA - Aesthetic Visual Analysis Dataset

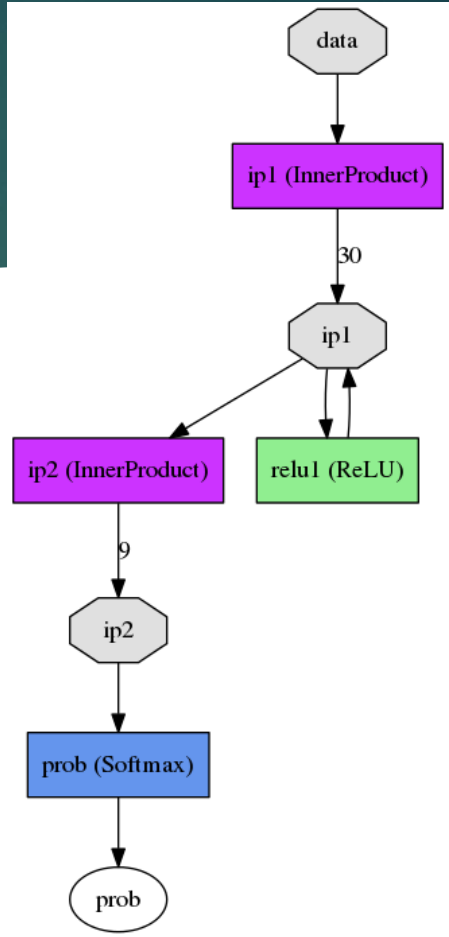
- 250,000 images from dpchallege.com - digital photography contests where users vote and rate pictures.
- Each image has an average of 210 votes. Normally distributed
- I scraped 154k images
  - Started random, then focused on downloading more "bad" photos to even out the ratio of good/bad
  - Eventually used 50k for training, by only using photos outside the range 5±0.5
  - Reducing the data set to more polarizing pictures will give higher accuracy in classification

```
average score:  5.17057359393
std dev:  0.763671155425
total photos downloaded: 154026
good photos (avg > 5): 81865
bad photos (avg <= 5): 72161
very good photos (avg > 6) 22595
very bad photos (avg <= 4) 7919
```



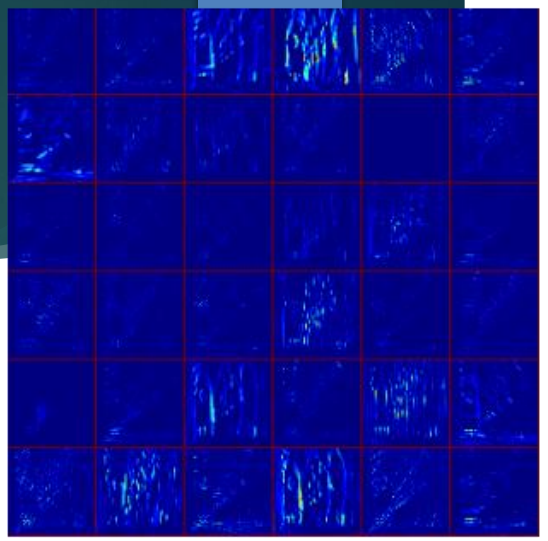
Building the net

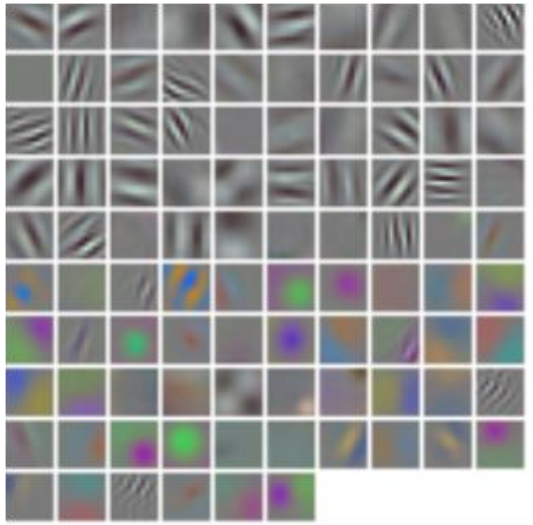
- Start from CaffeNet's prototext (model architecture spec.)
- Remove the final fully connected layer that maps it to 1000 categories for image classification
- Replace with new FC layer with 2 classifications
- Map new classification labels to FC layer
- Feed in training data to the net ImageData layer
- Copy weights
- Write Solver file



Layers Used

- Data layer – image data input
- Convolution – convolves the input image with a set of learnable filters, each producing one feature map in the output image. Used for extracting visual features every where in the photo.
- Fully Connected – full connections to all the neurons in the previous layer. Inner product is calculated together for output of the class scores .
- Dropout – regularization to prevent overfitting
- Probs, label, loss, acc – the output of the net depending on phase. Will output probability (deploy), with labels, loss, accuracy (train and test)

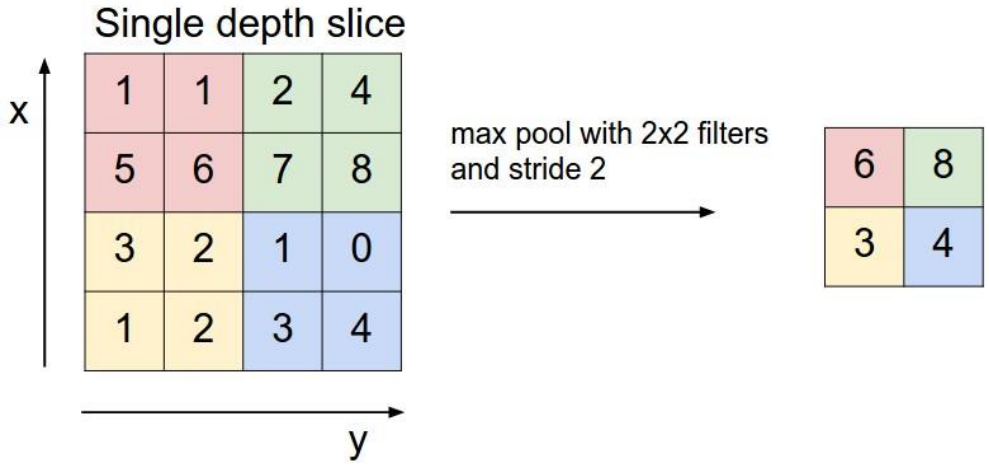


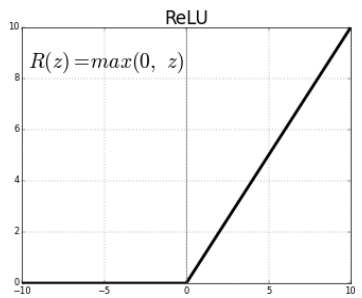


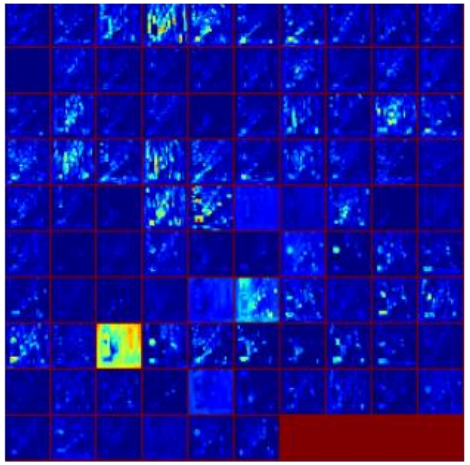
Top: conv1 output
Bottom: conv1 filters

Layers cont.

- ReLU non-linearity – unbounded non-saturating activation function between neurons. Preferred over sigmoid due to easy to compute for faster learning, and low activation of random weighted neurons.
- Normalization (LRN) - implements lateral inhibition, makes excited neurons stand out.
- Max Pooling – reduce the spatial size after conv to reduce the amount of parameters and computation in the network. Also controls overfitting.





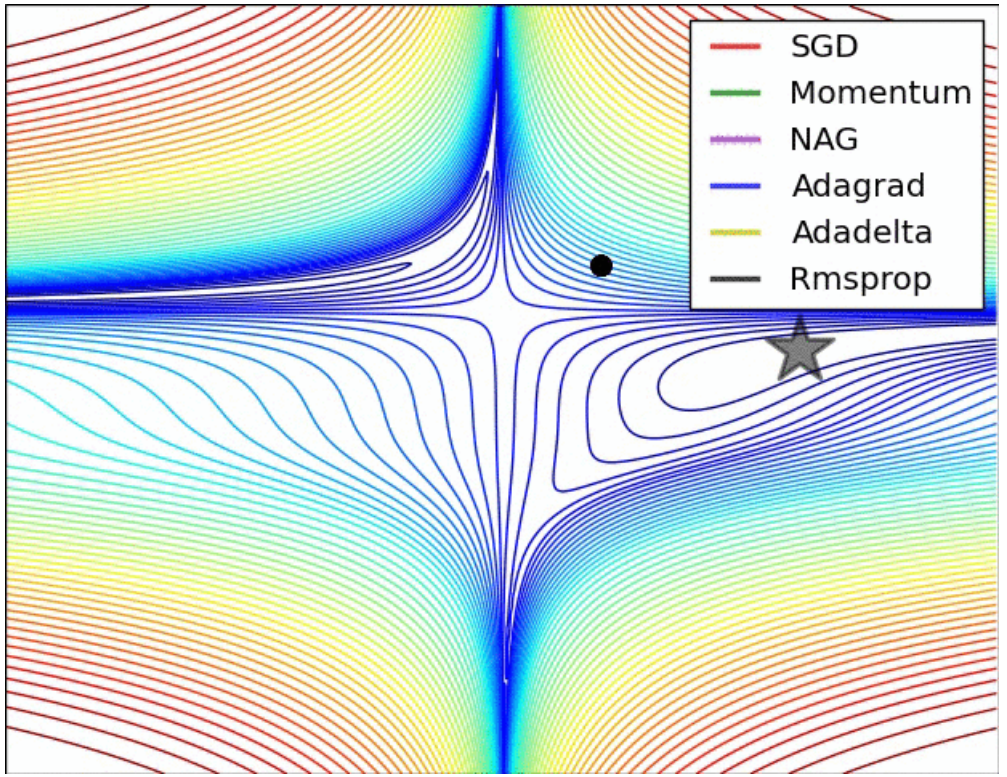


Top: max pooling example
Bottom: ReLU, Pool1 after conv1

Solvers – Stochastic Optimization

- Algorithm used: Adam
- Solver iteratively optimizes by forward / back propagation and updating parameters
- (periodically) evaluates the test networks
- snapshots the model and solver state throughout the optimization

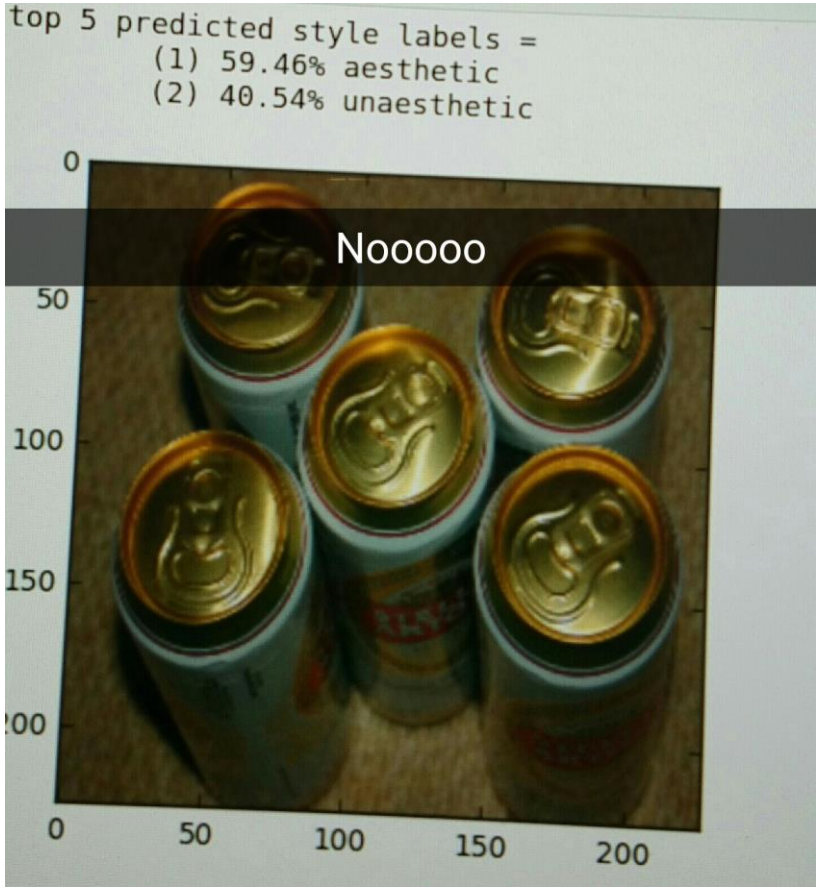
- Each iteration:
  - Calls network forward to compute output and loss
  - Network backward to compute gradients
  - Incorporates gradients into parameter updates
  - Updates solver state according to learning rate
- Computes the weights from init. to learned model



Animation: different optimization algorithms rate of convergence to minimum

Training the net - attempt 1

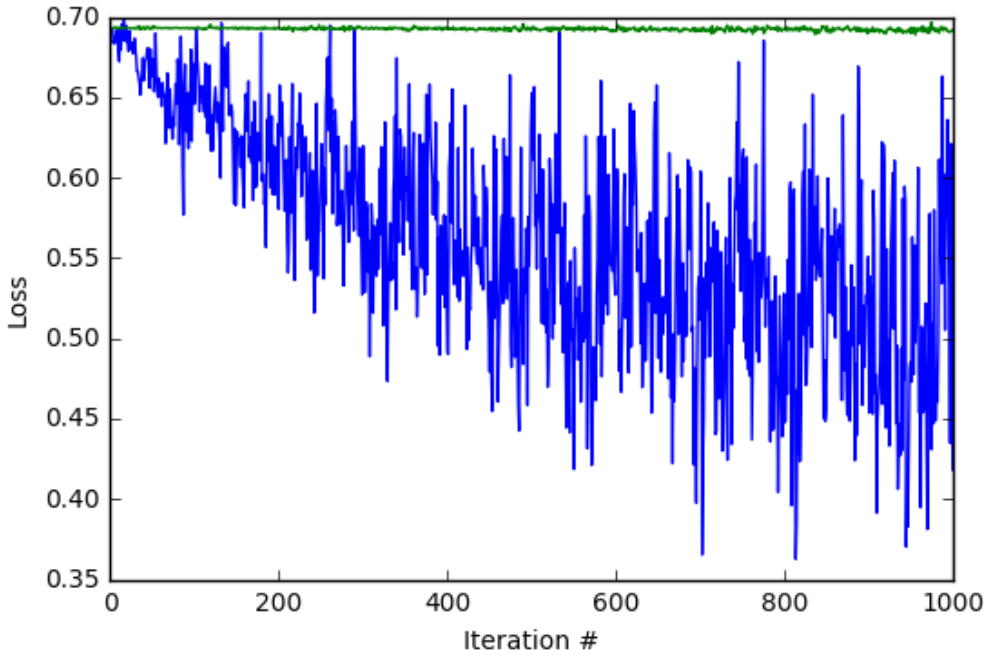
- Fed the newly built network all the photos I had at the time ~100k
- Did not vet the data.
  - Did not check ratio of bad photos to good photos (was about 75/25)
  - Did not check the average rating (ambiguous data, score 4.99 vs 5.01?)
- Did not understand batch size (how many photos evaluated at once)
- Loss and accuracy did not converge (frustrating!)
- Was like flipping a coin for prediction



Training the net 2

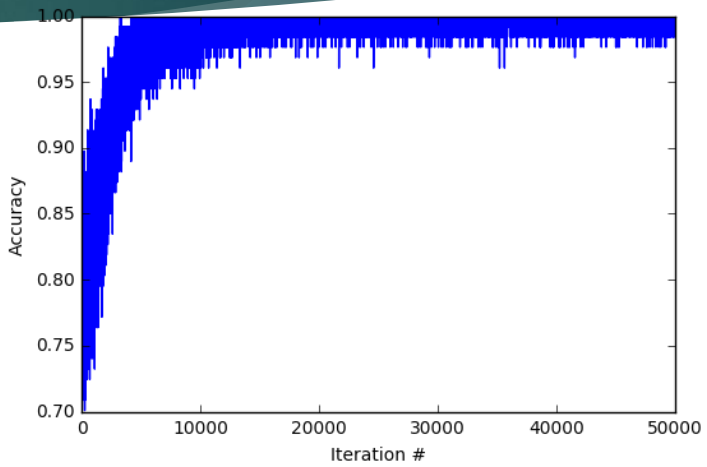
- Reduced dataset to 300 photos only picking the best of the best and worst of the worst.
  - Net converged to 100% accuracy, 0 loss
- Increased training data set to 20k photos, with images with scores 4- and 6+
- Used "Adam" optimization
- Reduced learning rate to 1e-5
- Converged very fast! High accuracy.
- Ran 50k epochs, but converged loss to 0 around 10k.
- Saved snapshot at 5 and 10k

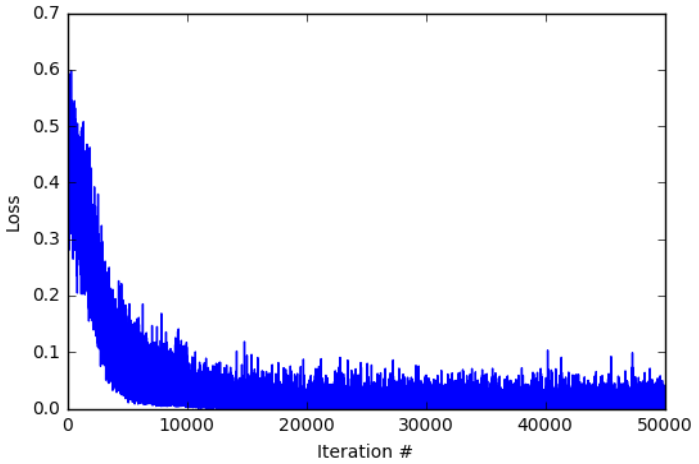
Training the new FC layer only, all other layers are frozen with lr_mult = 0.
Blue: Net with weights loaded with CaffeNet
Green: Net without any weights initalized



Training the net 3

- Increased training data set to 50k photos, only images with a score of 5.5 and higher, or 4.5 and lower.
- Copied weights from previous 4-6 run at 5k snapshot
- Used custom learning weights on FC layer. Higher than other layers
- Ran 50k epochs, but converged around 10k again.
- Wobbly loss, not perfect. Can be improved with lower learning rates. I think by reducing LR multiplier and using standard learning rates





Figures: Training end-to-end network. All layers learning. Fully connected layer learning the fastest.
Top: Accuracy
Bottom: Loss

Accuracy, trained from ImageNet initialization: 87.4%

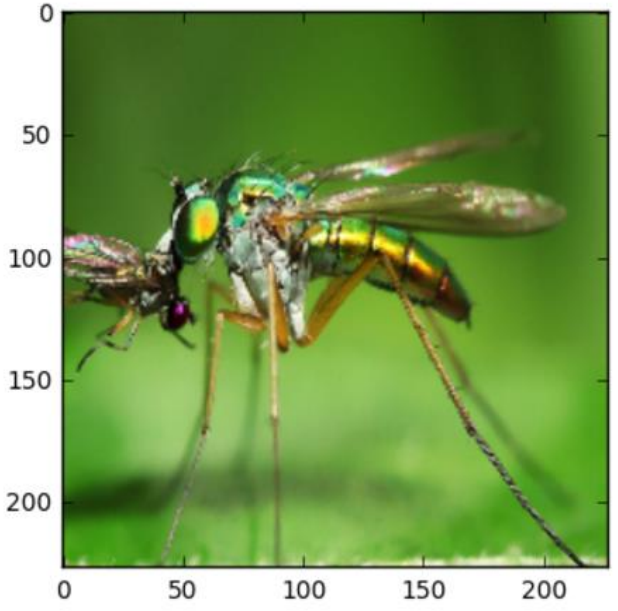
Results on the AVA dataset

Very accurate! 87.4%

Penalizes blur, overexposure, and selfies!

Very satisfied with results

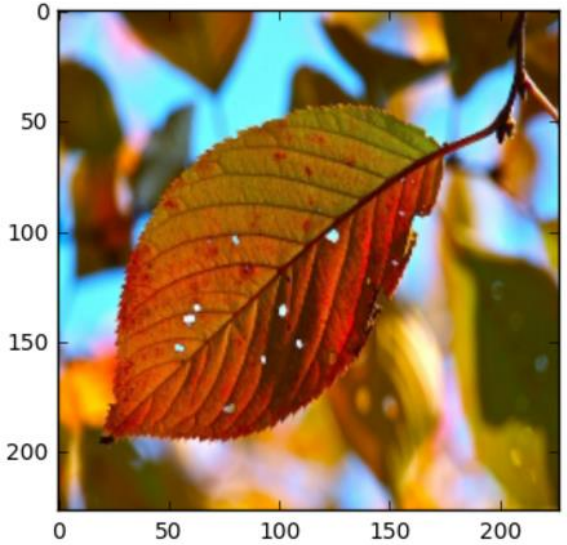
2nd highest rated photo in AVA. Score 8.024



```
disp_aesthetic_preds(net, image,pred[0])

top 2 predicted aesthetics labels =
        (1) 100.00% aesthetic
        (2)  0.00% unaesthetic
```

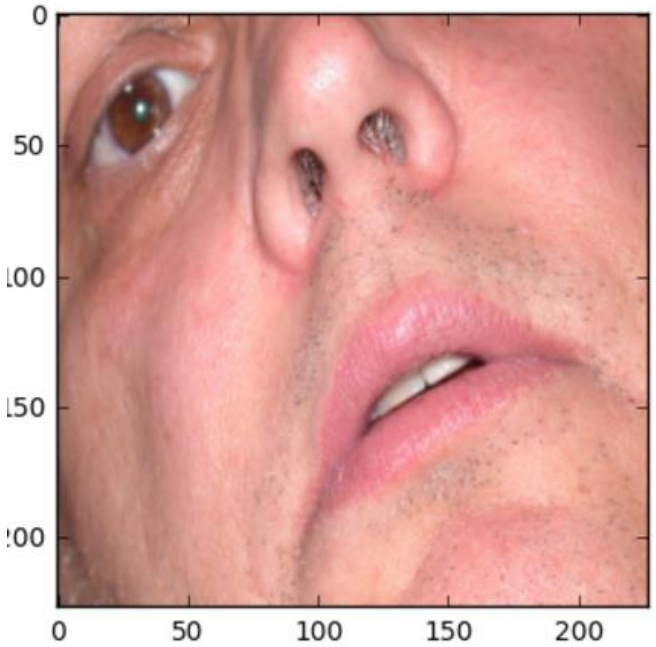
Mislabeled, but good level of uncertainty. Score from AVA: 5.43



```
disp_aesthetic_preds(net, image,pred[0])

op 2 predicted aesthetics labels =
        (1) 54.46% unaesthetic
        (2) 45.54% aesthetic
```

Score from AVA: 3.65/10

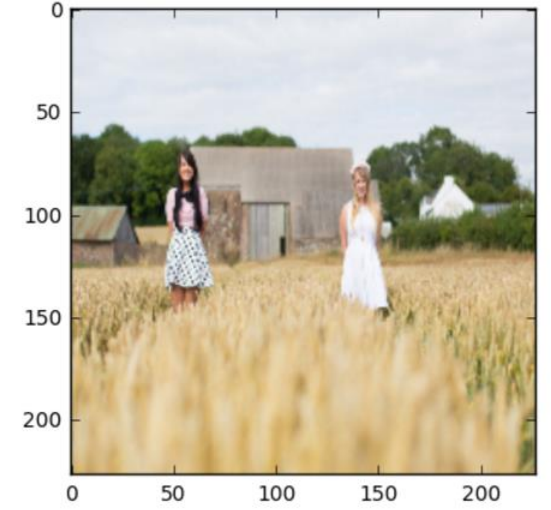


```
sp_aesthetic_preds(net, image,pred[0])

p 2 predicted aesthetics labels =
        (1) 100.00% unaesthetic
        (2)  0.00% aesthetic
```

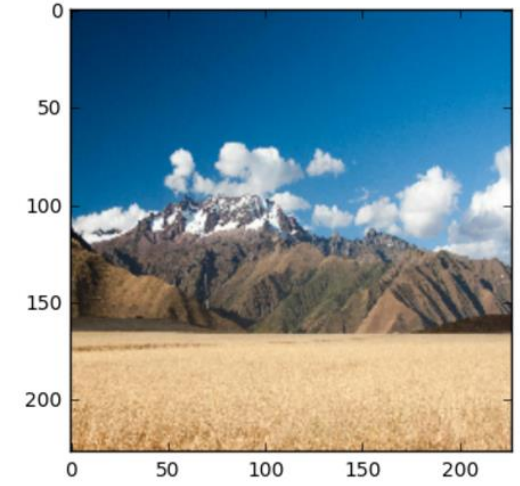
Results from outside the training set

- Extremely accurate!!!
- Have not found a case where I disagree. Spooky accurate
- Does not like Instagram models! Likes Gigi Hadid a lot
- Can tell if a photo is professional or amateur very well
- Live Demo.



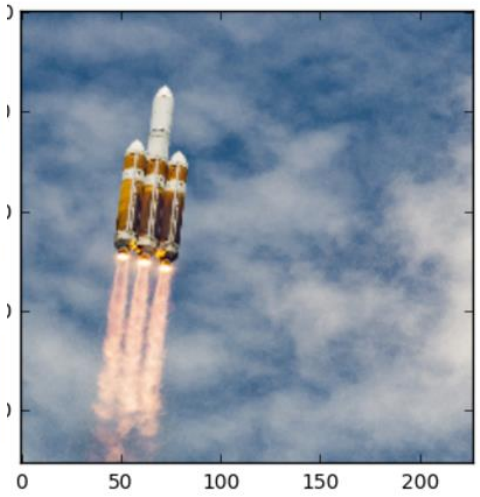
```
disp_aesthetic_preds(net, image,pred[0])
```

```
top 2 predicted aesthetics labels =
      (1) 100.00% unaesthetic
      (2)  0.00% aesthetic
```



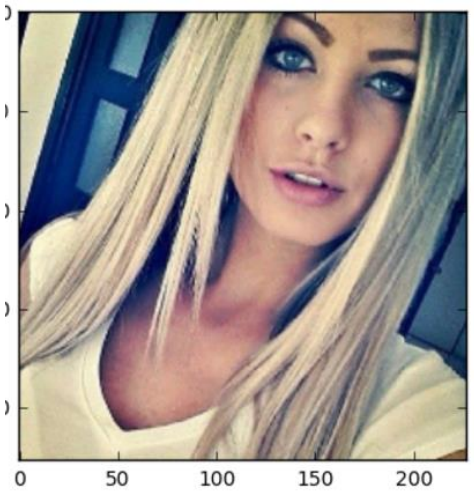
```
disp_aesthetic_preds(net, image,pred[0])
```

```
top 2 predicted aesthetics labels =
      (1) 88.18% aesthetic
      (2) 11.82% unaesthetic
```



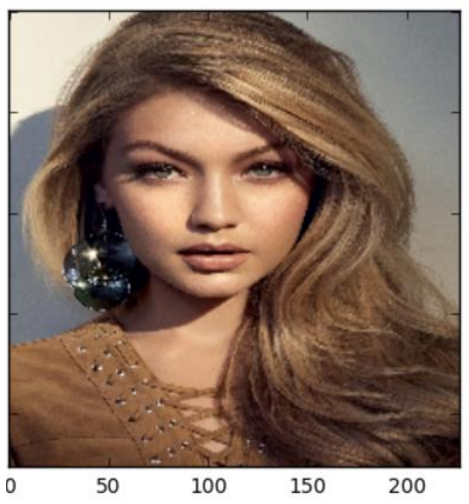
```
_aesthetic_preds(net, image,pred[0])
```

```
2 predicted aesthetics labels =
      (1) 98.89% aesthetic
      (2)  1.11% unaesthetic
```



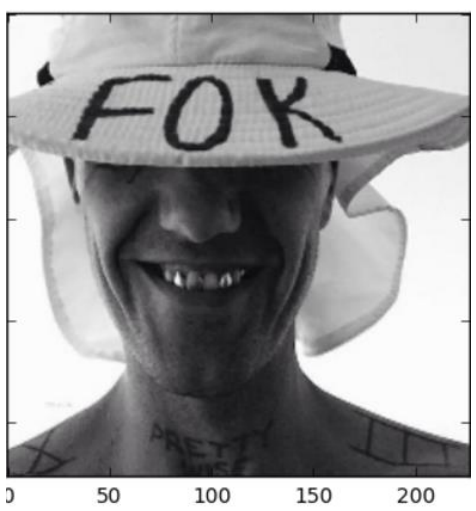
```
_aesthetic_preds(net, image,pred[0])
```

```
2 predicted aesthetics labels =
      (1) 95.80% unaesthetic
      (2)  4.20% aesthetic
```



```
_aesthetic_preds(net, image,pred[0])
```

```
 predicted aesthetics labels =
      (1) 100.00% aesthetic
      (2)  0.00% unaesthetic
```



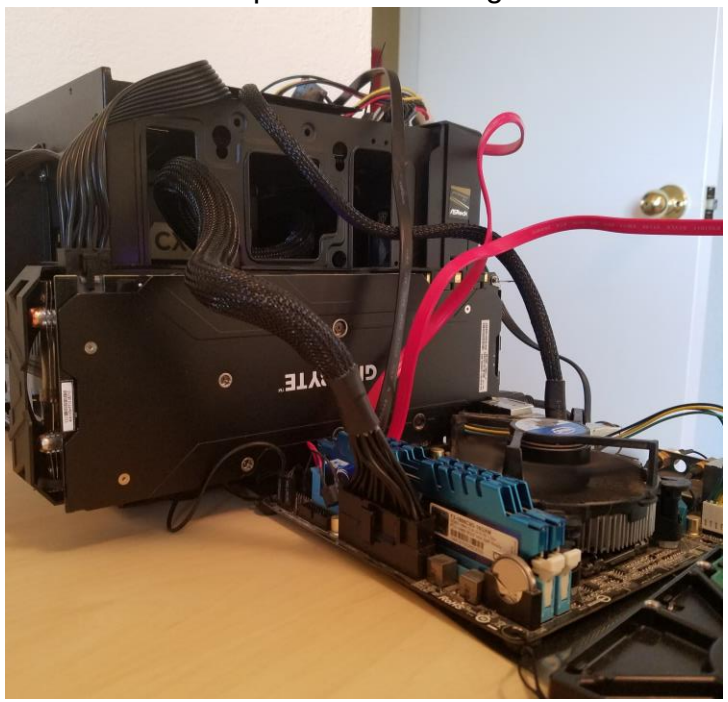

```
aesthetic_preds(net, image,pred[0])
```

```
 predicted aesthetics labels =
      (1) 99.08% unaesthetic
      (2)  0.92% aesthetic
```

Hardware Setup

- CPU: Intel i7-4790K
- RAM: 16 GB DDR3 1866MHz
- GPU: Gigabyte GTX 1070 G1 8GB
- HDD: WD 2TB 7200RPM
- OS: Ubuntu 16.04

GTX 1070 Graphics card too big to fit in case!



```
disp_aesthetic_preds(net, image,pred[0])
```

```
top 2 predicted aesthetics labels =
        (1) 88.77% aesthetic
        (2) 11.23% unaesthetic
```

Future work

- Finish downloading entire 250k dataset. Respecting robots.txt (maybe?)
- Implement a 3-way classification to label the middle section of average scoring photos.
- Train on a larger dataset (full AVA) try to get high accuracy
- Understand training parameters better to get better loss convergence
- Publish the finished network on Caffe Model Zoo
- Install Nvidia DIGITS software (interactive deep learning system) https://developer.nvidia.com/digits to see training visually
- Host the neural net classification as a webapp and accessible tool to the outside world.

Special thanks to Prof Maya Ackerman for the flexible project schedule
Thanks to R. Gruemmer and Avinash More for consultation on linear regression

Conclusion

- 150k photos downloaded
  - Removed ambiguous or non-discerning photos
  - 50k used to train the net.
- Loss and accuracy converged to optimal in 10k epochs.
- Accuracy of 87% on the test set.
- Very agreeable classification on photos outside of the test set.
- Satisfied with results, don't feel further training is necessary.
- Will work to host as web app.

Sources

- AVA Dataset http://www.lucamarchesotti.com/
- Caffe http://caffe.berkeleyvision.org/
- Stanford CS 231n CNN http://cs231n.github.io/
- Dropout https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf
- Adam - https://arxiv.org/abs/1412.6980
- Buck Graphic by designfordisorder.tumblr.com