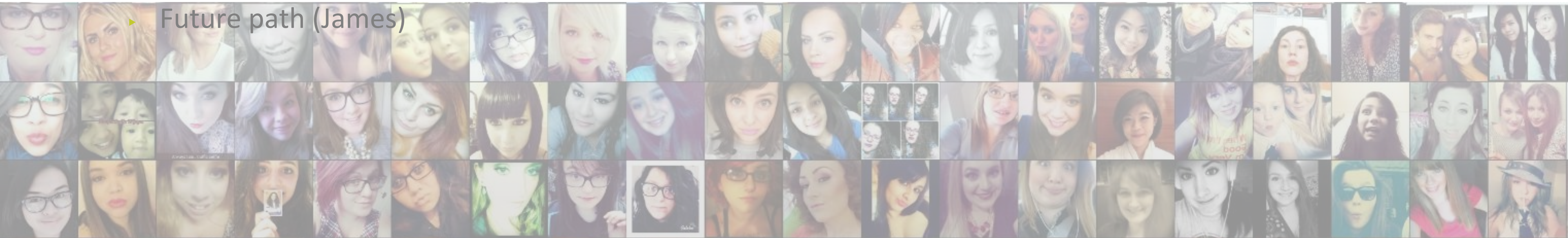
Facial Keypoints Detection

Megan Jasek, Charles Kekeh, James King, & Beth Partridge

W207 Spring 2016

Agenda

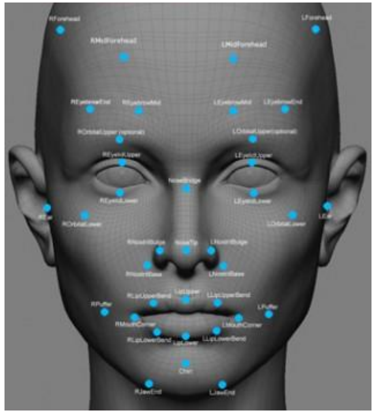
- Overview of competition (Beth)
- Overfitting (Charles)
- Tools & Environment (Megan)
- Data augmentation (James)
- Models
  - Pragmatic approach - Beth
  - non-NN and non-convolved NN models - Megan
  - single NN plus Analysis - Charles
- Final results (James)
- Future path (James)

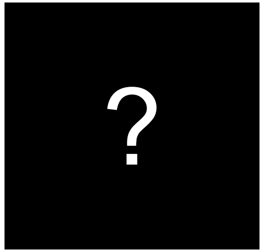


Kaggle Facial Keypoints Detection Competition - Overview

- Key building block for many applications
- Very challenging
  - Feature variation person-to-person
  - 3D pose
  - Size
  - Position
  - Viewing angle
  - Illumination conditions

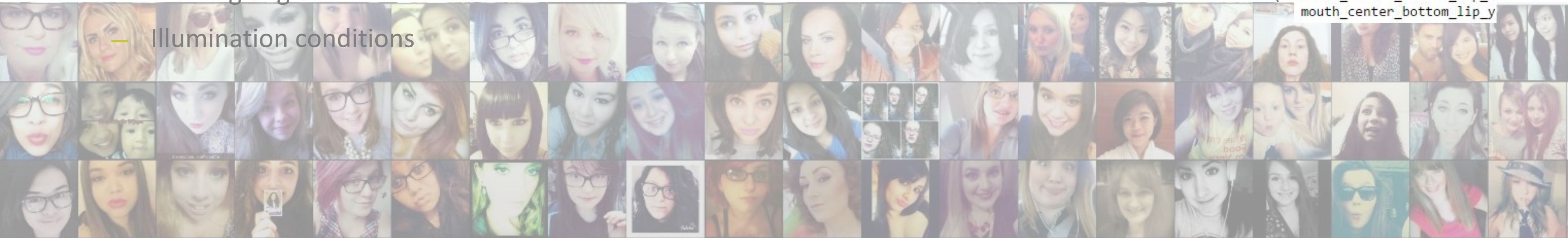
96x96 pixel faces



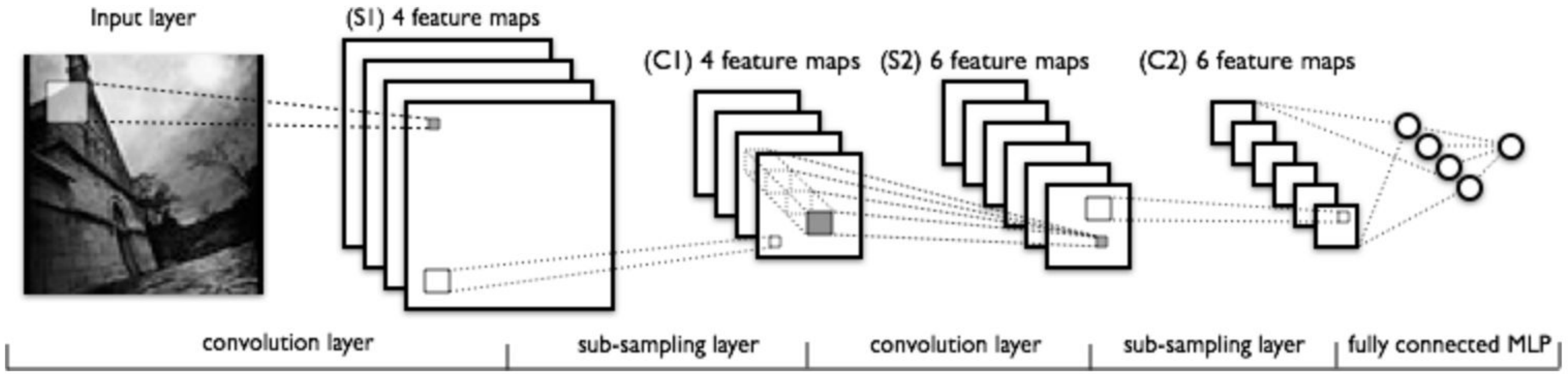


15 keypoints

```
left_eye_center_x
left_eye_center_y
right_eye_center_x
right_eye_center_y
left_eye_inner_corner_x
left_eye_inner_corner_y
left_eye_outer_corner_x
left_eye_outer_corner_y
right_eye_inner_corner_x
right_eye_inner_corner_y
right_eye_outer_corner_x
right_eye_outer_corner_y
left_eyebrow_inner_end_x
left_eyebrow_inner_end_y
left_eyebrow_outer_end_x
left_eyebrow_outer_end_y
right_eyebrow_inner_end_x
right_eyebrow_inner_end_y
right_eyebrow_outer_end_x
right_eyebrow_outer_end_y
nose_tip_x
nose_tip_y
mouth_left_corner_x
mouth_left_corner_y
mouth_right_corner_x
mouth_right_corner_y
mouth_center_top_lip_x
mouth_center_top_lip_y
mouth_center_bottom_lip_x
mouth_center_bottom_lip_y
```



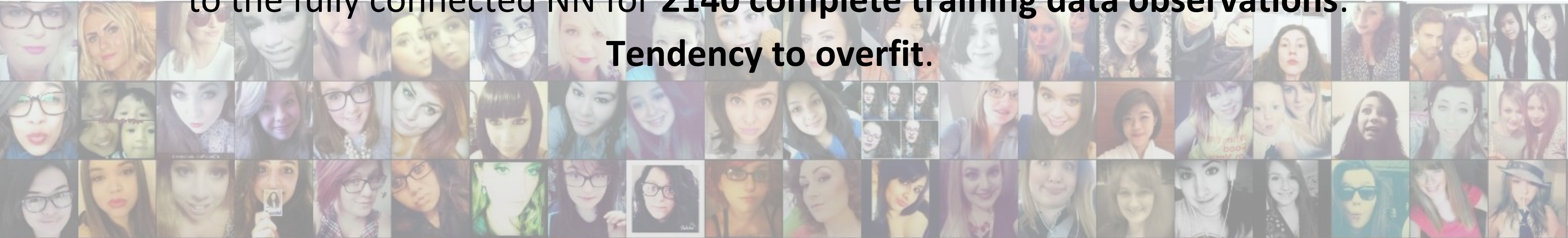
Overfitting



Typical convolutional layer architecture: 32 * 64 * 128 feature maps

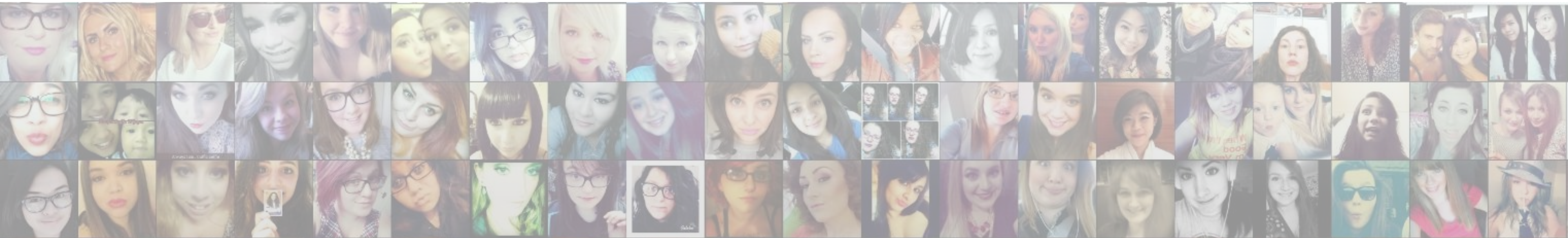
3 * 3 patch widths and (2, 2) max pooling: 128 feature frames of size 10 * 10 pixels on the last feature map layer. That's **12800 features** presented to the fully connected NN for **2140 complete training data observations**.

Tendency to overfit.



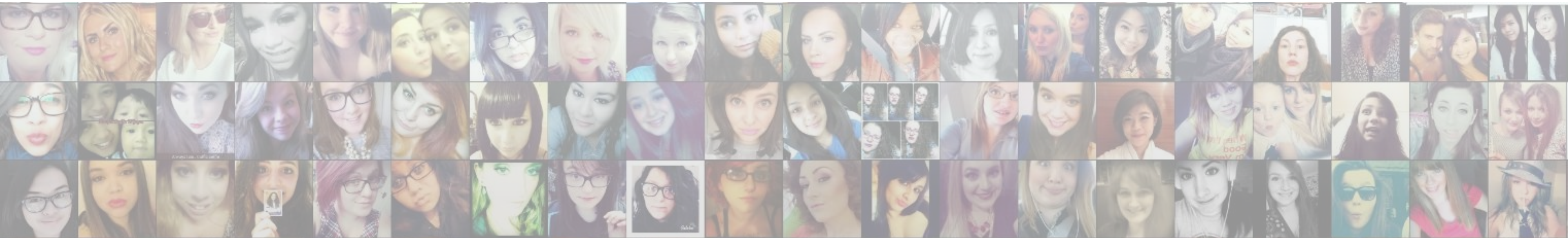
Tools & Environment

- Theano vs Lasagne
  - Theano - Python library that allows you to define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays efficiently (symbolic language)
  - Lasagne - lightweight library to build and train neural networks in Theano
    - Difficult to install (stackoverflow.com)
    - Easy to use, easy to understand and easy to extend, to facilitate use in research
- GPU - AWS (expensive)
  - g2.2xlarge. G2 Instances are backed by 1 x NVIDIA GRID GPU (Kepler GK104) and 8 x hardware hyperthreads from an Intel Xeon E5-2670
  - Mixed results. Need to give GPU enough to do for it to be effective
    - Still takes hours to run



Tools & Environment - Lasagne code example

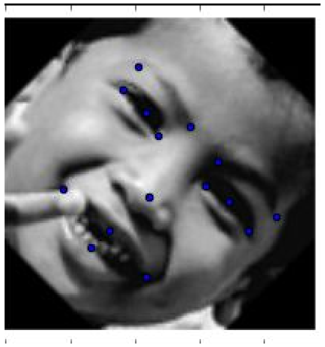
```
# create a small convolutional neural network
from lasagne.nonlinearities import leaky_rectify, softmax
network = lasagne.layers.InputLayer((None, 3, 32, 32), input_var)
network = lasagne.layers.Conv2DLayer(network, 64, (3, 3), nonlinearity=leaky_rectify)
network = lasagne.layers.Conv2DLayer(network, 32, (3, 3), nonlinearity=leaky_rectify)
network = lasagne.layers.Pool2DLayer(network, (3, 3), stride=2, mode='max')
network = lasagne.layers.DenseLayer(lasagne.layers.dropout(network, 0.5),
                                    128, nonlinearity=leaky_rectify,
                                    W=lasagne.init.Orthogonal())
network = lasagne.layers.DenseLayer(lasagne.layers.dropout(network, 0.5),
                                    10, nonlinearity=softmax)
```

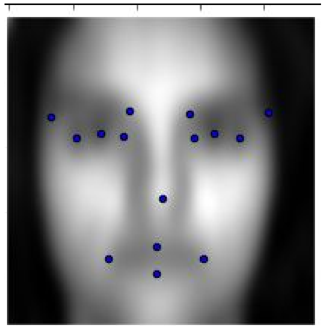


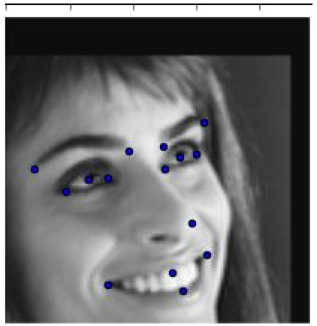
Data Augmentation

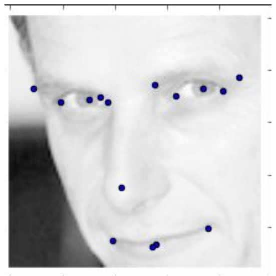
Overfitting can theoretically be reduced by adding more training data.

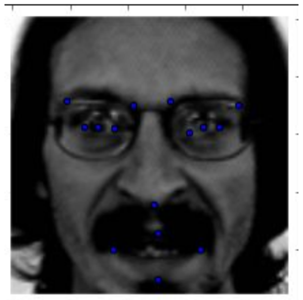
If no more are available, we can create new labeled training data by "tweaking" existing data and adjusting known keypoints to fit.

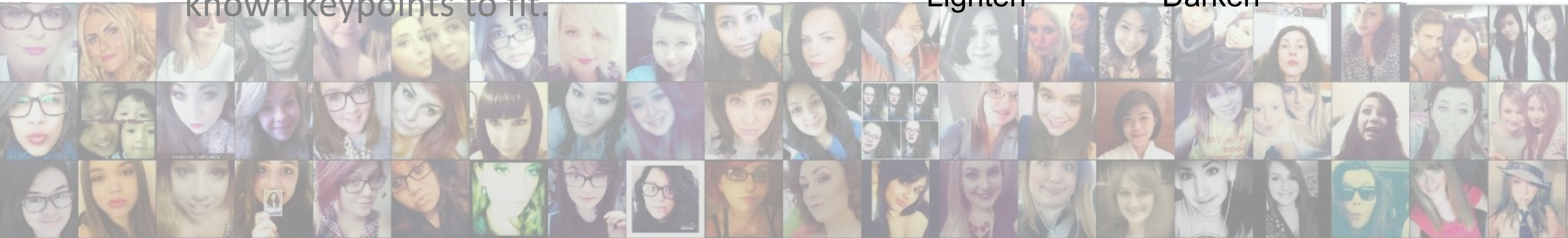

Rotation


Blur


Shift


Lighten

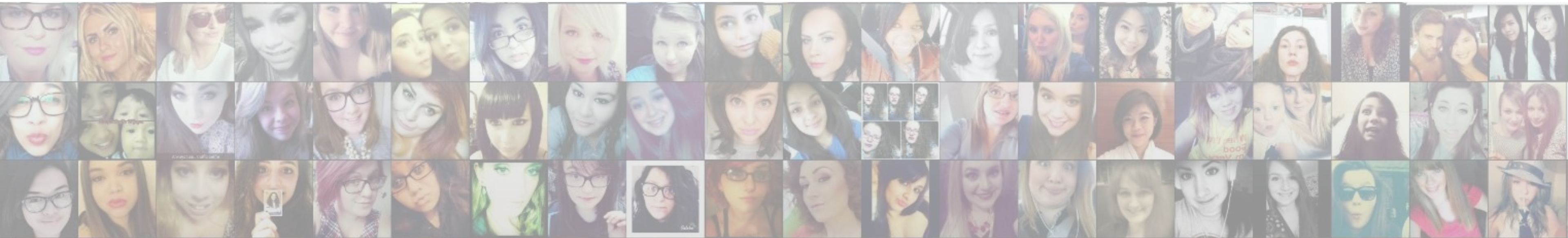

Darken



Data Augmentation

Our data augmentation had one of two results.   Most commonly, it caused our models to diverge and output ‘nan’ over and over for $1 an hour.

When the augmentations were “toned down” it seemed to increase the time significantly without helping the results much.

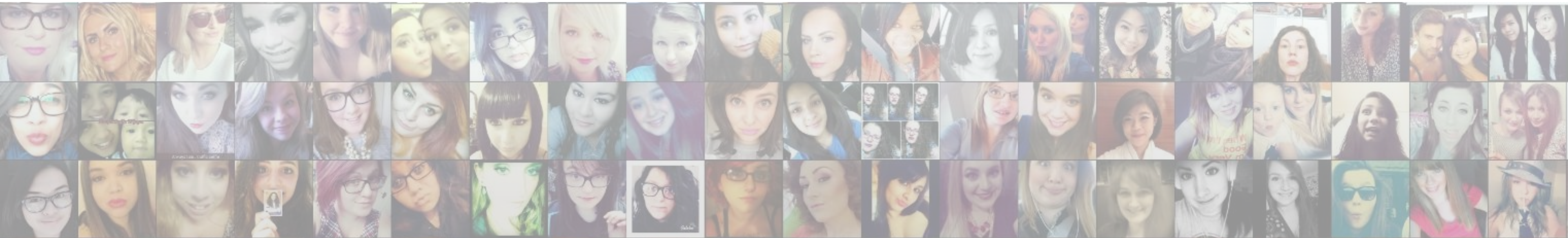


Pragmatic Approach

1. Start with working example
2. Research past facial recognition work
3. Create test plan
4. Experiment with first parameter, keep best case then move on to the next one

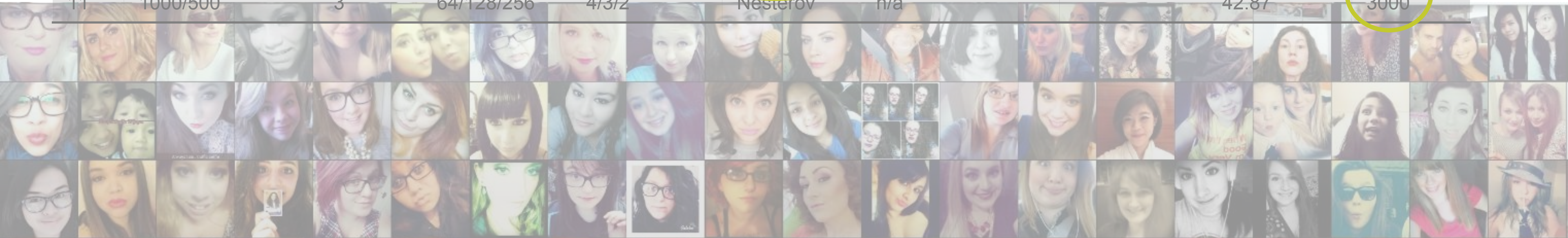
Common Neural Net "Knobs"

- Number of hidden layers
- Type of hidden layers
- Number of nodes per hidden layer
- Activation functions for each layer
- Dropout percentage
- Cost function itself
- Regularization parameter in cost function
- Lambda value
- Convolution kernel size and stride length
- Pooling layers: kernel size and function
- Learning rate
- Momentum
- # epochs
- Training set size and variations
- Cleanliness of training set
- Weight initialization
- Optimization method



Convolution Model Comparisons

	Nodes per hidden layer	# conv layers	# filters per conv layer	Conv kernel sizes	Update method	Dropout	Val Loss @ 300	Dur per epoch (sec)	# epochs
1	1000/500	2	48/96	2/2	Nesterov	n/a	.02055	22.90	300
2	1000/500	4	24/48/96/192	4/2/2/2	Nesterov	n/a	.02056	17.56	300
3	1000/500	3	32/64/128	4/3/2	Nesterov	n/a	.01082	19.45	300
4	1000/1000	3	32/64/128	4/3/2	Nesterov	n/a	.02056	19.48	300
5	1000/500	3	64/128/256	4/3/2	Nesterov	n/a	.00387	42.87	300
6	1000/1000	3	64/128/256	4/3/2	Nesterov	n/a	.02056	42.93	300
7	1000/500	3	64/128/256	7/5/3	Nesterov	n/a	.00436	43.80	300
8	1000/500	3	64/128/256	4/3/2	Nesterov	Conv layers 0.1/0.2/0.3	.00499	43.77	300
9	1000/1000	3	64/128/256	4/3/2	Nesterov	Input layer 0.15	.00443	42.88	300
10	1000/500	3	64/128/256	4/3/2	Adagrad	n/a	.	43.14	300
11	1000/500	3	64/128/256	4/3/2	Nesterov	n/a		42.87	3000

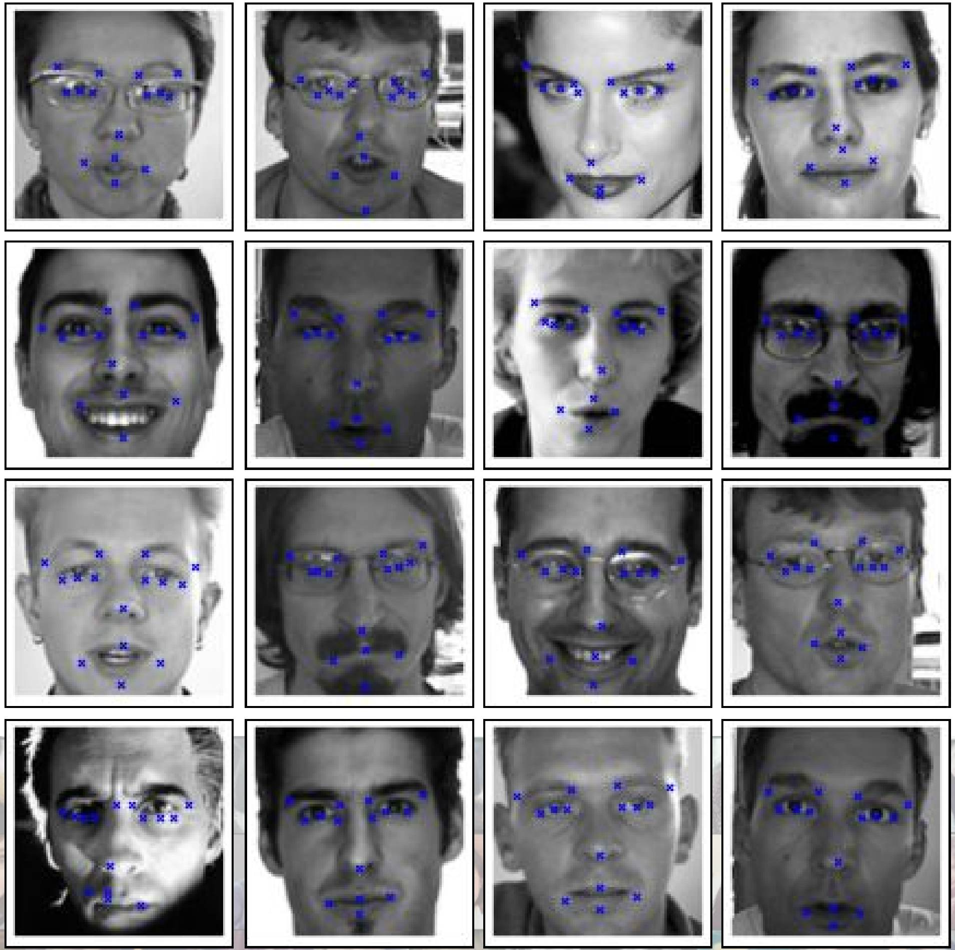
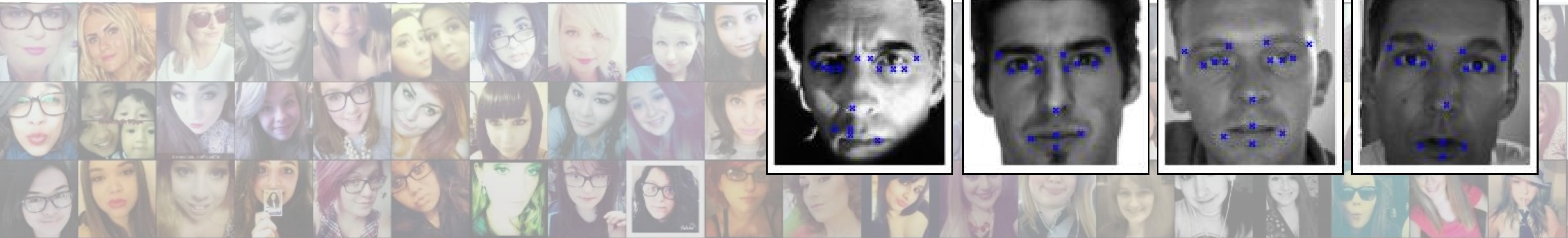


Non-NN models: Means

- RMSE = (mean(sqrt(y - y_pred)))^2
- Use the Means
- **RMSE = 3.1949**

Non-NN models: Linear Regression

- RMSE = (mean(sqrt(y - y_pred)))^2
- Linear Regression
- **RMSE = 2.6168**

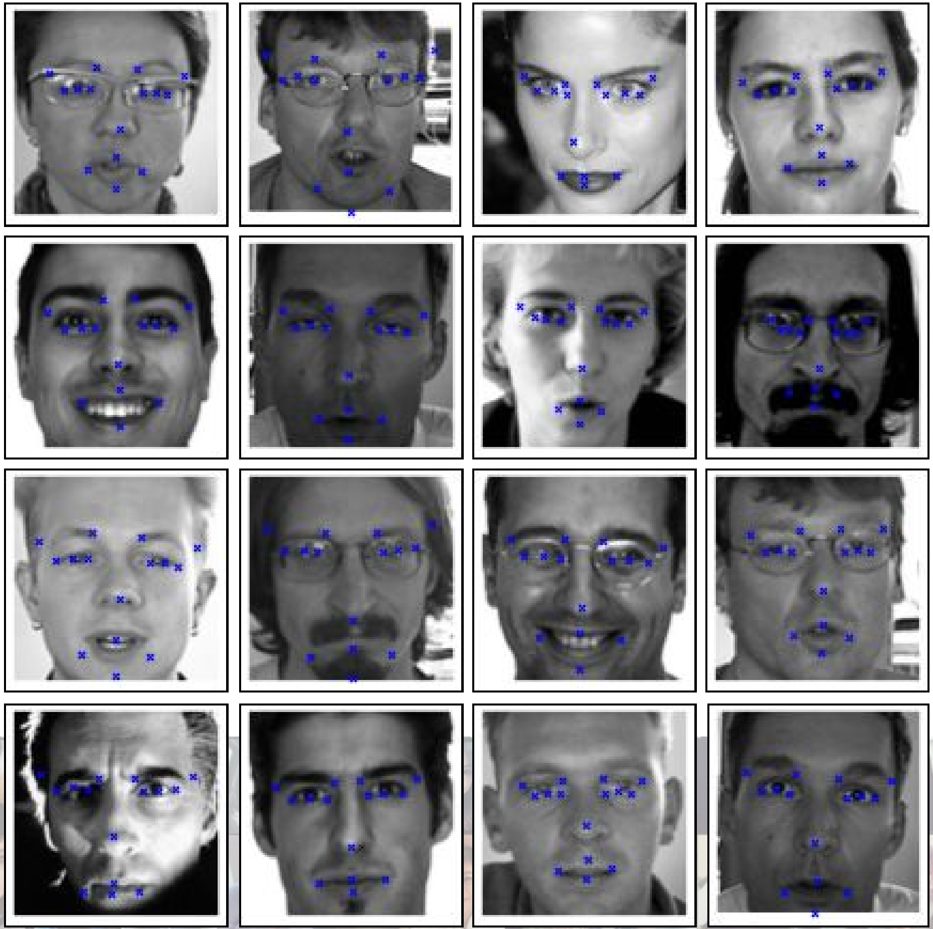



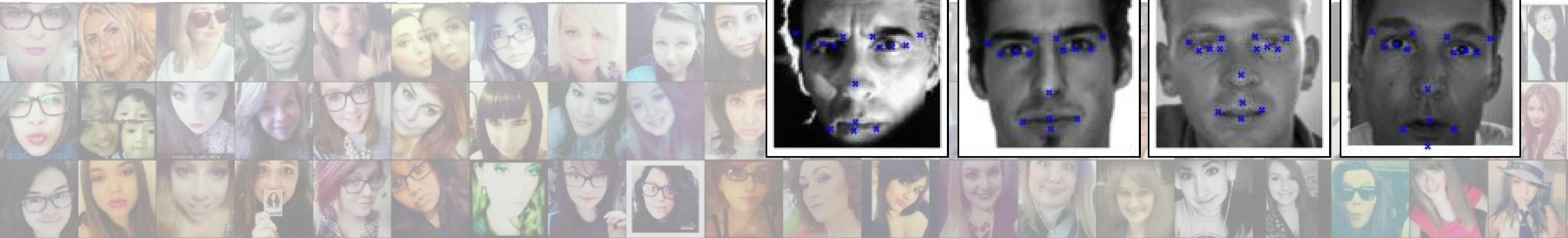
NN models: Non-Convolutional NN

- RMSE = (mean(sqrt(y - y_pred)))^2

- NN with 1 hidden layer (200 nodes)
- 100 epochs (stochastic GD)

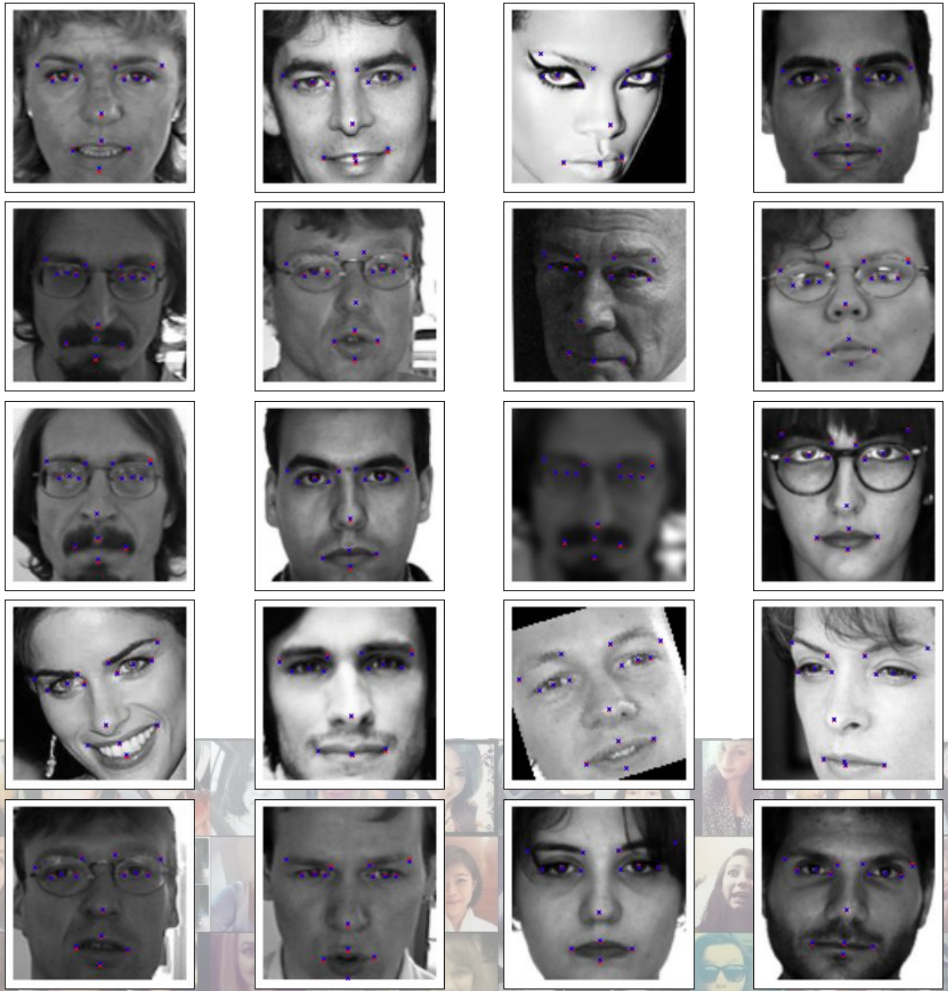
- start: RMSE = 5.7269
- end: **RMSE = 3.3448**

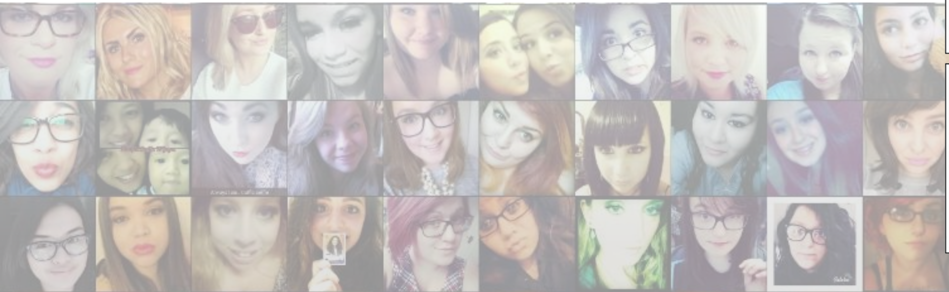




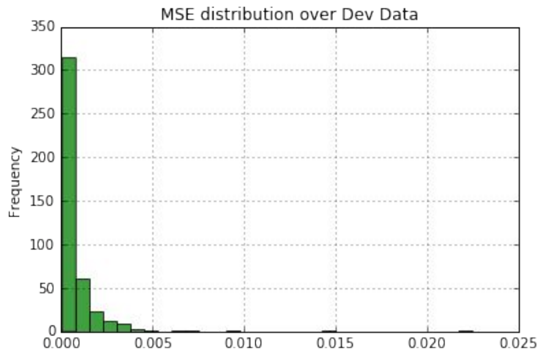
NN models: Convolutional NN

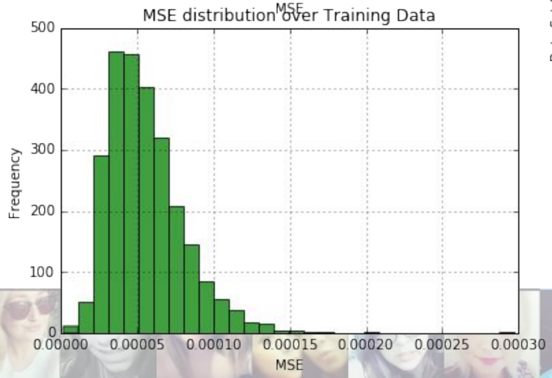
- RMSE = (mean(sqrt(y - y_pred)))^2
- Convo NN
  - 32*64*128 (3*3) patch size convo layers
  - 1000*1000*30 fully connected NN
- 100 epochs (stochastic GD)
- start: RMSE = 3.47
- end: **RMSE = 1.423**

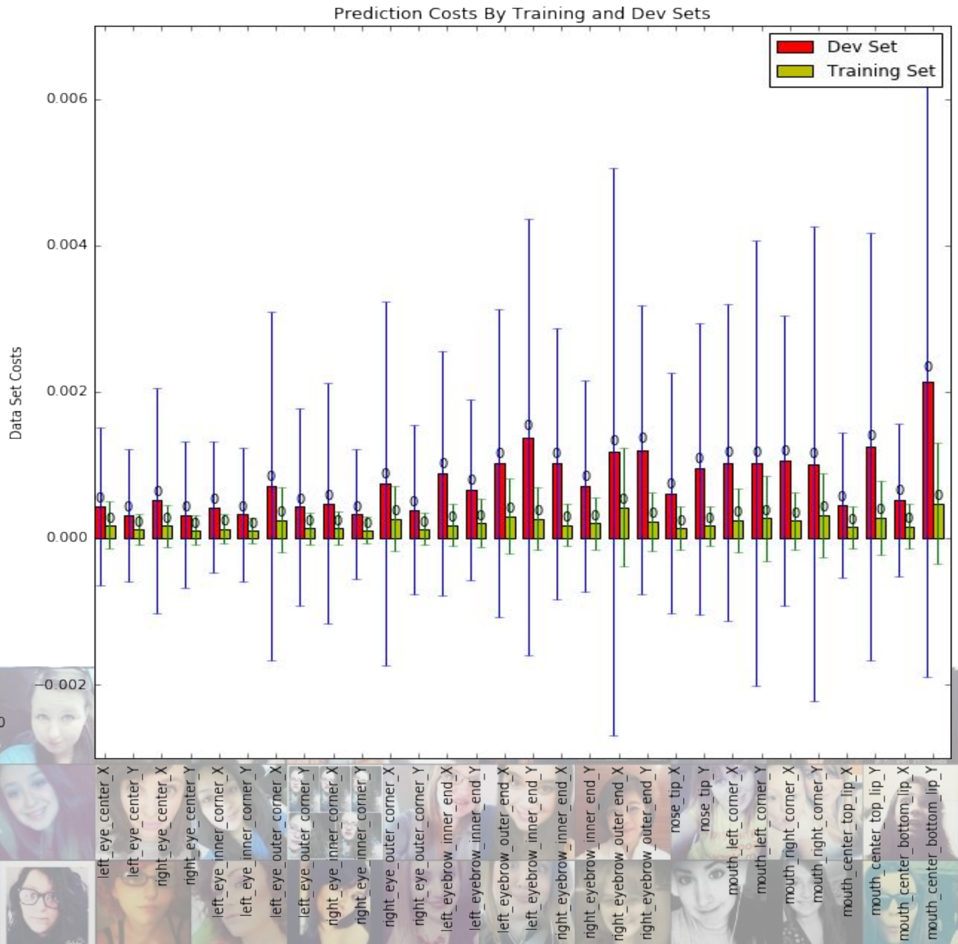




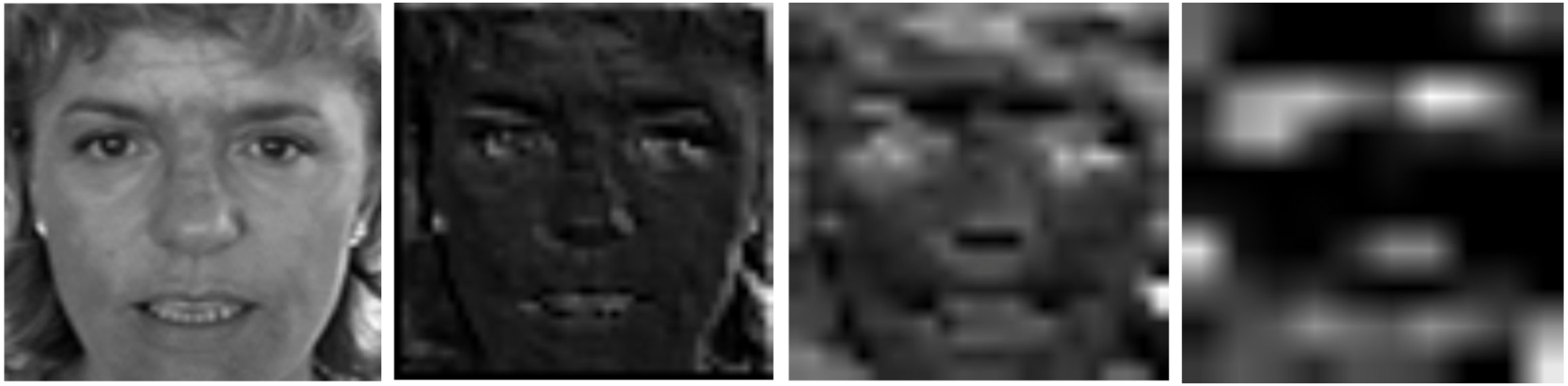
NN models: Convolutional NN Analysis

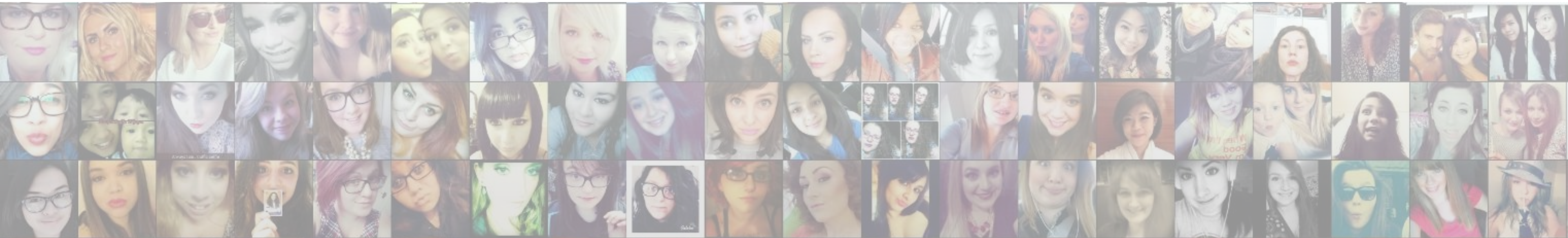






NN models: Convolutional NN - Convolutional Filters

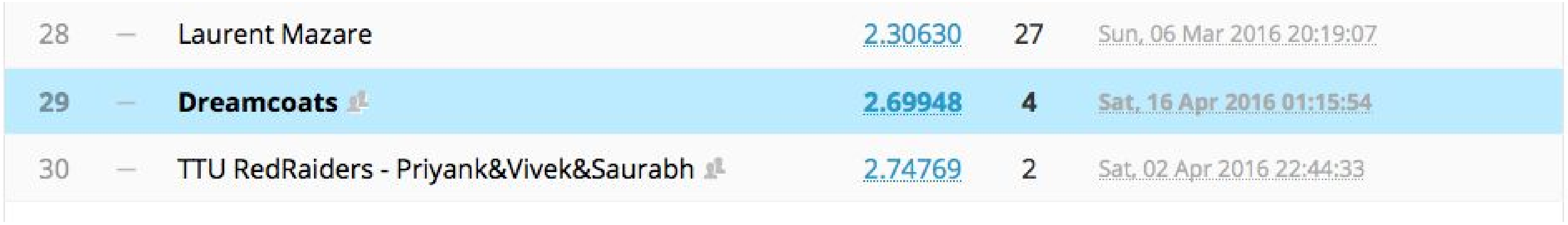




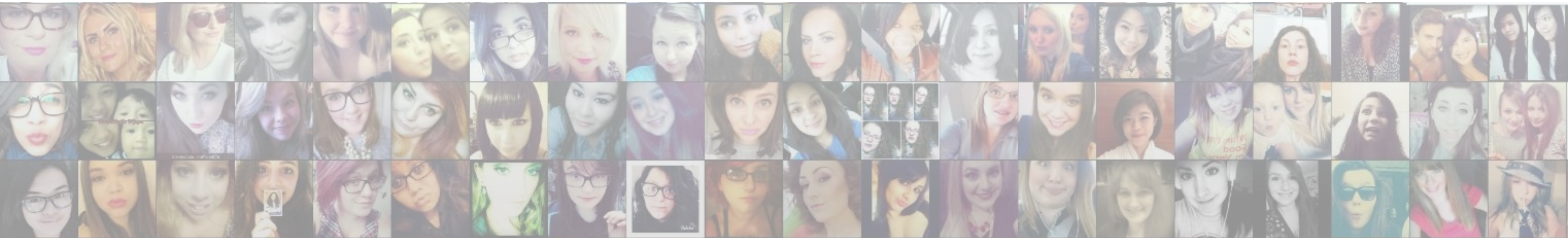
Final results

Our best model was a set of convolutional neural nets, one for each keypoint, using an augmented data set. This enabled us to overcome the problem of inconsistent amounts of training data for each keypoint. Even with a modest net size, we landed 29th place.

Not nearly as impressive as our classmates' #3, but not terrible, either.



28	—	Laurent Mazare	2.30630	27	Sun, 06 Mar 2016 20:19:07
29	—	**Dreamcoats**	**2.69948**	**4**	**Sat, 16 Apr 2016 01:15:54**
30	—	TTU RedRaiders - Priyank&Vivek&Saurabh	2.74769	2	Sat, 02 Apr 2016 22:44:33



Future path

The obvious next step would be to train a set of deep nets, one for each keypoint, and merge the results into a final net which would adjust predictions slightly by the position of other predictions.

This might help the ‘problem points’ a bit.

But we have new classes starting in 2 weeks...