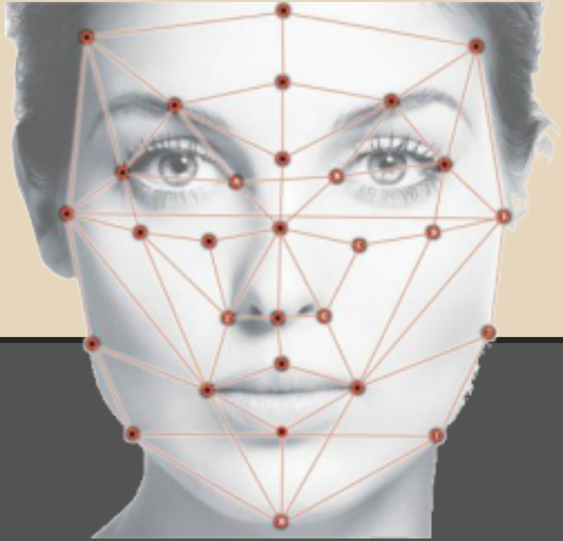
Facial Keypoints Detection:
An Effort to Top the Kaggle Leaderboard

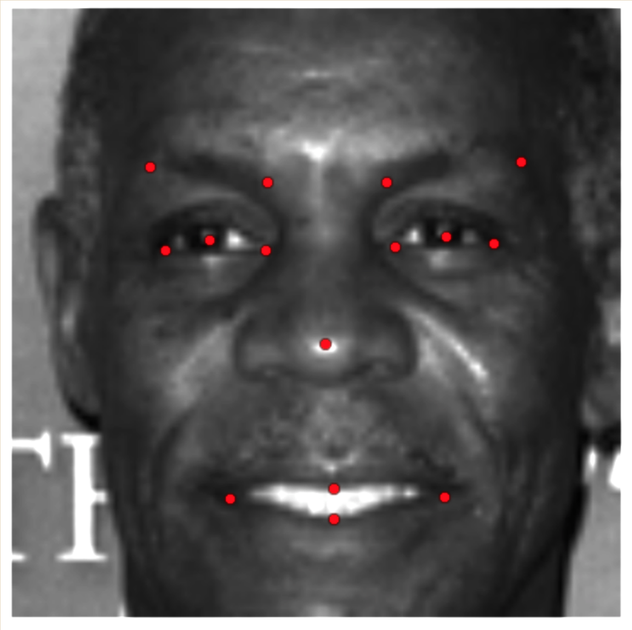


MIDS W207 | Summer 2015 | Final Project

Chris Dailey | Younghak Jang | Marguerite Oneto | Lei Yang

The Challenge

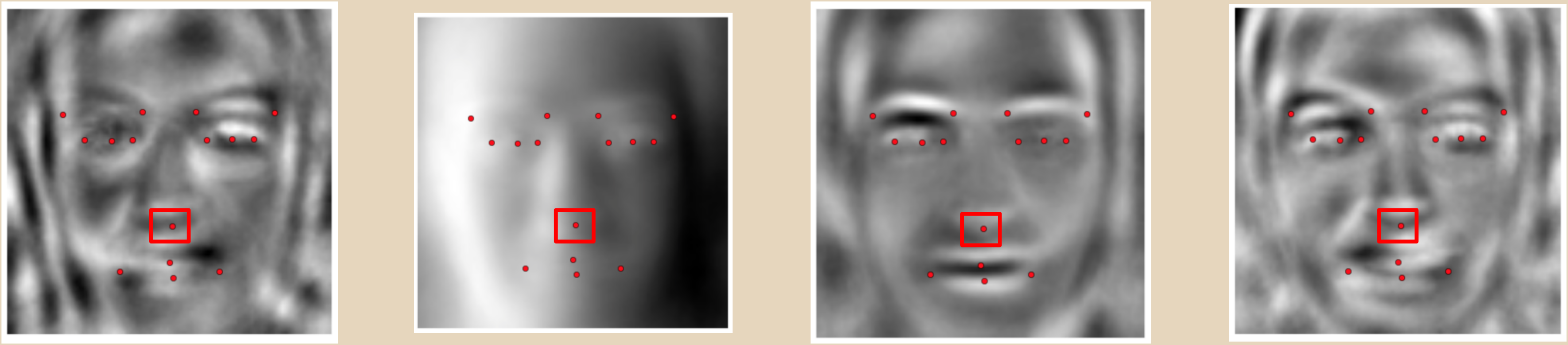
Predict the x-y coordinates of 15 keypoints on the face:



Initial Kaggle Submission

Baseline: Mean Patch Searching

Step 1 - Feature Engineering: Use PCA to create 125 eigenfaces



Step 2 - Train: Create "Golden Patches" -- the standard of comparison

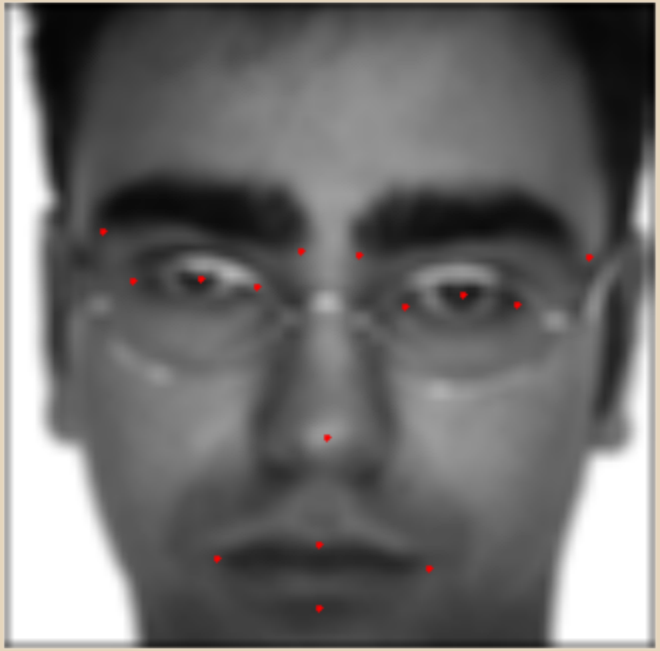
Step 3 - Predict: Compare patches from test images to Golden Patches. Closest wins!

Next Step: Feature Engineering

FE1: Histogram Stretching

- Provides Better Contrast

FE2: Gaussian Blurring



- Reduce local (high frequency) noise: e.g. glass effect

FE3: Flipping the Image

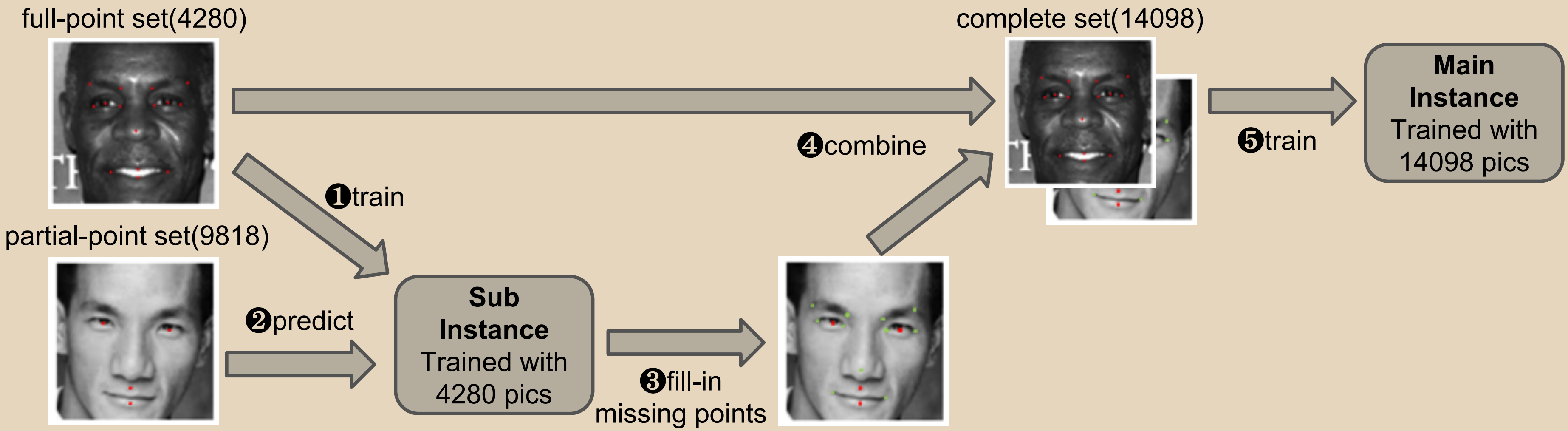
- Familiar to eyes, new to computer
- Increase training sample with no cost → reduce the chance of overfitting

FE4: Keypoint Grouping

- Group images with as many common keypoints as possible → implicitly encode the geometric constraints between points.
- Train model separately for 2 training groups, and predict on test data
  - Model 1: 4 points x 14000 images
  - Model 2: 15 points x 4280 images
- Synthesize predictions from 2 models to obtain final prediction
  - Weighted average for common point prediction

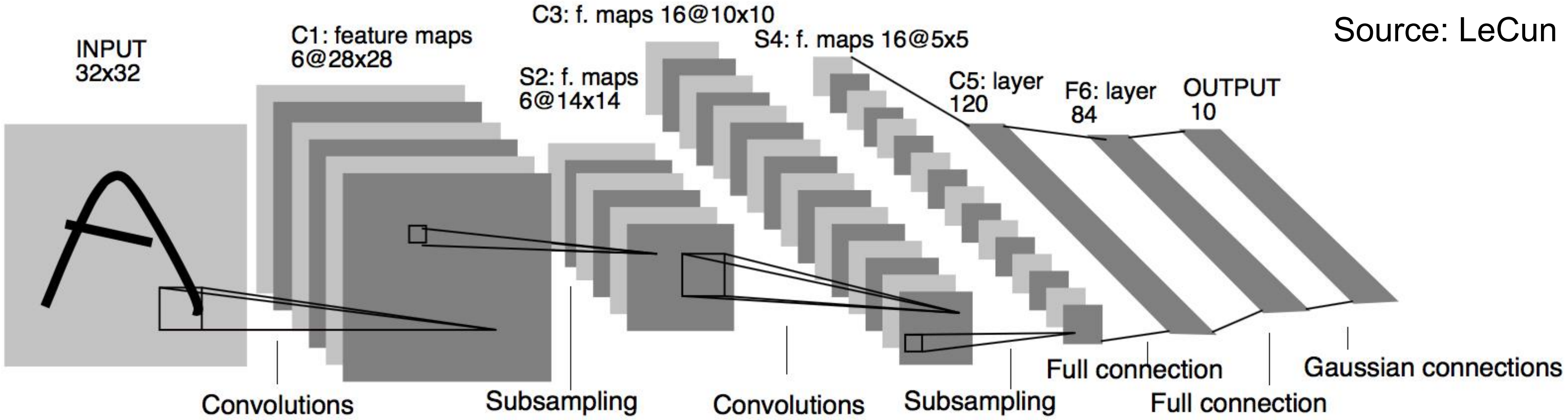
FE5: Use Predictions as Train Data

- Split train set into full-point(4280) and partial-point(9818) Subset
- Train an instance with full-point set and use it to predict on the partial-point set
- Fill the missing points in partial-point set with predictions and merge it with full-point set to get complete set(14098)



Next Step: The Model

Model: Convolution Neural Network



Source: LeCun 98'

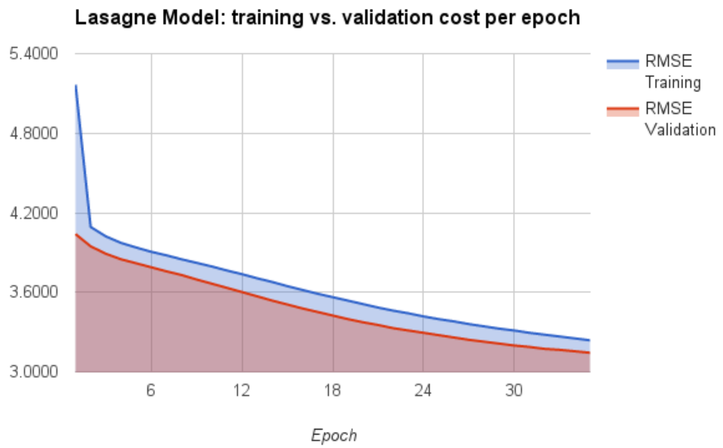
- 3 convolution and subsampling layers: (96x96) → (11x11)
- 2 fully connected neural layers: 600 neurons
- Direct output of coordinate prediction (no activation function)

The Lasagne Model

```python
def getCNN(n_output):
    net = NeuralNet(
        layers=[
            ('input', layers.InputLayer),
            ('conv1', layers.Conv2DLayer),
            ('pool1', layers.MaxPool2DLayer),
            ('conv2', layers.Conv2DLayer),
            ('pool2', layers.MaxPool2DLayer),
            ('conv3', layers.Conv2DLayer),
            ('pool3', layers.MaxPool2DLayer),
            ('hidden4', layers.DenseLayer),
            ('hidden5', layers.DenseLayer),
            ('output', layers.DenseLayer),
            ],
        input_shape=(None, 1, 96, 96),
        # 3 convoluational layer
        conv1_num_filters=32, conv1_filter_size=(3, 3), pool1_pool_size=(2, 2),
        conv2_num_filters=64, conv2_filter_size=(2, 2), pool2_pool_size=(2, 2),
        conv3_num_filters=128, conv3_filter_size=(2, 2), pool3_pool_size=(2, 2),
        # 2 fully connected hidden layer
        hidden4_num_units=500, hidden5_num_units=500,
        # fully connected output layer, no activation function to give continuous output
        output_num_units=n_output, output_nonlinearity=None,

        update_learning_rate=0.02,
        update_momentum=0.80,

        regression=True,
        max_epochs=35,
        verbose=1,
        )
```



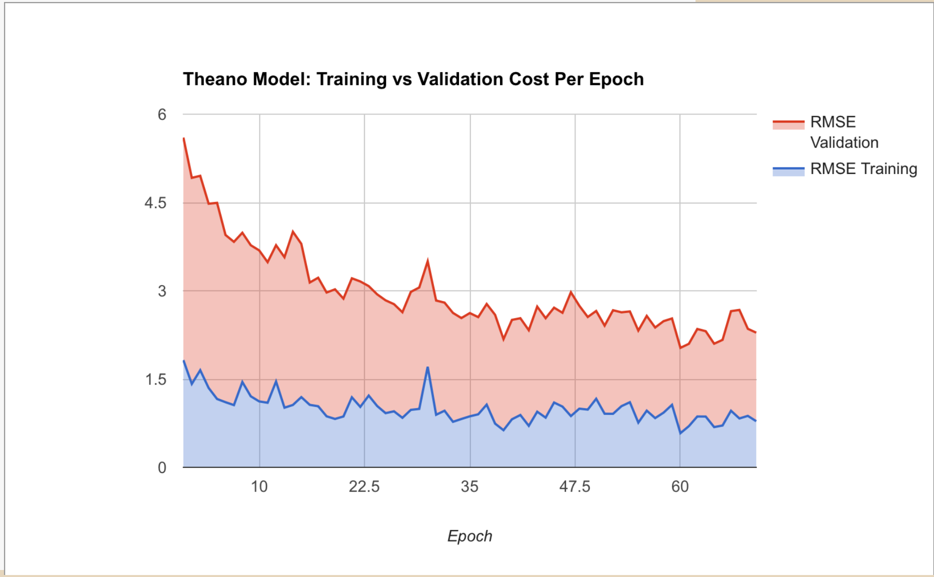
The Theano Model

```
    self.cost = T.sum(T.sqr(Y - y_hat_train)) #T.sqrt(T.mean(T.sqr(Y - y_hat_train)))
    update = self._backprop(self.cost, self.params)
    self.train = theano.function(inputs=[X, Y], outputs=self.cost, updates=update, allow_input_downcast=True)
    self.predict = theano.function(inputs=[X], outputs=y_hat_predict, allow_input_downcast=True)

def _model(self, X, w_1, w_4, w_5, p_1, p_2):
    l1 = self._dropout(T.flatten(max_pool_2d(T.maximum(conv2d(X, w_1, border_mode='full'),0.), (2, 2)), outdim=2), p_1)
    l4 = self._dropout(T.maximum(T.dot(l1, w_4), 0.), p_2)
    return T.dot(l4, w_5)

def _dropout(self, X, p=0.):
    if p > 0:
        X *= self.srng.binomial(X.shape, p=1 - p)
        X /= 1 - p
    return X

def _backprop(self, Cost, w, alpha=0.0001, rho=0.66, epsilon=1e-6):
    grads = T.grad(cost=Cost, wrt=w)
    updates = []
    for w1, grad in zip(w, grads):
        # adding gradient scaling
        acc = theano.shared(w1.get_value() * 0.0)
        acc_new = rho * acc + (1 - rho) * grad ** 2
        gradient_scaling = T.sqrt(acc_new + epsilon)
        grad = grad / gradient_scaling
        updates.append((acc, acc_new))
        updates.append((w1, w1 - grad * alpha))
    return updates
```



Results

The Benefits of Feature Engineering

Feature Engineering Benchmark (3-Layer Lasagne Model): Training Time / Accuracy

No.	Method Applied	Train Set Size	# of epochs	mini-batch size	Train Time	RMSE	Improvement
0.	No feature engineering	2140	100	100	96min	3.85	baseline
1.	Histogram Stretching	2140	100	100	125min	3.66	5.0%
2.	#1 + Gaussian Blur	2140	100	100	110min	3.73	3.1%
3.	#1 + Gaussian Blur	2140	100	50	121min	3.52	8.6%
4.	#1 + Gaussian Blur	2140	200	100	244min	3.58	7.0%
5.	#2 + Horizontal Flip	4280	200	100	364min	3.36	12.7%
6.	Predict partial training set and combine with full-point set	14098	200	100	1235min	3.55	7.8%

The Kaggle Leaderboard

	RMSE	Place
Initial Submission	5.79	42nd
Lasagne Model	3.91	34th
Theano Model	2.92	13th

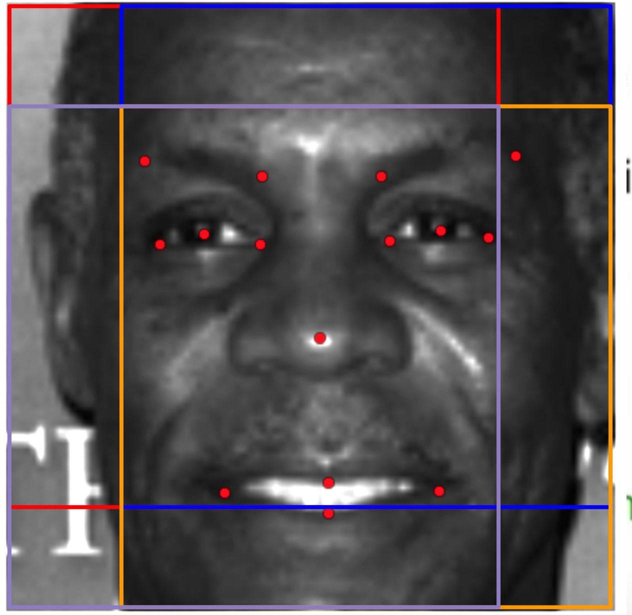
Parsimony Wins!

Future Work

Next Steps: Another FE Trick

Tricks that significantly improve generalization

- Train on random 224x224 patches from the 256x256 images to get more data. Also use left-right reflections of the images.
    - At test time, combine the opinions from ten different patches: The four 224x224 corner patches plus the central 224x224 patch plus the reflections of those five patches.



Next Steps: More Computing Power

- Diagnose speed issues
- Better utilize GPUs: EC2, CUDA
- Would allow for more experimentation
    - Alternative cost function for smoother convergence
    - Alternative hyperparameters