Detecting Facial Keypoints

using Conv Nets

Alex Byron Minghu
Siddarth Yuhang

1.Overview

Problem Statement

The goal is to detect 15 locations on the human face (keypoints) given a digital image.

- Images are 96x96 pixels
- Each keypoint is given as an x and y coordinate
- A set of labeled images was provided for training a model
- Not all training images had all keypoints labeled

Challenges

This type of problem is a key building block for many applications and is very challenging for the following reasons:

- Feature variation person to person
- 3D pose
- Position
- Viewing angle
- Illumination conditions

Not enough data to create a generic model that was not overfit. With potentially 100,000 features in our models and only 2,000 complete training images, there was be a tendency to overfit the model.

Solution Approach

Python Packages:

- Theano
- Lasange
- nolearn

EC2 Setup:

- Amazon EC2 g2.2xlarge GPU server
- NVIDIA GPU with 1,536 CUDA cores

Our Final Solution

Data Cleaning

Histogram Normalization

Conv Net Architecture:

  6 * 12 * 36 feature maps

  (5X5), (7X7), (9X9) Conv filter
  No pooling

  Two 500 hidden layers

  Dropout after each Conv Layer, and first Hidden Layer (0.1, 0.2, 0.3, 0.5)

2. Introduction

Convolutional Neural Nets (ConvNN)

ConvNNs pioneered for optical character recognition (LeCun et al. 1989)

Large scale object classification using GPUs (Krizhevsky, Sutskever, and Hinton 2012)

Face detection (Sun, Wang, and Tang 2013)

Object detection (Girshick et al. 2013)

Pedestrian detection (Sermanet et al. 2013)

Human posture detection (Toshev and Szegedy 2013)

Playing Go (Clark and Storkey 2014)

Training a Convolutional Net

Data spelunking

Preprocessing

Augmentation

Choose an architecture

Training

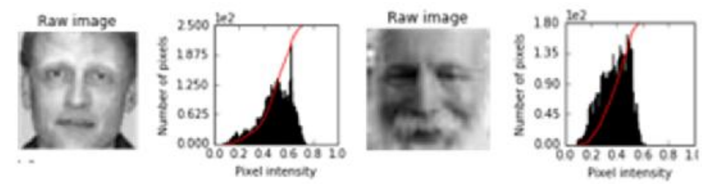
Optimization/Regularization

Data Spelunking

The contrast of some images can be enhanced:
- Solution: Normalization and whitening

Two label sets with different Labelling criteria:
- Solution: Data cleaning and separation

Less training examples than parameters:
- Solution: Data augmentation

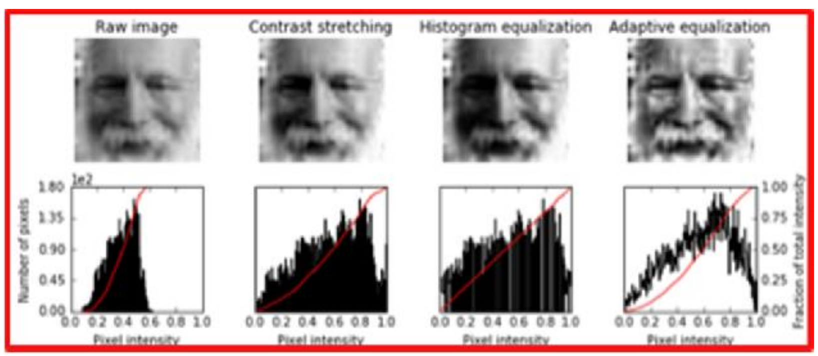


Data Processing - Normalization

Histogram stretching: rescale images to include intensities that fall within certain percentiles
img_rescale = exposure.rescale_intensity(img, in_range=(0, 0.8))

Histogram equalization: spreads out the most frequent intensity values in an image
img_eq = exposure.equalize_hist(img)

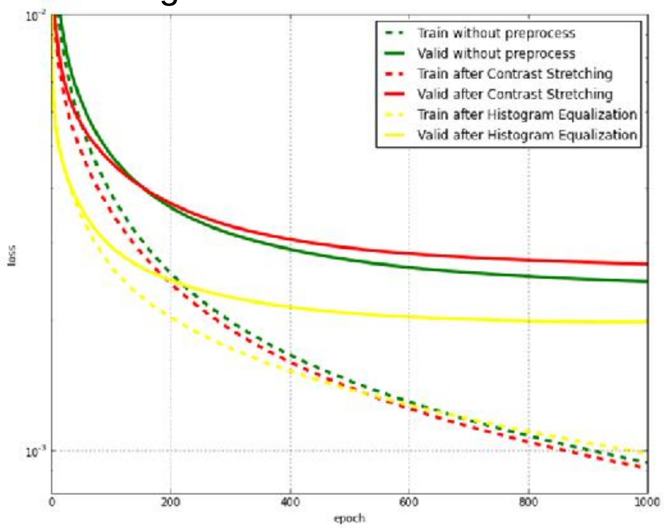
Adaptive equalization: automatically adapts to time-varying properties of the communication channel
e.g. img_adapteq = exposure.equalize_adapthist(img, clip_limit=0.03)



Data Processing - Normalization
-- single hidden layer as the example

Histogram Equalization slightly improved the model accuracy.

Training and validation loss curves



Training and validation loss of last epoch

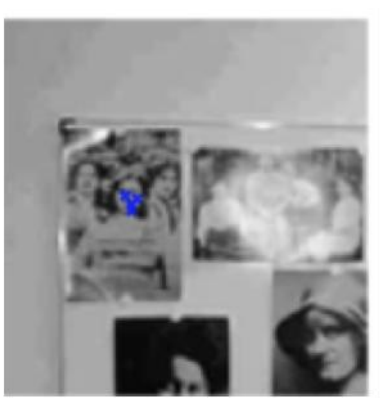
Preprocess	Train_loss	Validation_loss	Duration_time(s)
Without Preprocessing	0.00094	0.00244	0.09263
Contrast stretching	0.00091	0.00268	0.09288
Histogram equalization	0.00099	0.00197	0.09257

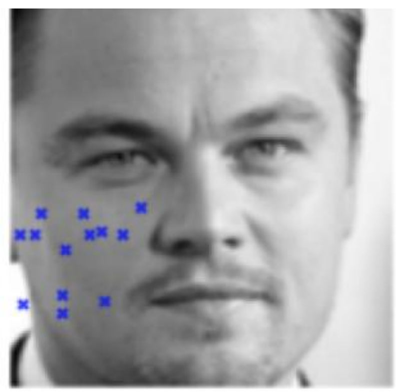
Data Processing - Data Cleaning

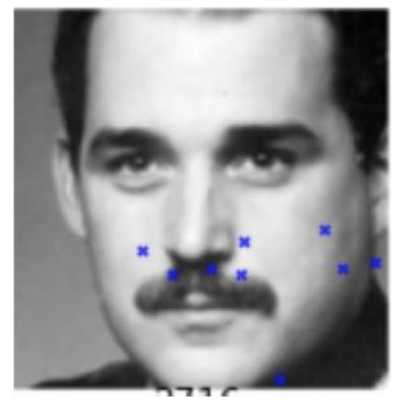
Multiple faces

Misplaced labels

Poor image quality







Data Augmentation

Generate new labeled data from existing training data

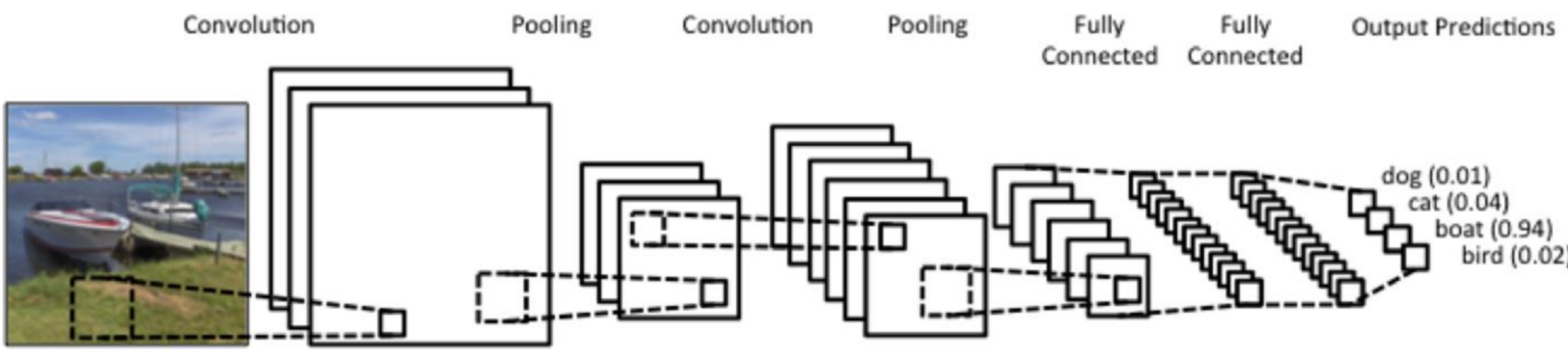
- **Mirroring data along y-axis**
- Shifting
- Stretching
- Cropping
- Changing brightness
- Slight rotation (a few degrees)

Training Loss & Validation Loss

- Cross-validation
  - Used instead of conventional validation (split data into training and test data)
  - Want to maximize the amount of training data
- Training loss
  - Loss on training data
  - Measures how well model fits training data
  - What the ConvNN wants to minimize
- Validation loss
  - Loss on withheld validation data

3. Base Conv Net Architectures

Architecture of a Basic Conv Net



Architecture of a Basic Conv Net

```
input_shape=(None, 1, 96, 96),
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),
hidden4_num_units=500, hidden5_num_units=500, output_num_units=30,
output_nonlinearity=None,
```

```
  input             (None, 1, 96, 96)       produces    9216 outputs
  conv1             (None, 32, 94, 94)      produces  282752 outputs
  pool1             (None, 32, 47, 47)      produces   70688 outputs
  conv2             (None, 64, 46, 46)      produces  135424 outputs
  pool2             (None, 64, 23, 23)      produces   33856 outputs
  conv3             (None, 128, 22, 22)     produces   61952 outputs
  pool3             (None, 128, 11, 11)     produces   15488 outputs
  hidden4           (None, 500)             produces     500 outputs
  hidden5           (None, 500)             produces     500 outputs
  output            (None, 30)              produces      30 outputs
```

```
  epoch    train loss    valid loss    train/val  dur
-------  ------------  ------------  -----------  -----
750       0.00144       0.00175      0.82118  2.94s
```

from Daniel's work http://danielnouri.org/notes/2014/12/17/using-convolutional-neural-nets-to-detect-facial-keypoints-tutorial/

Net6 Architecture

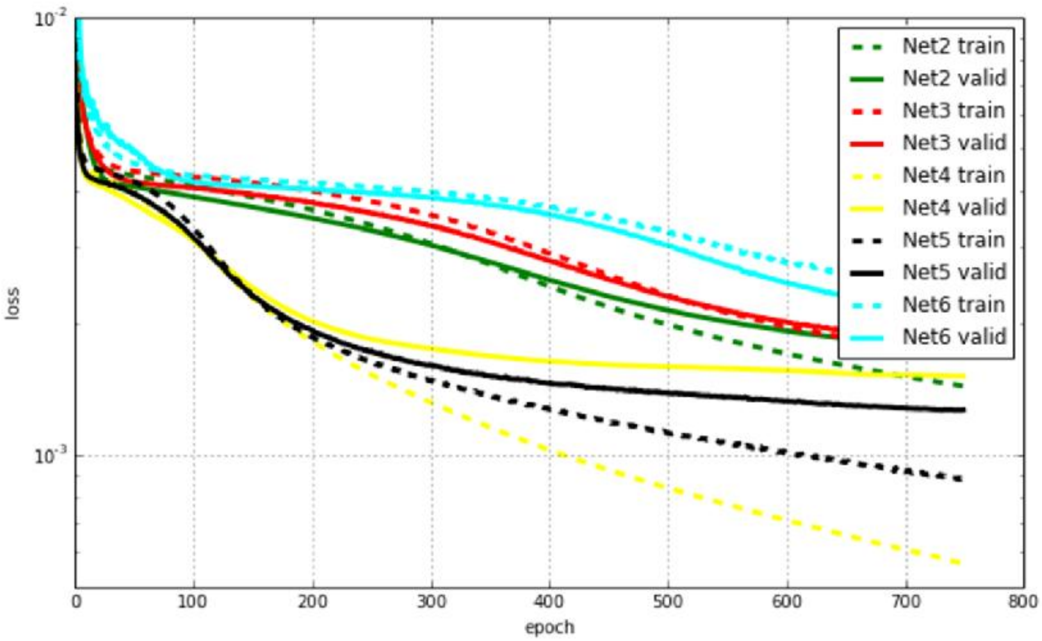
```
input_shape=(None, 1, 96, 96),
conv1_num_filters=32, conv1_filter_size=(3, 3), pool1_pool_size=(2, 2),
dropout1_p=0.1,
conv2_num_filters=64, conv2_filter_size=(2, 2), pool2_pool_size=(2, 2),
dropout2_p=0.2,
conv3_num_filters=128, conv3_filter_size=(2, 2), pool3_pool_size=(2, 2),
dropout3_p=0.3,
hidden4_num_units=500,
dropout4_p=0.5,
hidden5_num_units=500,output_num_units=30,output_nonlinearity=None,
```

```
input                          (None, 1, 96, 96)           produces    9216 outputs
  conv1                        (None, 32, 94, 94)          produces  282752 outputs
  pool1                        (None, 32, 47, 47)          produces   70688 outputs
  dropout1                     (None, 32, 47, 47)          produces   70688 outputs
  conv2                        (None, 64, 46, 46)          produces  135424 outputs
  pool2                        (None, 64, 23, 23)          produces   33856 outputs
  dropout2                     (None, 64, 23, 23)          produces   33856 outputs
  conv3                        (None, 128, 22, 22)         produces   61952 outputs
  pool3                        (None, 128, 11, 11)         produces   15488 outputs
  dropout3                     (None, 128, 11, 11)         produces   15488 outputs
  hidden4                      (None, 500)                 produces     500 outputs
  dropout4                     (None, 500)                 produces     500 outputs
  hidden5                      (None, 500)                 produces     500 outputs
  output                       (None, 30)                  produces      30 outputs
```

```
  epoch    train loss    valid loss    train/val  dur
-------  ------------  ------------  -----------  -----
750             0.00239       0.00210      1.13988  3.01s
```

from Daniel's work http://danielnouri.org/notes/2014/12/17/using-convolutional-neural-nets-to-detect-facial-keypoints-tutorial/

Comparison of Different ConvNets



Net2: Basic ConvNN

Net3: Basic ConvNN + **data augmentation**

Net4: Basic ConvNN + **adaptive learning rate & momentum**

Net5: Basic ConvNN + data argumentation
+ adaptive learning rate
& momentum

Net6: Basic ConvNN + data argumentation
+ adaptive learning rate
& momentum
+ **dropout**

Though the performance of Net6 was worse than that of Net5,
we decided to use Net6 as baseline for our explorations

4. Exploring Hyperparameters & Conv Net architectures

Parameter Tuning for Deep Learning

Hyper-parameters:

Learning rate

Mini-batch size

#training iteration

Momentum

Regularization coefficient

… ...

ConvNet Architecture:

Layer pattern

#filter

Receptive field (filter size)

stride

zero-padding

Pooling size

Dropout rate

Examples of Parameter Search
-- #hidden_unit & filter_size of conv1

Train & validation losses of last epco for various hidden units of (simple neural net terminated at 1000 epochs as demonstration)

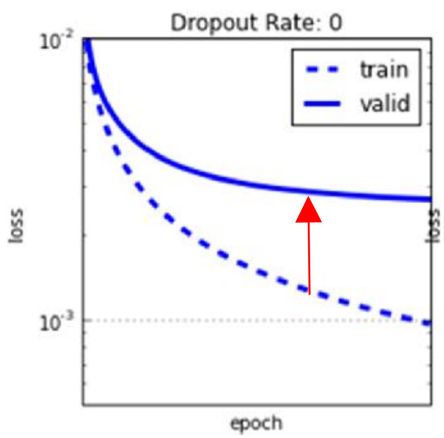
Num_hidden_layer	Train_loss	Validation_loss	Duration_time(s)
100	0.00234	0.00302	0.07048
200	0.00141	0.00252	0.08022
300	0.00107	0.00245	0.0907
400	0.00099	0.00259	0.09771
500	0.00094	0.00253	0.10468
600	0.00091	0.00256	0.12092
700	0.00083	0.00266	0.12741
800	0.00081	0.00277	0.13913
900	0.00072	0.00279	0.15141
1000	0.00069	0.00283	0.14717

Train & validation losses of last epco for various filter sizes of the 1st convolutional layer (terminated at 150 epochs as demonstration)

conv1_filter_size	Train_loss	Validation_loss	Duration_time(s)
(3, 3)	0.00294	0.00286	2.88116
(4, 4)	0.0028	0.00276	2.95509
(5, 5)	0.00284	0.00275	2.89244
(6, 6)	0.00248	0.00243	3.10353
(7, 7)	0.00227	0.00228	2.90226
(8, 8)	0.00235	0.0024	2.87946
(9, 9)	0.00227	0.0023	3.44563
(10, 10)	0.00221	0.00227	3.61742
(11, 11)	0.00235	0.0024	3.50213
(12, 12)	0.00211	0.00216	3.75244

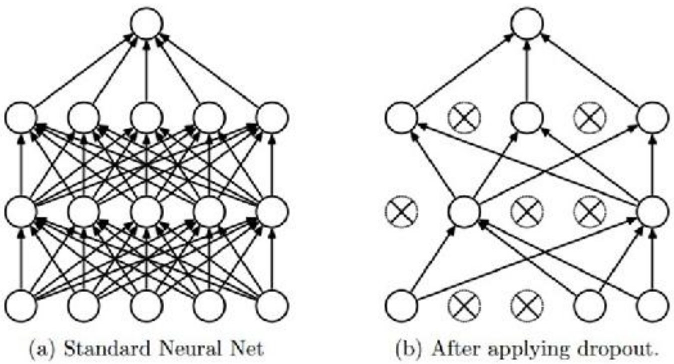
Visual Inspection during Parameter Tuning

Curves of training & valid. losses

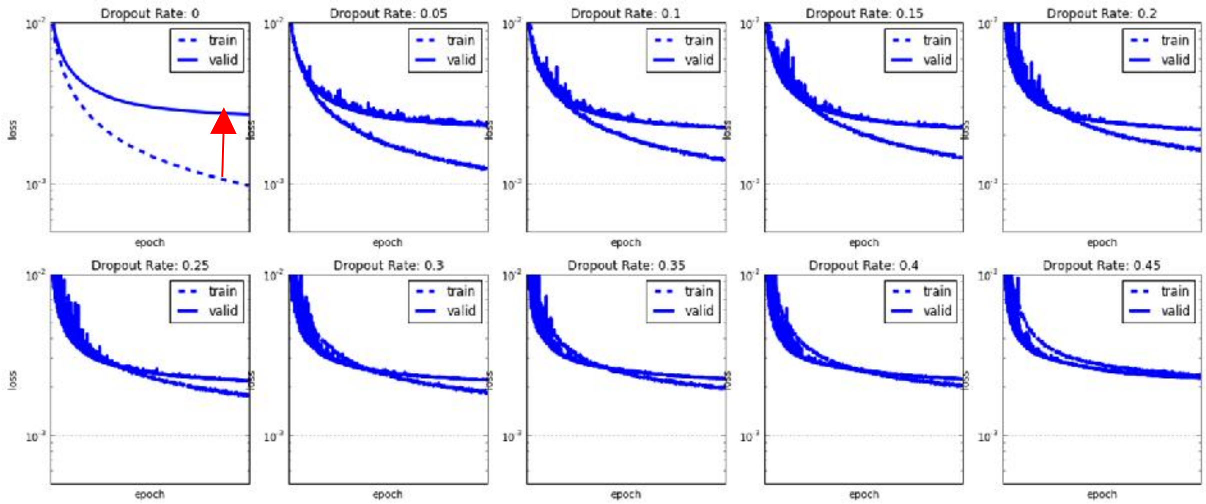


Visualization of Conv1's weights (basic ConvNet conv1_filter_size=(3, 3), conv1_num_filters=32)

Visual Inspection -- dropout rate as another example



The influence of dropout rate on train & valid losses of a simple single-layer-hidden NN



Dropout_rate	Train_loss	Validation_loss	Duration_time(s)
0	0.00097	0.00268	0.09222
0.05	0.00127	0.00229	0.09355
0.1	0.0014	0.00224	0.09424
0.15	0.00146	0.00225	0.0904
0.2	0.0016	0.00217	0.0947
0.25	0.00175	0.00217	0.09424
0.3	0.00186	0.00219	0.09033
0.35	0.00196	0.00223	0.09403
0.4	0.00201	0.00223	0.09369
0.45	0.00232	0.00227	0.09407

Augment Training Set by Pre-process with DCT-iDCT-LPF-HPF

Augment training data by DCT transform:

2d-DCT -> HPF (set coefs to 0 in the top left corner of image in dct domain) -> 2d iDCT

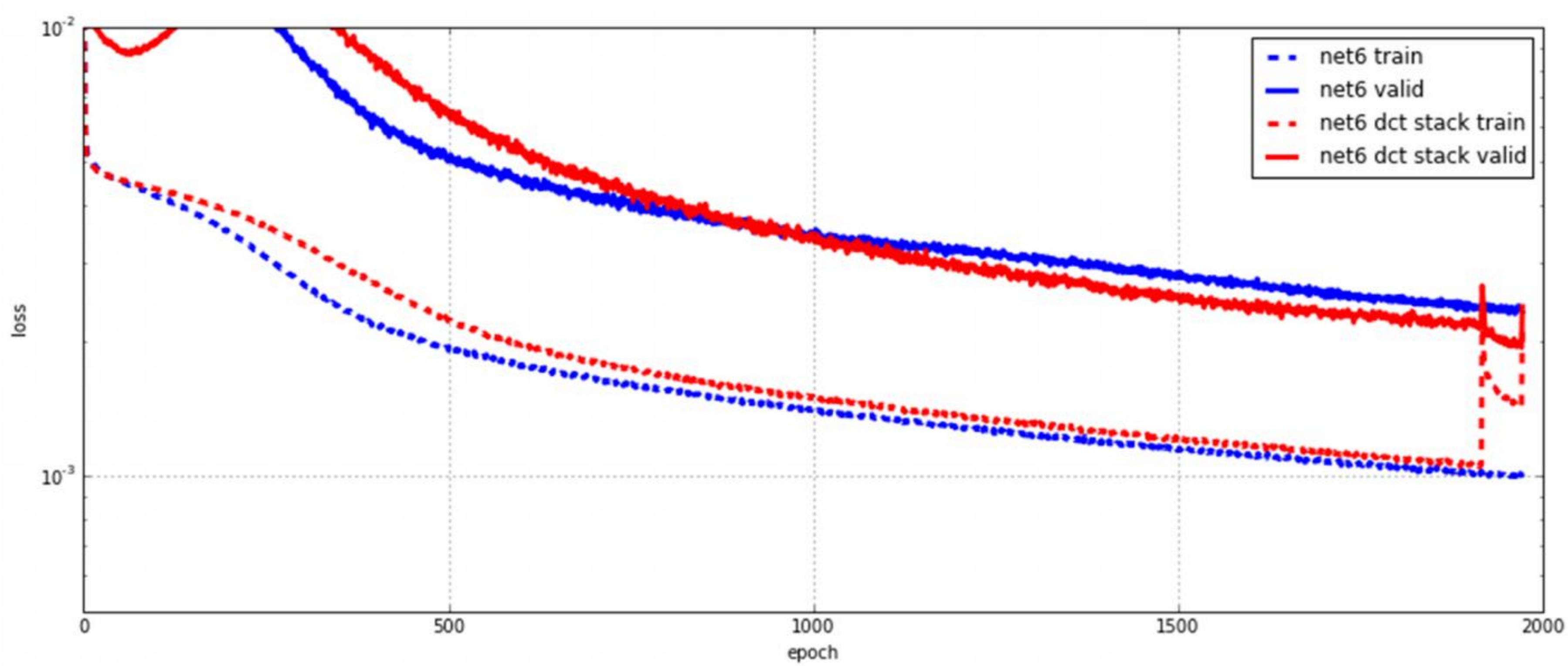
2d-DCT -> LPF (set coefs to 0 in the bottom right corner of image in dct domain) -> 2d iDCT

DCT and iDCT are both normalized.

Also we tried different window size of when doing HPF and LPF.

Result: Augmenting training set with DCT/HPF/LPF actually makes the result (validation error rate) of net6 slightly worse.

Augment Training Set by Pre-process with DCT-iDCT-LPF-HPF Result



Adjust Hidden Layer

Net2 has two FC hidden layer, each with 500 nodes, followed by an output layer (30 nodes)

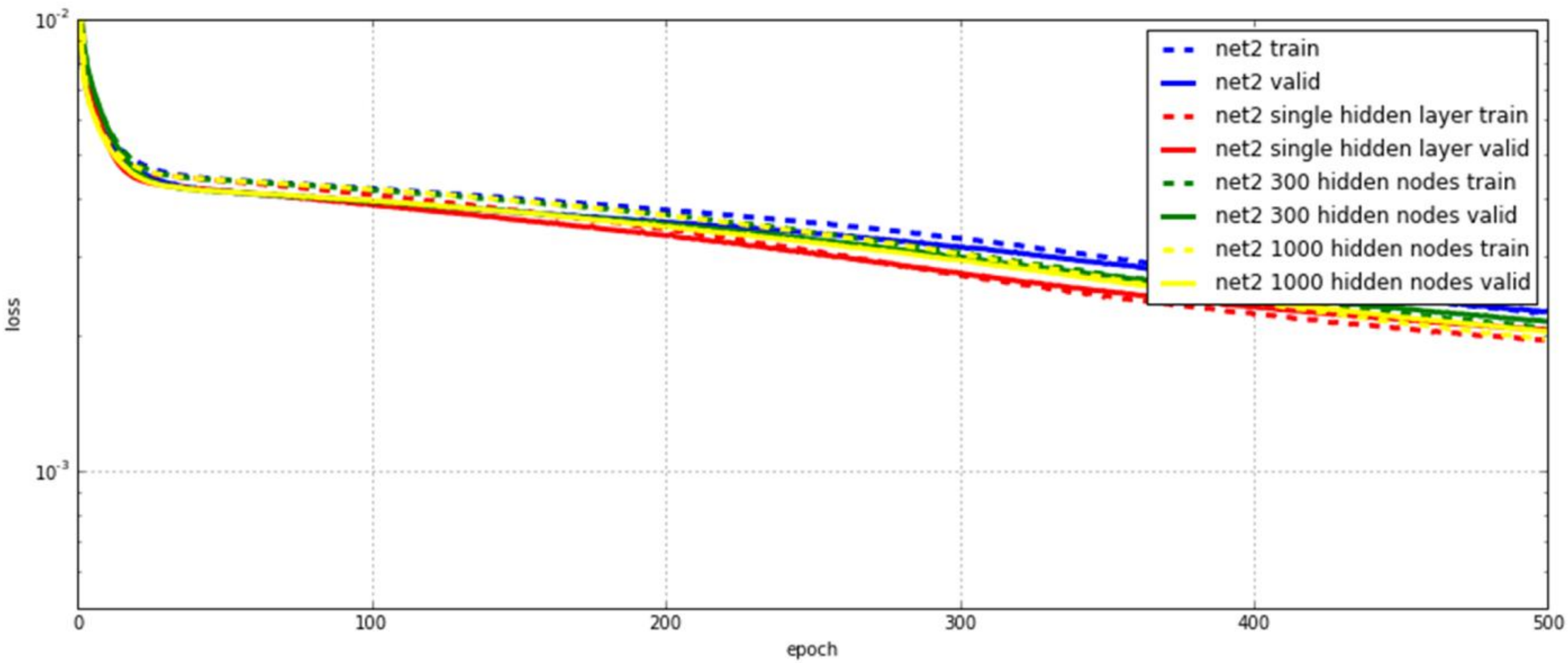
If one of the hidden layers is completely removed, overfitting becomes severe. (**Even though the initial result shows little to no impact.**)

Adjusting number of nodes in the hidden layer (300, 500, 1000) has not noticeable impact.

Conclusion: Since learning is mostly happened in conv layers, but FC layers contains the majority of weights, reducing the number of nodes in hidden layer is desirable (Occam's razor).

Finding the optimal number of nodes in each hidden layer and optimal number of hidden layers require more work. (For example, the Microsoft paper on MNIST data set indicates one hidden layer of 100 nodes is sufficient for the task.)

Adjust Hidden Layer Result



Adjust Non-Linear function

Change frmo RECT to Sigmoid and Tanh.

Make things a little worse.

Stack conv layer - ConvNet_Stack

```
conv_net_stack

  input                       (None, 1, 96, 96)            produces    9216 outputs
  conv1                       (None, 32, 94, 94)           produces  282752 outputs
  conv2                       (None, 64, 90, 90)           produces  518400 outputs
  pool3                       (None, 64, 45, 45)           produces  129600 outputs
  dropout3                    (None, 64, 45, 45)           produces  129600 outputs
  conv5                       (None, 128, 44, 44)          produces  247808 outputs
  pool6                       (None, 128, 22, 22)          produces   61952 outputs
  dropout6                    (None, 128, 22, 22)          produces   61952 outputs
  hidden5                     (None, 500)                  produces     500 outputs
  output                      (None, 30)                   produces      30 outputs
```

Stack conv layer - ConvNet_Stack (continued)

Stack conv layer (with different filter size and depth) without pooling and dropout in between.

Running out of GPU memory easily if stacking more than two layers of conv layer, or if the filter size and depth is big.

Improving result significantly.

Stack conv layer, increase filter size - ConvNet_Stack_Increase_Filter_Size

```
conv_net_stack_increase_filter_size:

  input                      (None, 1, 96, 96)           produces    9216 outputs
  conv1                      (None, 32, 94, 94)          produces  282752 outputs
  conv2                      (None, 64, 90, 90)          produces  518400 outputs
  pool3                      (None, 64, 45, 45)          produces  129600 outputs
  dropout3                   (None, 64, 45, 45)          produces  129600 outputs
  conv5                      (None, 128, 44, 44)         produces  247808 outputs
  pool6                      (None, 128, 22, 22)         produces   61952 outputs
  dropout6                   (None, 128, 22, 22)         produces   61952 outputs
  hidden5                    (None, 500)                 produces     500 outputs
  output                     (None, 30)                  produces      30 outputs
```

ConvNet_Stack_Increase_Filter_Size (continued)

Comparing to ConvNet_Stack, increasing the filter size in the first two conv layers (from 2 * 2 and 3 * 3 to 3 * 3 and 5 * 5) improves the result.

ConvNet_Stack_Increase_Filter_Size (continued)

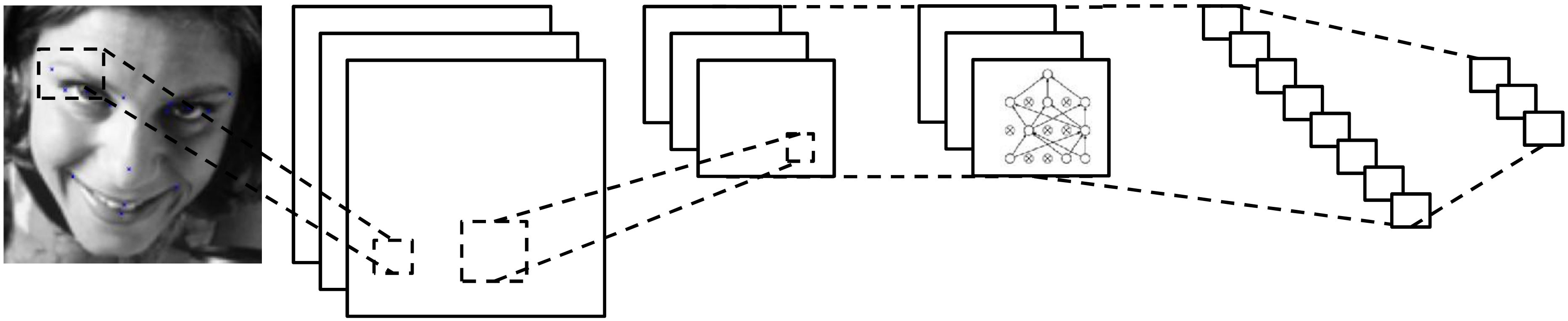
Finding the optimal stack structure and filter size requires more experiment.

More elaborated model demands more GPU p

http://www.tomshardware.com/news/openai-nvidia-dgx-1-ai-supercomputer,32476.html

Nvidia’s DGX-1

Conv Net Experiments



Input:

96 X 96

Feature Maps:

32 X 94 X 94

Feature Maps:

32 X 47 X 47

Feature Maps:

32 X 47 X 47

Fully Connected:

500

Output:

30

Convolution Filter Size
3 X 3

Pooling Filter Size
2 X 2

Dropout
0.1

# of layers

# of layers

Conv Net Experiments

Experiment	Conv Filter Size	Num of Feature Maps	Pooling Filter	Hidden Layer / Dropout	Train Loss	Val Loss
Net6	(3X3), (2X2), (2X2)	32, 64, 128	(2X2), (2X2), (2X2)	500, 500 0.1, 0.2, 0.3, 0.5	0.001902	0.001607
Large pool			(4X4), (6X6), (8X8)		0.00444	0.00428
No pools		6, 12, 24	Removed!		0.00108	0.00136
Large filters, no pools	(5X5), (5X5), (5X5)	6, 12, 24	Removed!		0.00108	0.00109
Large filters, no pools, remove hidden layer	(5X5), (7X7), (9X9)	6, 12, 36	Removed!	500 0.1, 0.2, 0.3	0.00037	0.00105
Replace pools with conv layers	(5X5), (5X5), (5X5)	6, 12, 32	Replaced with Conv Filter (2X2) with stride length = 2		0.00324	0.00273
XLarge filter size, Replace pools with conv layers	(5X5), (7X7), (9X9)	32, 64, 128	Replaced with Conv Filter (2X2) with stride length = 2		0.00226	0.00180
XLarge filters size, medium feature maps, no pool	**(5X5), (7X7), (9X9)**	**6, 12, 36**	**Removed!**		**0.00100**	**0.00107**

Conv Net Experiments

Experiment	Conv Filter Size	Num of Feature Maps	Pooling Filter	Hidden Layer / Dropout	Train Loss	Val Loss
Net6	(3X3), (2X2), (2X2)	32, 64, 128	(2X2), (2X2), (2X2)	500, 500 0.1, 0.2, 0.3, 0.5	0.001902	0.001607
Large pool			(4X4), (6X6), (8X8)		0.00444	0.00428
No pools		6, 12, 24	Removed!		0.00108	0.00136
Large filters, no pools	(5X5), (5X5), (5X5)	6, 12, 24	Removed!		0.00108	0.00109
Large filters, no pools, remove hidden layer	(5X5), (7X7), (9X9)	6, 12, 36	Removed!	500 0.1, 0.2, 0.3	0.00037	0.00105
Replace pools with conv layers	(5X5), (5X5), (5X5)	6, 12, 32	Replaced with Conv Filter (2X2) with stride length = 2		0.00324	0.00273
XLarge filter size, Replace pools with conv layers	(5X5), (7X7), (9X9)	32, 64, 128	Replaced with Conv Filter (2X2) with stride length = 2		0.00226	0.00180
XLarge filters size, medium feature maps, no pool	**(5X5), (7X7), (9X9)**	**6, 12, 36**	**Removed!**		**0.00100**	**0.00107**

Net6LP - Large Pools

Hypothesis: Typical pooling sizes are 2x2 or no max-pooling except that very large input images may need 4x4 pooling in the lower-layers. The increasing size of pooling will reduce the dimension of the signal, which may result in throwing away too much information.

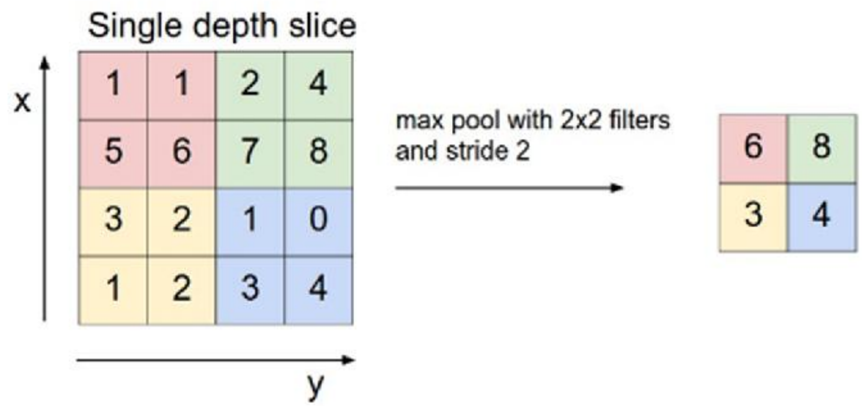
```
input_shape=(None, 1, 96, 96),
conv1_num_filters=32, conv1_filter_size=(3, 3), pool1_pool_size=(4, 4),
dropout1_p=0.1, # !
conv2_num_filters=64, conv2_filter_size=(2, 2), pool2_pool_size=(6, 6),
dropout2_p=0.2, # !
conv3_num_filters=128, conv3_filter_size=(2, 2), pool3_pool_size=(8, 8),
dropout3_p=0.3, # !
hidden4_num_units=500,
dropout4_p=0.5, # !
hidden5_num_units=500,
output_num_units=30,
output_nonlinearity=None,
```

Net6LP - Large Pools - Results

Results!

```
# Neural Network with 371,502 learnable parameters
  #  name      size
---  --------  --------
  0  input     1x96x96
  1  conv1     32x94x94
  2  pool1     32x23x23
  3  dropout1  32x23x23
  4  conv2     64x22x22
  5  pool2     64x5x5
  6  dropout2  64x5x5
  7  conv3     128x4x4
  8  pool3     128x1x1
  9  dropout3  128x1x1
 10  hidden4   500
 11  dropout4  500
 12  hidden5   500
 13  output    30
```

```
 epoch    trn loss    val loss    trn/val   dur
-------  ----------  ----------  ---------  -----
1      0.10663     0.04631    2.30228  2.44s
250      0.00446     0.00425    1.04823  2.45s
500      0.00445     0.00426    1.04323  2.44s
750      0.00445     0.00428    1.03873  2.45s
1000      0.00444     0.00428    1.03701  2.45s
```



Hypothesis: Typical pooling sizes are 2x2 or no max-pooling, except very large input images may need 4x4 pooling in the lower-layers. The increasing size of pooling will reduce the dimension of the signal, which may result in throwing away too much information.

The poorer performance of Net6LP with larger pooling size is consistent with our initial hypothesis.

Can we remove pooling?

Accepted as a workshop contribution at ICLR 2015

STRIVING FOR SIMPLICITY:
THE ALL CONVOLUTIONAL NET

Jost Tobias Springenberg*, Alexey Dosovitskiy*, Thomas Brox, Martin Riedmiller
Department of Computer Science
University of Freiburg
Freiburg, 79110, Germany
{springj, dosovits, brox, riedmiller}@cs.uni-freiburg.de

To conclude, we highlight a few key observations that we made in our experiments:

With modern methods of training convolutional neural networks very simple architectures may perform very well: a network using nothing but convolutions and subsampling matches or even slightly outperforms the state of the art on CIFAR-10 and CIFAR-100. A similar architecture shows competitive results on ImageNet.

In particular, as opposed to previous observations, including explicit (max-)pooling operations in a network does not always improve performance of CNNs. This seems to be especially the case if the network is large enough for the dataset it is being trained on and can learn all necessary invariances just with convolutional layers.

We propose a new method of visualizing the representations learned by higher layers of a convolutional network. While being very simple, it produces sharper visualizations of descriptive image regions than the previously known methods, and can be used even in the absence of 'switches' – positions of maxima in max-pooling regions.

http://arxiv.org/pdf/1412.6806v3.pdf

Conv Net Experiments

Experiment	Conv Filter Size	Num of Feature Maps	Pooling Filter	Hidden Layer / Dropout	Train Loss	Val Loss
Net6	(3X3), (2X2), (2X2)	32, 64, 128	(2X2), (2X2), (2X2)	500, 500 0.1, 0.2, 0.3, 0.5	0.001902	0.001607
Large pool			(4X4), (6X6), (8X8)		0.00444	0.00428
No pools		6, 12, 24	Removed!		0.00108	0.00136
Large filters, no pools	(5X5), (5X5), (5X5)	6, 12, 24	Removed!		0.00108	0.00109
Large filters, no pools, remove hidden layer	(5X5), (7X7), (9X9)	6, 12, 36	Removed!	500 0.1, 0.2, 0.3	0.00037	0.00105
Replace pools with conv layers	(5X5), (5X5), (5X5)	6, 12, 32	Replaced with Conv Filter (2X2) with stride length = 2		0.00324	0.00273
XLarge filter size, Replace pools with conv layers	(5X5), (7X7), (9X9)	32, 64, 128	Replaced with Conv Filter (2X2) with stride length = 2		0.00226	0.00180
XLarge filters size, medium feature maps, no pool	**(5X5), (7X7), (9X9)**	**6, 12, 36**	**Removed!**		**0.00100**	**0.00107**

Net6NPLF

C) No pooling, decrease number of filters, try 6, 12, 24 Feature Maps

```
conv1_num_filters=6, conv1_filter_size=(5, 5), # pool1_pool_size=(2, 2),
dropout1_p=0.1, # !
conv2_num_filters=12, conv2_filter_size=(5, 5), # pool1_pool_size=(2, 2),
dropout2_p=0.2, # !
conv3_num_filters=24, conv3_filter_size=(5, 5), # pool1_pool_size=(2, 2),
dropout3_p=0.3, # !
hidden4_num_units=500,
dropout4_p=0.5, # !
hidden5_num_units=500,
output_num_units=30,
```

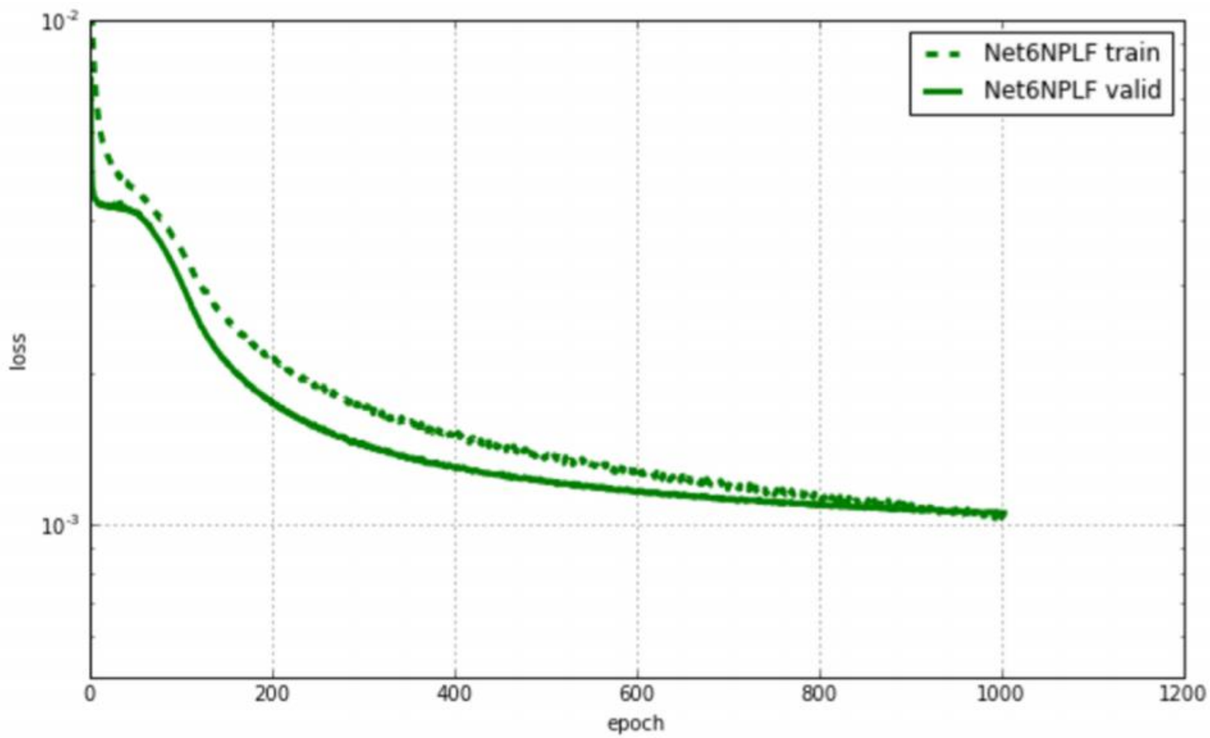
Net6NPLF Results

Results!

```
# Neural Network with 84,947,222 learnable parameters
## Layer information
  #  name      size
---  --------  --------
  0  input     1x96x96
  1  conv1     6x92x92
  2  dropout1  6x92x92
  3  conv2     12x88x88
  4  dropout2  12x88x88
  5  conv3     24x84x84
  6  dropout3  24x84x84
  7  hidden4   500
  8  dropout4  500
  9  hidden5   500
 10  output    30
  epoch    trn loss    val loss    trn/val   dur
-------  ----------  ----------  ---------  -----
1      0.05670     0.00718    7.89818  8.45s
250      0.00188     0.00157    1.19734  8.50s
500      0.00135     0.00122    1.10687  8.49s
750      0.00115     0.00111    1.03713  8.50s
1000     0.00108     0.00109    0.99082  8.50s
```

Net6NPLF Results

Results!

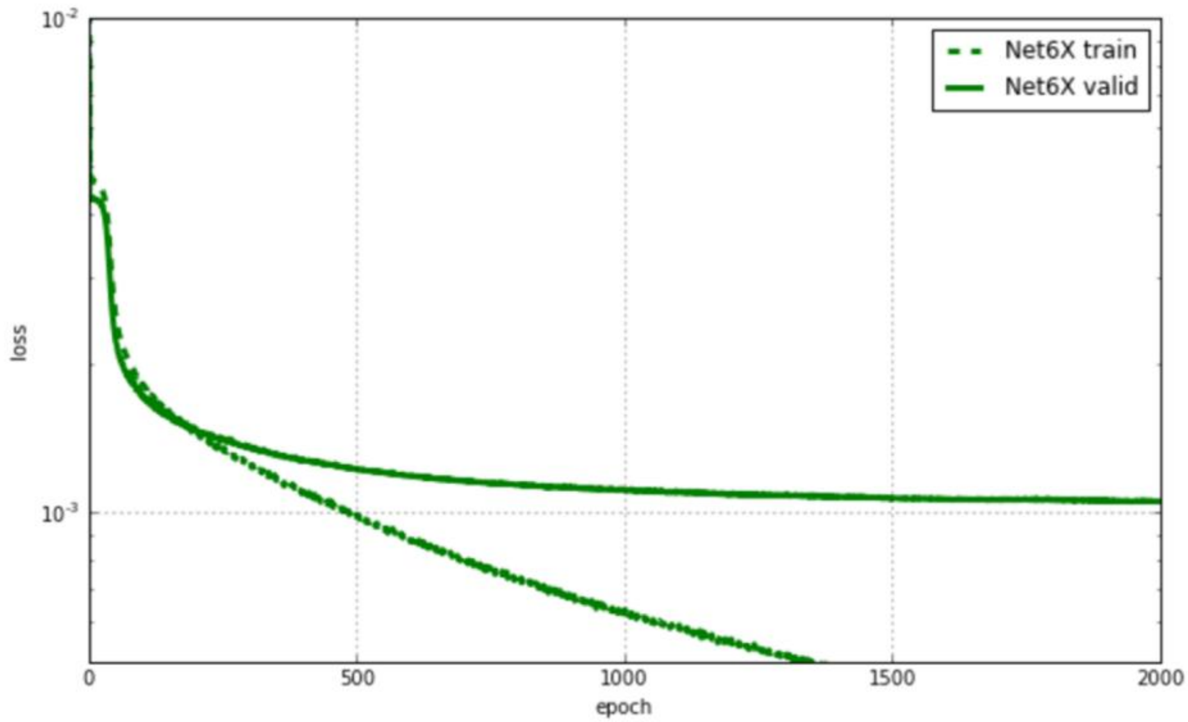


Conv Net Experiments

Experiment	Conv Filter Size	Num of Feature Maps	Pooling Filter	Hidden Layer / Dropout	Train Loss	Val Loss
Net6	(3X3), (2X2), (2X2)	32, 64, 128	(2X2), (2X2), (2X2)	500, 500 0.1, 0.2, 0.3, 0.5	0.001902	0.001607
Large pool			(4X4), (6X6), (8X8)		0.00444	0.00428
No pools		6, 12, 24	Removed!		0.00108	0.00136
Large filters, no pools	(5X5), (5X5), (5X5)	6, 12, 24	Removed!		0.00108	0.00109
Large filters, no pools, remove hidden layer	(5X5), (7X7), (9X9)	6, 12, 36	Removed!	500 0.1, 0.2, 0.3	0.00037	0.00105
Replace pools with conv layers	(5X5), (5X5), (5X5)	6, 12, 32	Replaced with Conv Filter (2X2) with stride length = 2		0.00324	0.00273
XLarge filter size, Replace pools with conv layers	(5X5), (7X7), (9X9)	32, 64, 128	Replaced with Conv Filter (2X2) with stride length = 2		0.00226	0.00180
XLarge filters size, medium feature maps, no pool	**(5X5), (7X7), (9X9)**	**6, 12, 36**	**Removed!**		**0.00100**	**0.00107**

Net6RHL Results

Results!



Overfitting

Conv Net Experiments

Experiment	Conv Filter Size	Num of Feature Maps	Pooling Filter	Hidden Layer / Dropout	Train Loss	Val Loss
Net6	(3X3), (2X2), (2X2)	32, 64, 128	(2X2), (2X2), (2X2)	500, 500 0.1, 0.2, 0.3, 0.5	0.001902	0.001607
Large pool			(4X4), (6X6), (8X8)		0.00444	0.00428
No pools		6, 12, 24	Removed!		0.00108	0.00136
Large filters, no pools	(5X5), (5X5), (5X5)	6, 12, 24	Removed!		0.00108	0.00109
Large filters, no pools, remove hidden layer	(5X5), (7X7), (9X9)	6, 12, 36	Removed!	500 0.1, 0.2, 0.3	0.00037	0.00105
Replace pools with conv layers	(5X5), (5X5), (5X5)	6, 12, 32	Replaced with Conv Filter (2X2) with stride length = 2		0.00324	0.00273
XLarge filter size, Replace pools with conv layers	(5X5), (7X7), (9X9)	32, 64, 128	Replaced with Conv Filter (2X2) with stride length = 2		0.00226	0.00180
XLarge filters size, medium feature maps, no pool	**(5X5), (7X7), (9X9)**	**6, 12, 36**	**Removed!**		**0.00100**	**0.00107**

Net6RPXL

Replace pooling with conv layers of stride length 2, with large number of filters

```
conv1_num_filters=32, conv1_filter_size=(5, 5), #pool1_pool_size=(4, 4),
conv_rpool1_num_filters=32, conv_rpool1_filter_size=(2, 2), conv_rpool1_stride=2, # !
dropout1_p=0.1,  # !

conv2_num_filters=64, conv2_filter_size=(7, 7), #pool2_pool_size=(4, 4),
conv_rpool2_num_filters=64, conv_rpool2_filter_size=(2, 2), conv_rpool2_stride=2, # !
dropout2_p=0.2,  # !

conv3_num_filters=128, conv3_filter_size=(9, 9), #pool3_pool_size=(4, 4),
conv_rpool3_num_filters=128, conv_rpool3_filter_size=(2, 2), conv_rpool3_stride=2,# !
dropout3_p=0.3,  # !

hidden4_num_units=500,
dropout4_p=0.5,  # !
hidden5_num_units=500,
output_num_units=30, output_nonlinearity=None,
```

Net6RPXL

Results!

```
# Neural Network with 3,421,198 learnable parameters

## Layer information

  #  name         size
---  -----------  ---------
  0  input        1x96x96
  1  conv1        32x92x92
  2  conv_rpool1  32x46x46
  3  dropout1     32x46x46
  4  conv2        64x40x40
  5  conv_rpool2  64x20x20
  6  dropout2     64x20x20
  7  conv3        128x12x12
  8  conv_rpool3  128x6x6
  9  dropout3     128x6x6
 10  hidden4      500
 11  dropout4     500
 12  hidden5      500
 13  output       30

  epoch    trn loss    val loss    trn/val   dur
-------  ----------  ----------  ---------  ------
1     0.09322     0.01231    7.57231  14.28s
250     0.00428     0.00400    1.06990  14.27s
500     0.00331     0.00279    1.18513  14.31s
750     0.00243     0.00193    1.25843  14.35s
1000     0.00226     0.00180    1.25192  14.29s
```

Conv Net Experiments

Experiment	Conv Filter Size	Num of Feature Maps	Pooling Filter	Hidden Layer / Dropout	Train Loss	Val Loss
Net6	(3X3), (2X2), (2X2)	32, 64, 128	(2X2), (2X2), (2X2)	500, 500 0.1, 0.2, 0.3, 0.5	0.001902	0.001607
Large pool			(4X4), (6X6), (8X8)		0.00444	0.00428
No pools		6, 12, 24	Removed!		0.00108	0.00136
Large filters, no pools	(5X5), (5X5), (5X5)	6, 12, 24	Removed!		0.00108	0.00109
Large filters, no pools, remove hidden layer	(5X5), (7X7), (9X9)	6, 12, 36	Removed!	500 0.1, 0.2, 0.3	0.00037	0.00105
Replace pools with conv layers	(5X5), (5X5), (5X5)	6, 12, 32	Replaced with Conv Filter (2X2) with stride length = 2		0.00324	0.00273
XLarge filter size, Replace pools with conv layers	(5X5), (7X7), (9X9)	32, 64, 128	Replaced with Conv Filter (2X2) with stride length = 2		0.00226	0.00180
XLarge filters size, medium feature maps, no pool	**(5X5), (7X7), (9X9)**	**6, 12, 36**	**Removed!**		**0.00100**	**0.00107**

CNet_NPLFXL

XLarge filters size, medium feature maps, no pool

```
conv1_num_filters=6, conv1_filter_size=(5, 5), # pool1_pool_size=(2, 2),
dropout1_p=0.1, # !
conv2_num_filters=12, conv2_filter_size=(7, 7), # pool1_pool_size=(2, 2),
dropout2_p=0.2, # !
conv3_num_filters=36, conv3_filter_size=(9, 9), # pool1_pool_size=(2, 2),
dropout3_p=0.3, # !
hidden4_num_units=500,
dropout4_p=0.5, # !
hidden5_num_units=500,
output_num_units=30,
```

CNet_NPLFXL

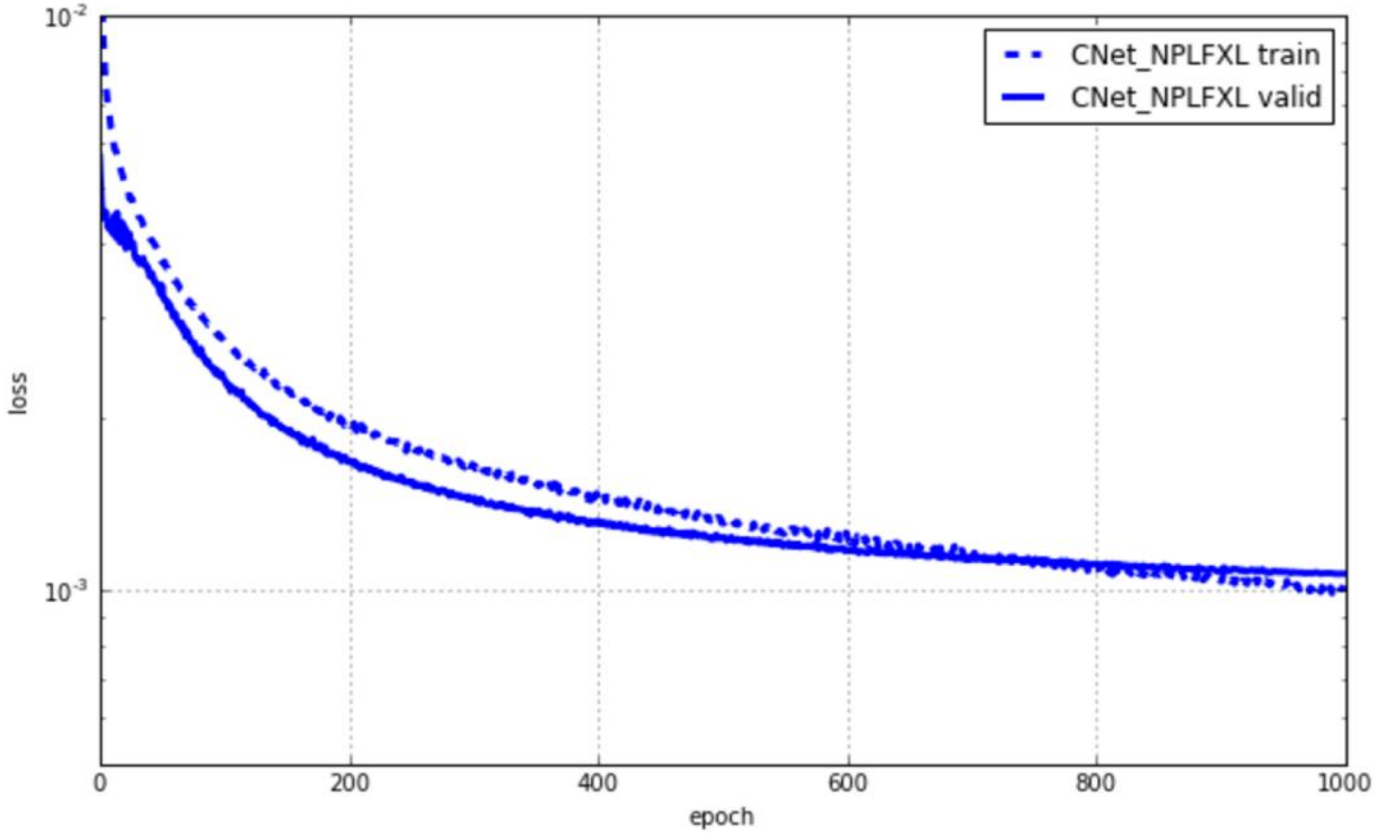
Results!

```
# Neural Network with 109,816,754 learnable parameters
## Layer information
  #  name      size
---  --------  --------
  0  input     1x96x96
  1  conv1     6x92x92
  2  dropout1  6x92x92
  3  conv2     12x86x86
  4  dropout2  12x86x86
  5  conv3     36x78x78
  6  dropout3  36x78x78
  7  hidden4   500
  8  dropout4  500
  9  hidden5   500
 10  output    30


 epoch    trn loss    val loss    trn/val   dur
-------  ----------  ----------  ---------  ------
  1      0.06979     0.00568   12.28723  18.88s
250      0.00178     0.00154    1.15592  18.90s
500      0.00131     0.00124    1.06097  18.94s
750      0.00111     0.00112    0.98713  18.95s
1000      0.00100     0.00107    0.93156  18.93s
```

CNet_NPLFXL

Results!



Conv Net Experiments

Experiment	Key Learning
Net6	
Large pool	Throwing away too much information
No pools	Good results, but accuracy can be improved
Large filters, no pools	**85M parameters, took longer to train, but great accuracy and generalization**
Large filters, no pools, remove hidden layer & dropout	No hidden layer with dropout caused dramatic overfitting
Replace pools with conv layers	Using a stride length of 2 also resulted in huge loss of information
XLarge filter size, Replace pools with conv layers	Better, but still didn't make up for loss of information
XLarge filters size, medium # of feature maps, no pool	**100M parameters, long training time, best results**

7. Conclusion

Key Learnings

Pooling layers are lossy and can be delayed/removed from early layers

Number of parameters increase the complexity of the network as well as the computation time pretty quickly

Handcrafted features or simple pre-processing do little improvement over an additional convolutional layer

Dropout was critical in preventing overfitting

Current score (without Kaggle test-set) based on MSE loss value of 0.00107 would be an RMSE of 1.57012

Further improvements

More:

- Sophisticated Data Cleaning
- Sophisticated pre-processing
- Careful optimisation for computation costs such as explore conversion of fully-connected layers to convolutional

Explore use of trained networks:

- to label more training data
- to predict bad training data for manual review

Explore use of visualization techniques that reveal input stimuli exciting

Thanks!

U.C. Berkeley
School of Information
MIDS w207 - Summer 2016
Under the Instruction of Professor Todd Holloway

See: https://github.com/yhzhao/W207KaggleFaceFeature

alex

minghu

Siddharth Singhal
siddharth.singhal@berkeley.edu

yuhang

Byron Tang
tangbyron@berkeley.edu

References

https://www.kaggle.com/c/facial-keypoints-detection/details/deep-learning-tutorial

http://markus.com/install-theano-on-aws/

http://danielnouri.org/notes/2014/12/17/using-convolutional-neural-nets-to-detect-facial-keypoints-tutorial/

http://cs231n.github.io/convolutional-networks/

http://deeplearning.stanford.edu/tutorial/

Appendix

Experiments - Net6NP

No pooling, decrease number of filters to account for large number of parameters

```
conv1_num_filters=6, conv1_filter_size=(3, 3), # pool1_pool_size=(2, 2),
dropout1_p=0.1, # !
conv2_num_filters=12, conv2_filter_size=(2, 2), # pool2_pool_size=(2, 2),
dropout2_p=0.2, # !
conv3_num_filters=24, conv3_filter_size=(2, 2), # pool3_pool_size=(2, 2),
dropout3_p=0.3, # !
hidden4_num_units=500,
dropout4_p=0.5, # !
hidden5_num_units=500,
output_num_units=30,
```

Experiments - Net6NP - Results

Results!

```
# Neural Network with 101,835,566 learnable parameters
#  name      size
---  --------  --------
  0  input     1x96x96
  1  conv1     6x94x94
  2  dropout1  6x94x94
  3  conv2     12x93x93
  4  dropout2  12x93x93
  5  conv3     24x92x92
  6  dropout3  24x92x92
  7  hidden4   500
  8  dropout4  500
  9  hidden5   500
 10  output    30

  epoch    trn loss    val loss    trn/val   dur
-------  ----------  ----------  ---------  -----
1      0.05052     0.01757    2.87644  5.14s
250      0.00221     0.00197    1.11873  5.14s
500      0.00152     0.00156    0.97522  5.14s
750      0.00122     0.00143    0.85226  5.14s
1000     0.00108     0.00136    0.78980  5.14s
```

Architecture of A Single Hidden Layer NN

```
input_shape=(None, 9216), # 96x96 input pixels per batch
hidden_num_units=100, # number of units in hidden layer
output_nonlinearity=None, # output layer uses identity function
output_num_units=30, # 30 target values
```

```
input                  (None, 9216)           produces    9216 outputs
  hidden               (None, 100)            produces     100 outputs
  output               (None, 30)             produces      30 outputs
  epoch    train loss    valid loss    train/val  dur
-------  ------------  ------------  -----------  -----
750       0.00156       0.00280      0.55823  0.07s
```

Experiments - Net6RHL

No pooling, large filters, increase number of filters, remove one hidden layer

```
conv1_num_filters=6, conv1_filter_size=(5, 5), # pool1_pool_size=(2, 2),
dropout1_p=0.1, # !
conv2_num_filters=12, conv2_filter_size=(7, 7), # pool2_pool_size=(2, 2),
dropout2_p=0.2, # !
conv3_num_filters=36, conv3_filter_size=(9, 9), # pool3_pool_size=(2, 2),
dropout3_p=0.3, # !
# hidden4_num_units=500,
# dropout4_p=0.5, # !
hidden5_num_units=500,
output_num_units=30,
```

Experiments - Net6RHL Results

Results!

```
# Neural Network with 109,566,254 learnable parameters
## Layer information
  #  name      size
---  --------  --------
  0  input     1x96x96
  1  conv1     6x92x92
  2  dropout1  6x92x92
  3  conv2     12x86x86
  4  dropout2  12x86x86
  5  conv3     36x78x78
  6  dropout3  36x78x78
  7  hidden5   500
  8  output    30

  epoch    trn loss    val loss    trn/val  dur
-------  ----------  ----------  ---------  ------
1     0.08458     0.00579   14.60905  18.84s
250     0.00148     0.00155    0.96113  18.93s
500     0.00110     0.00128    0.85610  18.92s
750     0.00084     0.00118    0.71464  18.93s
1000     0.00069     0.00112    0.62124  18.93s
1250     0.00053     0.00109    0.48676  18.93s
1500     0.00045     0.00107    0.42472  18.93s
2000     0.00037     0.00105    0.34997  18.95s
```

Experiments - Net6RP

Replace pooling with conv layers of stride length 2

```
conv1_num_filters=6, conv1_filter_size=(5, 5), #pool1_pool_size=(4, 4),
conv_rpool1_num_filters=6, conv_rpool1_filter_size=(2, 2), conv_rpool1_stride=2, # !
dropout1_p=0.1,  # !

conv2_num_filters=12, conv2_filter_size=(5, 5), #pool2_pool_size=(4, 4),
conv_rpool2_num_filters=12, conv_rpool2_filter_size=(2, 2), conv_rpool2_stride=2, # !
dropout2_p=0.2,  # !

conv3_num_filters=32, conv3_filter_size=(5, 5), #pool3_pool_size=(4, 4),
conv_rpool3_num_filters=32, conv_rpool3_filter_size=(2, 2), conv_rpool3_stride=2,# !
dropout3_p=0.3,  # !

hidden4_num_units=500,
dropout4_p=0.5,  # !
hidden5_num_units=500,
output_num_units=30, output_nonlinearity=None,
```

Experiments - Net6RP

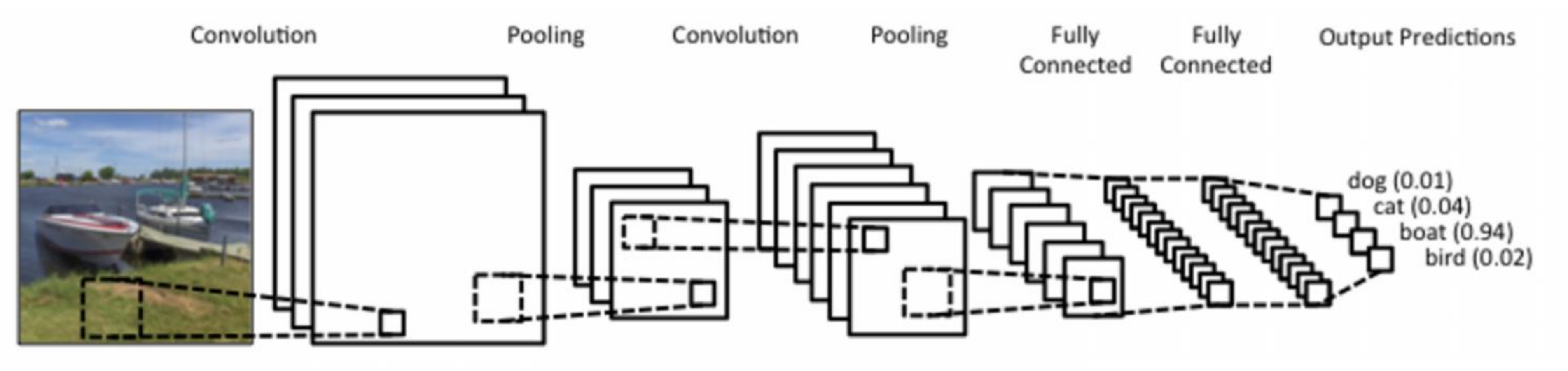
Results!

```
# Neural Network with 1,306,496 learnable parameters
## Layer information
  #  name         size
---  -----------  --------
  0  input        1x96x96
  1  conv1        6x92x92
  2  conv_rpool1  6x46x46
  3  dropout1     6x46x46
  4  conv2        12x42x42
  5  conv_rpool2  12x21x21
  6  dropout2     12x21x21
  7  conv3        32x17x17
  8  conv_rpool3  32x8x8
  9  dropout3     32x8x8
 10  hidden4      500
 11  dropout4     500
 12  hidden5      500
 13  output       30

 epoch    trn loss    val loss    trn/val  dur
-------  ----------  ----------  ---------  -----
1     0.12775     0.07193    1.77603  2.96s
250     0.00446     0.00424    1.05232  2.97s
500     0.00443     0.00420    1.05564  2.96s
750     0.00395     0.00355    1.11268  2.97s
1000     0.00324     0.00273    1.18705  2.97s
1250     0.00284     0.00237    1.19976  2.97s
1500     0.00262     0.00216    1.21448  2.96s
1750     0.00243     0.00201    1.21067  2.97s
2000     0.00233     0.00190    1.22529  2.95s
```

Net6 Architecture

Convolutional layers are not fully connected
Weights are shared between a subset of neurons in the convolutional layer
Pooling is a static subsampling of inputs

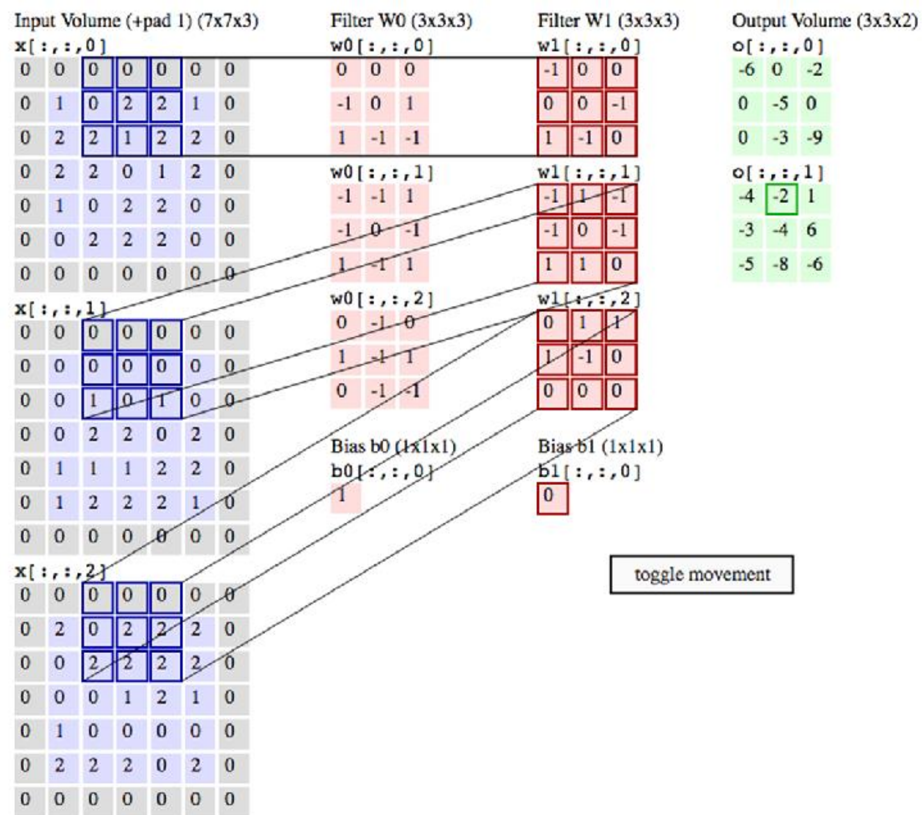


Convolution

Filtering is a slicing window function applied to a matrix.

The data is continous. Because the facial keypoints are so close together - we aren't trying to find a digit.

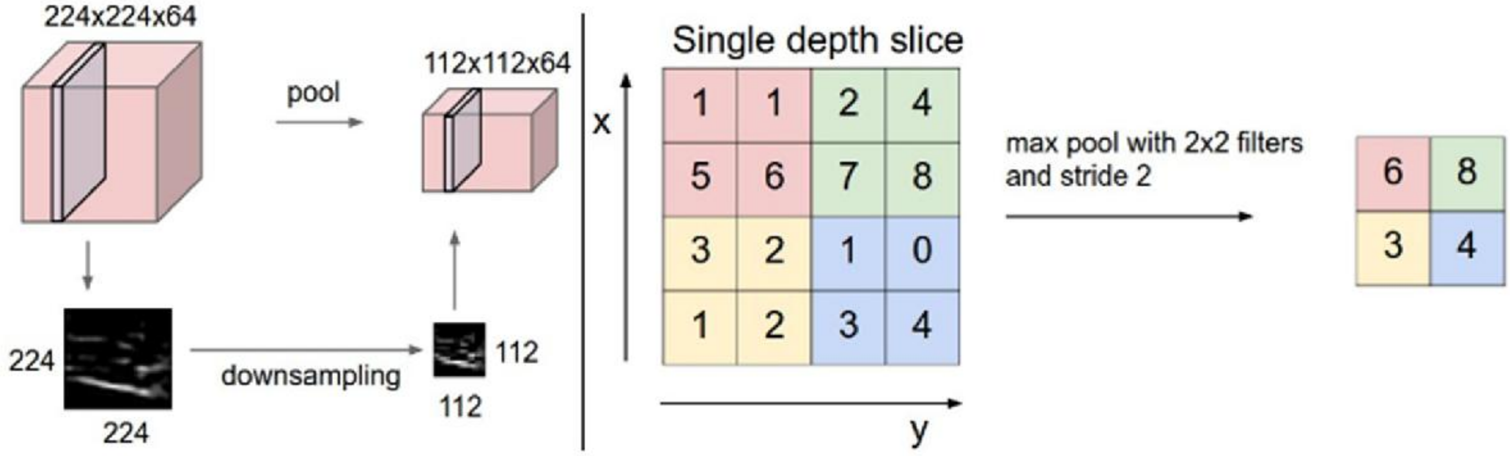
Thus, we keep the stride length below 3



http://deeplearning.stanford.edu/tutorial/ && http://cs231n.github.io/convolutional-networks/

Pooling

Pooling's function is to simplify the information from the convolutional layer. Additionally, it can prevent overfitting. Recent research (Striving for Simplicity: The All Convolutional Net), propose that we remove the pooling layer all together, and go with repeated convolutional layers, with larger strides.



http://neuralnetworksanddeeplearning.com/chap6.htm && http://cs231n.github.io/convolutional-networks/