Convolutional Neural Networks

Deep Learning in Python Part 3

What is convolution?

- Convolution is an important operation from signal processing and analysis
- Think of your favorite audio effect - let's say it's an "echo"
- All effects can be thought of as filters, or what we call them in machine learning sometimes, kernels - we'll call it h(t) or w(t)

$$\begin{tikzcd} x(t) \arrow[r] & \boxed{h(t)} \arrow[r] & y(t) \end{tikzcd}$$

Matrix Multiplication

$$y = Wx$$

$$y_i = \sum_j w_j x_{i-j}$$

Why Convolution?

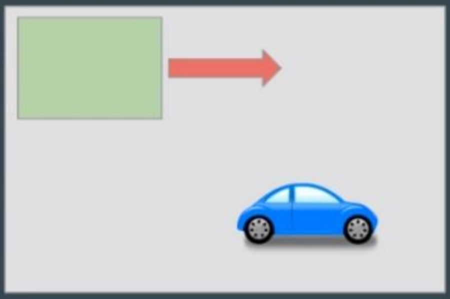
- Images are special types of inputs
- We can see them with our own eyes, so we know what the neural network should be robust to
- Perfect example: translational invariance

Why Convolution?

- We know these are both cars
- But a feedforward neural network has different weights everywhere
- If we only trained it on the left image, it wouldn't recognize the right

Why Convolution?

- What does convolution do?
- Multiplies the same weights everywhere on the image
- Suppose this is a feature finder that identifies the car
- i.e. outputs very +ve # when it sees a car, very -ve # otherwise
- Then it will find a car no matter where it is on the image



Why Convolution?

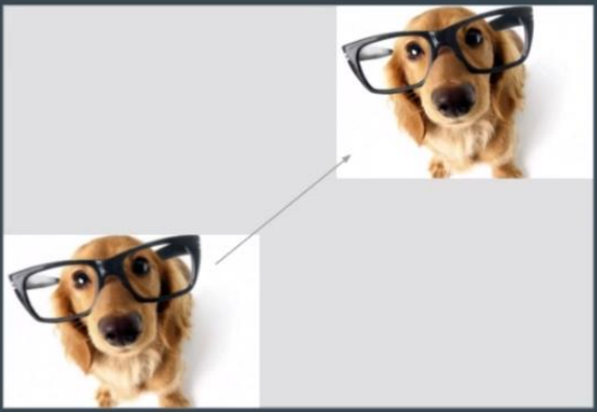
Recall:

Feedforward neural networks contain multiple layers which allow each layer to find successively complex features

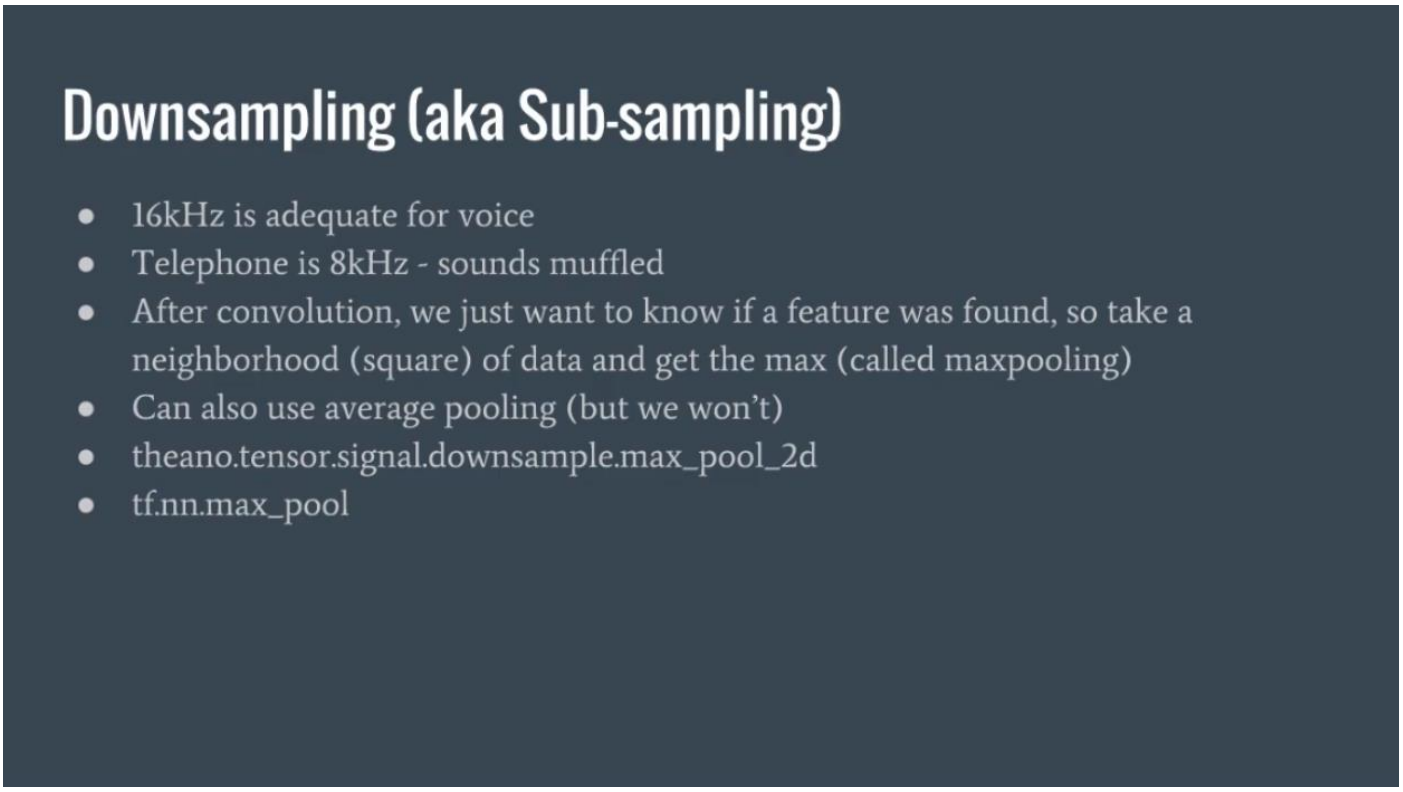
Layers of convolutions behave the same. First layer will find small, simple features anywhere on the image (e.g. a line). Next layer will find more complex features, and so on.

Why do convolution?

- Think of it as a sliding window or sliding filter
- Doesn't matter where the dog is
- It's still a dog
- "Translational invariance"

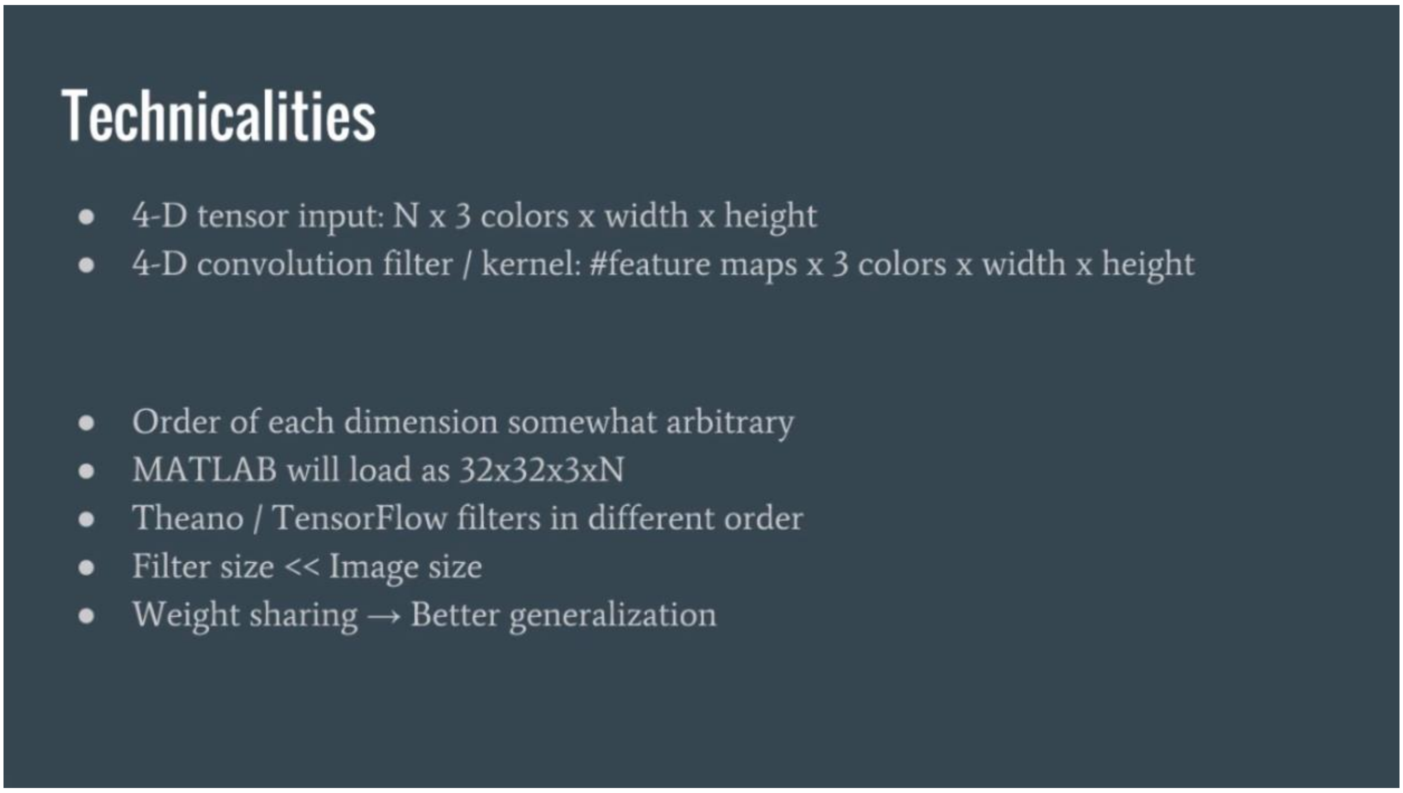


- Question to think about:
- How to solve rotational invariance?



Downsampling (aka Sub-sampling)

- 16kHz is adequate for voice
- Telephone is 8kHz - sounds muffled
- After convolution, we just want to know if a feature was found, so take a neighborhood (square) of data and get the max (called maxpooling)
- Can also use average pooling (but we won't)
- theano.tensor.signal.downsample.max_pool_2d
- tf.nn.max_pool

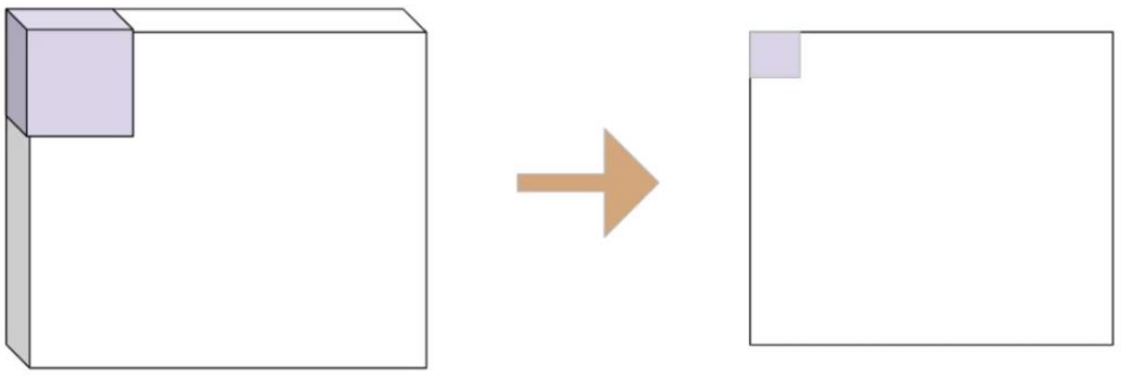


Technicalities

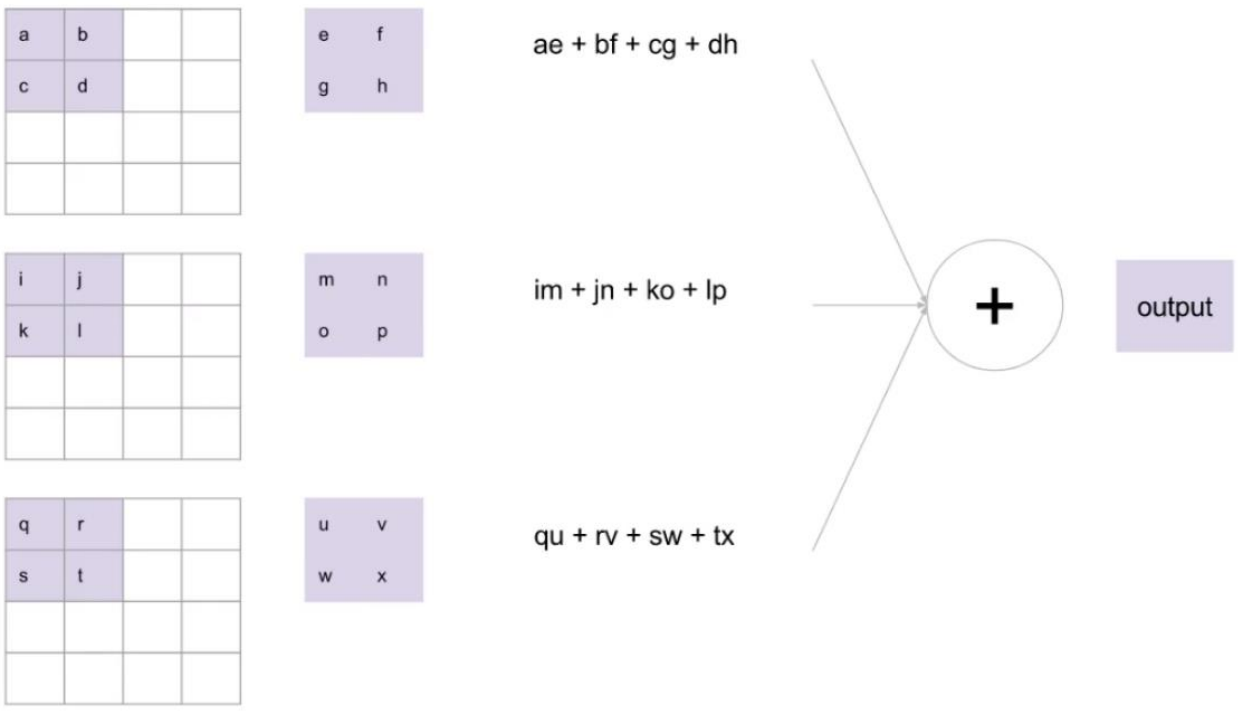
- 4-D tensor input: N x 3 colors x width x height
- 4-D convolution filter / kernel: #feature maps x 3 colors x width x height

- Order of each dimension somewhat arbitrary
- MATLAB will load as 32x32x3xN
- Theano / TensorFlow filters in different order
- Filter size << Image size
- Weight sharing → Better generalization

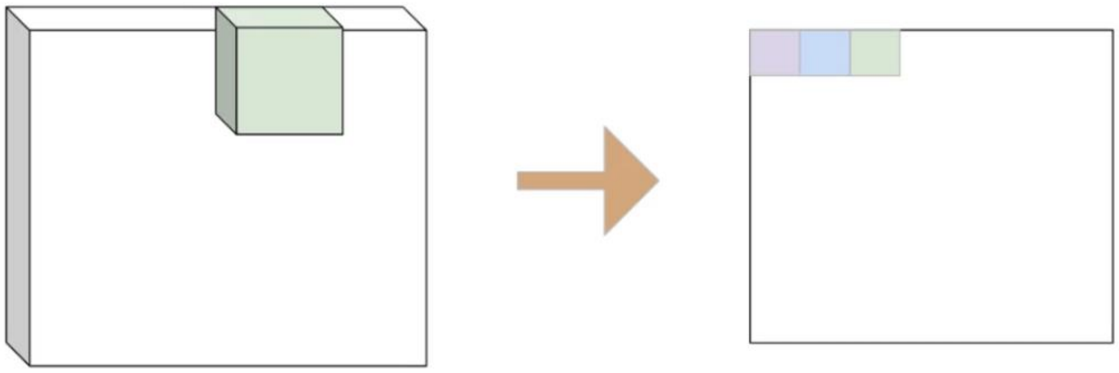
Convolution with 3-D input



$$out[i,j] = \sum_{k_1=1}^{K} \sum_{k_2=1}^{K} \sum_{c=1}^{C} in[i-k_1, j-k_2, c] filter[c, k_1, k_2]$$



Convolution with 3-D input



- Opposite problem now! Input is 3-D, output is 2-D!

Why is the output 2-D?

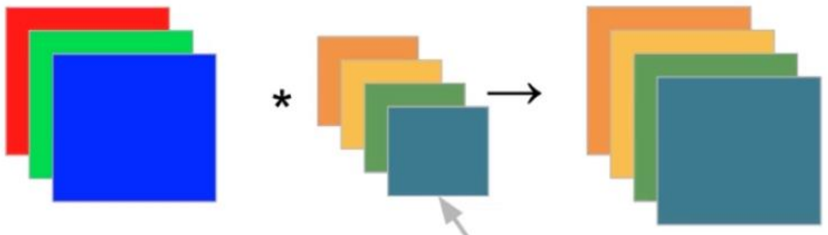
- Filter shape is C x K x K
- Let's apply the same concept as before
- Use multiple color filters of size C x K x K that do different things
- E.g. a "color vertical edge finder" + a "color horizontal edge finder"
- Stacked size: C x K x K x 2

$$out[i,j] = \sum_{k_1=1}^{K} \sum_{k_2=1}^{K} \sum_{c=1}^{C} in[i-k_1, j-k_2, c] \boxed{filter[c, k_1, k_2]}$$

Generalizing the concept

Note: exact order of dimensions differs in different libraries
We can think of it as "C2 different 3-D filters of size C1xKxK"

- Input image: H x W x C1
- Filter: C1 x K x K x C2
- Output image: H x W x C2

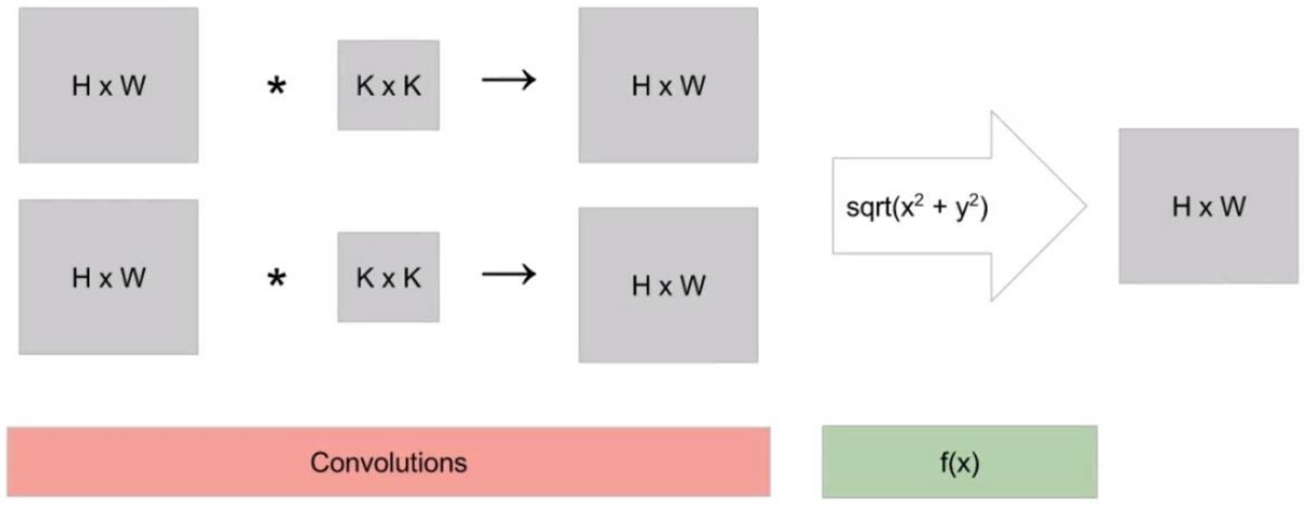


Actually, it's not so easy to draw a 4-D object so we are hiding the C1 dimension. =)

Final notes

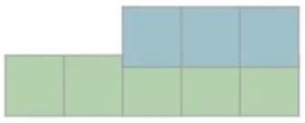
- In this example only, we used "edge detection" with 2 feature maps
- In the real-world, we may have 100s or 1000s of feature maps
- The # of feature maps is a hyperparameter, and must be chosen using hyperparameter optimization techniques
- No guarantee or expectation that the filters a CNN finds will do edge detection
- We optimize the filters using backprop / gradient descent on the cost, a central principle in machine learning
- We don't want to manually decide what features the filters should find, because that is suboptimal compared to what's found automatically

One more final note



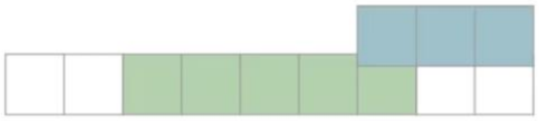
Convolution Modes

- “Valid” vs. “Same” vs. “Full”
- Valid mode: filter never goes outside the input
- If:
  - Input length = N
  - Filter length = K
- Then:
  - Output length = N - K + 1



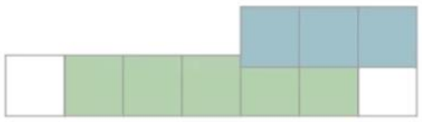
Convolution Modes

- Full mode: filter allowed to go outside the input, far enough so that there is at least still 1 overlapping element
- If:
  - Input length = N
  - Filter length = K
- Then:
  - Output length = N + K - 1



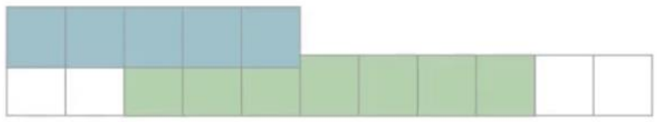
Convolution Modes

- Same mode: padding is set such that input length == output length
- If:
  - Input length = N = 5
  - Filter length = K = 3
- Then:
  - Padding length = P = 1
  - Note: also called "half-padding" in Theano since P = floor(K / 2)



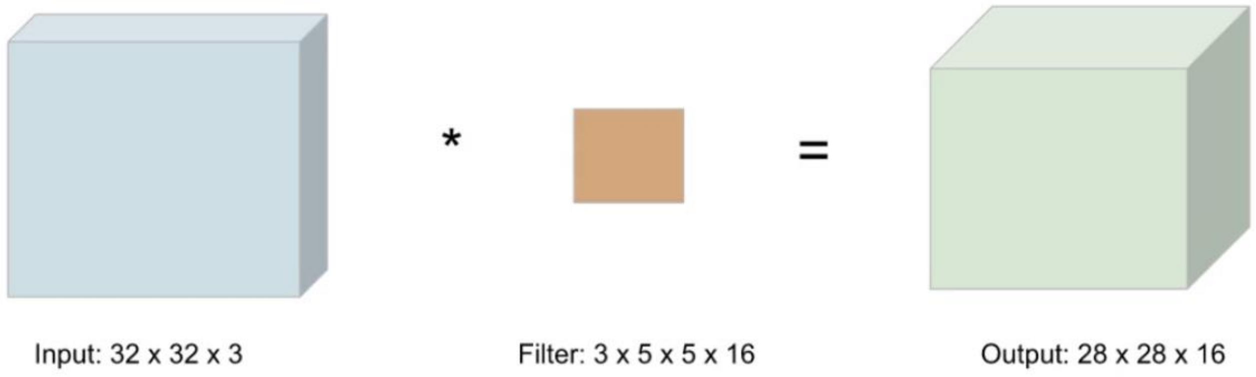
Convolution Modes

- Same mode: padding is set such that input length == output length
- If:
  - Input length = N = 7
  - Filter length = K = 5
- Then:
  - Padding = 2



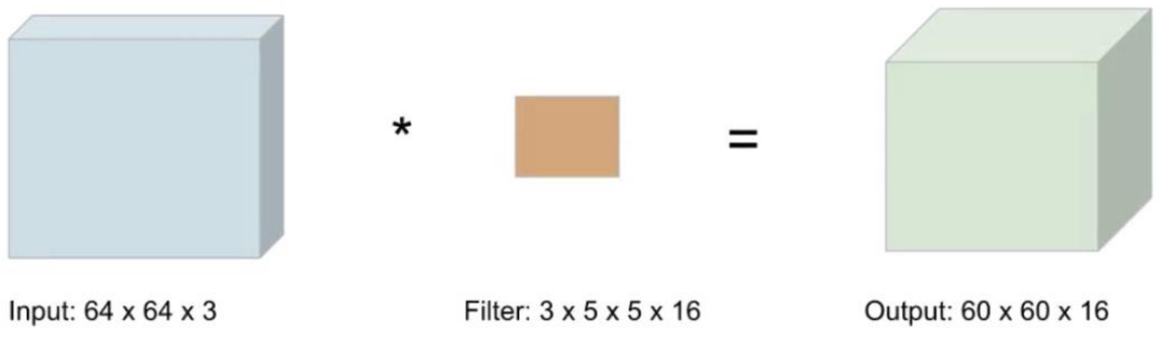
Example of 1 convolution

Assuming "valid mode": 32 - 5 + 1 = 28

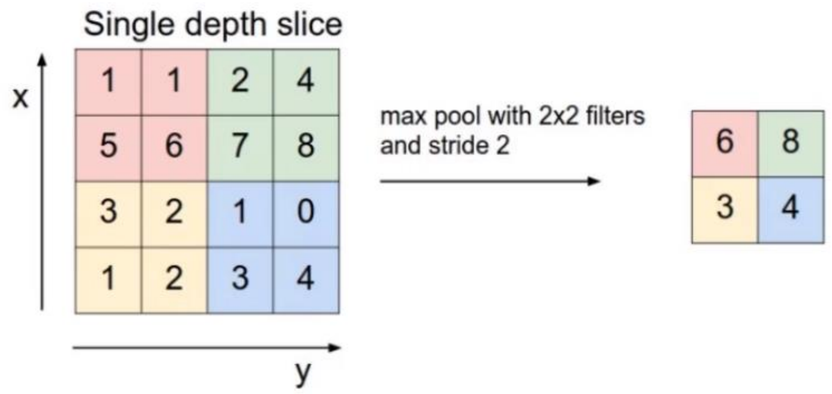


Important note

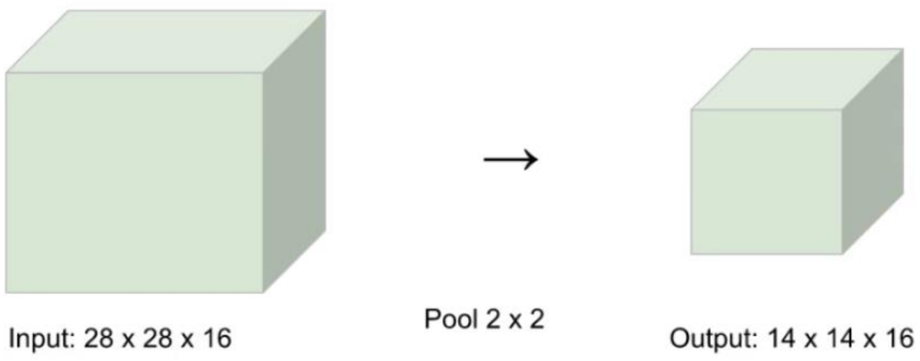
- Convolution always works, no matter the size of the input (the output size is just calculated accordingly)
- Note how this is not the case for a FC / dense layer. If W shape is 784 x 1000, then the input must be a vector of size 784



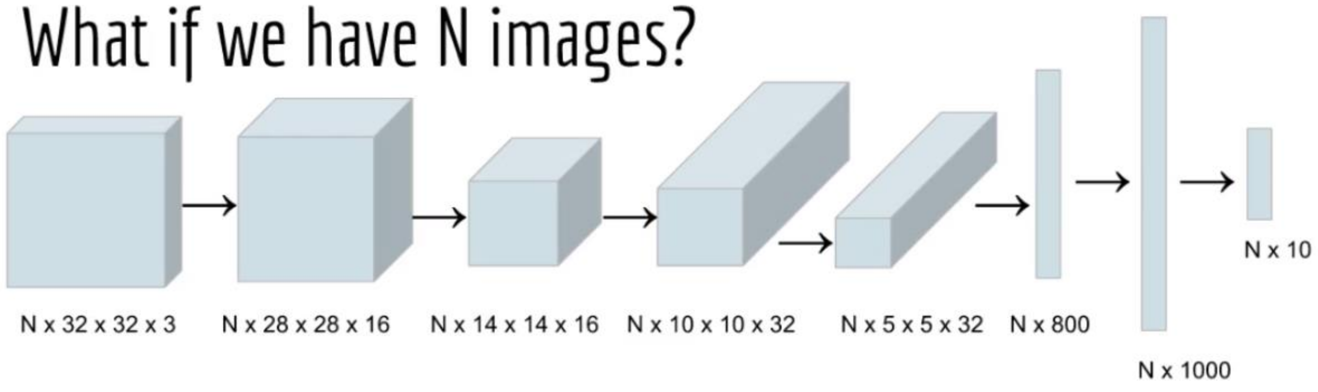
Pooling

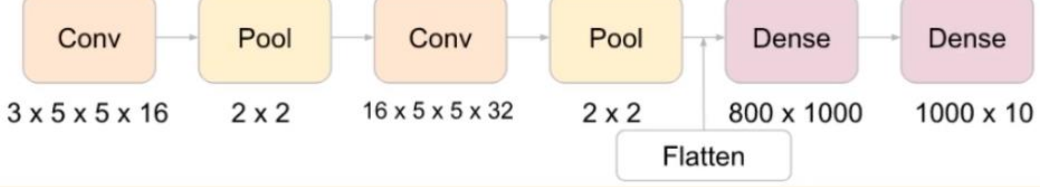


Pooling (our example)



What if we have N images?



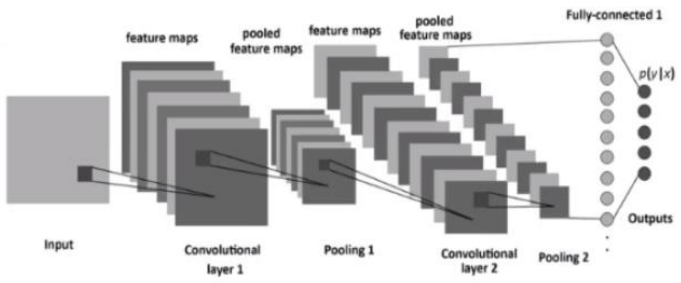


How do we choose filter sizes?

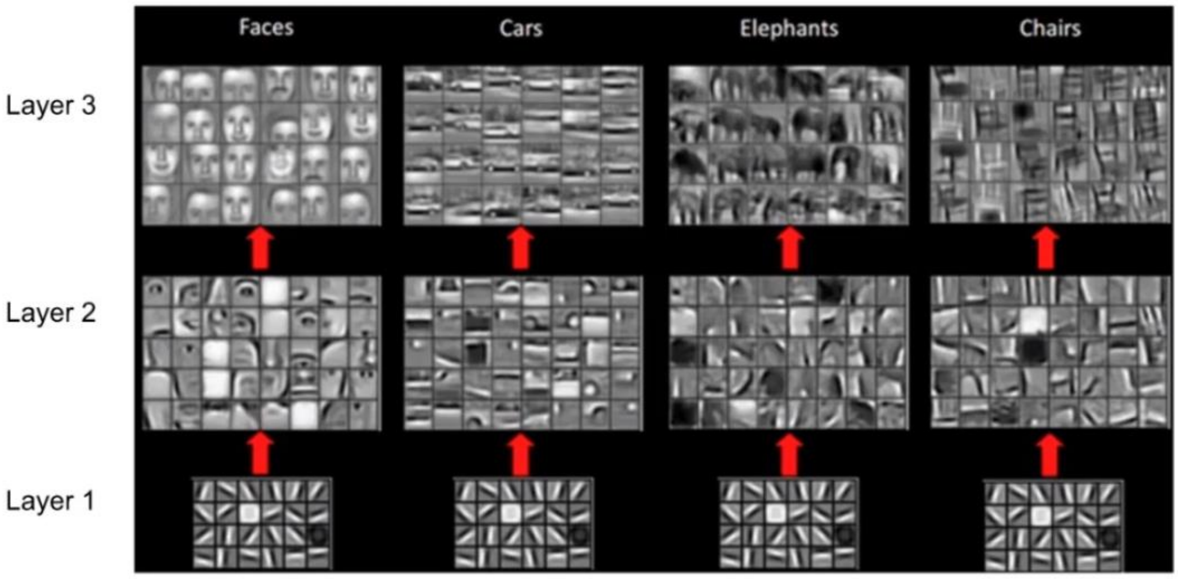
- These are called "hyperparameters", generally must be chosen via experimentation (methods we discussed in prerequisites such as grid search)
- General pattern:

#1 - Image shrinks due to pooling, filters generally stay about the same

#2 - Number of feature maps increases.

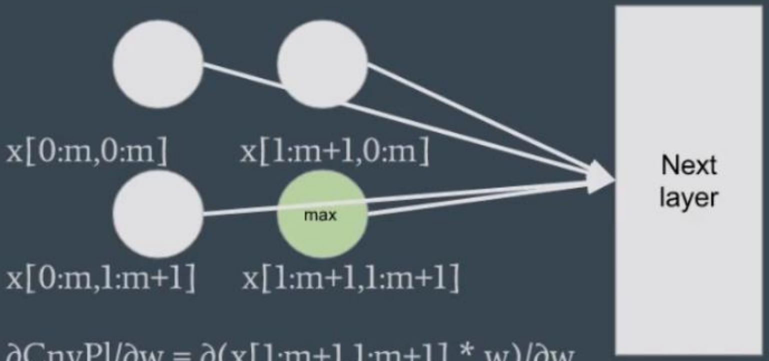


What has a CNN learned?



Max Pooling

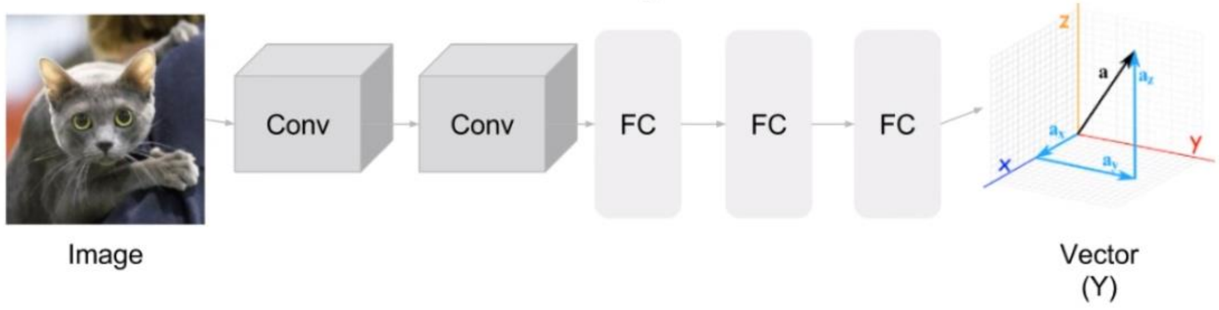
- If it influences an output, it should be updated
- A weight should be updated by the error signal at any nodes it affects



$\partial CnvPl/\partial w = \partial(x[1:m+1,1:m+1] * w)/\partial w$

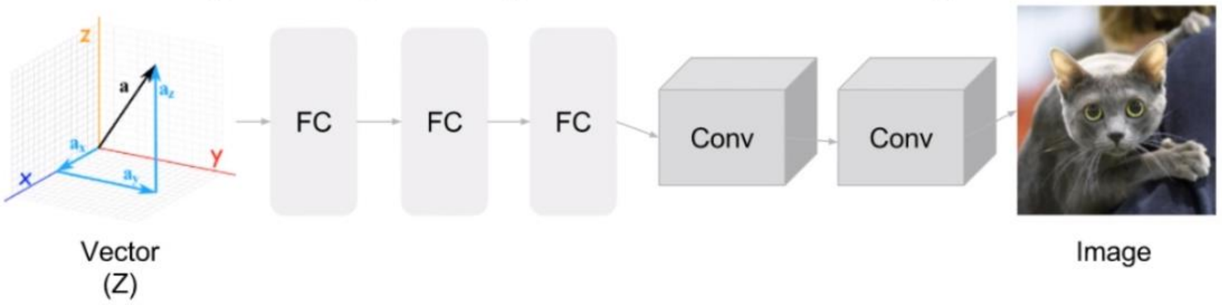
Exception to Pattern #1

- Exception: Generating images
- CNN for classification: Pattern = image → vector



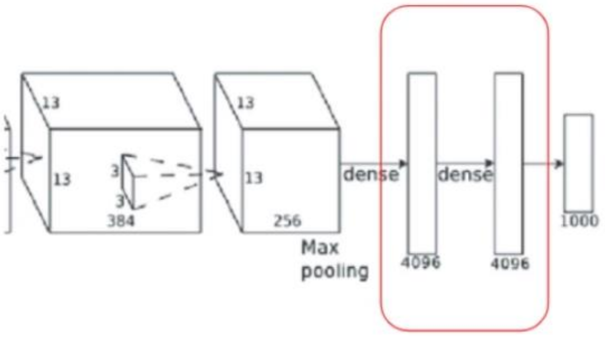
Exception to Pattern #1

- Exception: Generating images
- CNN for generating an image: Pattern = vector → image



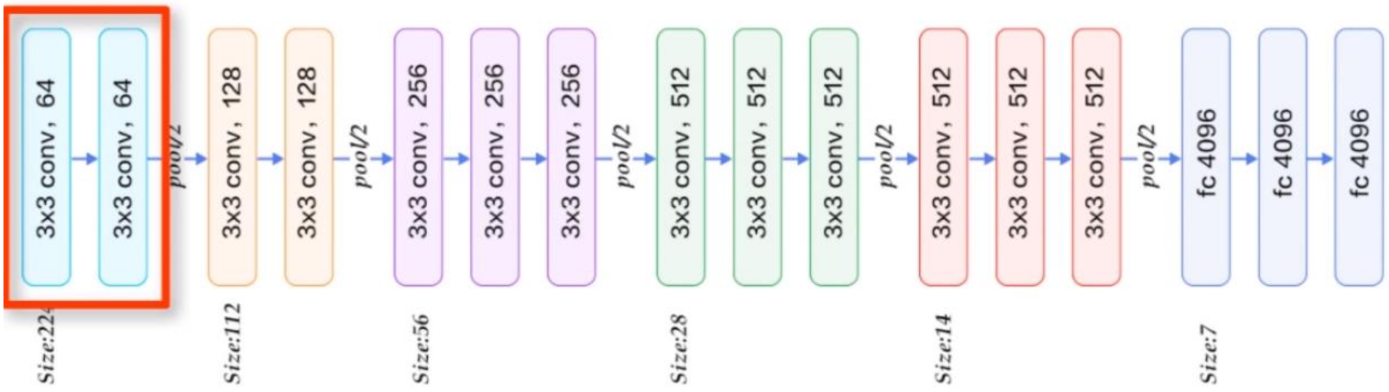
Fully connected layers

- Typically either all hidden layers are the same size or decrease (same size is more common - research has found it doesn't overfit)
- Same # of hidden units per layer also allows us less choices - only have to decide 2 things: # hidden layers, # hidden units



VGG

- There are a number of different architectures that fall under VGG, named by the # of layers they have (e.g. VGG16)

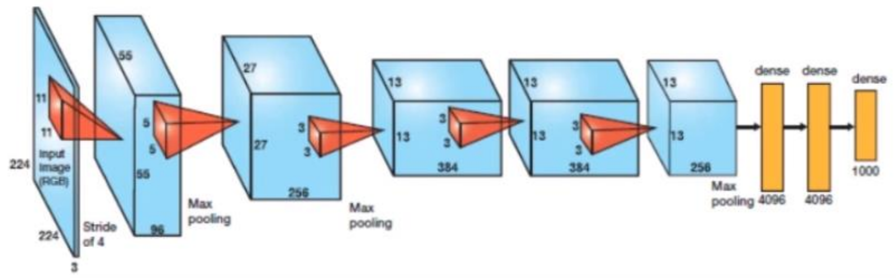


VGG

- If you want to read more about VGG16, check out extra_reading.txt:
  - *VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION*
  - ImageNet 2014
- VGG = Visual Geometry Group

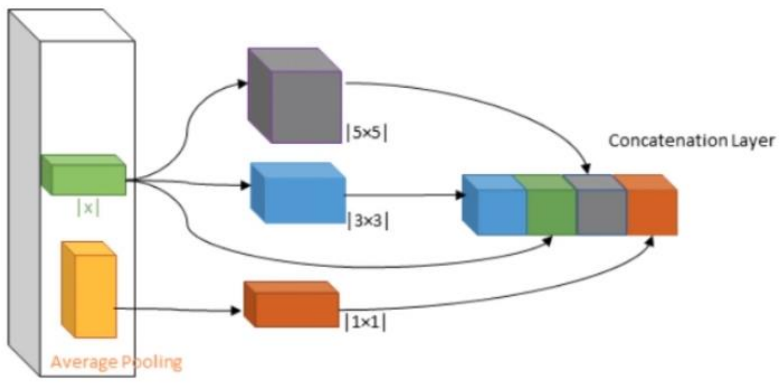
AlexNet

- Geoff Hinton is listed as an author
- 5 conv, 3 FC
- Why smaller than VGG? Appeared in 2012 ImageNet competition
- General pattern: Networks get deeper and deeper
- Extra_reading.txt: *ImageNet Classification with Deep Convolutional Neural Networks*



Inception module

- Instead of having to choose filter size, we convolve with multiple filter sizes, concat the result along depth dimension
- extra_reading.txt: "*Going Deeper with Convolutions*"



Goals of the Project

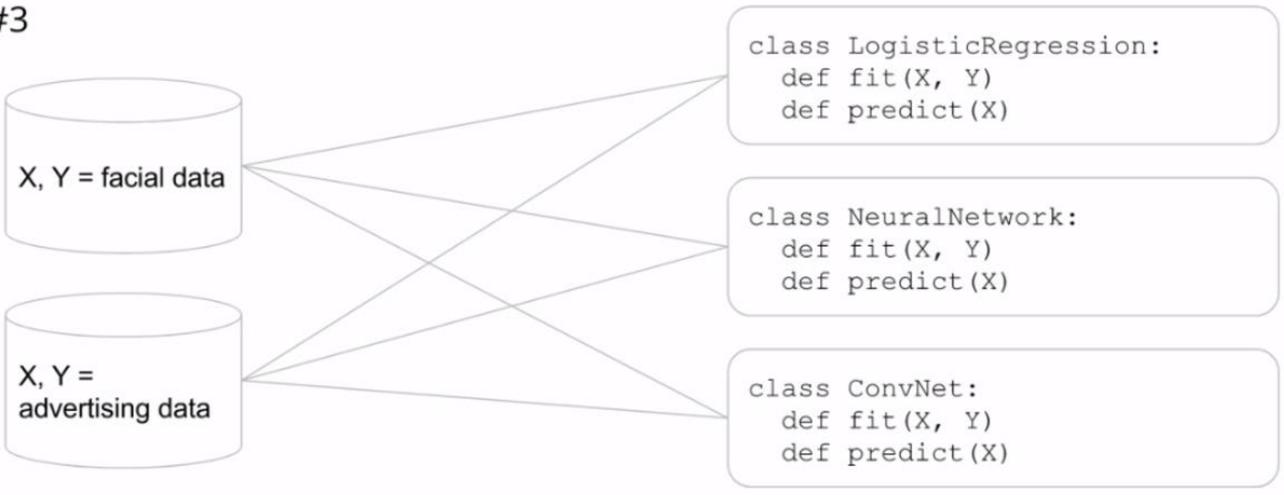
#2

- Make the model class-based
- Mimic the Sci-Kit Learn interface
- All supervised SKLearn models work like this:

```
model = MyModel()
model.fit(X, Y)
model.predict(X)
model.score(X, Y)
```

Goals of the Project

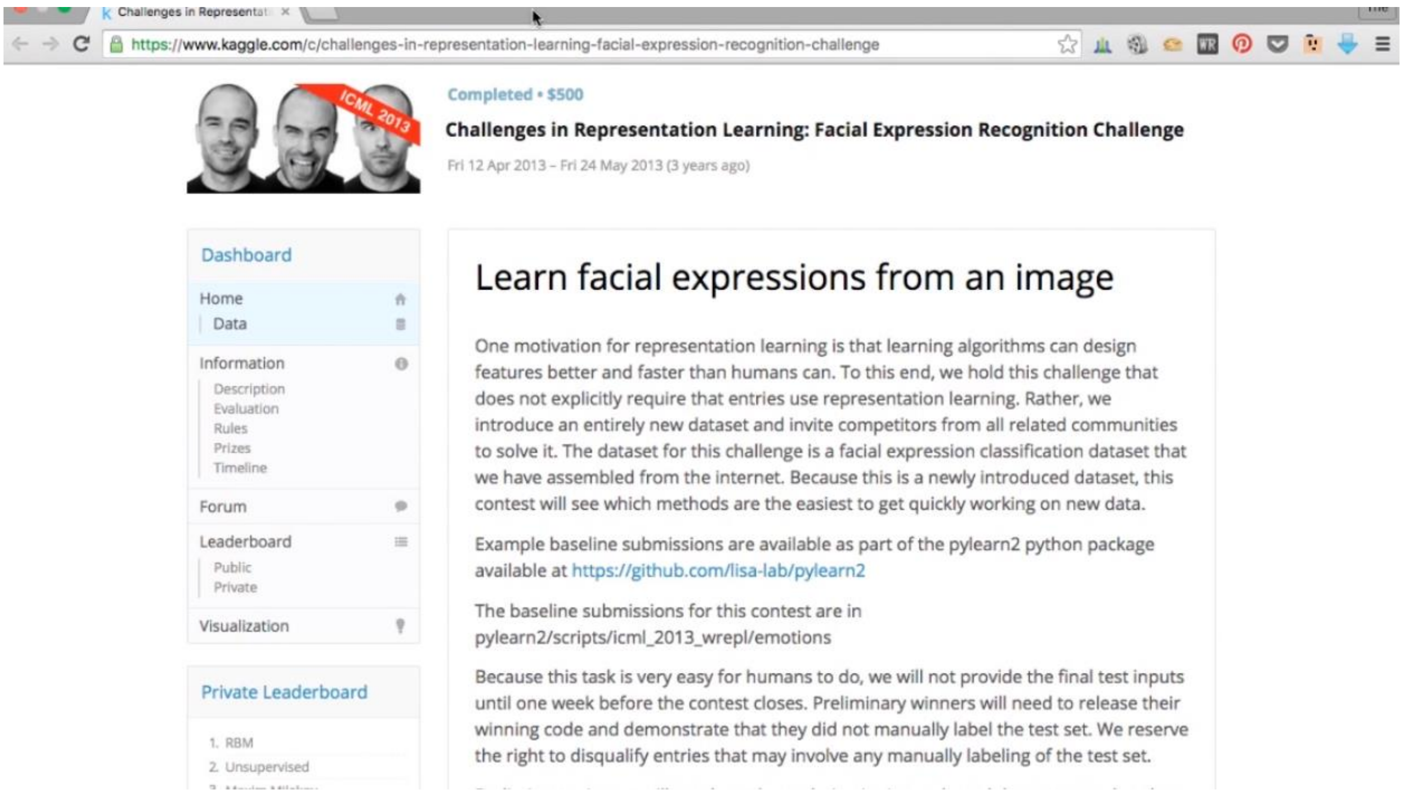
#3



```
class LogisticRegression:
  def fit(X, Y)
  def predict(X)
```

```
class NeuralNetwork:
  def fit(X, Y)
  def predict(X)
```

```
class ConvNet:
  def fit(X, Y)
  def predict(X)
```


https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge

Completed • $500

Challenges in Representation Learning: Facial Expression Recognition Challenge

Fri 12 Apr 2013 – Fri 24 May 2013 (3 years ago)

Learn facial expressions from an image

One motivation for representation learning is that learning algorithms can design features better and faster than humans can. To this end, we hold this challenge that does not explicitly require that entries use representation learning. Rather, we introduce an entirely new dataset and invite competitors from all related communities to solve it. The dataset for this challenge is a facial expression classification dataset that we have assembled from the internet. Because this is a newly introduced dataset, this contest will see which methods are the easiest to get quickly working on new data.

Example baseline submissions are available as part of the pylearn2 python package available at https://github.com/lisa-lab/pylearn2

The baseline submissions for this contest are in pylearn2/scripts/icml_2013_wrepl/emotions

Because this task is very easy for humans to do, we will not provide the final test inputs until one week before the contest closes. Preliminary winners will need to release their winning code and demonstrate that they did not manually label the test set. We reserve the right to disqualify entries that may involve any manually labeling of the test set.

The data

- 48x48 grayscale images
- Faces are centered, approx same size
- 7 classes:
  - 0 = Angry
  - 1 = Disgust
  - 2 = Fear
  - 3 = Happy
  - 4 = Sad
  - 5 = Surprise
  - 6 = Neutral
- 3 column CSV: label, pixels (space-separated), train/test

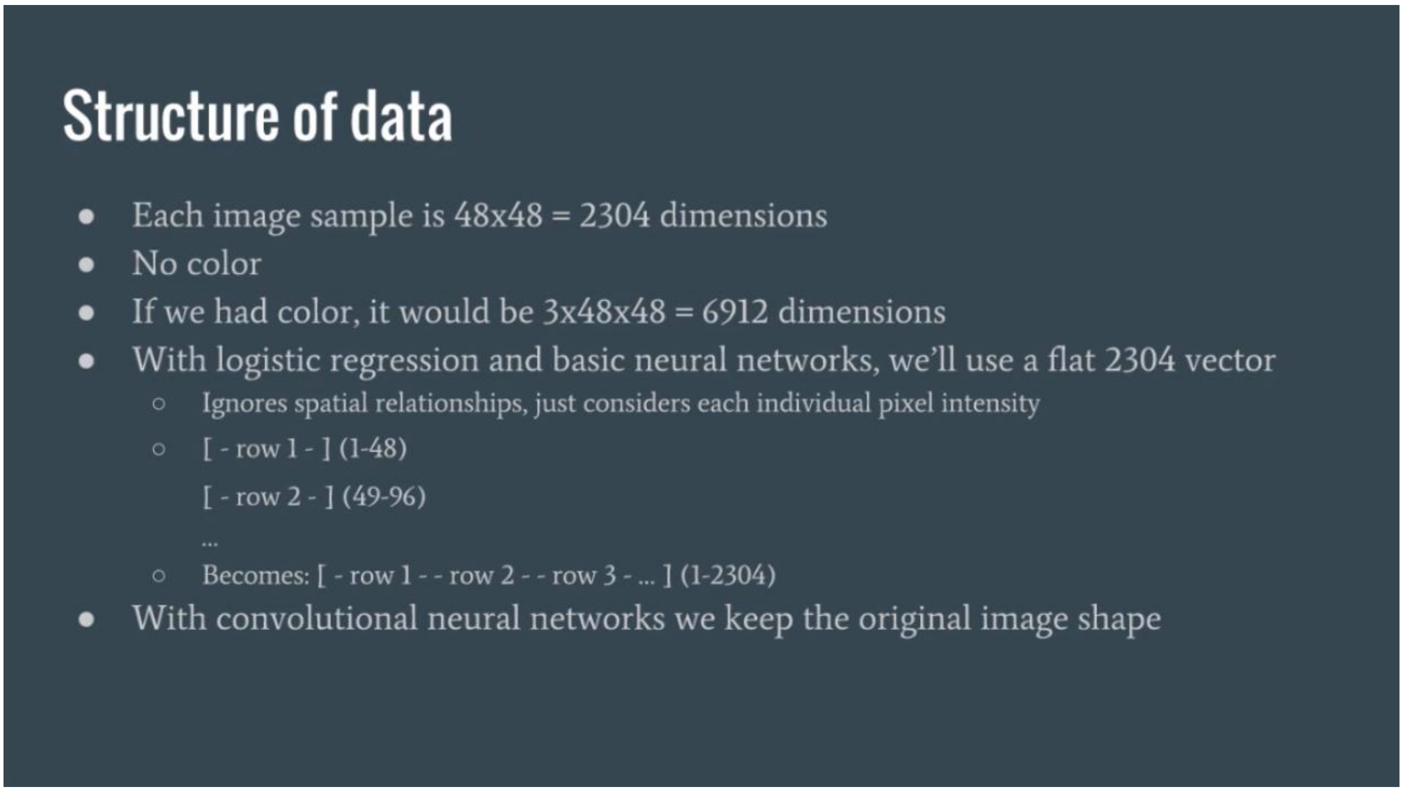
Logistic Regression

- We've only learned binary classification at this point
- Will focus on class 0 and 1
- 4953 samples of class 0, 547 samples of class 1
  - What would my classification rate be if I just choose 0 every time?

2-class problem vs. 7-class problem

- When we switch to softmax (which we will in Deep Learning part 1), will the problem get easier or harder?
- 2 class:
  - Guess at random - expect 50% error
- 7 class:
  - Guess at random - expect 6/7 = 86% error
- K class: 1/K chance of being correct

Kaggle top score: ~70% correct

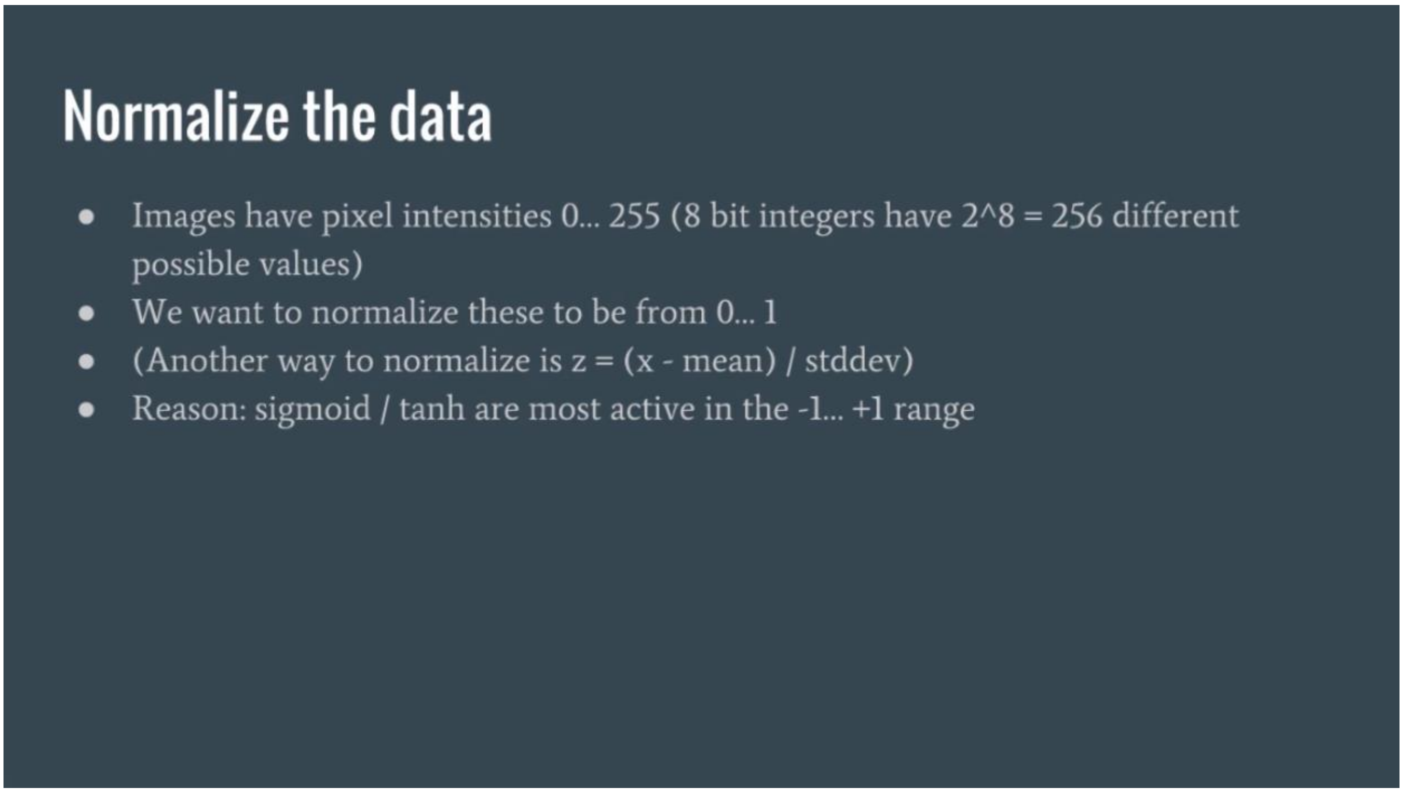


Structure of data

- Each image sample is 48x48 = 2304 dimensions
- No color
- If we had color, it would be 3x48x48 = 6912 dimensions
- With logistic regression and basic neural networks, we'll use a flat 2304 vector
  - Ignores spatial relationships, just considers each individual pixel intensity
  - [- row 1 -] (1-48)

    [- row 2 -] (49-96)

    ...
  - Becomes: [- row 1 - - row 2 - - row 3 - ...] (1-2304)
- With convolutional neural networks we keep the original image shape



Normalize the data

- Images have pixel intensities 0... 255 (8 bit integers have 2^8 = 256 different possible values)
- We want to normalize these to be from 0... 1
- (Another way to normalize is z = (x - mean) / stddev)
- Reason: sigmoid / tanh are most active in the -1... +1 range

Challenges

- Deep learning doesn't really give you plug-and-chug black boxes
- Correct learning rate?
  - Too high → NaN
  - Too low → Slow convergence
- Correct regularization
  - Too high → Just makes W very small, ignores actual pattern
  - Too low → Cost may still explode to NaN
- How many epochs?
  - Stop too early → Steep slope, not at minimum error
  - Stop too late → No effect on error, takes too long

Class Imbalance

- In facial expression recognition problem, we have 547 samples from class 1 and 4953 samples from class 0 (class 1 is the only class with substantially less samples)
- Why is this a problem?
- Medical testing:
  - Disease is present in 1% of population
  - Predict "no disease" every time → classifier is 99% accurate
  - Hasn't learned anything
- But we don't use accuracy to train, we use cross-entropy
  - Same problem

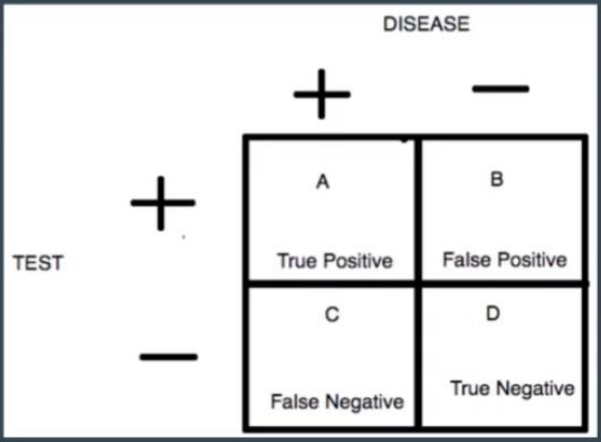
Solving class imbalance

- Suppose we have 1000 samples from class 1, 100 samples from class 2
- Method 1) Pick 100 samples from class 1, now we have 100 vs. 100
- Method 2) Repeat class 2 10 times, now we have 1000 vs. 1000
- Same *expected* error rate
- But method 2 is better (less variance, more data)
- Other options to expand class 2:
    - Add Gaussian noise
    - Add invariant transformations (shift left, right, rotate, etc.)

Other accuracy measures

- Measures can allow for class imbalance
- Medical field and information retrieval
- Assumes binary classification
- Basic idea:
- Maximize TP, TN
- Minimize FP, FN

Note: classification rate =
(TP+TN)/(TP+TN+FP+FN)



Sensitivity and Specificity

- Used in medical field
- Sensitivity:

True positive rate = TPR = TP / (TP + FN)

- Specificity:

True negative rate = TNR = TN / (TN + FP)

Precision and Recall

- Used in information retrieval
- Precision

TP / (TP + FP)

- Recall

TP / (TP + FN)

- Note: recall = sensitivity

F1-score

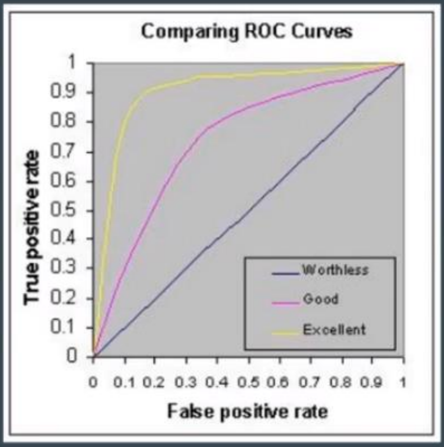
- Combines precision and recall into a balanced measure

F1 = 2 * (precision * recall) / (precision + recall)

- Harmonic mean of precision and recall

ROC and AUC

- In logistic regression we use 0.5 as a threshold, makes sense because P(Y=1|X) > 0.5 → guess 1, P(Y=1|X) < 0.5 → guess 0
- But we can use any threshold
- Different thresholds → different TPR and FPR
- Receiver operating characteristic (ROC curve is a plot of these for every value of threshold between 0 and 1)
- Area under curve = AUC
    - 1 = perfect classifier, 0.5 = random guessing


Comparing ROC Curves
True positive rate
False positive rate
Worthless
Good
Excellent