Deep Learning

Europython 2016 - Bilbao

G. French
University of East Anglia

Image montages from http://www.image-net.org

Focus:

Mainly image processing

This talk is more about the principles and the maths than code

Got to fit this into 1 hour!

What we’ll cover

Theano
What it is and how it works

What is a neural network?
The basic model; the multi-layer perceptron

Convolutional networks
Neural networks for computer vision

Lasagne
The Lasagne neural network library

Notes for building neural networks
A few tips on building and training neural networks

OxfordNet / VGG and transfer learning
Using a convolutional network trained by the VGG group at Oxford University and re-purposing it for your needs

Talk materials

Github Repo (originally for PyData London):

https://github.com/Britefury/deep-learning-tutorial-pydata2016

The notebooks are viewable on Github

Intro to Theano and Lasagne slides:

https://speakerdeck.com/britefury

https://speakerdeck.com/britefury/intro-to-theano-and-lasagne-for-deep-learning

Amazon AMI (Use GPU machine)

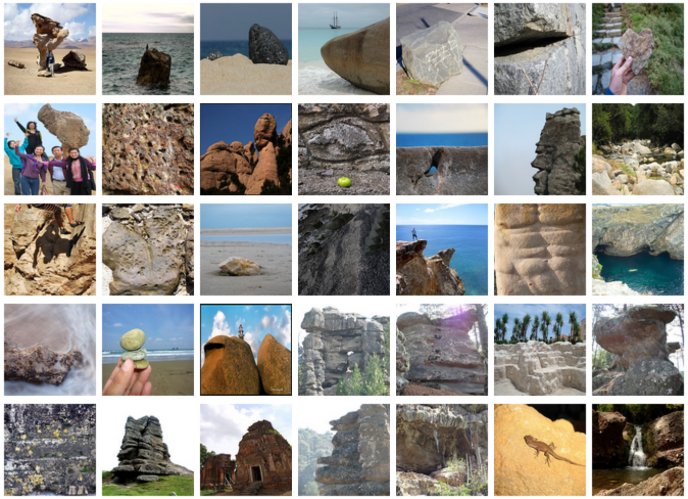
AMI ID: `ami-e0048af7`

AMI Name:

`Britefury deep learning - Ubuntu-14.04 Anaconda2-4.0.0 Cuda-7.5 cuDNN-5 Theano-0.8 Lasagne Fuel`

ImageNet

Image classification dataset



~1,000,000 images
~1,000 classes

Ground truths prepared manually
through Amazon Mechanical Turk

ImageNet Top-5 challenge:

You score if ground truth class is one
your top 5 predictions

ImageNet in 2012

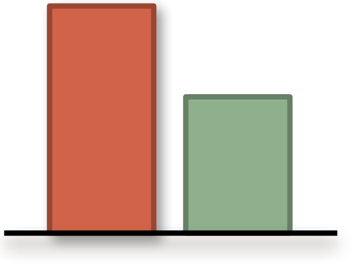
Best approaches used hand-crafted features (SIFT, HOGs, Fisher vectors, etc) + classifier

Top-5 error rate: ~25%

Then the game changed.

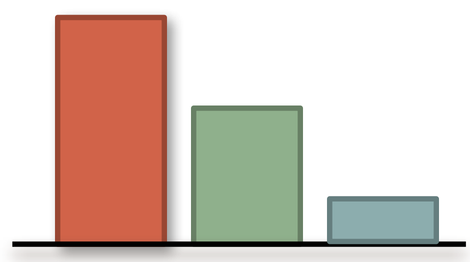
Krizhevsky, Sutskever and Hinton;
ImageNet Classification with Deep Convolutional Neural networks
[Krizhevsky12]

Top-5 error rate of ~15%



In the last few years, more modern networks have achieved better results still [Simonyan14, He15]

Top-5 error rates of ~5-7%



I hope this talk will give you an idea of how!

Theano

Neural network software comes in two flavours:

Neural network toolkits

Expression compilers

Neural network toolkit

Specify structure of neural network in terms of layers

Expression compilers

Lower level

Describe the mathematical expressions behind the layers

More powerful and flexible

Theano

An expression compiler

Write NumPy style expressions

Compiles to either C (CPU) or CUDA (nVidia GPU)

Intro to Theano and Lasagne slides:

https://speakerdeck.com/britefury

https://speakerdeck.com/britefury/intro-to-theano-and-lasagne-for-deep-learning

There is much more to Theano

For more information:

http://deeplearning.net/tutorial
http://deeplearning.net/software/theano

There are others

Tensorflow – developed by Google – is gaining popularity *fast*

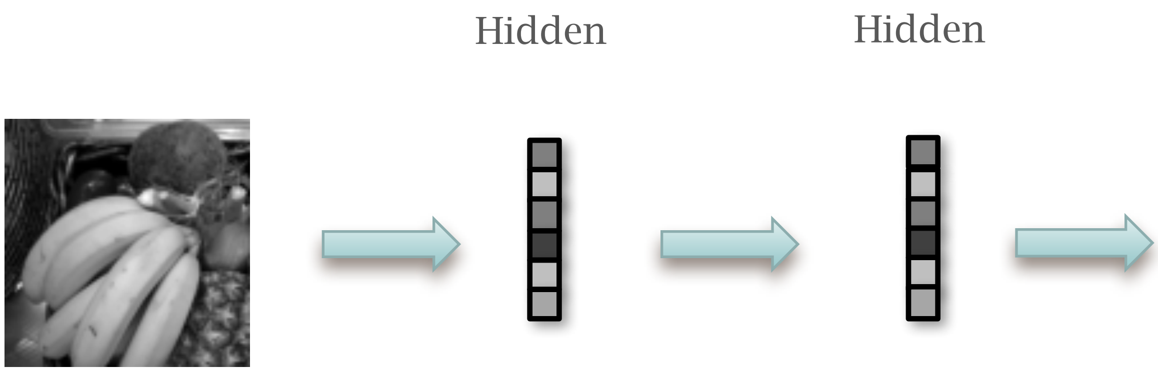
What is a neural network?

Multiple layers

Data propagates through layers

Transformed by each layer

Neural network image classifier



Class probabilities

$P(cat) = 0.003$

$P(dog) = 0.002$

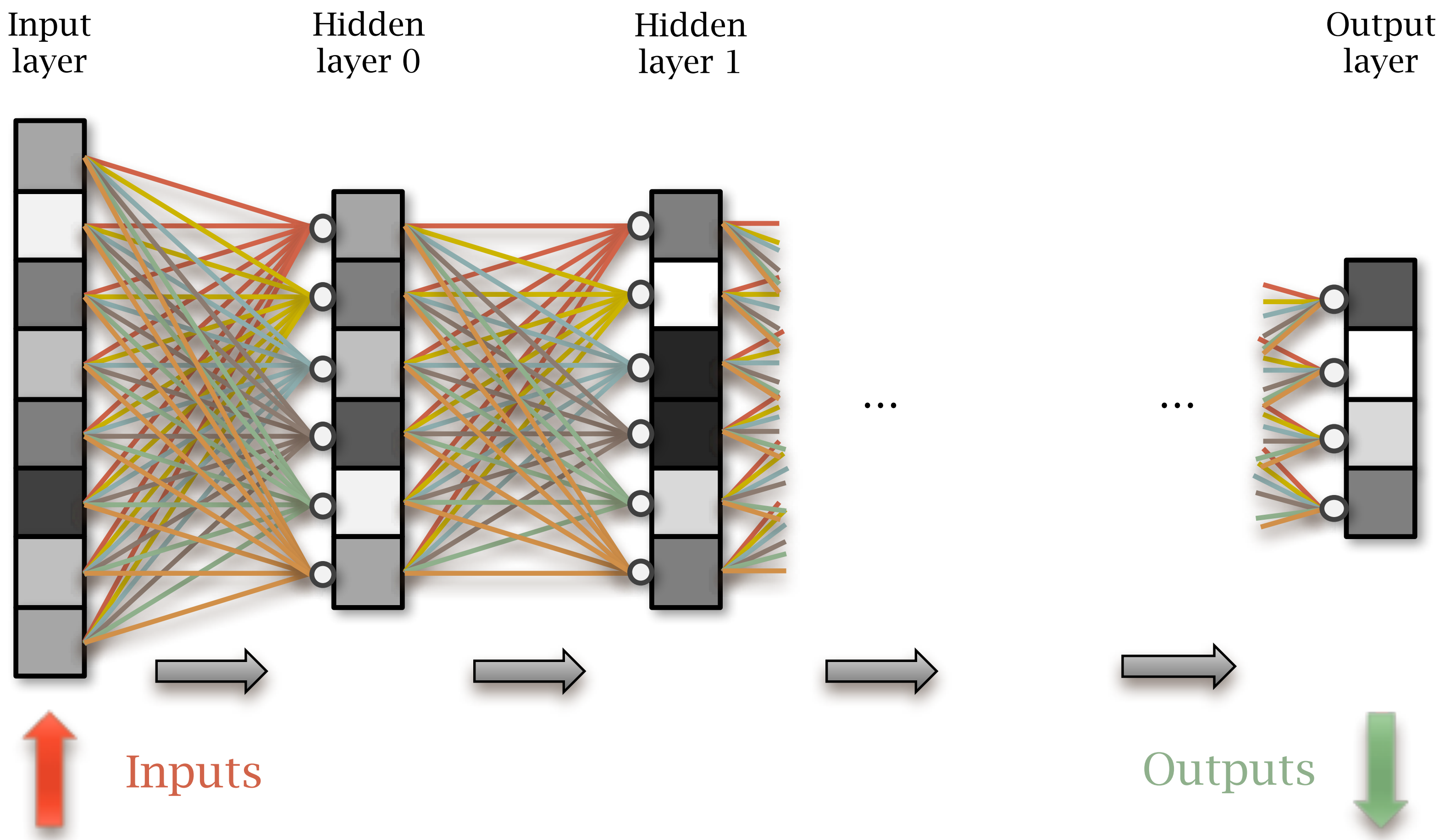
$P(car) = 0.005$

$\boxed{P(bananas) = 0.9}$

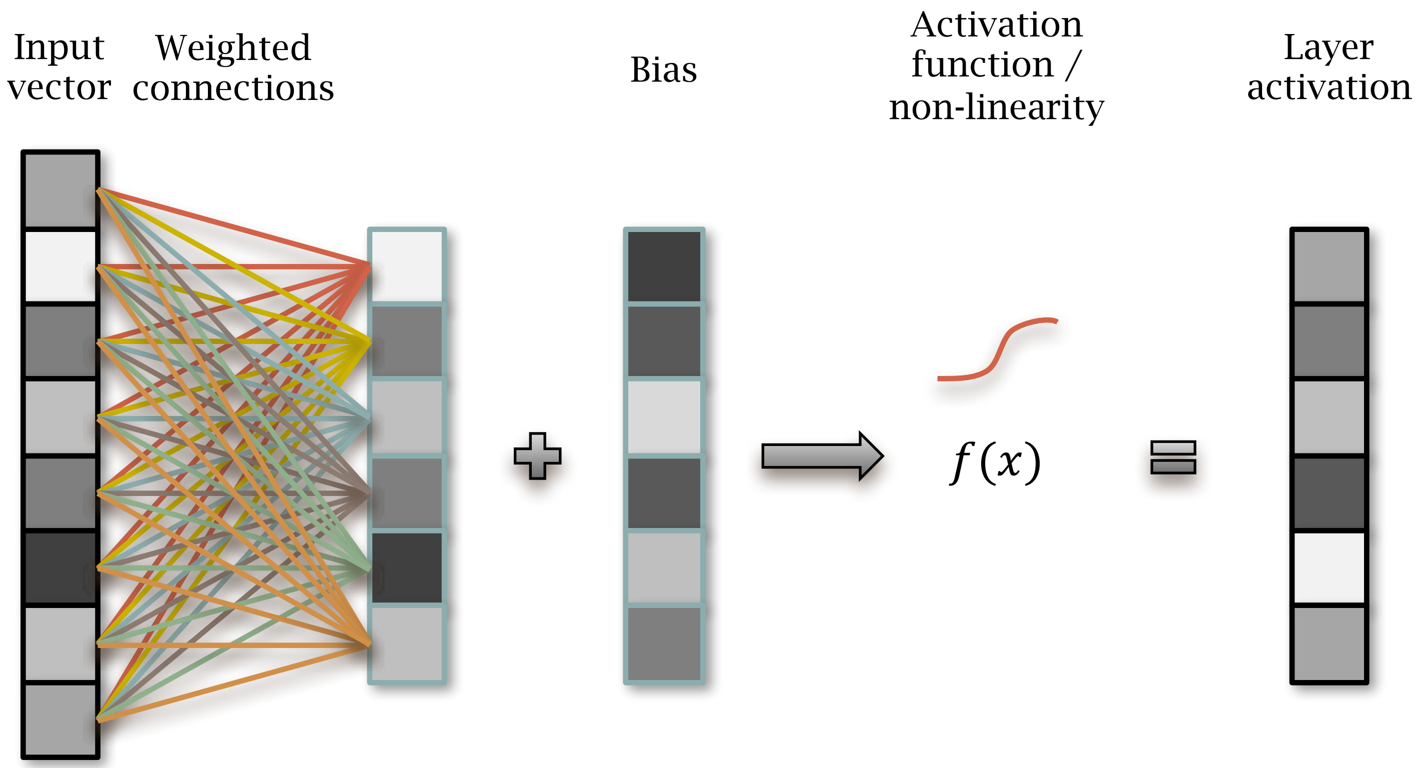
Inputs

Outputs

Neural network

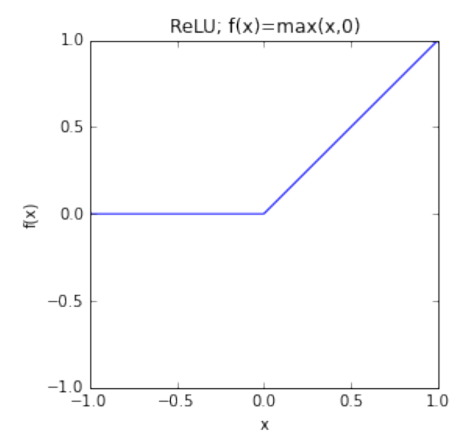


Single layer of a neural network



x = input (M-element vector)
y = output (N-element vector)
W = weights parameter (NxM matrix)
b = bias parameter (N-element vector)
f = non-linearity (a.k.a. activation function); normally *ReLU* but can be *tanh* or *sigmoid*

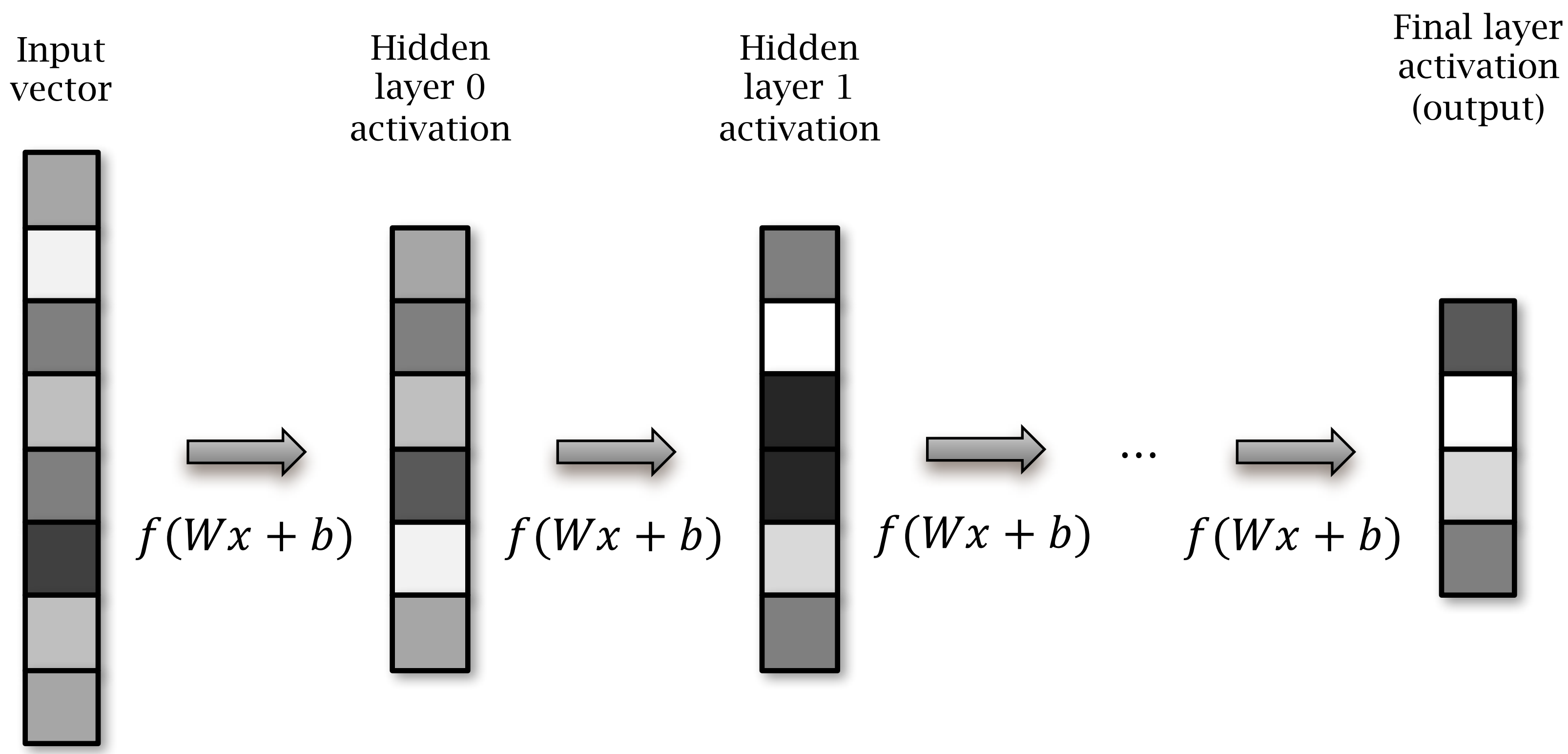
$$y = f(Wx + b)$$



In a nutshell:

$$y = f(Wx + b)$$

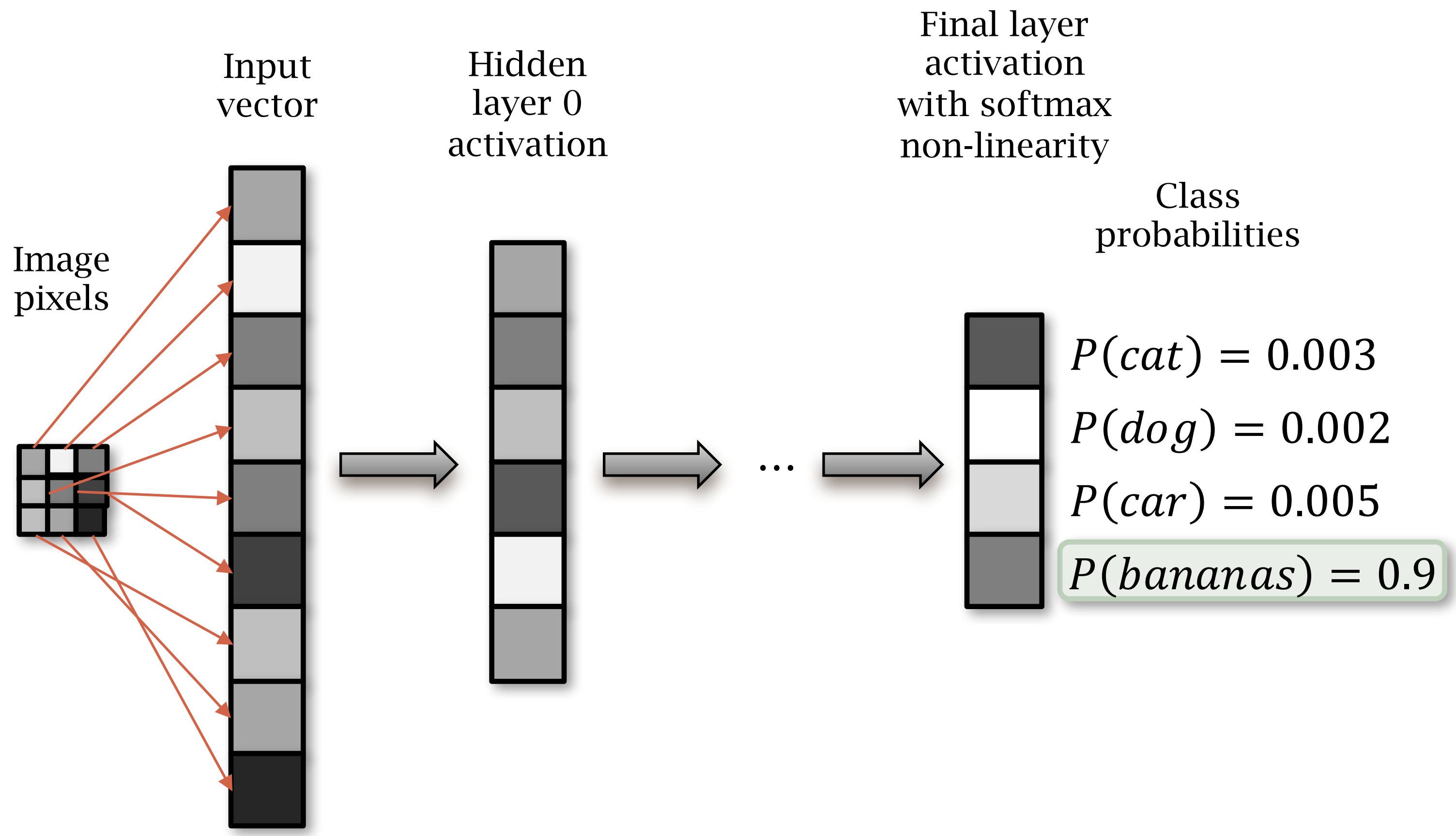
Repeat for each layer



In mathematical notation:

$$
\begin{gathered}
y_0 = f(W_0 x + b_0) \\
y_1 = f(W_1 y_0 + b_1) \\
\cdots \\
y_L = f(W_L y_{L-1} + b_L)
\end{gathered}
$$

As a classifier



Summary; a neural network is:

Built from layers, each of which is:

a matrix multiplication,
then add bias,
then apply non-linearity.

Training a neural network

Learn values for parameters; W and b
(for each layer)

Use back-propagation

Initialise weights randomly (more on this later)

Initialise biases to 0

For each example x_{train} from training set

evaluate network prediction y_{pred} given the training input; $x = x_{train}$

Measure cost c (error); difference between y_{pred} and ground truth output y_{train}

Classification
(which of these categories best describes this?)

Final layer: softmax as non-linearity f; output vector of class probabilities

Cost: negative-log-likelihood / categorical cross-entropy

Regression
(quantify something, real-valued output)

Final layer: no non-linearity / identity as f

Cost: Sum of squared differences

Reduce cost c (also known as loss) using gradient descent

Compute the derivative (gradient) of *cost* w.r.t. parameters (all W and b)

Theano performs symbolic differentiation for you!

```
dCdW = theano.grad(cost, W)
```

(other toolkits – such as Torch and Tensorflow – can also do this)

Update parameters:

$$W_0' = W_0 - \gamma \frac{dc}{dW_0}$$

$$b_0' = b_0 - \gamma \frac{dc}{db_0}$$

γ = learning rate

Randomly split the training set into *mini-batches* of ~100 samples.

Train on a *mini-batch* in a single step. The *mini-batch cost* is the mean of the *costs* of the samples in the *mini-batch.*

Training on *mini-batches* means that ~100 samples are processed in parallel – very good for running GPUs that do lots of operations in parallel

Training on all examples in the training set is called an *epoch*

Run multiple *epochs* (often 200-300)

Summary; train a neural network:

Take *mini-batch* of training samples
Evaluate (run/execute) the network
Measure the average error/cost across *mini-batch*
Use gradient descent to modify parameters to reduce *cost*
REPEAT ABOVE UNTIL DONE

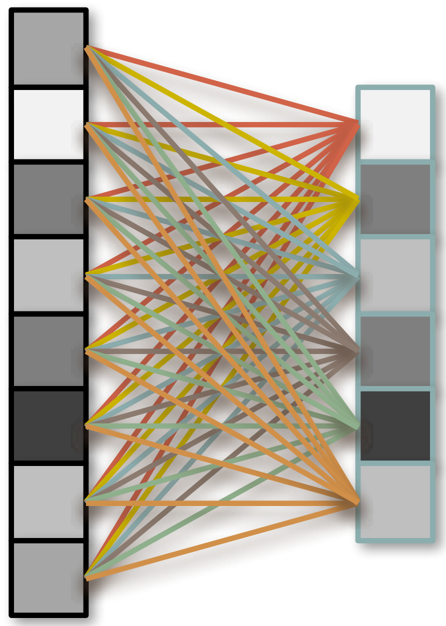
Multi-layer perceptron

Simplest network architecture

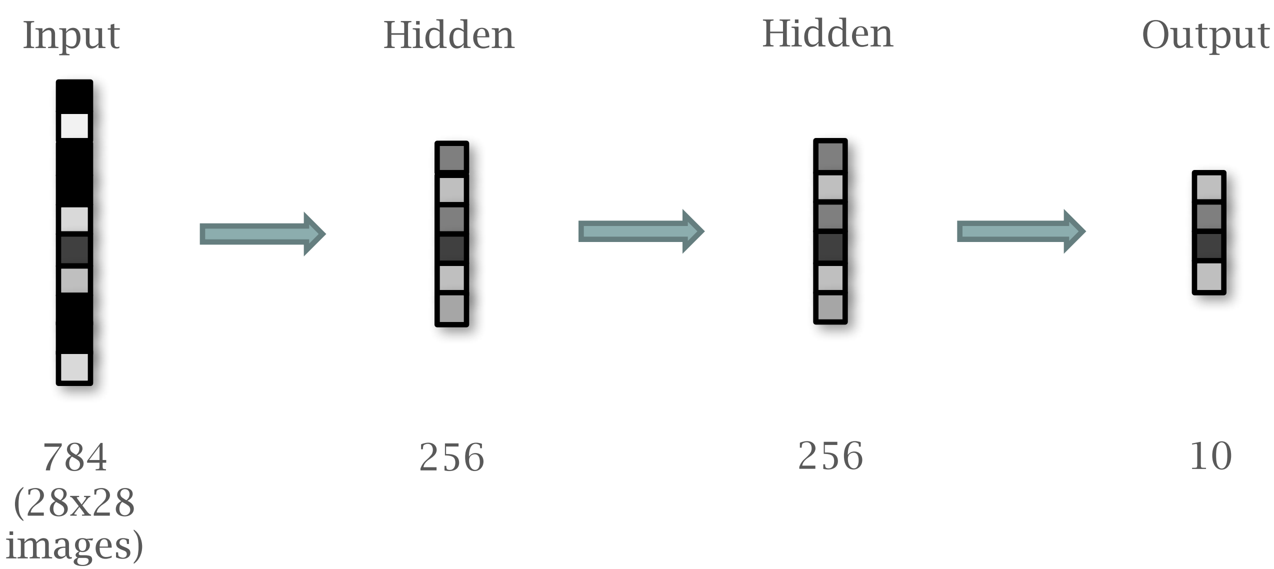
Nothing we haven’t seen so far

Uses only fully-connected / dense layers

Dense layer: each unit is connected too all units in previous layer



(Obligatory) MNIST example: 2 hidden layers, both 256 units after 300 iterations over training set: 1.83% validation error



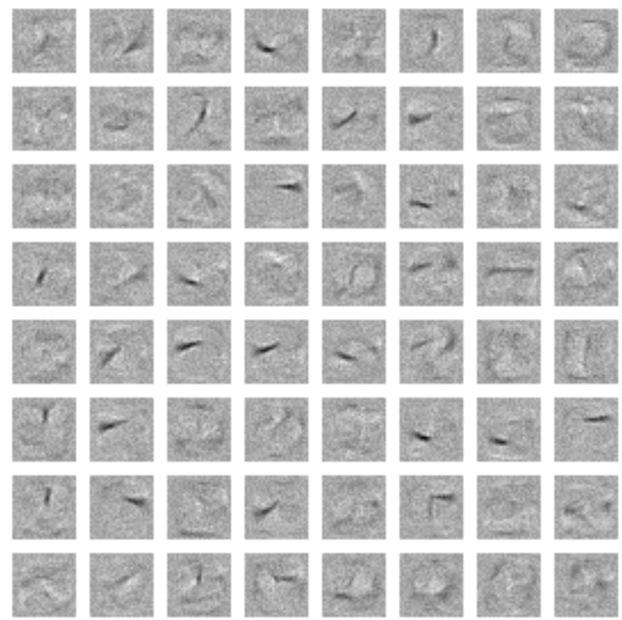
MNIST is quite a special case

Digits nicely centred within image

Scaled to approx. same size

The fully connected networks so far have a weakness:

No translation invariance; learned features are *position dependent*



For more general imagery:
requires a training set large enough to see all features in all possible positions…

Requires network with enough units to represent this…

Convolutional networks

Convolution

Slide a convolution kernel over an image

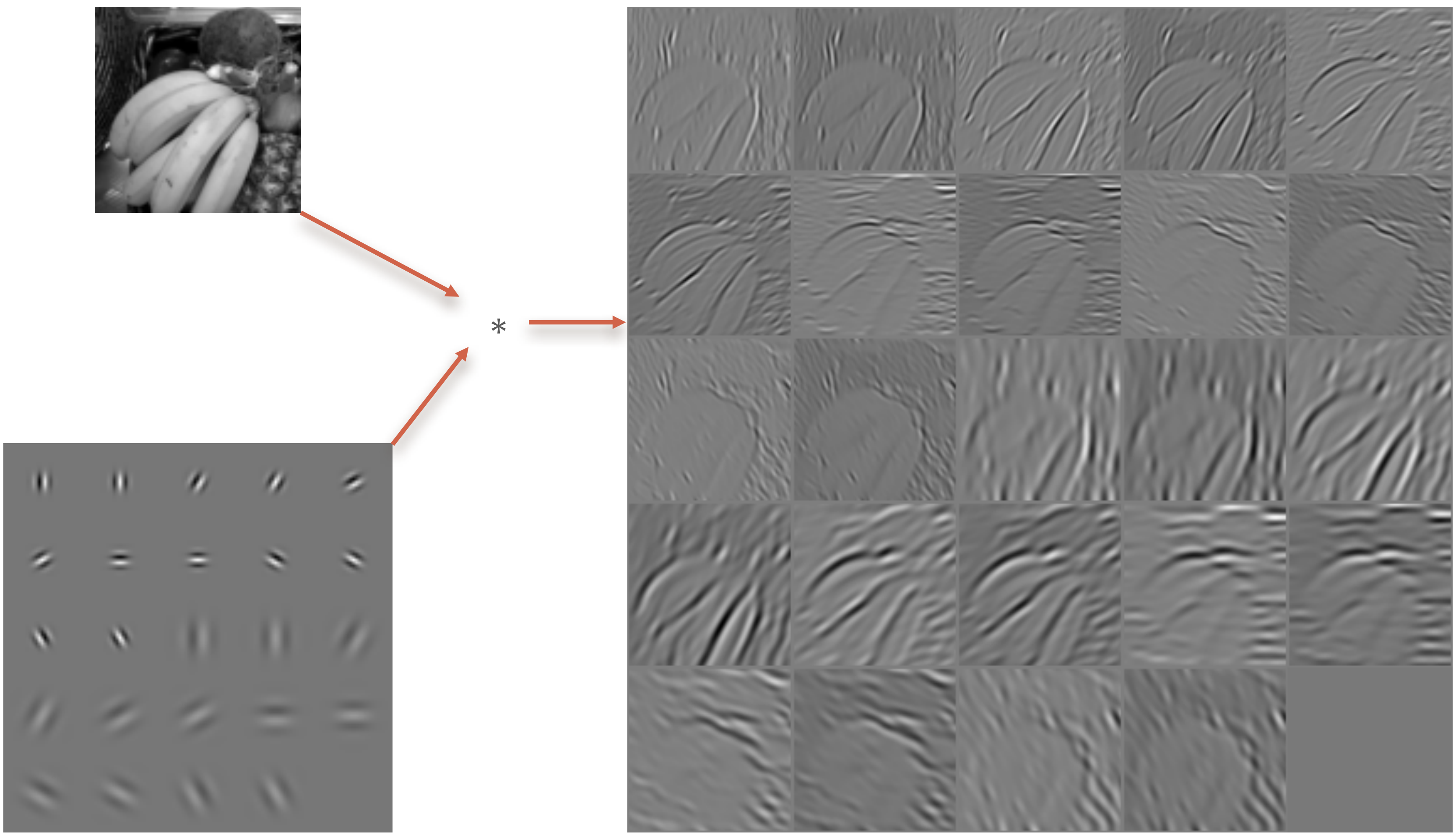
Multiply image pixels by kernel pixels and sum

Convolution

Convolutions are often used for feature detection

A brief detour…

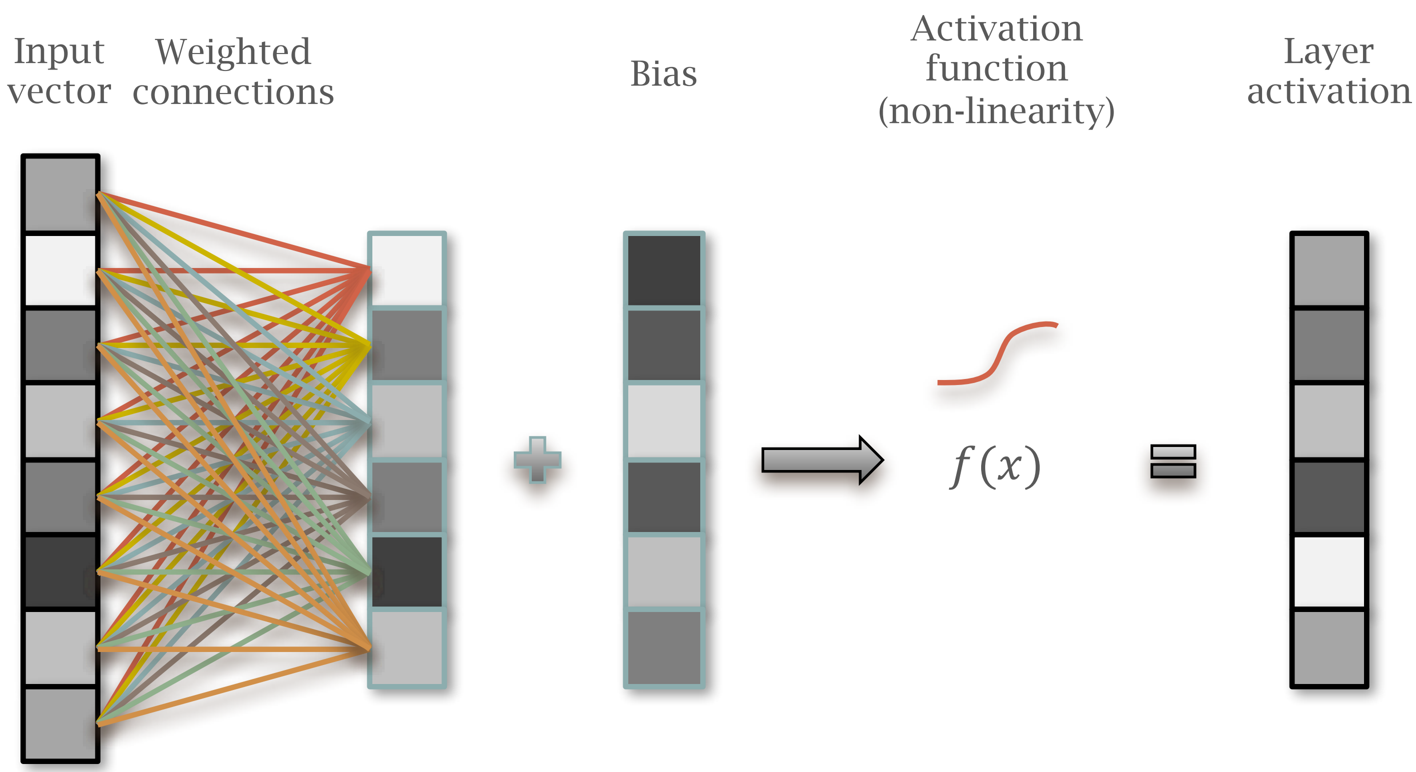
Gabor filters



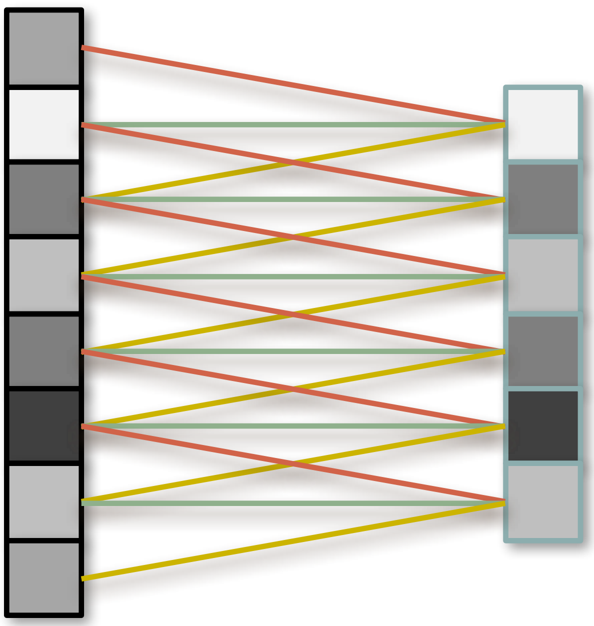
Back on track to…

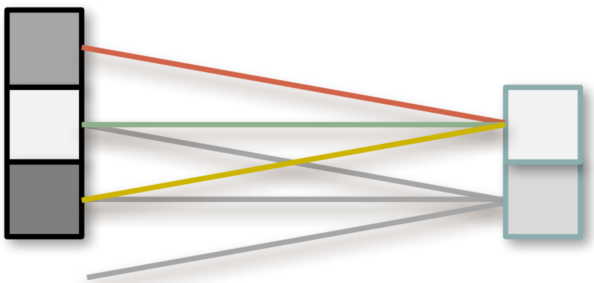
Convolutional networks

Recap: FC (fully-connected) layer



Convolutional layer

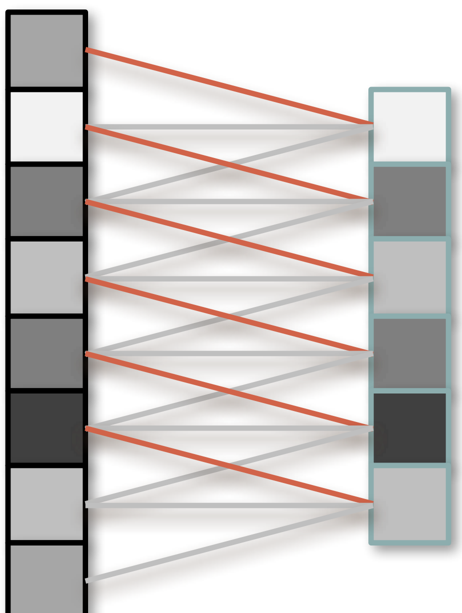




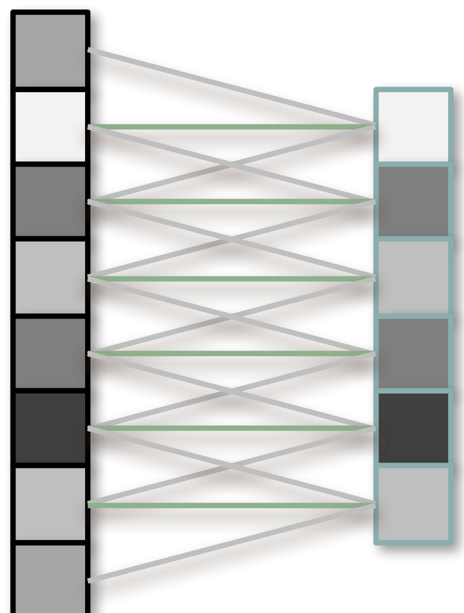
Each unit only connected to units in its neighbourhood

Convolutional layer

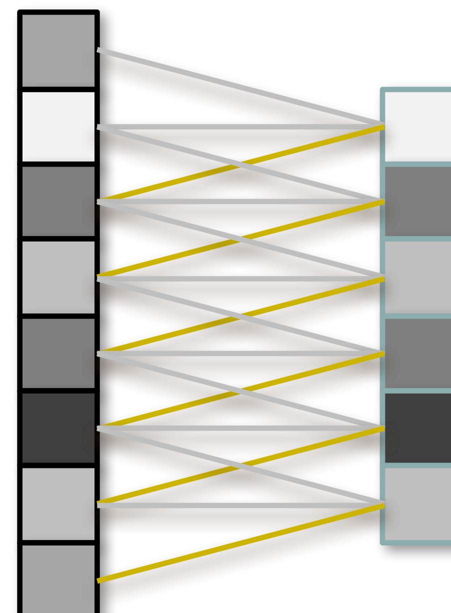
Weights are shared



Red weights have same value



As do greens…

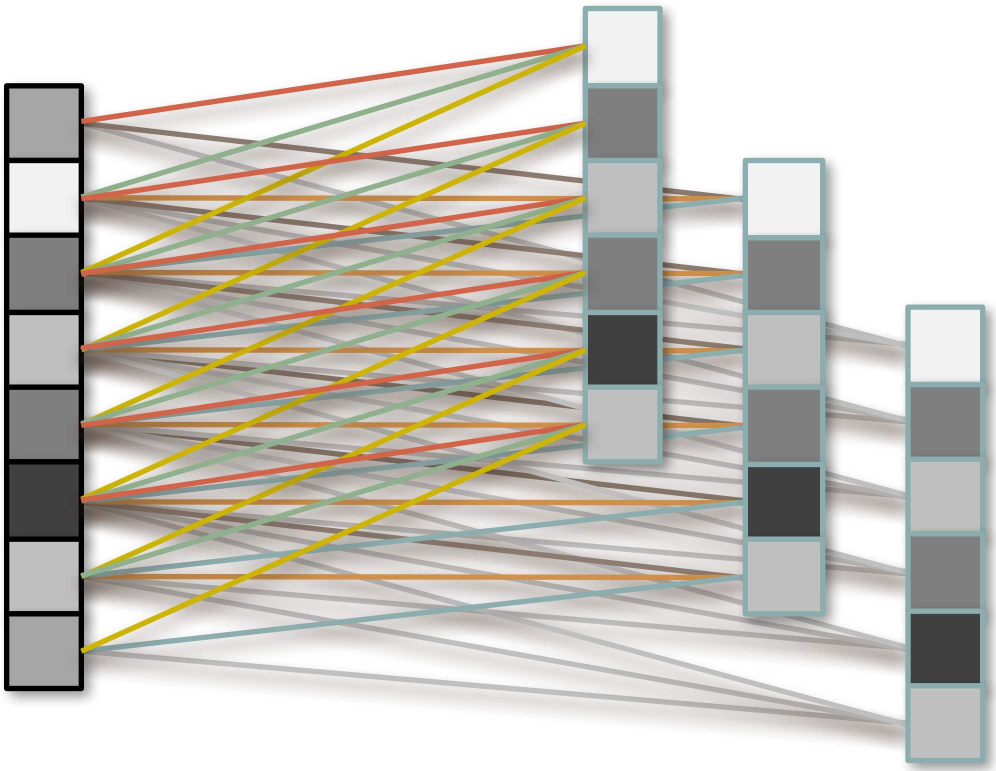


And yellows

The values of the weights form a convolution kernel

For practical computer vision, more an one kernel must be used to extract a variety of features

Convolutional layer



Different weight-kernels:

Output is image with multiple channels

Note

Each kernel connects to pixels in ALL channels in previous layer

Still

$$y = f(Wx + b)$$

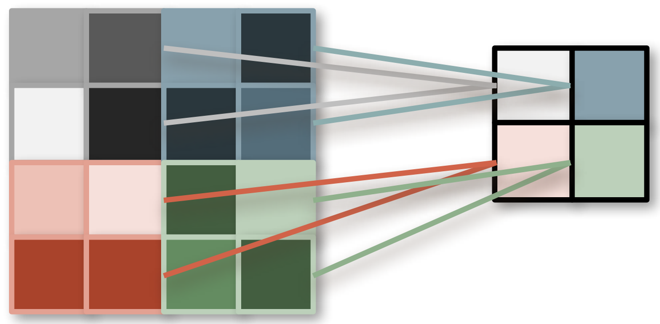
As convolution can be expressed as multiplication by weight matrix

Down-sampling

In typical networks for computer vision, we need to shrink the resolution after a layer, by some constant factor

Use max-pooling or striding

Down-sampling: max-pooling *'layer'* [Ciresan12]



Take maximum value from each 2 x 2 pooling region (p x p) in the general case

Down-samples image by factor p

Operates on channels independently

Down-sampling: striding

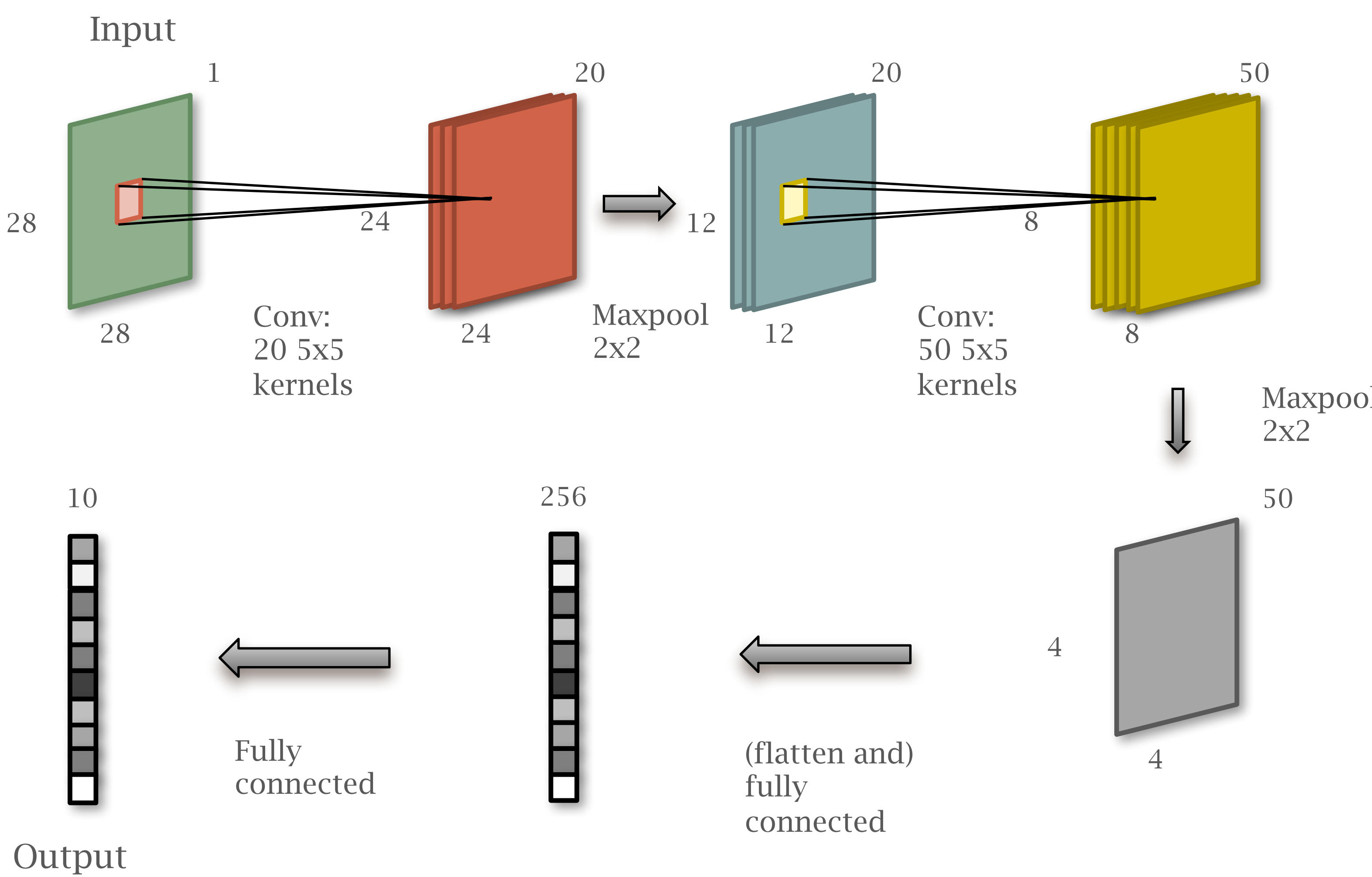
Can also down-sample using strided convolution; generate output for 1 in every n pixels

Faster, can work as well as max-pooling

Example:

A Simplified LeNet [LeCun95] for MNIST digits

Simplified LeNet for MNIST digits

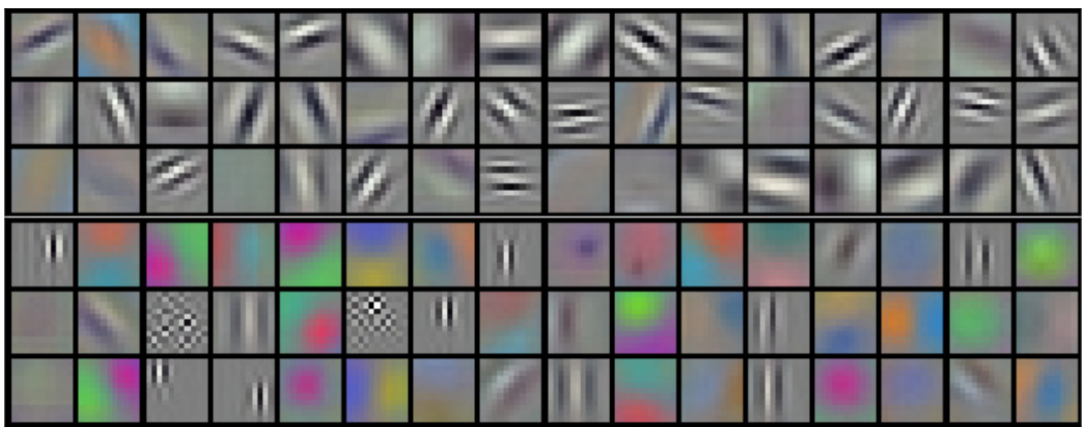


after 300 iterations over training set:
99.21% validation accuracy

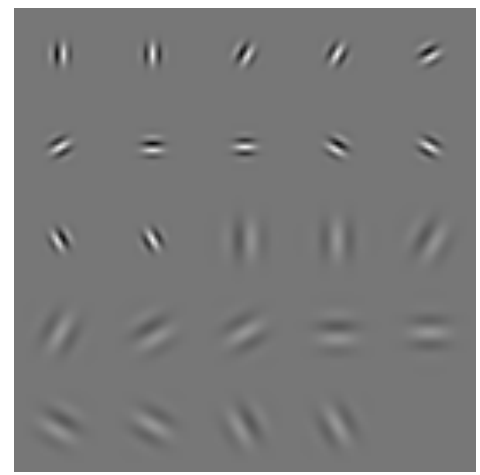
Model	Error
FC64	2.85%
FC256--FC256	1.83%
20C5--MP2--50C5--MP2--FC256	0.79%

What about the learned kernels?

Image taken from paper [Krizhevsky12]
(ImageNet dataset, not MNIST)



Gabor filters



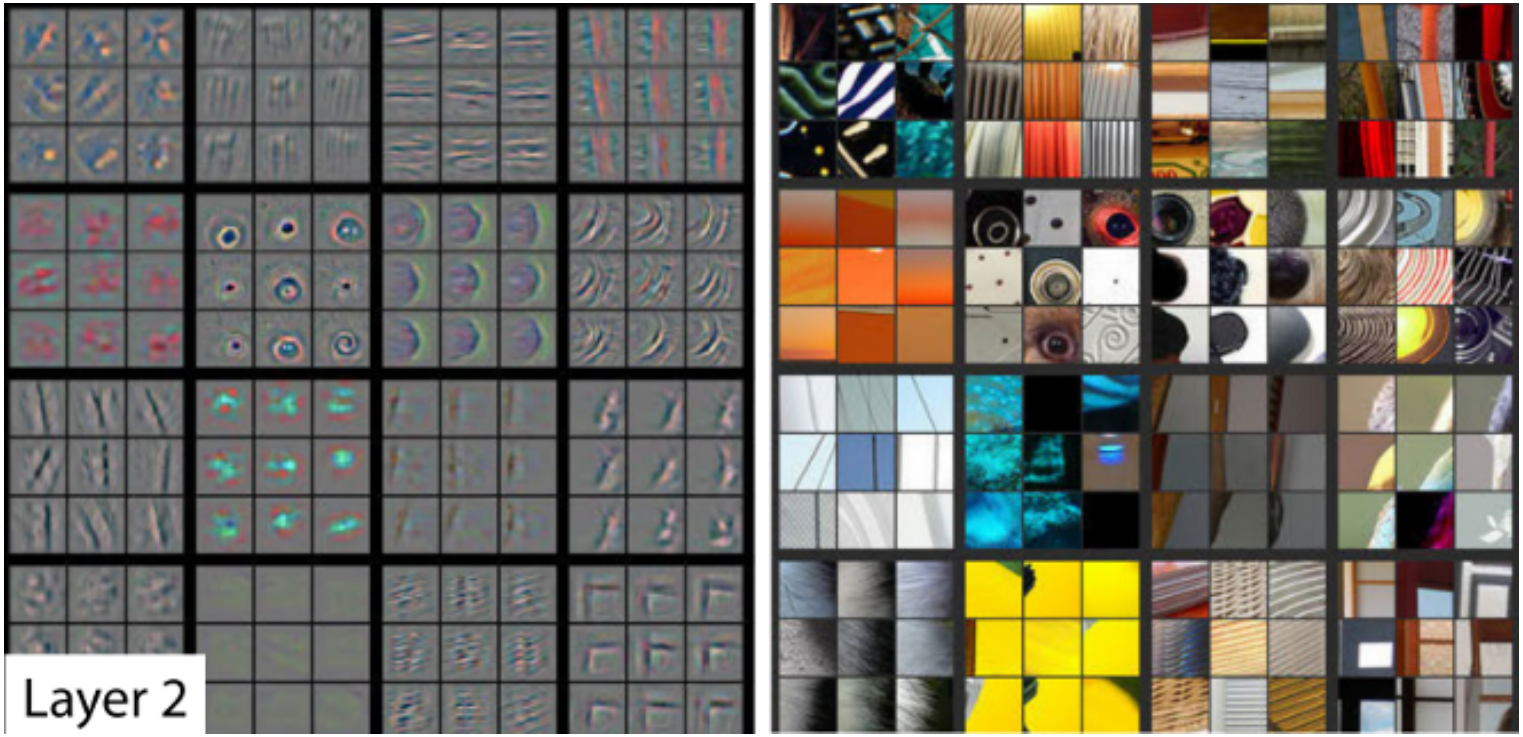


Image taken from [Zeiler14]

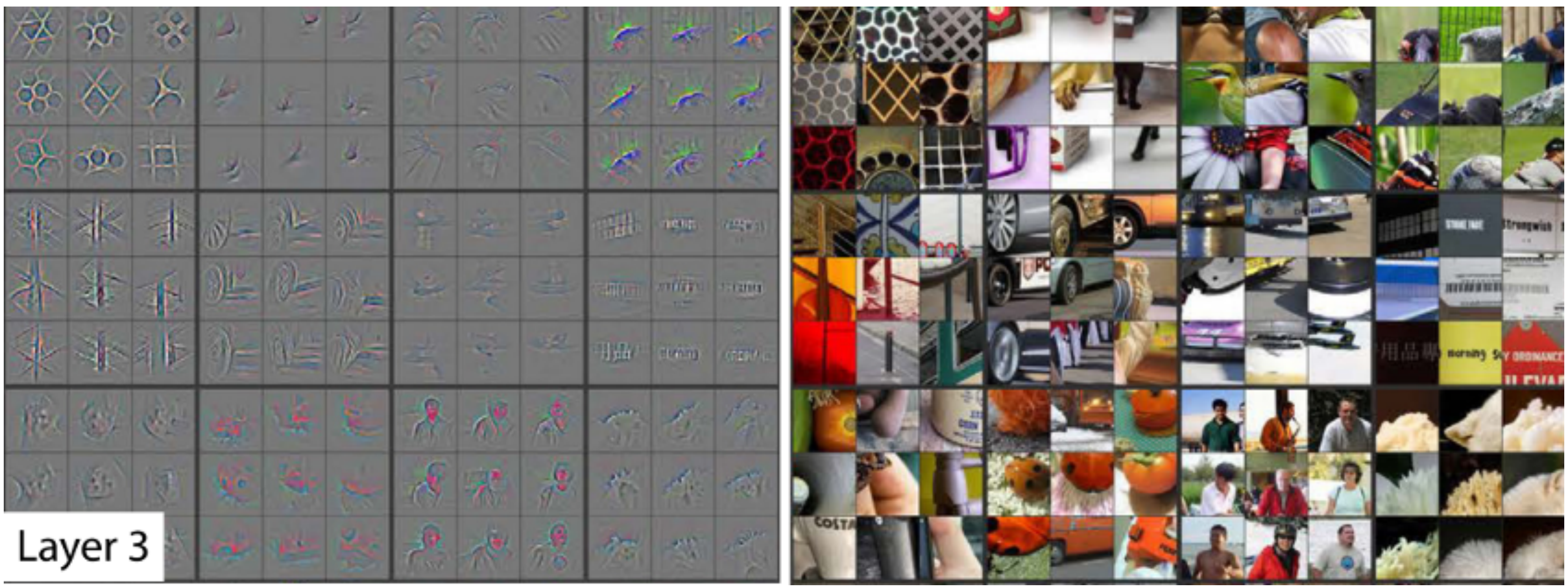


Image taken from [Zeiler14]

Lasagne

Specifying your network as mathematical expressions is powerful but low-level

Lasagne is a neural network library built on Theano

Makes building networks with Theano much easier

Provides API for:

constructing layers of a network

getting Theano expressions
representing output, loss, etc.

Lasagne is quite a thin layer on top of Theano, so understanding Theano is helpful

On the plus side, implementing custom layers, loss functions, etc is quite doable.

Intro to Theano and Lasagne slides:

https://speakerdeck.com/britefury

https://speakerdeck.com/britefury/intro-to-theano-and-lasagne-for-deep-learning

Notes for building and training neural networks

Neural network architecture (OxfordNet / VGG style)

#	Layer
	Input: 3 x 224 x 224 (RGB image, zero-mean)
1	64C3
2	64C3
	MP2
3	128C3
4	128C3
	MP2

64C3 = 3x3 conv, 64 filters
MP2 = max-pooling, 2x2

Early part

Blocks consisting of:

A few convolutional layers, often 3x3 kernels
- *followed by* -
Down-sampling; max-pooling or striding

#	Layer
	Input: 3 x 224 x 224 (RGB image, zero-mean)
1	64C3
2	64C3
	MP2
3	128C3
4	128C3
	MP2

Notation:

64C3
convolutional layer with 64 3x3 filters

MP2
max-pooling, 2x2

#	Layer
	Input: 3 x 224 x 224 (RGB image, zero-mean)
1	64C3
2	64C3
	MP2
3	128C3
4	128C3
	MP2

Note

after down-sampling, double the number of convolutional filters

#	Layer
	Input: 3 x 224 x 224 (RGB image, zero-mean)
1	64C3
2	64C3
	MP2
3	128C3
4	128C3
	MP2
	FC256
	FC10

Later part:

After blocks of convolutional and down-sampling layers:

Fully-connected (a.k.a. dense) layers

#	Layer
	Input: 3 x 224 x 224 (RGB image, zero-mean)
1	64C3
2	64C3
	MP2
3	128C3
4	128C3
	MP2
	FC256
	FC10

Notation:

FC256
fully-connected layer with 256 channels

#	Layer
	Input: 3 x 224 x 224 (RGB image, zero-mean)
1	64C3
2	64C3
	MP2
3	128C3
4	128C3
	MP2
	FC256
	FC10

Overall

Convolutional layers detect feature in various positions throughout the image

#	Layer
	Input: 3 x 224 x 224 (RGB image, zero-mean)
1	64C3
2	64C3
	MP2
3	128C3
4	128C3
	MP2
	FC256
	FC10

Overall

Fully-connected / dense layers use features detected by convolutional layers to produce output

Could also look at architectures developed by others, e.g. Inception by Google, or ResNets by Micrsoft for inspiration

Batch normalization

Batch normalization [Ioffe15] is recommended in most cases

Necessary for deeper networks (> 8 layers)

Speeds up training; cost drops faster per-epoch, although epochs take longer (~2x in my experience)

Can also reach lower error rates

Layers can magnify or shrink magnitudes of values. Multiple layers can result in exponential increase/decrease.

Batch normalisation maintains constant scale throughout network

Insert into convolutional and fully-connected layers

after matrix multiplication/convolution, *before* the non-linearity

Lasagne batch normalization inserts itself into a layer before the non-linearity, so its nice and easy to use:

```
lyr = lasagne.layers.batch_norm(lyr)
```

DropOut

Normally necessary for training (turned off at predict/test time)

Reduces over-fitting

Over-fitting is a well-known problem in machine learning, affects neural networks particularly

A model over-fits when it is very good at correctly predicting samples in training set but fails to generalise to samples outside it

DropOut [Hinton12]

During training, randomly choose units to ‘drop out’ by setting their output to 0, with probability P, usually around 0.5

(compensate by multiplying values by $\frac{1}{1-P}$)

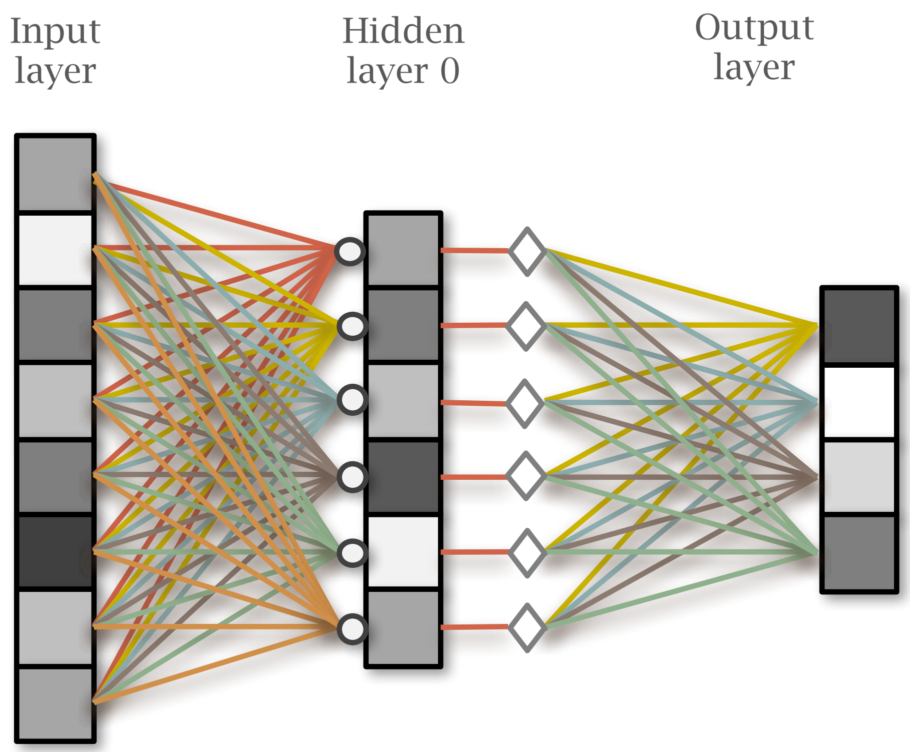
During test/predict:

Run as normal (DropOut turned off)

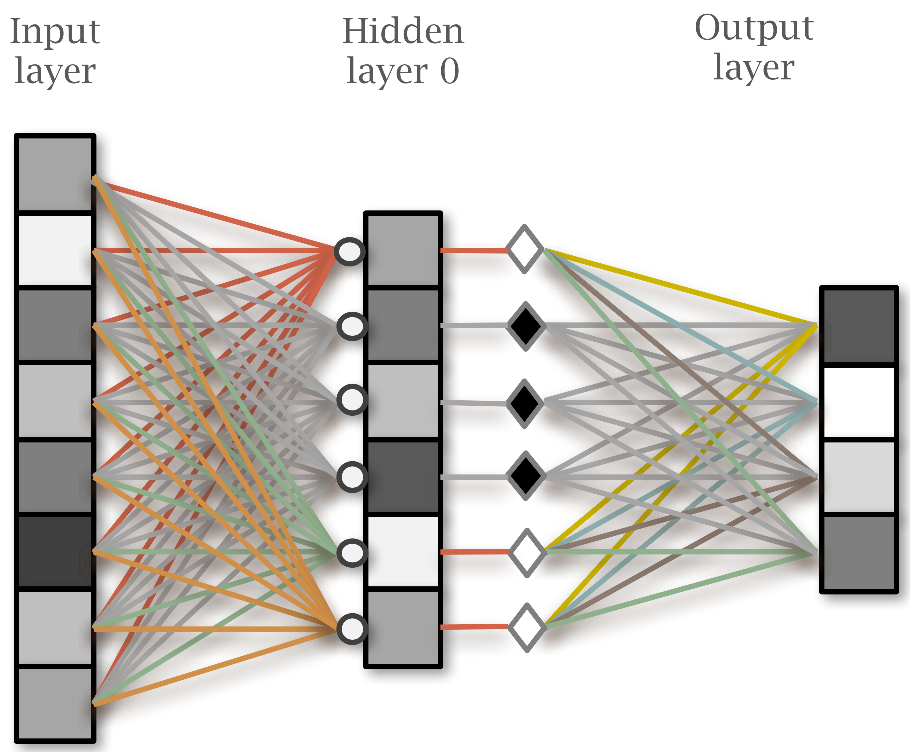
Normally applied after later, fully connected layers

```
lyr = lasagne.layers.DenseLayer(lyr, num_units=256)
lyr = lasagne.layers.DropoutLayer(lyr, p=0.5)
```

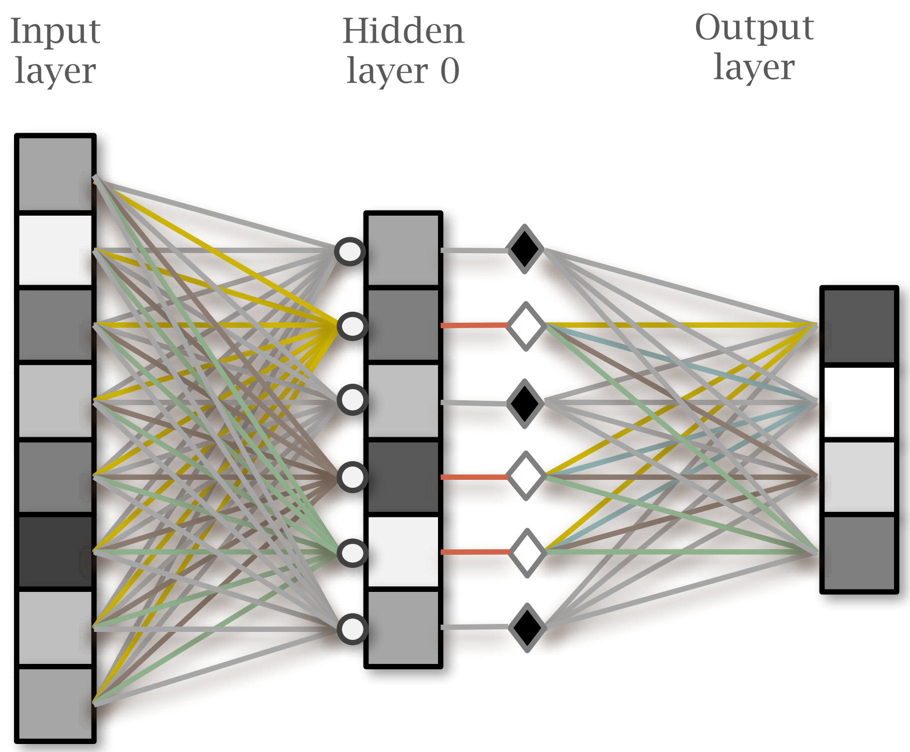
Dropout OFF



Dropout ON (1)



Dropout ON (2)



Turning on a different subset of units for each sample:

causes units to learn more robust features that cannot rely on the presence of other specific features to cover for flaws

Dataset augmentation

Reduce over-fitting by enlarging training set

Artificially modify existing training samples to make new ones

For images:

Apply transformations such as move, scale, rotate, reflect, etc.

Data standardisation

Neural networks train more effectively when training data has:

zero-mean
unit variance

Standardise input data

In case of regression, standardise output data too (don’t forget to invert the standardisation of network predictions!)

Standardisation

Extract samples into an array

In case of images, extract all pixels from all sampls, keeping R, G & B channels *separate*

Compute distribution and standardise

Either:

Zero the mean and scale std-dev to 1, per channel (RGB for images)

$$x' = \frac{x - \mu(x)}{\sigma(x)}$$

When training goes wrong and what to look for

Loss becomes `NaN`

(ensure you track the loss after each epoch so you can watch for this!)

Classification error rate equivalent of random guess (its not learning)

Learns to predict constant value;
optimises constant value for best loss

A constant value is a local minimum
that the network won’t get out of
(neural networks ‘cheat’ like crazy!)

Neural networks (most) often *DON'T* learn what you want or expect them to

Local minima will be the bane of your existence

Designing a computer vision pipeline

Simple problems may be solved with just a neural network

Not sufficient for more complex problems

(neural networks aren’t a silver bullet; don’t believe the hype)

Theoretically possible to use a single network for a complex problem

if you have *enough* training data (often an impractical amount)

For more complex problems, the problem should be broken down

Example

Identifying right whales, by Felix Lau
2nd place in Kaggle competition

http://felixlaumon.github.io/2015/01/08/kaggle-right-whale.html

Identifying right whales, by Felix Lau

The first naïve solution – training a classifier to identify individuals – did not work well

Region-based saliency map revealed that the network had ‘locked on’ to features in the ocean shape rather than the whales

Lau’s solution:

Train a localiser neural network to locate the whale in the image

Lau's solution:

Train a keypoint finder neural network to locate two keypoints on the whale's head to identify its orientation

Lau’s solution:

Train classifier neural network on oriented and cropped whale head images

OxfordNet / VGG and transfer learning

Using a pre-trained network

Use Oxford VGG-19; the 19-layer model

1000-class image classifier, trained on ImageNet

Can download CC licensed weights from (in Caffe format):
http://www.robots.ox.ac.uk/~vgg/research/very_deep/

GitHub repo contains code that downloads a Python version form:
http://s3.amazonaws.com/lasagne/recipes/pretrained/imagenet/vgg19.pkl

VGG models are simple but effective

Consist of:

3x3 convolutions
2x2 max pooling
fully connected

#	Layer
	Input: 3 x 224 x 224 (RGB image, zero-mean)
1	64C3
2	64C3
	MP2
3	128C3
4	128C3
	MP2
5	256C3
6	256C3
7	256C3
8	256C3
	MP2

#	Layer
9	512C3
10	512C3
11	512C3
12	512C3
	MP2
13	512C3
14	512C3
15	512C3
16	512C3
	MP2
17	FC4096 (dropout 50%)
18	FC4096 (dropout 50%)
19	FC1000 soft-max

Exercise / Demo

Classifying an image with VGG-19

Transfer learning (network re-use)

Training a neural network is notoriously data-hungry

Preparing training data with ground truths is expensive and time consuming

What if we don’t have enough training data to get good results?

The ImageNet dataset is huge; millions of images with ground truths

What if we could somehow use it to help us with a different task?

Good news:

we can!

Transfer learning

Re-use part (often most) of a pre-trained network for a new task

Example; can re-use part of VGG-19 net for:

Classifying images with classes that weren't part of the original ImageNet dataset

Example; can re-use part of VGG-19 net for:

Localisation (find location of object in image)

Segmentation (find exact boundary around object in image)

Transfer learning: how to

Take existing network such as VGG-19

#	Layer
	Input: 3 x 224 x 224 (RGB image, zero-mean)
1	64C3
2	64C3
	MP2
3	128C3
4	128C3
	MP2
5	256C3
6	256C3
7	256C3
8	256C3
	MP2

#	Layer
9	512C3
10	512C3
11	512C3
12	512C3
	MP2
13	512C3
14	512C3
15	512C3
16	512C3
	MP2
17	FC4096 (drop 50%)
18	FC4096 (drop 50%)
19	FC1000 soft-max

Remove last layers e.g. the fully-connected ones

(just 17,18,19; those in the left box are hidden here for brevity!)

#	Layer
9	512C3
10	512C3
11	512C3
12	512C3
	MP2
13	512C3
14	512C3
15	512C3
16	512C3
	MP2

Build new randomly initialise layers to replace them

(the number of layers created their size is only for illustration here)

#	Layer
9	512C3
10	512C3
11	512C3
12	512C3
	MP2
13	512C3
14	512C3
15	512C3
16	512C3
	MP2
17	FC1024 (drop 50%)
18	FC21 soft-max

Transfer learning: training

Train the network with your training data, only learning parameters for the *new* layers

Transfer learning: fine-tuning

After learning parameters for the *new* layers, fine-tune by learning parameters for the whole network to get better accuracy

Result

Nice shiny network with good performance that was trained with *much less* of *our* training data

Some cool work in the field that might be of interest

Visualizing and understanding convolutional networks [Zeiler14]

Visualisations of responses of layers to images

Visualizing and understanding convolutional networks [Zeiler14]

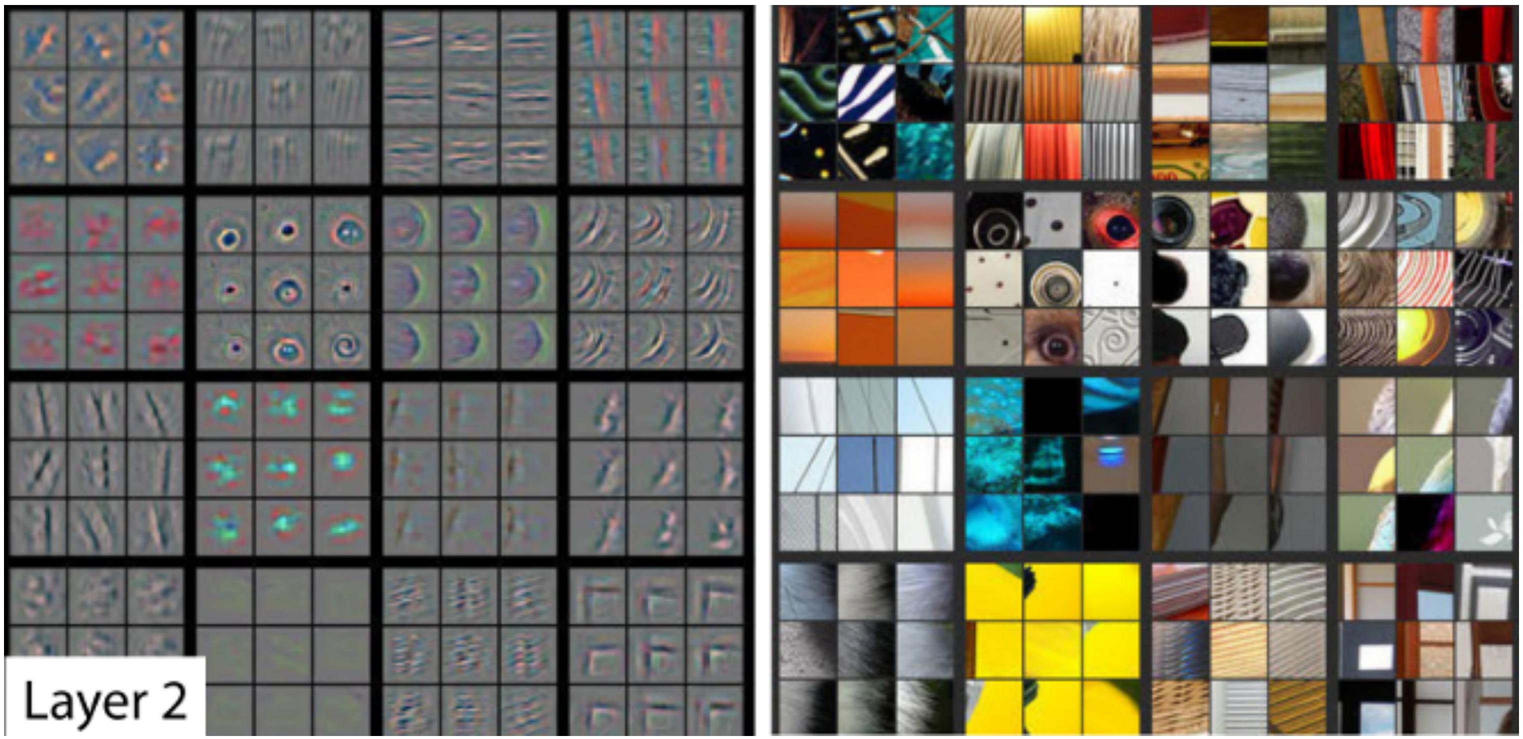


Image taken from [Zeiler14]

Visualizing and understanding convolutional networks [Zeiler14]

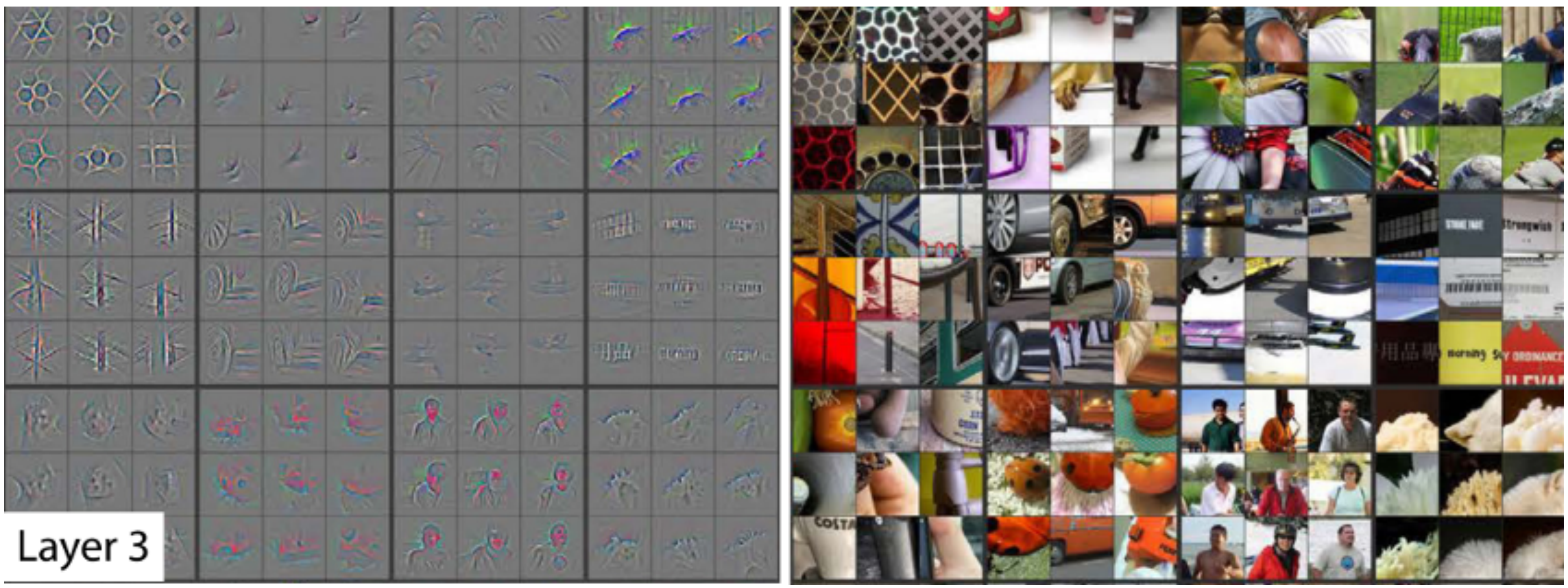


Image taken from [Zeiler14]

Deep Neural Networks are Easily Fooled: High Confidence Predictions in Recognizable Images [Nguyen15]

Generate images that are unrecognizable to human eyes but are recognized by the network

Deep Neural Networks are Easily Fooled: High Confidence Predictions in Recognizable Images [Nguyen15]

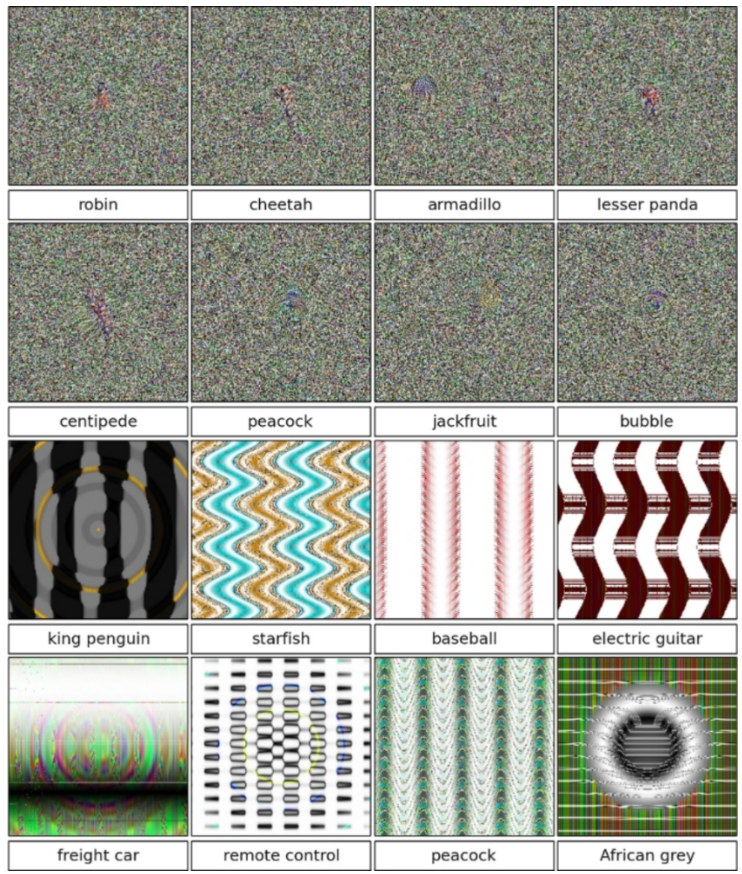


Image taken from [Nguyen15]

Learning to generate chairs with convolutional neural networks [Dosovitskiy15]

Network in reverse; orientation, design colour, etc parameters as input, rendered images as output training images

Learning to generate chairs with convolutional neural networks [Dosovitskiy15]

Image taken from [Dosovitskiy15]

A Neural Algorithm of Artistic Style [Gatys15]

Take an OxfordNet model [Simonyan14] and extract texture features from one of the convolutional layers, given a target style / painting as input
Use gradient descent to iterate photo – *not weights* – so that its texture features match those of the target image.

A Neural Algorithm of Artistic Style [Gatys15]

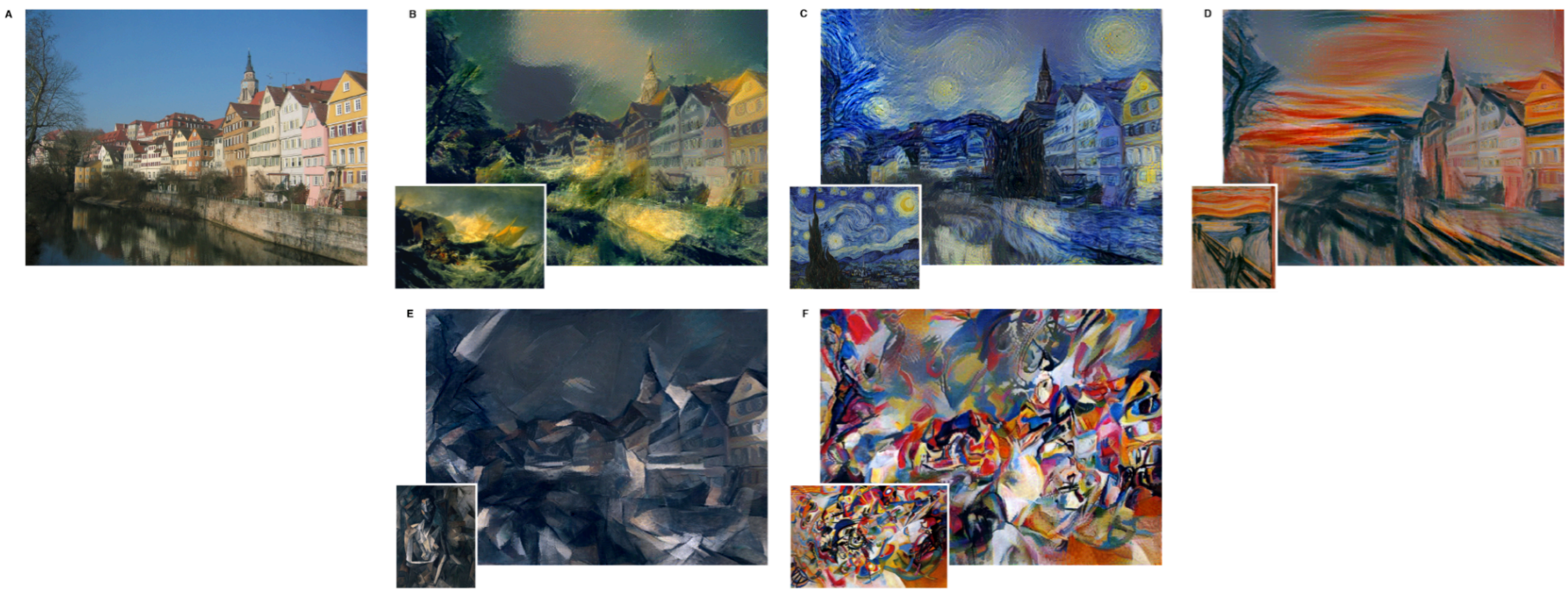


Image taken from [Gatys15]

Unsupervised representation Learning with Deep Convolutional Generative Adversarial Nets [Radford 15]

Train two networks; one given random parameters to generate an image, another to discriminate between a generated image and one from the training set

Generative Adversarial Nets [Radford15]

Images of bedrooms generated using
neural net
Image taken from [Radford15]

Generative Adversarial Nets [Radford15]

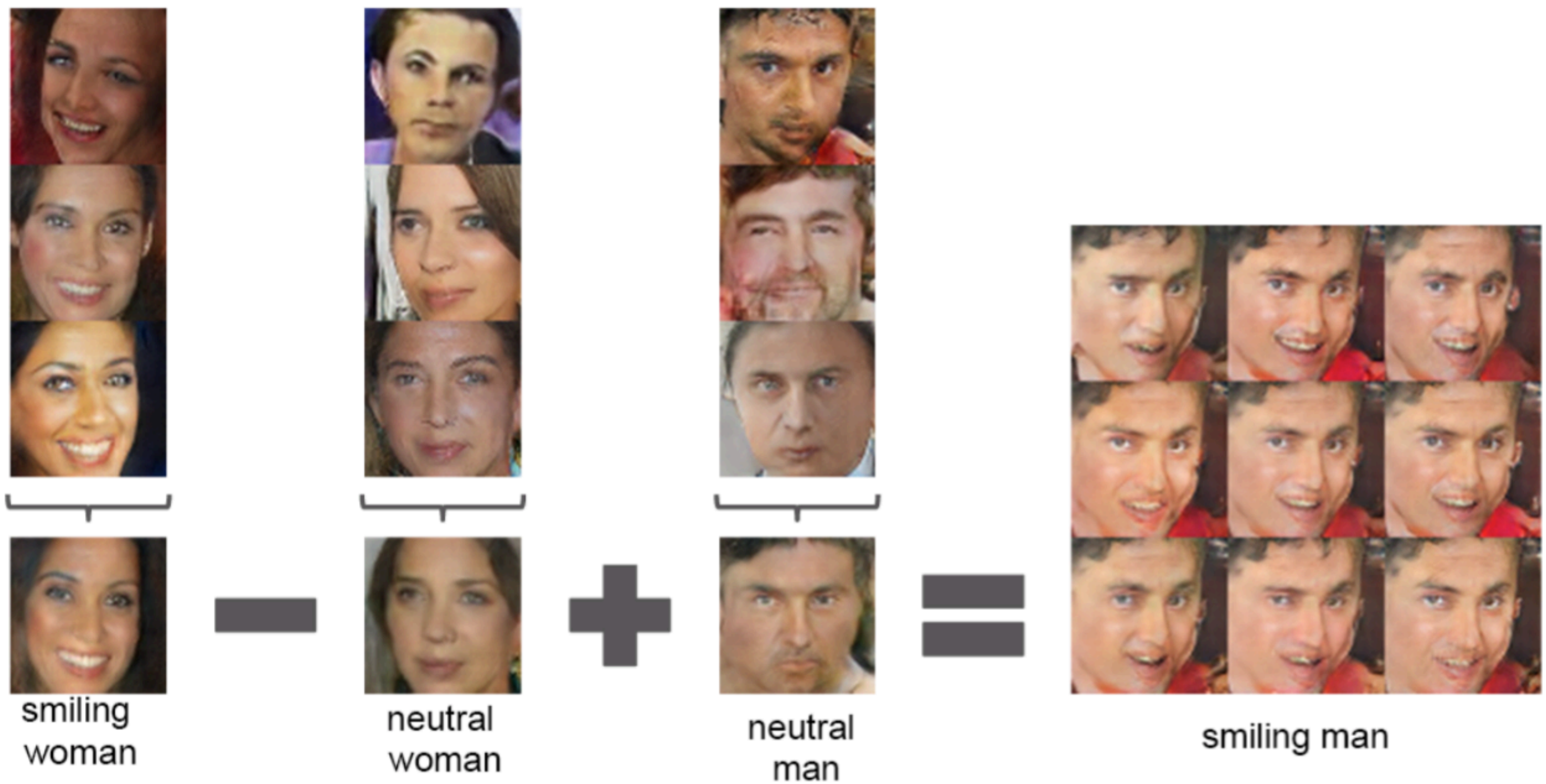


Image taken from [Radford15]

Hope you’ve found it helpful!

Thank you!

References

[Dosovitskiy15] Dosovitskiy, Springenberg and Box; *Learning to generate chairs with convolutional neural networks*, arXiv preprint, 2015

[Gatys15] Gatys, Echer, Bethge; *A Neural Algorithm of Artistic Style*, arXiv: 1508.06576, 2015

[He15a] He, Zhang, Ren and Sun; *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*, arXiv 2015

[He15b] He, Kaiming, et al. "Deep Residual Learning for Image Recognition." *arXiv preprint arXiv:1512.03385* (2015).

[Hinton12] G.E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever and R. R. Salakhutdinov; Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580, 2012.

[Ioffe15] Ioffe, S.; Szegedy C.. (2015). “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". *ICML 2015, arXiv:1502.03167*

[Jones87] Jones, J.P.; Palmer, L.A. (1987). "An evaluation of the two-dimensional gabor filter model of simple receptive fields in cat striate cortex". *J. Neurophysiol* **58** (6): 1233–1258

[Lin13] Lin, Min, Qiang Chen, and Shuicheng Yan. "Network in network." *arXiv preprint arXiv:1312.4400* (2013).

[Nesterov83] Nesterov, Y. A method of solving a convex programming problem with convergence rate O(1/sqr(k)). *Soviet Mathematics Doklady, 27:372–376* (1983).

[Radford15] Radford, Metz, Chintala; *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*, arXiv:1511.06434, 2015

[Sutskever13] Sutskever, Ilya, et al. On the importance of initialization and momentum in deep learning. *Proceedings of the 30th international conference on machine learning (ICML-13)*. 2013.

[Simonyan14] K. Simonyan and Zisserman; *Very deep convolutional networks for large-scale image recognition*, arXiv:1409.1556, 2014

[Wang14] Wang, Dan, and Yi Shang. "A new active labeling method for deep learning."*Neural Networks (IJCNN), 2014 International Joint Conference on*. IEEE, 2014.

[Zeiler14] Zeiler and Fergus; *Visualizing and understanding convolutional networks*, Computer Vision - ECCV 2014