Lecture 5 - Ensemble methods, neural nets and deep learning

See https://github.com/jbrinchmann/MLD2023 as usual

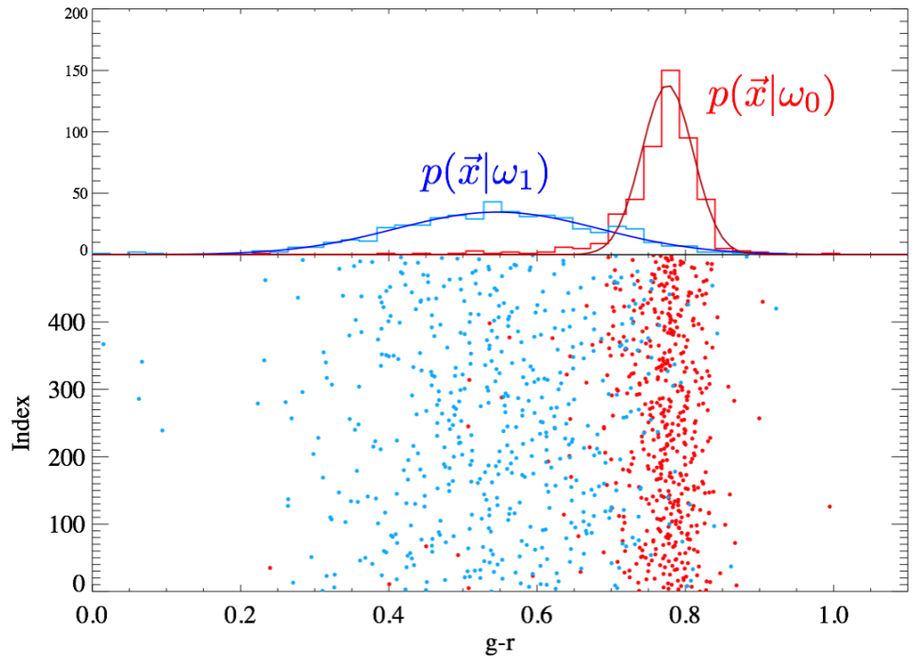
Lectures/Lecture 5 has the PDF for today

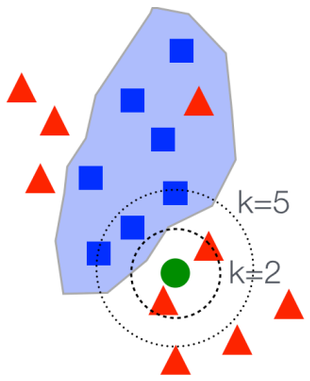
Lectures/Lecture 5/Notebooks has a number of example notebooks

Last lecture

Bayesian classification (naïve Bayes):

$$p(\omega_0|\vec{x}) > p(\omega_1|\vec{x}) \Rightarrow \text{Class 0 (elliptical)}$$
$$p(\omega_0|\vec{x}) < p(\omega_1|\vec{x}) \Rightarrow \text{Class 1 (spiral)}$$





k-nearest neighbours (classification or regression)

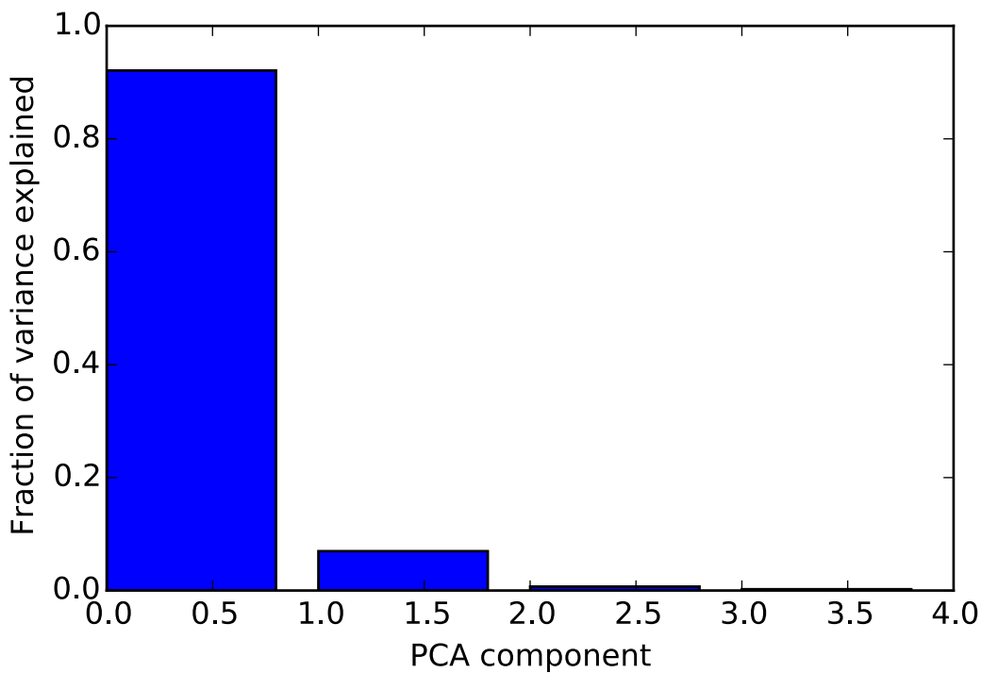
Standardizing data:

$$x_i \rightarrow \frac{x_i - \bar{x}}{\sigma_x}$$

Last lecture

Principal Component Analysis

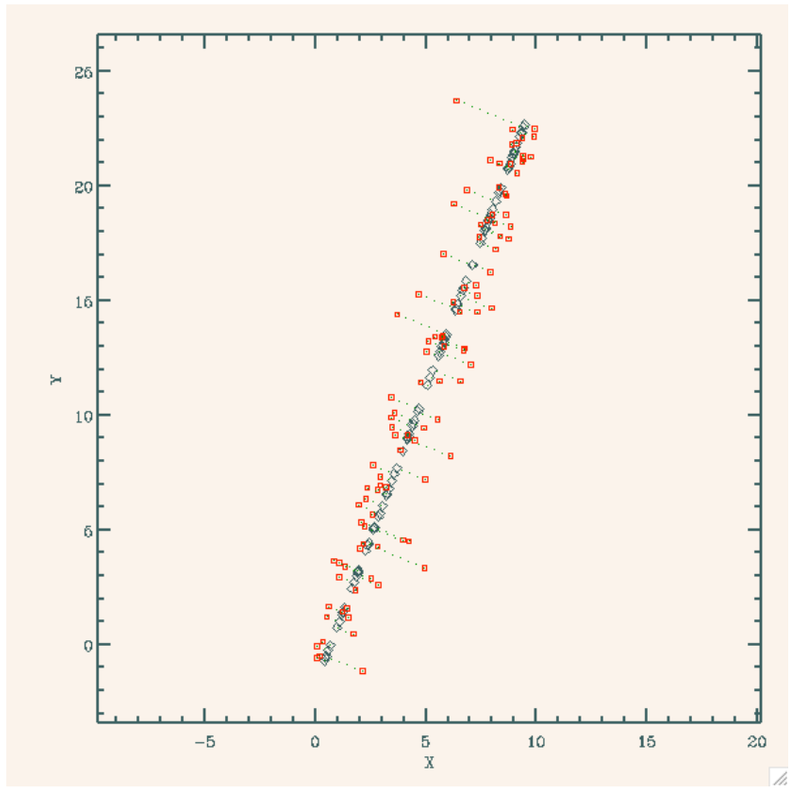
Finds directions where data vary the most, can be very useful for feature selection by looking at which components explain most of the variance

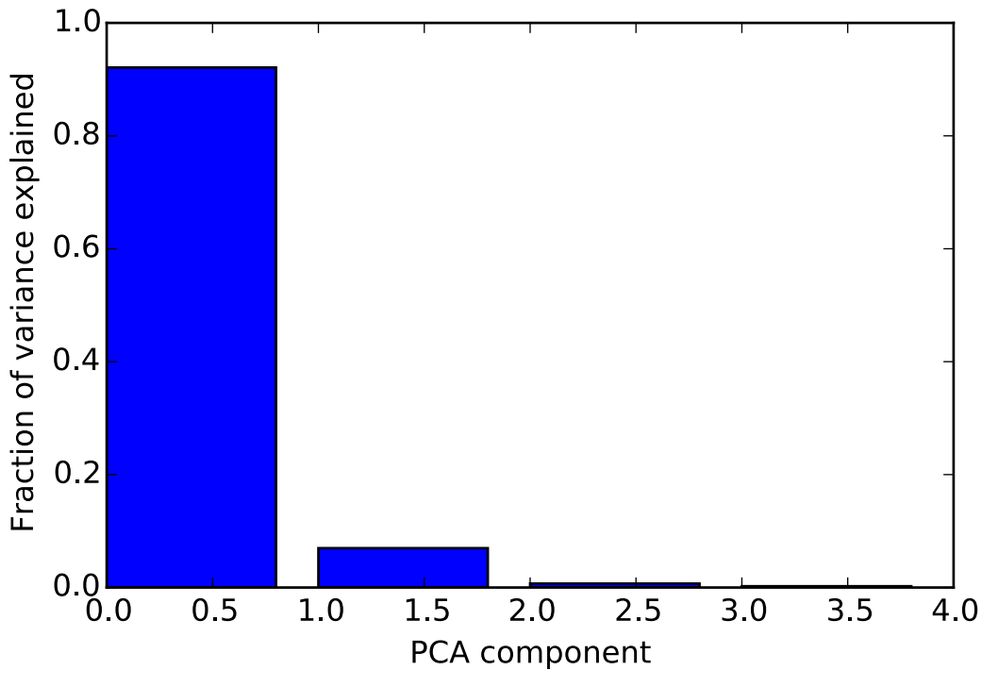


Last lecture

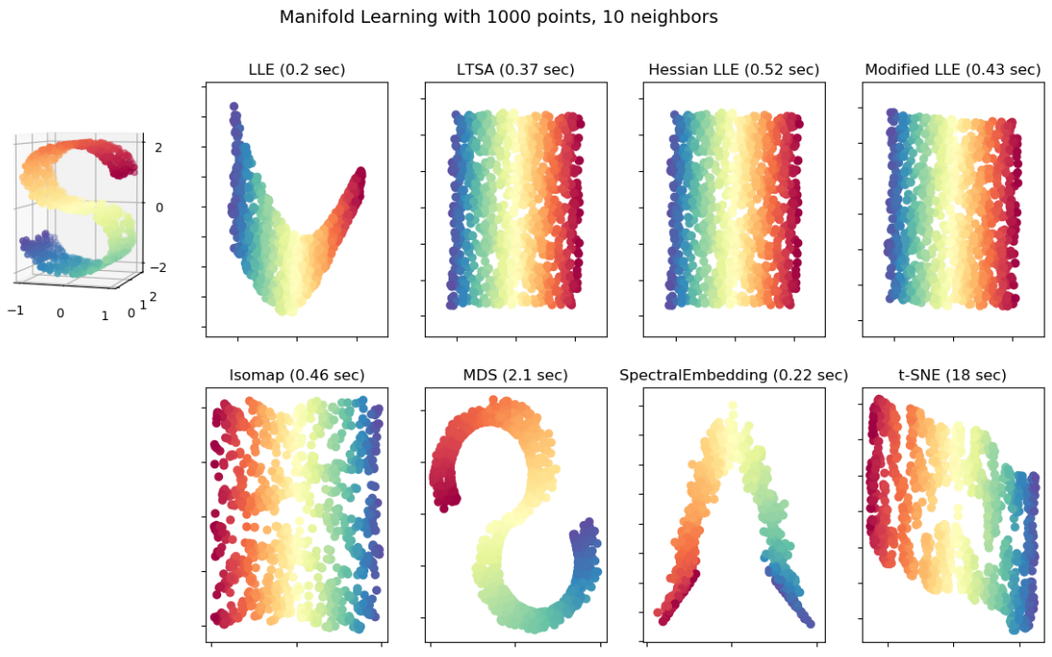
Principal Component Analysis

Finds directions where data vary the most, can be very useful for feature selection by looking at which components explain most of the variance

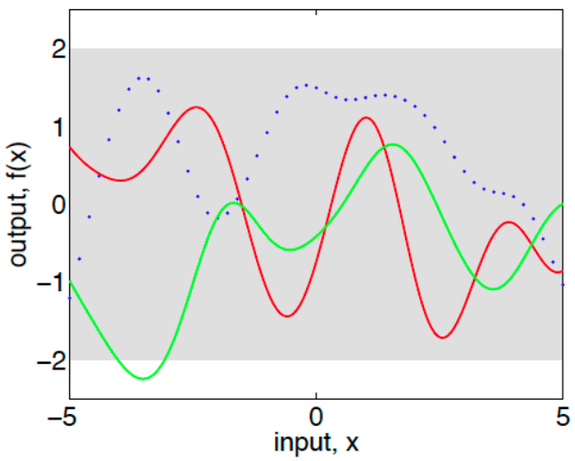




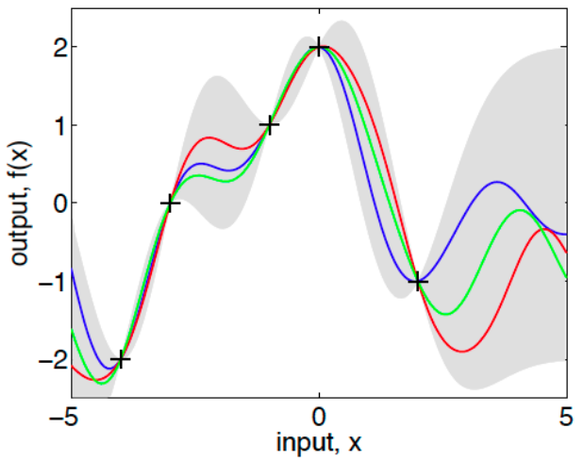
Last lecture



The zoo of the manifold learners



(a), prior



(b), posterior

Gaussian process regression

Some useful resources:

Very nice interactive examples + written explanation:

https://distill.pub/2019/visual-exploration-gaussian-processes/

A tidy and simple introduction with a pedagogic set of Python examples - and not just scikit-learn:

https://www.dominodatalab.com/blog/fitting-gaussian-process-models-python

The main textbook on Gaussian Processes (Rasmussen & Williams 2006) can be found here (with other resources):

http://gaussianprocess.org/

Neural networks

Literature:
Goodfellow et al (2016), “Deep Learning”

MIT course (2024): http://introtodeeplearning.com/

(with excellent notebooks at https://github.com/aamini/introtodeeplearning/)

Going from linear to non-linear

See Goodfellow et al, chap 6

$$y_i = \sum_j \alpha_j \phi_j(x_i)$$

Going from linear to non-linear

See Goodfellow et al, chap 6

$$y_i = \sum_j \alpha_j \phi_j(x_i)$$

1. If ϕ is generic and high-dimensional, this can fit training data extremely well, but generalisation tends to be poor.

Going from linear to non-linear

See Goodfellow et al, chap 6

$$y_i = \sum_j \alpha_j \phi_j(x_i)$$

1. If ϕ is generic and high-dimensional, this can fit training data extremely well, but generalisation tends to be poor.

2. By hand-crafting ϕ you can get good performance, but it can be extremely time-consuming.

Going from linear to non-linear

See Goodfellow et al, chap 6

$$y_i = \sum_j \alpha_j \phi_j(x_i)$$

1. If ϕ is generic and high-dimensional, this can fit training data extremely well, but generalisation tends to be poor.

2. By hand-crafting ϕ you can get good performance, but it can be extremely time-consuming.

3. In deep learning, the focus is on learning ϕ from the training data, so in practice we have a function

$$y = f(\mathbf{x}; \mathbf{w}, \boldsymbol{\theta}) = \phi(\mathbf{x}; \boldsymbol{\theta})^T \mathbf{w}$$

so we learn the function ϕ but with a more restricted set of functions than in 1. The weights $\mathbf{w}$ are then used to project down to the space we want the predictions in.

Types of tasks for a DL algorithm

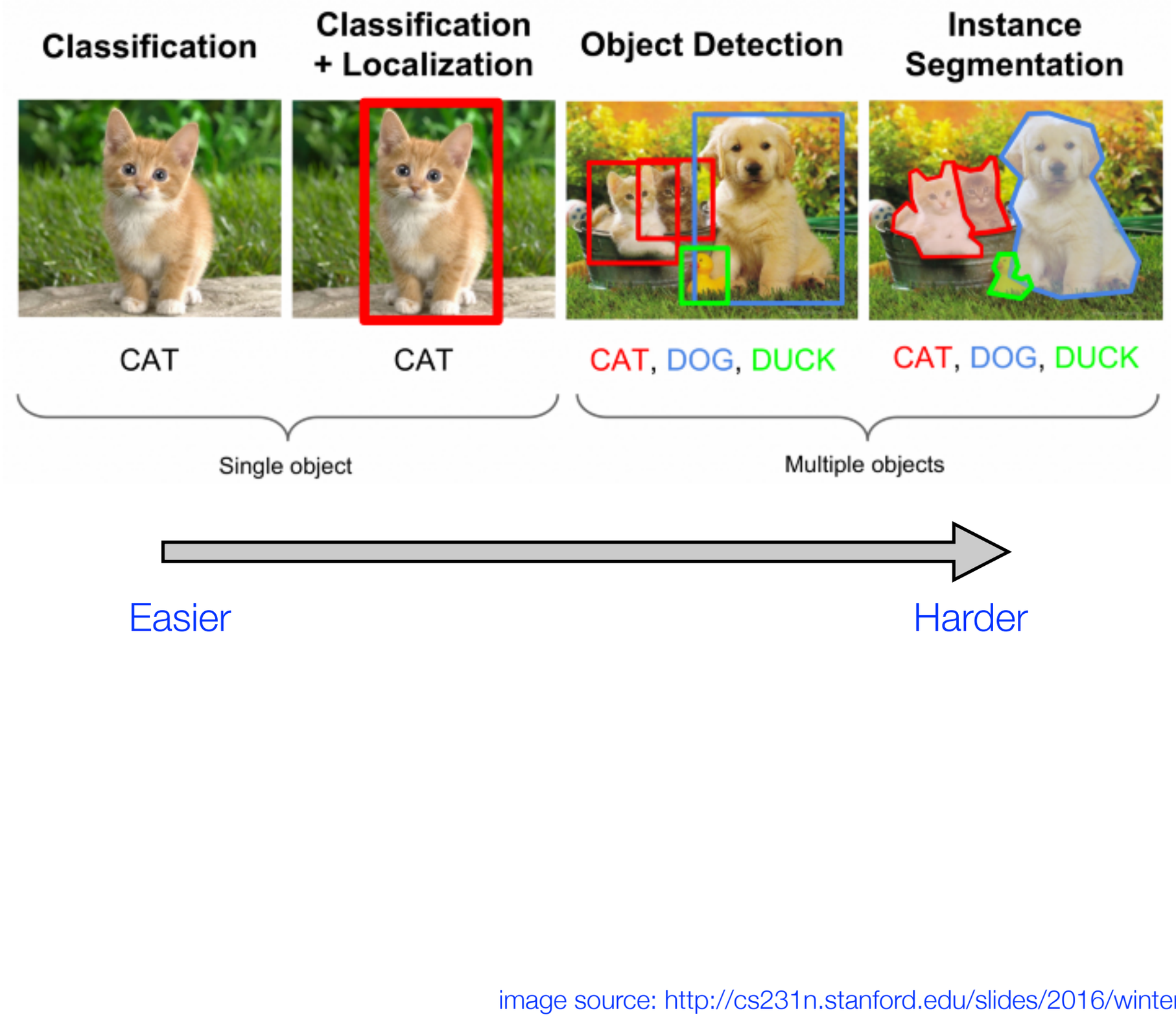
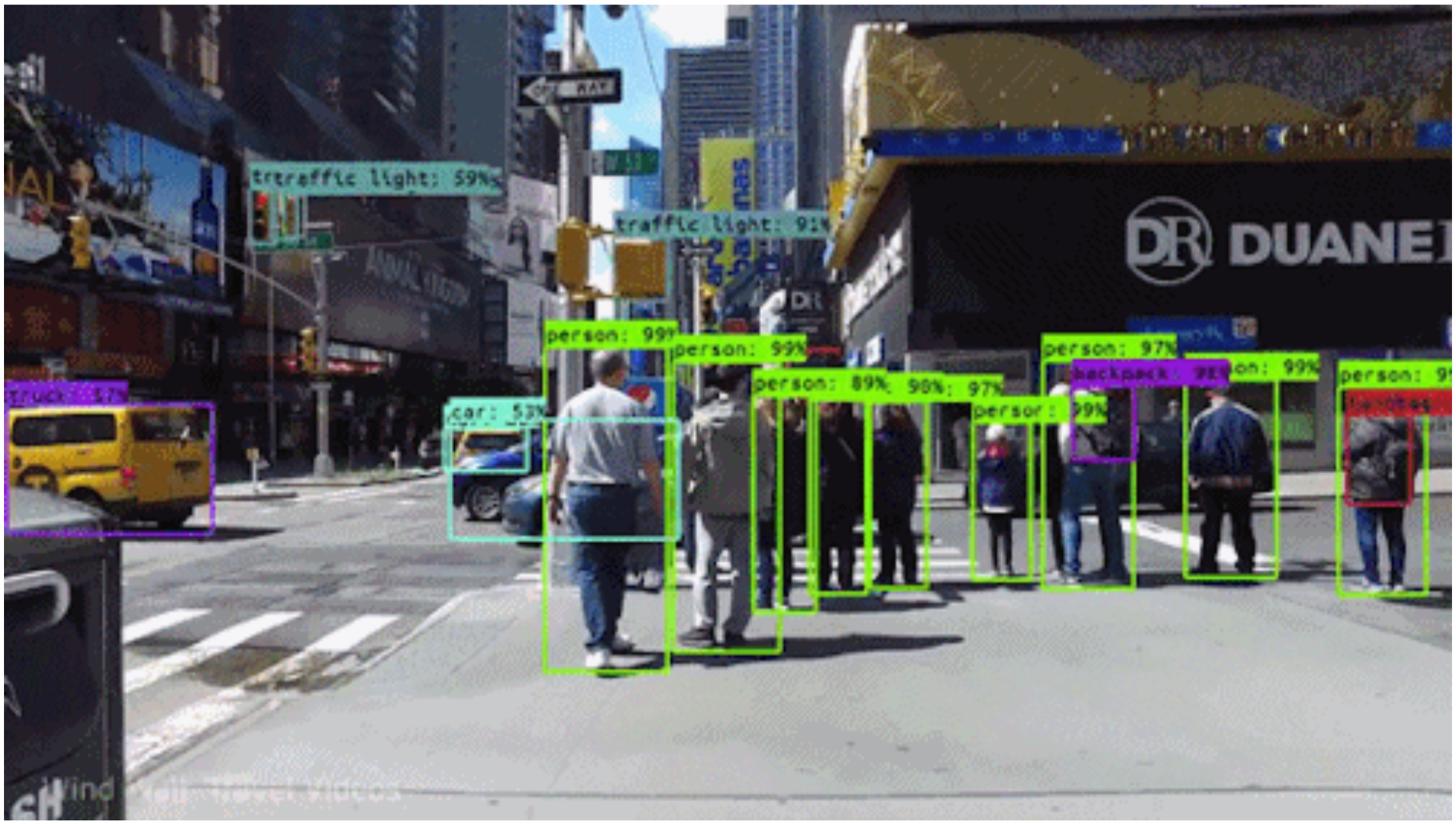


image source: http://cs231n.stanford.edu/slides/2016/winter1516_lecture8.pdf

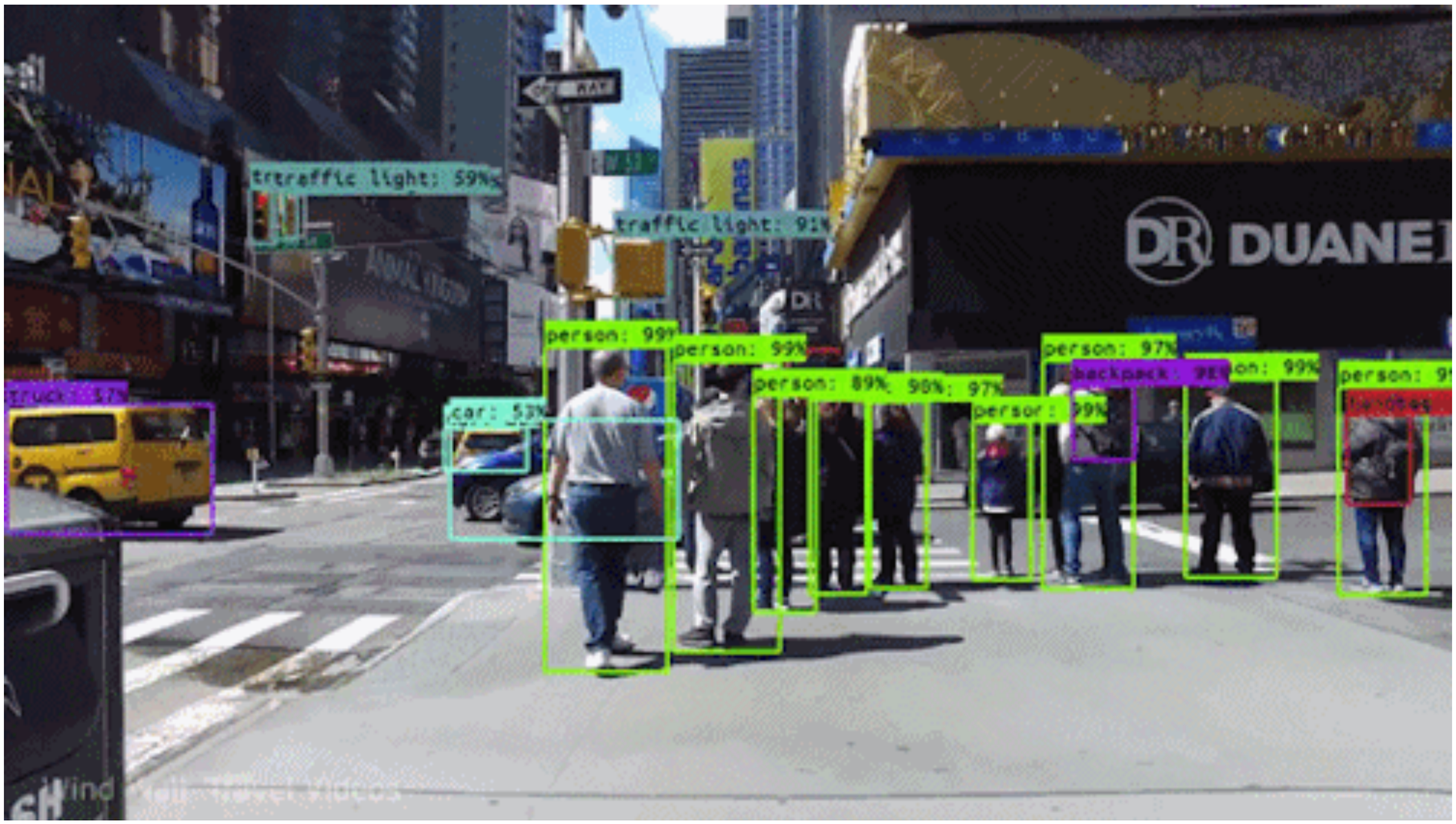
Deep learning - object detection



Accuracy matters… and it needs to be done quickly

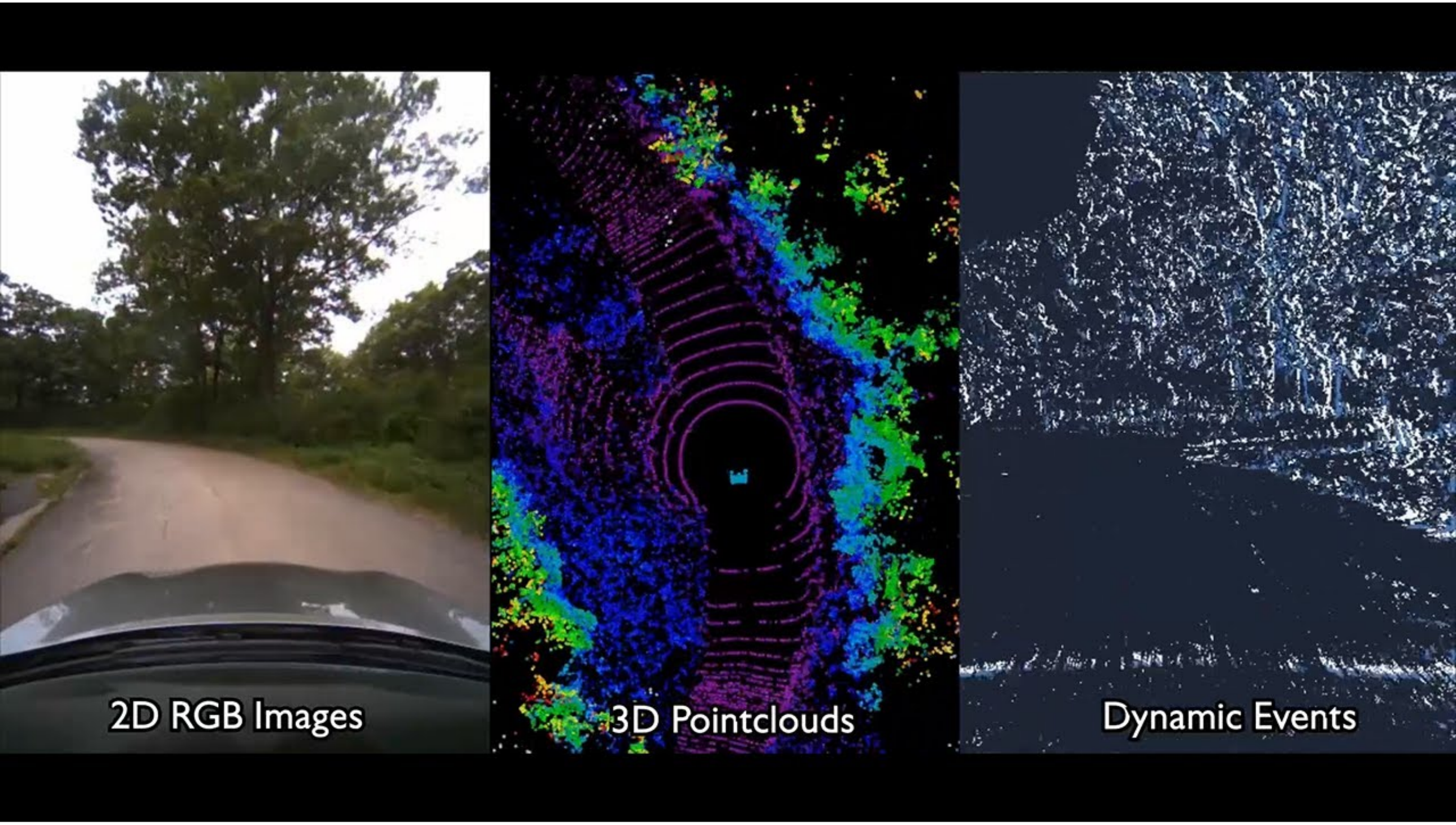
https://storage.googleapis.com/openimages/web/index.html

Deep learning - object detection

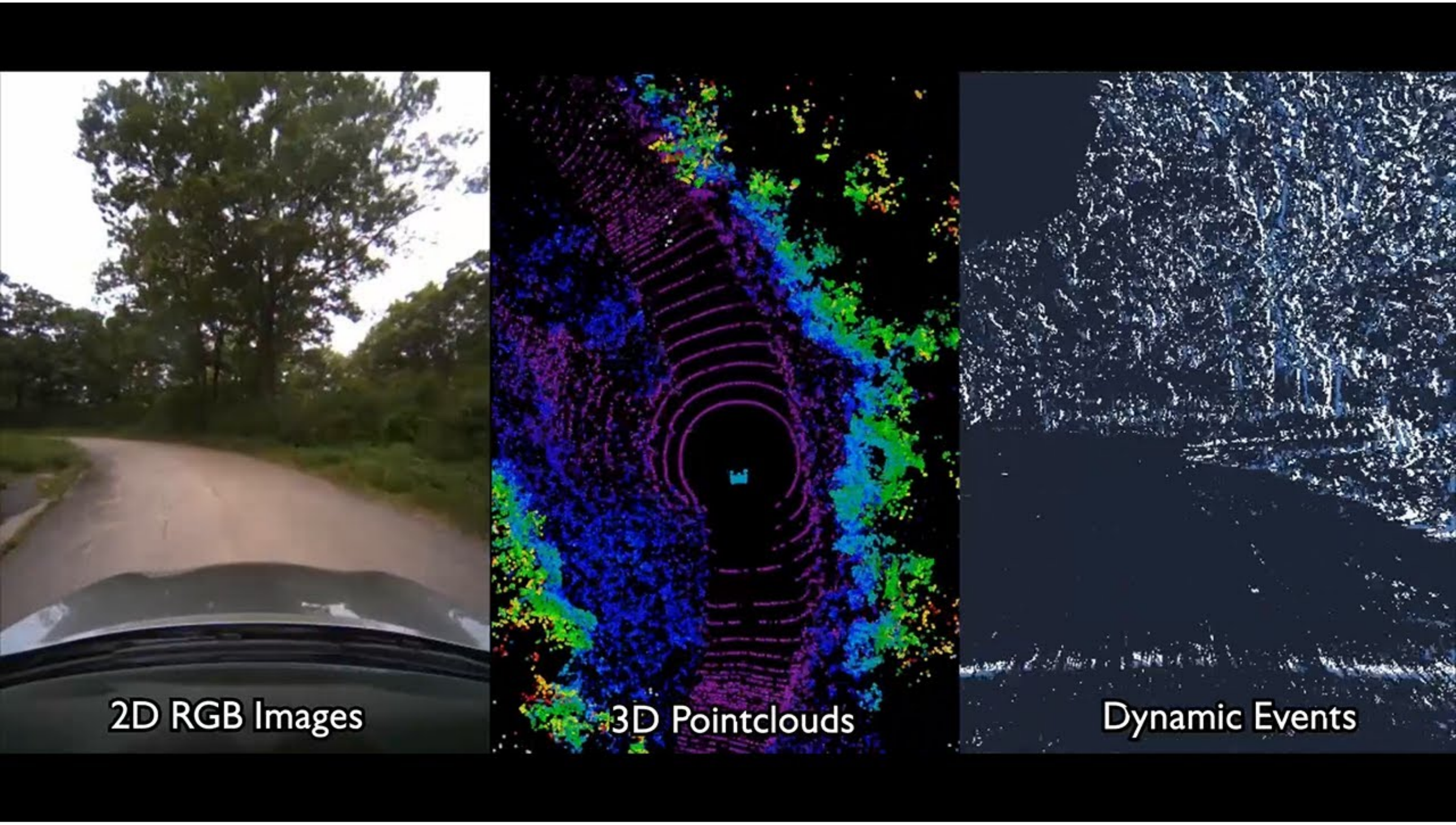


Accuracy matters… and it needs to be done quickly

https://storage.googleapis.com/openimages/web/index.html



VISTA - real life to simulated data - https://vista.csail.mit.edu/



VISTA - real life to simulated data - https://vista.csail.mit.edu/

Deep learning - what do we want?

What do we want?

$$P(\text{object} \mid \text{image})$$

The neural network will be our model and we need to train this using N examples. Since the dimension of the data can be large, this can be a very high-D PDF.

The trick then is to efficiently train this.

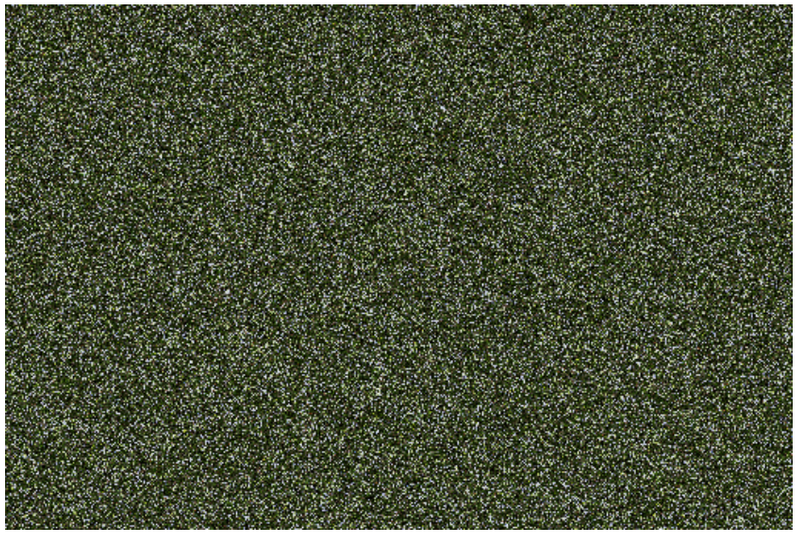
Manifold learning & deep learning

It seems that real images/data do not fully span this high-D space though:

D= 174592

Random resampling

D= 174592



Instead real images span only a small fraction of the full space - typically viewed as a manifold where images change smoothly.

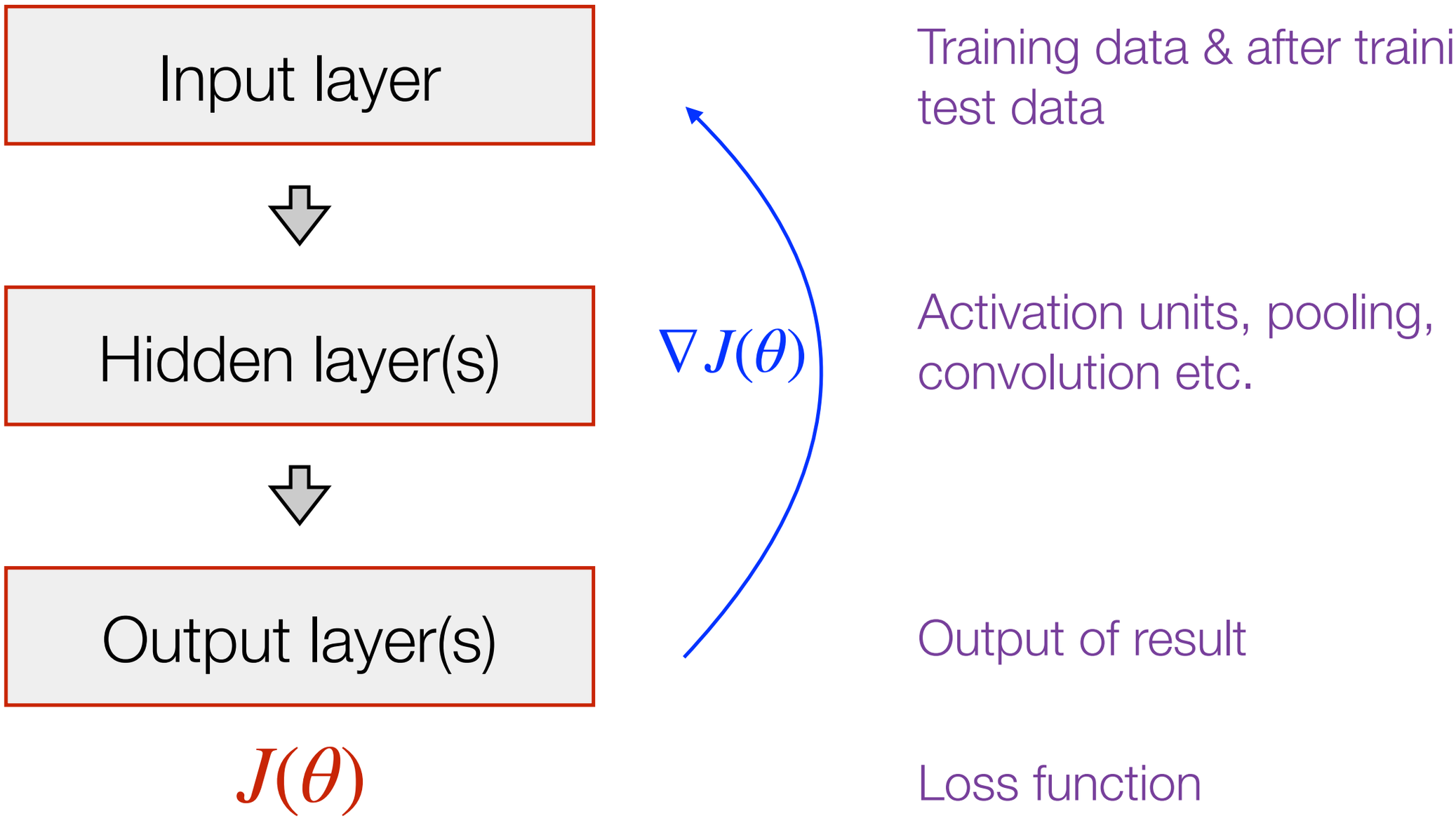
Manifold learning & deep learning

Instead real images span only a small fraction of the full space - typically viewed as a manifold where images change smoothly.

QMUL Multiview Face Dataset - http://www.eecs.qmul.ac.uk/~sgg/QMUL_FaceDataset/

Learning the PDF

The challenge is learning this structure - in deep learning we use a (neural) network (feed-forward network)

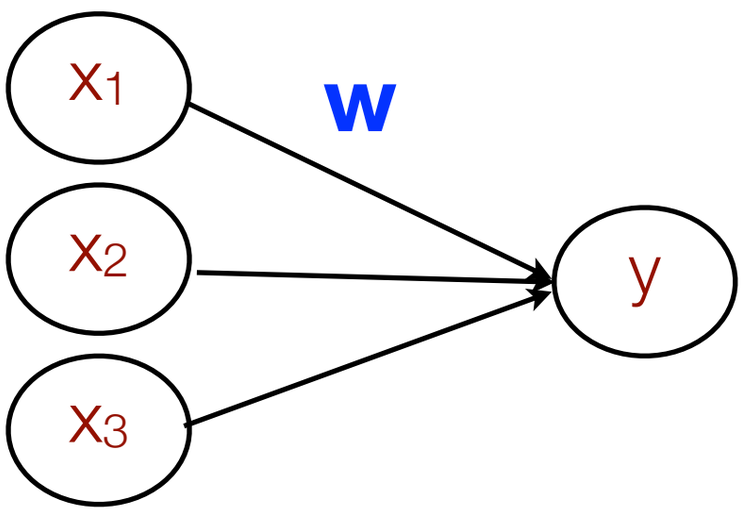


Linear Models - on our way to non-linearity

We can predict a value of a parameter y using a linear combination of an **x** vector & a set of basis functions $\boldsymbol{\phi}_i$:

$$y(\mathbf{x}, \mathbf{w}) = \sum_i w_i \phi_i(\mathbf{x})$$

This then represents a *transformation* of the input data **x**. And we can represent this as:



(if our x has 3 elements)

The simplest Neural Network

Now expand our previous method and create M different linear combinations of **x**, ie. create M different weight vectors $\mathbf{w}_j$:

$$z_j = h\left(\sum_i w_{i,j} x_i\right)$$

The h function is called the **activation function** in neural network contexts.

Activation functions

What function can we use?

Simplest: linear, but then the whole system is linear!

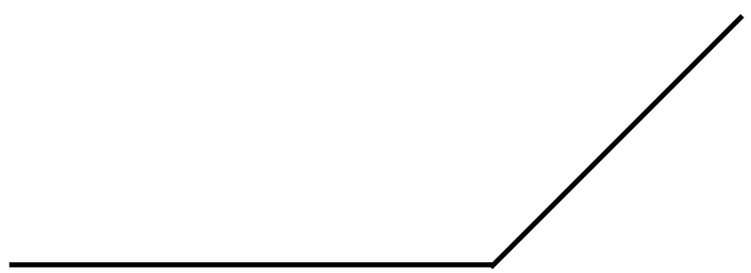
Activation functions

What function can we use?

Simplest: linear, but then the whole system is linear!

So it must be non-linear, but we would like it to behave much like a linear function. The modern choice is the rectified linear unit:

ReLU: $h(z) = \max\{0, z\}$

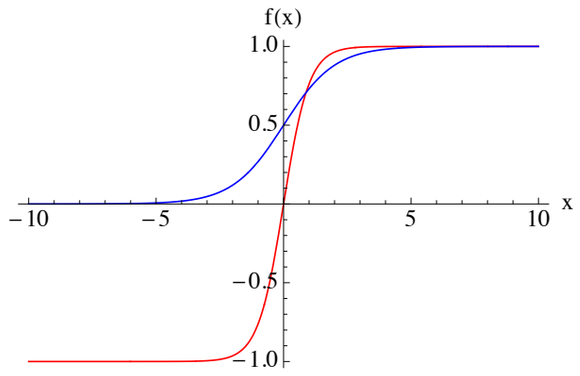


Activation functions

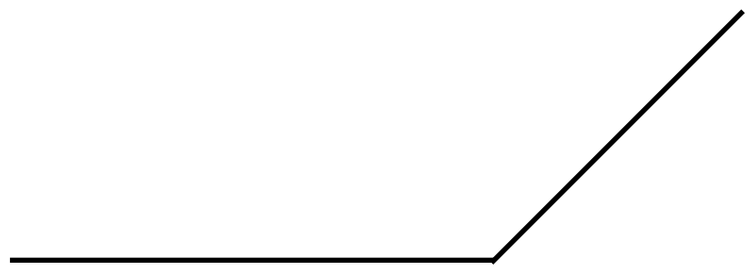
Why this form?

The issue is training: to train we need to calculate gradients

For this to be efficient, we prefer functions whose gradients are well away from zero.

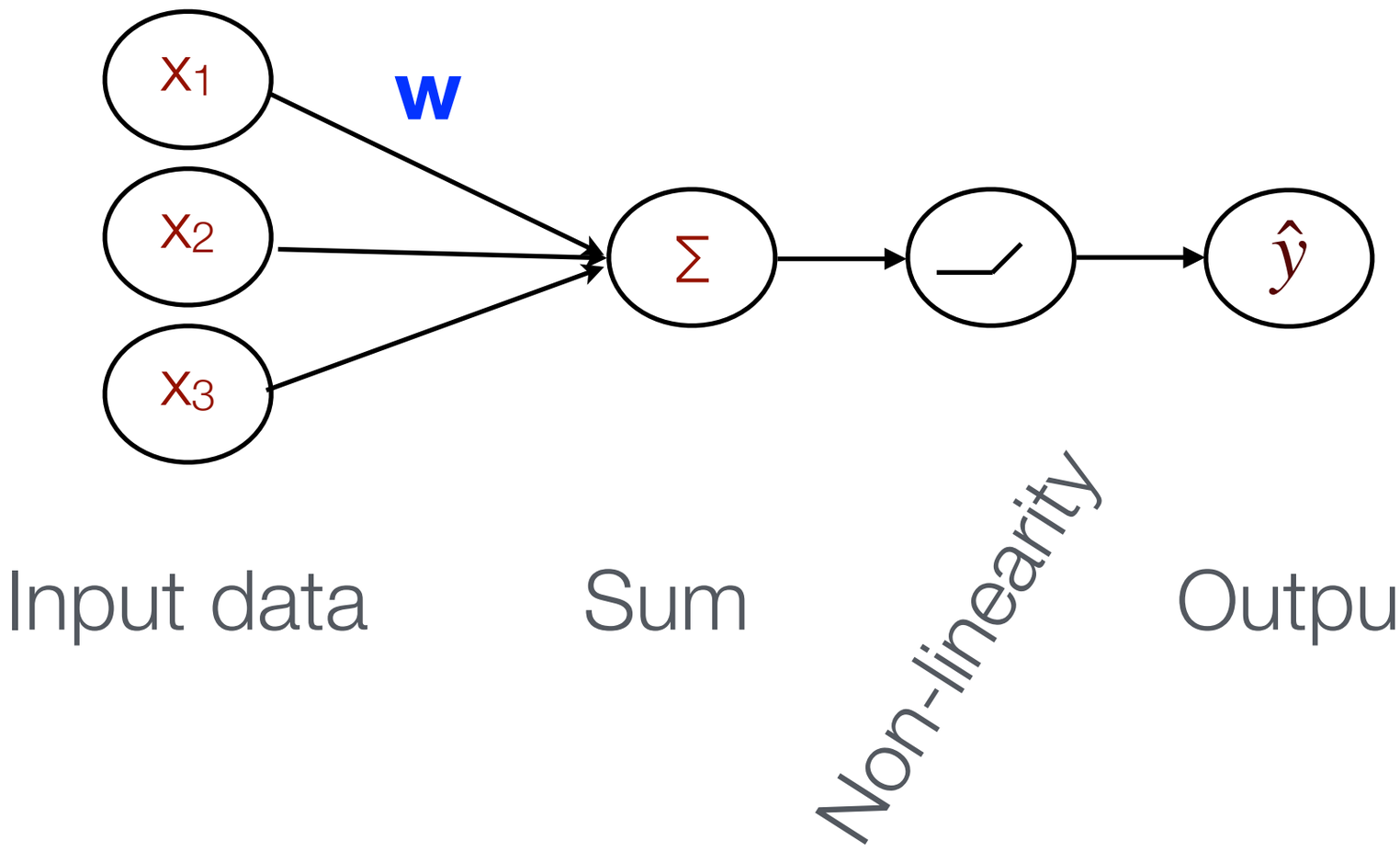


Less good - saturates



Good with start ≠ 0

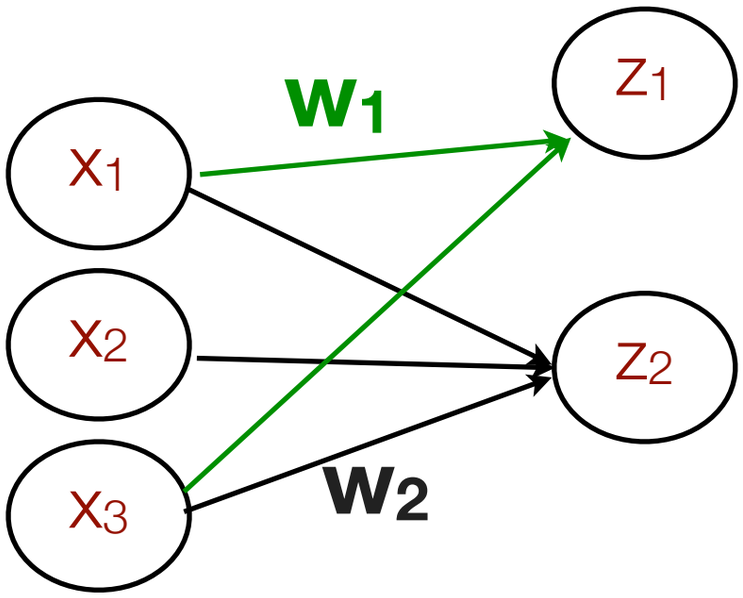
The perceptron (modern view)



The simplest Neural Network

Now expand our previous method and create M different linear combinations of **x**, ie. create M different weight vectors $\mathbf{w}_j$:

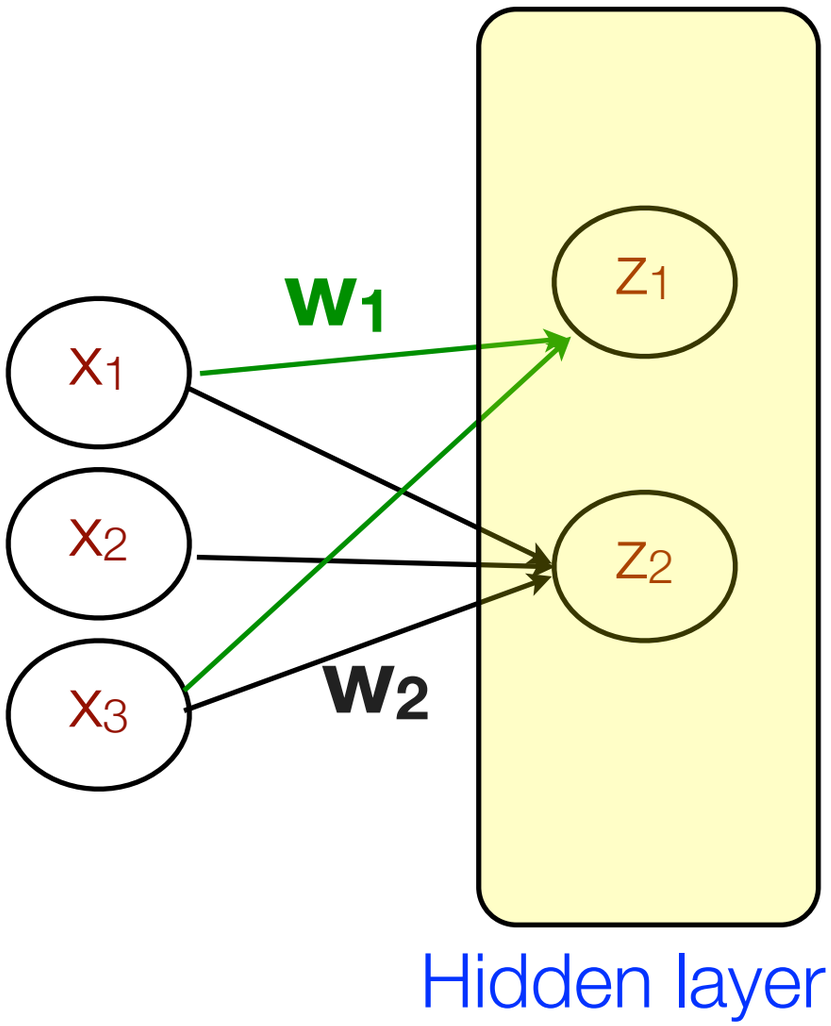
$$z_j = h\left(\sum_i w_{i,j} x_i\right)$$



The simplest Neural Network

Now expand our previous method and create M different linear combinations of **x**, ie. create M different weight vectors $\mathbf{w}_j$:

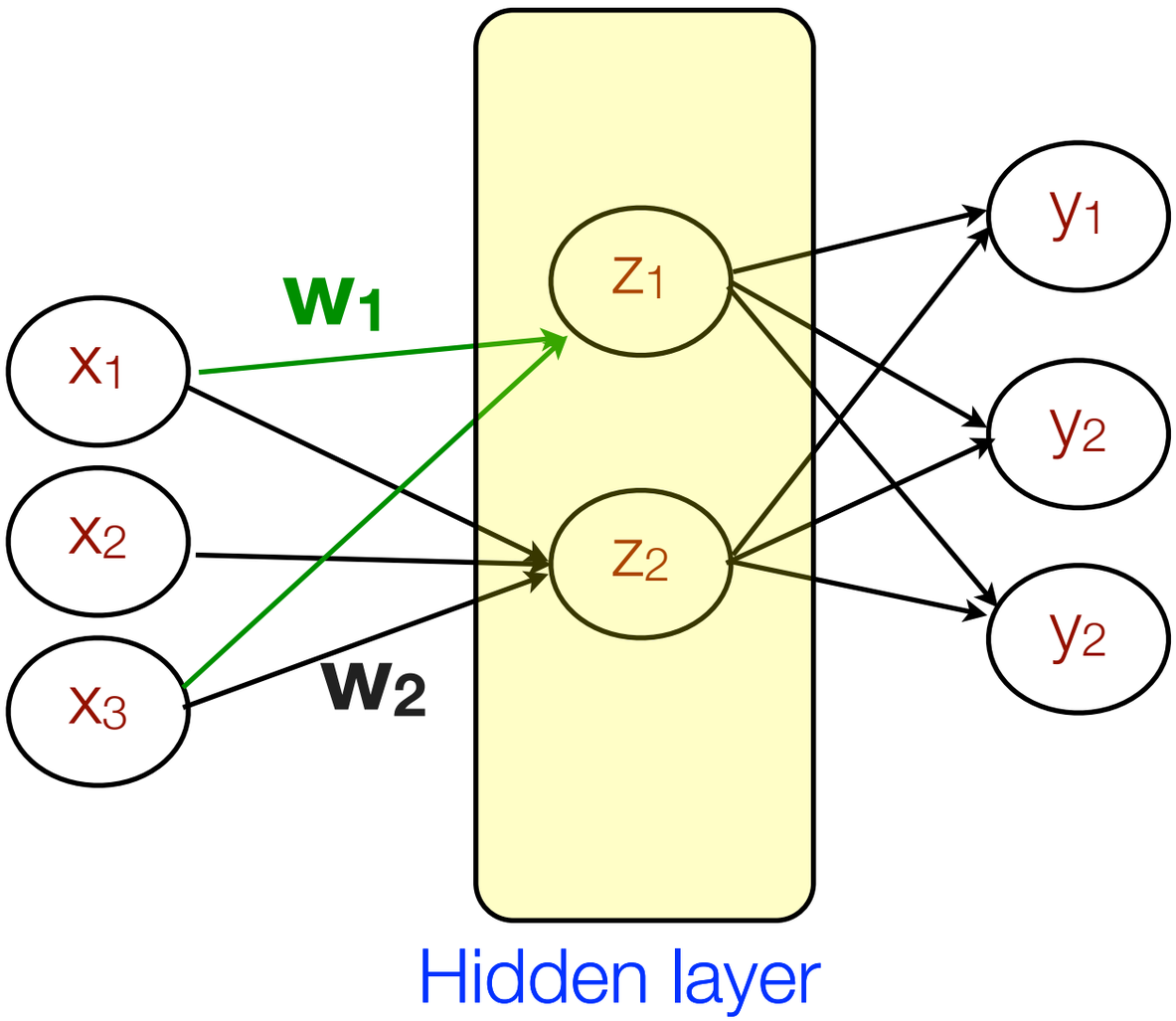
$$z_j = h\left(\sum_i w_{i,j} x_i\right)$$



The simplest Neural Network

Then we want to create one more layer - this is the *output* layer and consists of D outputs.

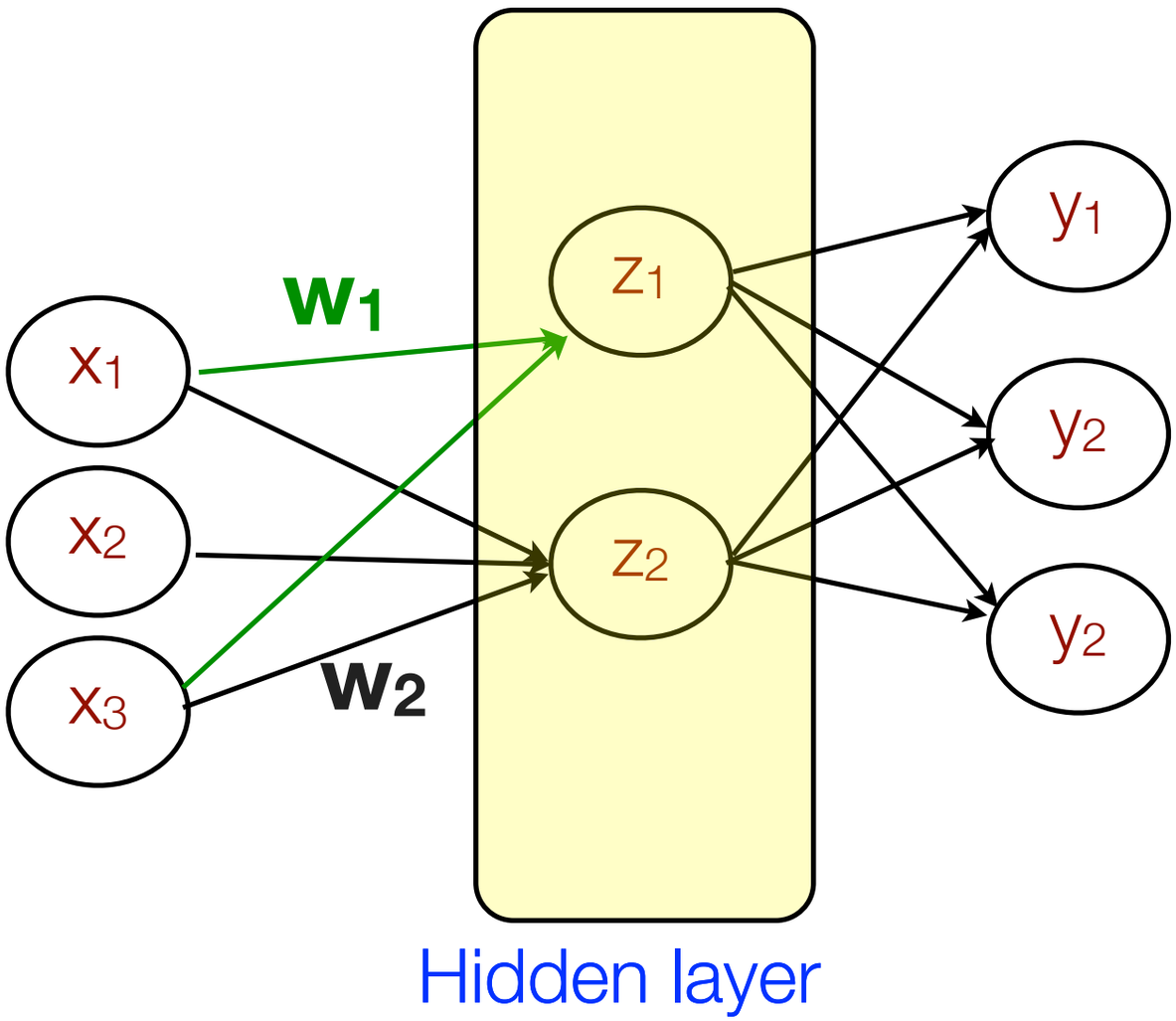
$$y_k = f\left(\sum_i v_{i,j} z_i\right)$$



Dense layer

Since all inputs and outputs are connected we call this a **dense** layer. It is a very common component of any neural network.

$$y_k = f\left(\sum_i v_{i,j} z_i\right)$$



The simplest Neural Network

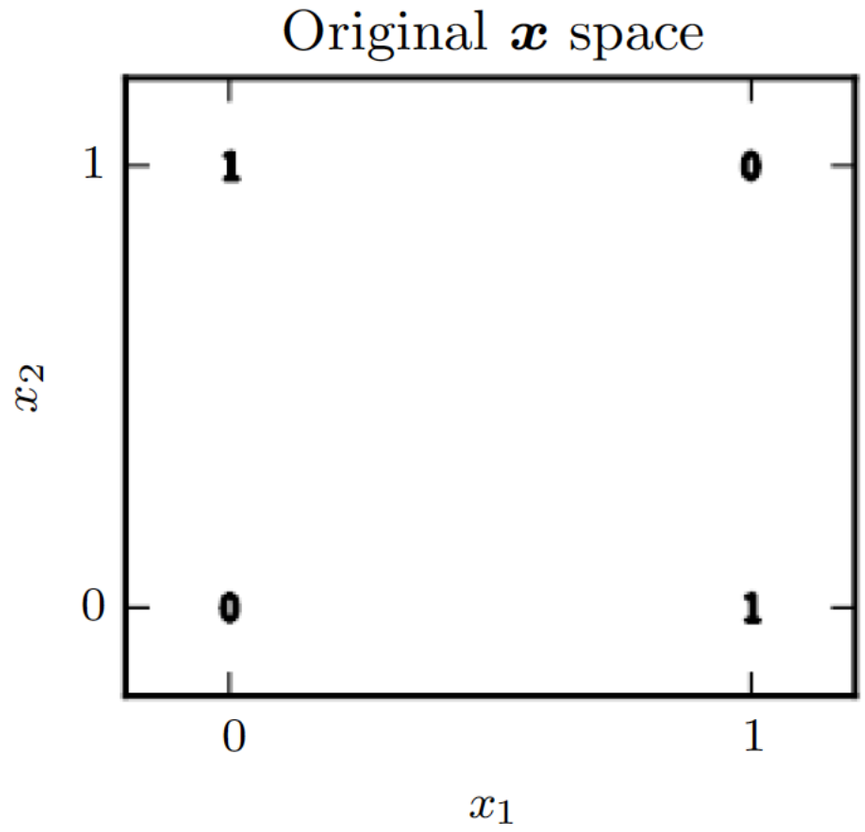
The full equation becomes:

$$y_k(\mathbf{x}, \mathbf{w}, \mathbf{v}) = f\left\{\sum_{j=1}^{M} v_{k,j}\ h\left(\sum_{i=1}^{N} w_{j,i} x_i\right)\right\}$$

For a set of inputs and a choice of weights - this can then be used to calculate y values easily. **The challenge is to determine the right choice of weights.** We will return to this when discussing deep learning.

Learning XOR

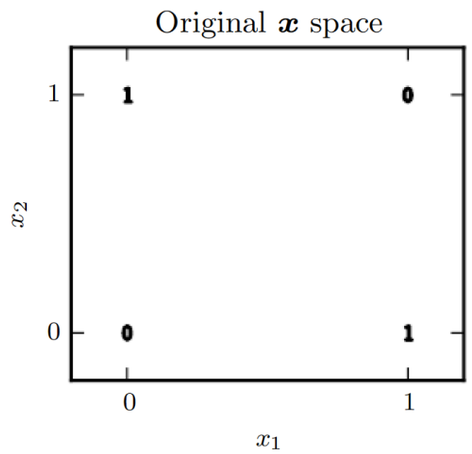
$$XOR(x, y) = \begin{array}{l} 1 \text{ if either } x = 1 \text{ or } y = 1, \text{ but not both} \\ 0 \text{ otherwise} \end{array}$$



We want to learn a function of x to fit this - obviously a linear function won't work!

Following Goodfellow et al, section 6.1

Learning XOR



weights: $\mathbf{w}$

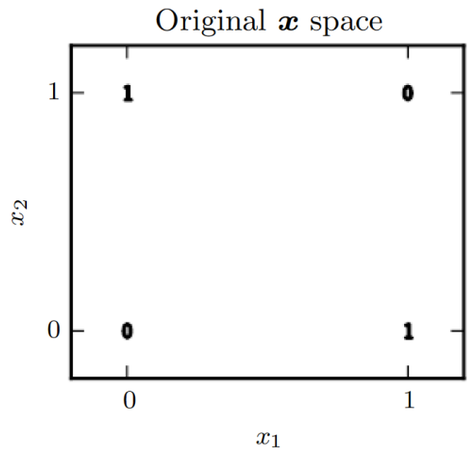
first layer: $\mathbf{h} = g\left(\mathbf{W}^T\mathbf{x} + \mathbf{c}\right)$

Using a ReLU:

$$f(\mathbf{x}; \mathbf{W}, \mathbf{c}, \mathbf{w}, b) = \mathbf{w}^T \max\left\{0, \mathbf{W}^T\mathbf{x} + \mathbf{c}\right\} + b$$

Following Goodfellow et al, section 6.1

Learning XOR



$$f(\mathbf{x}; \mathbf{W}, \mathbf{c}, \mathbf{w}, b) = \mathbf{w}^T \max\left\{0, \mathbf{W}^T\mathbf{x} + \mathbf{c}\right\} + b$$

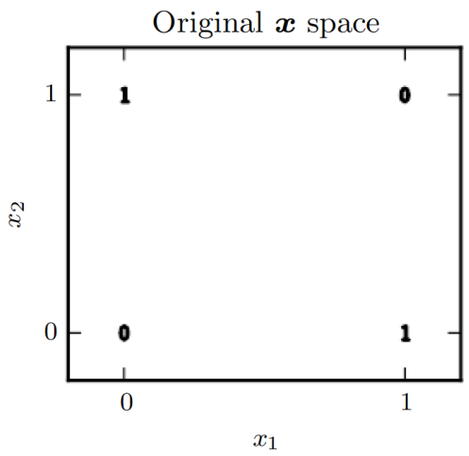
Let: $\mathbf{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$ $\mathbf{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$ $\mathbf{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$

Data: $\mathbf{X} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$ and: $\mathbf{XW} = \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{bmatrix}$

$$\mathbf{y} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

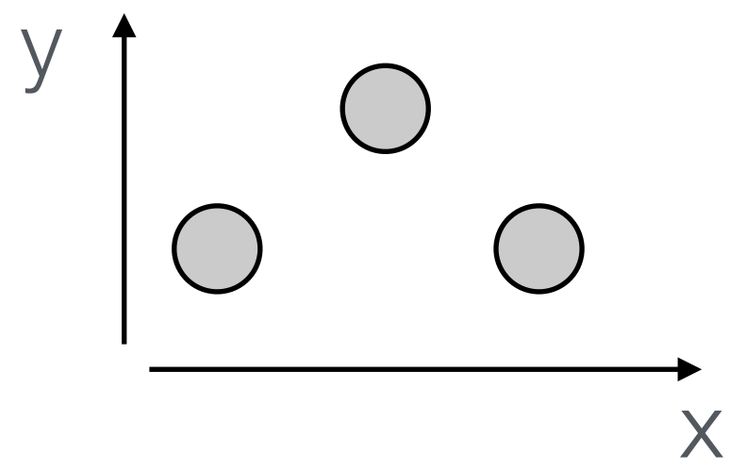
Following Goodfellow et al, section 6.1

Learning XOR

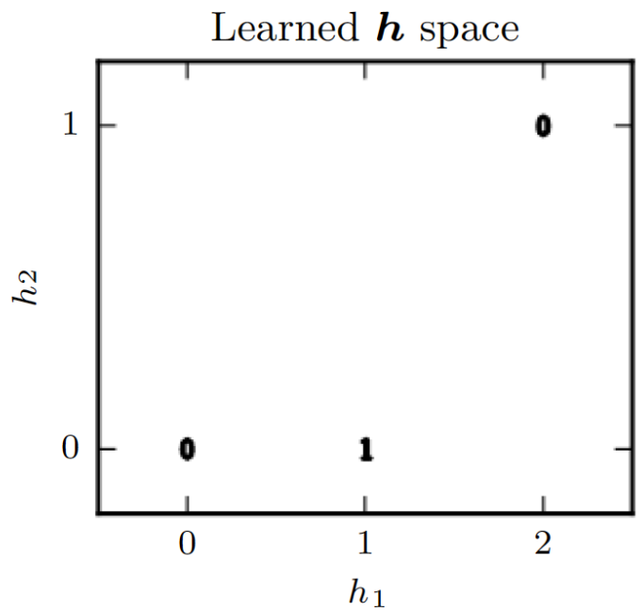


$$f(\mathbf{x}; \mathbf{W}, \mathbf{c}, \mathbf{w}, b) = \mathbf{w}^T \max\left\{0, \mathbf{W}^T \mathbf{x} + \mathbf{c}\right\} + b$$

$$\mathbf{XW} + \mathbf{c} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$$



$$\max\{0, \mathbf{XW} + \mathbf{c}\} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$$



Following Goodfellow et al, section 6.1

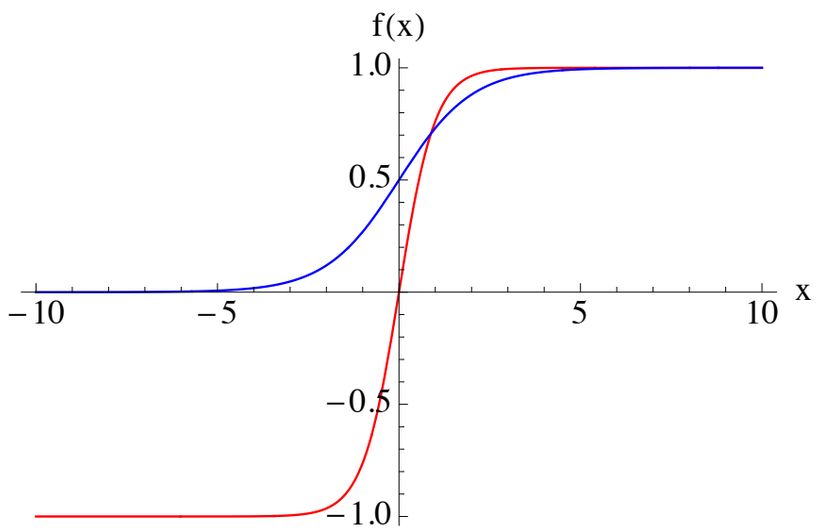
Output units - sigmoids

For classification cases, it is useful to have a function which takes -∞ to +∞ and maps it to -1 to 1 or 0 to 1. These are known as sigmoid functions. They have the advantage that they keep the signals from going off towards infinity and are useful as output units (in the past they were used as activation units too).

There are of course many of these, the most popular in our context are:

$$f(x) = \tanh(x)$$

Logistic: $$f(x) = \frac{1}{1 + e^{-x}}$$



Output units - linear functions

Another case is when you have a regression problem and want to produce a continuous output. In that case we use linear output units

Loss functions

MSE:

$$J(\theta) \propto \sum_{\mathbf{x}} \left(f^{\text{true}}(\mathbf{x}) - f(\mathbf{x}, \theta) \right)^2$$

Common earlier and sometimes used still but for many output units (sigmoids or softmax) this can lead to very bad performance.

Maximum likelihood/ cross-entropy

$$J(\theta) = -\ln \overline{p(\mathbf{y} \mid \mathbf{x}; \theta)}$$

The best general option - leads to MSE for Gaussian PDFs. The main advantage is that it undoes the exponentials in many PDFs.

What do you use as input?

You can give a full spectrum if you wish, or an image, but it could become time-consuming (but is what is typically used in deep learning) - can you do something else?

What do you use as input?

You can give a full spectrum if you wish, or an image, but it could become time-consuming (but is what is typically used in deep learning) - can you do something else?

You can instead provide a (judiciously chosen) set of measurements that summarise a particular image/ spectrum - **features**.

One possibility is to use **PCA** to select features - use these to reduce the dimension of the problem and input the PCA components as input variables.

You can also provide “Observables” - but make sure you don’t provide the same information many times!

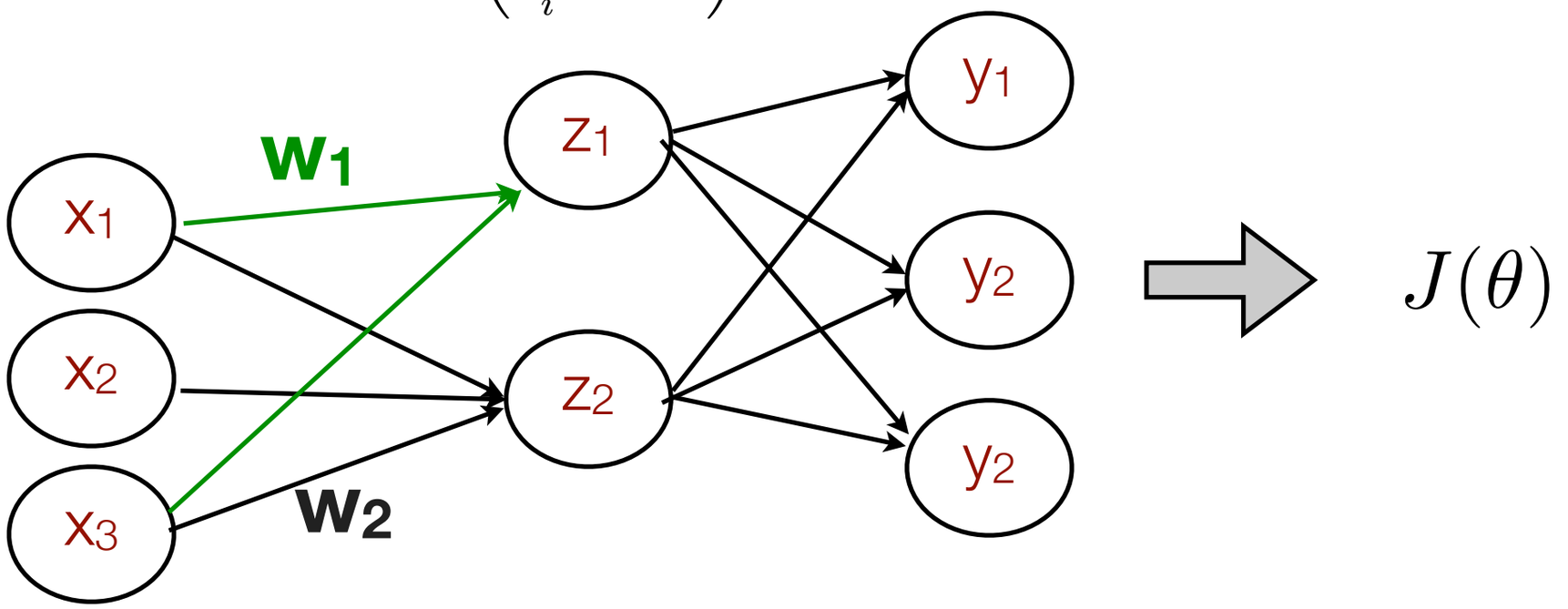
Forward- and back-propagation

Input

$$y_k = f\left(\sum_i v_{i,j} z_i\right)$$

Output

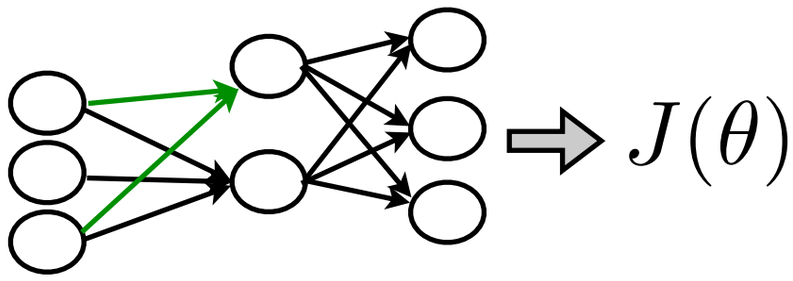
Loss function



Going from left to right is forward propagation

In this case, because there are no loops (cycles), the network is also a feed-forward network.

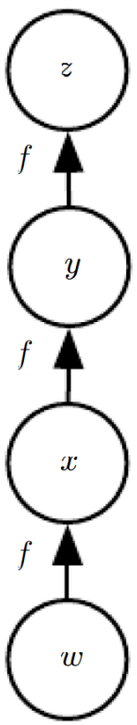
Forward- and back-propagation



But there will be errors made in the estimate.
How do we improve?

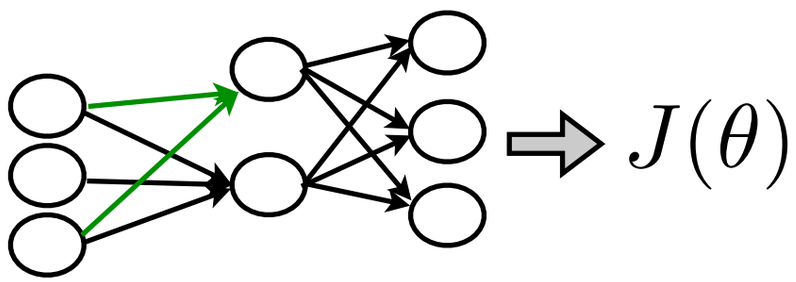
By going backwards!

We want to use gradient descent to update weights,
but for that we need the gradients:



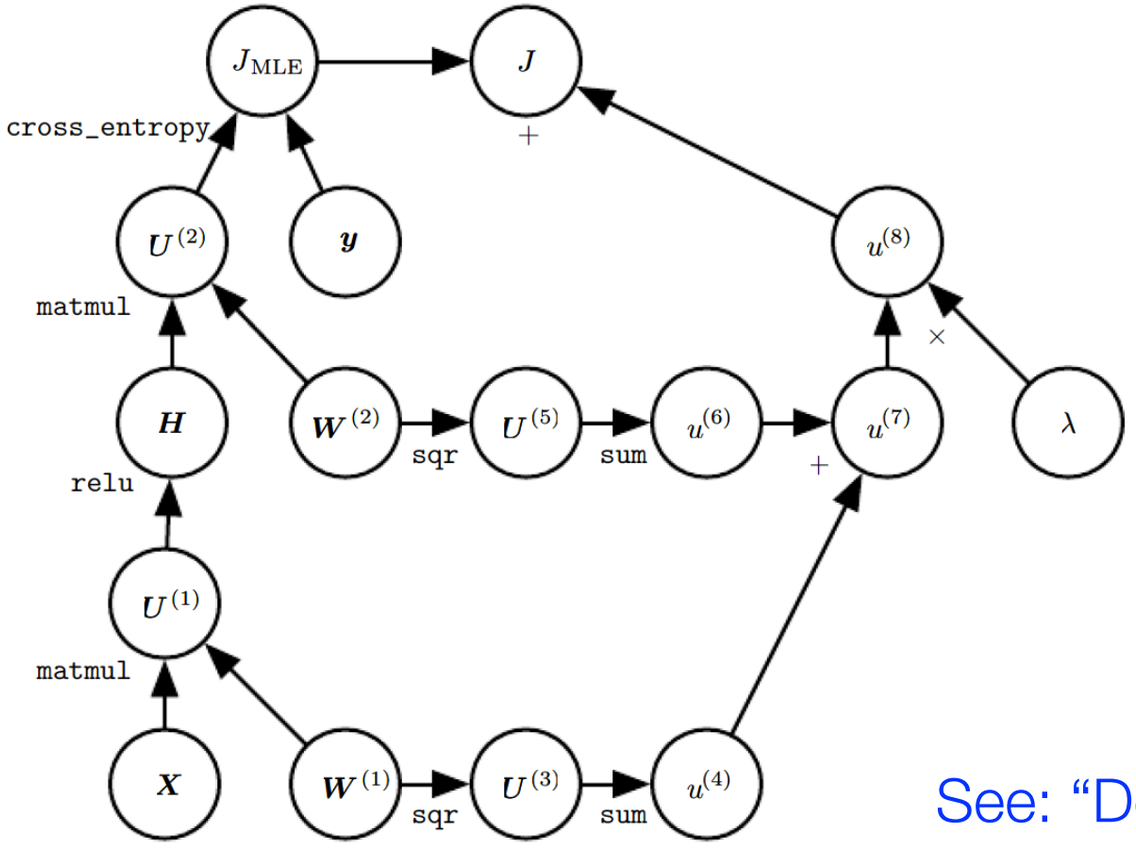
$$\frac{\partial z}{\partial w} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}\frac{\partial x}{\partial w}$$

Forward- and back-propagation



But there will be errors made in the estimate.
How do we improve?

The challenge is in slightly more complex situations:



Back-propagation is a clever algorithm to calculate these gradients efficiently and automatically.

See: “Deep learning”, Goodfellow et al (2016) for details.

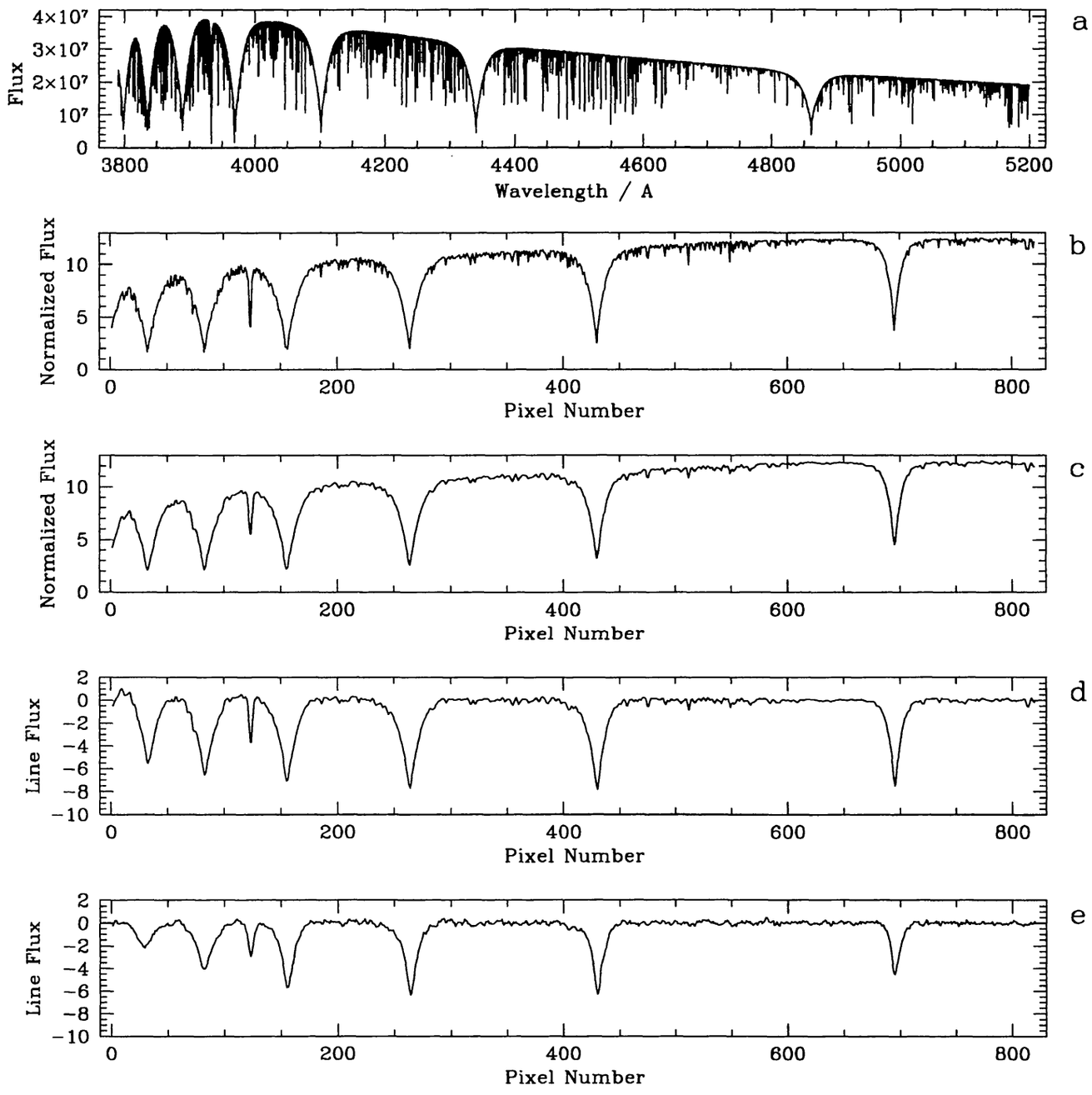
and https://www.ruder.io/optimizing-gradient-descent/ for details on optimization algorithms

Classifying spectra using artificial neural nets

A classical example:

Bailer-Jones et al (1997)

Input:



Classifying spectra using artificial neural nets

Bailer-Jones et al (1997)

Input: 821 fluxes
Neural net: 821 input units, 2 hidden layers with 5 weights each and 1 output unit

Classifying spectra using artificial neural nets

Bailer-Jones et al (1997)

Input: 821 fluxes
Neural net: 821 input units, 2 hidden layers with 5 weights each and 1 output unit

Combined 10 different neural nets (to check results, a kind of simplistic cross-validation).

Classifying spectra using artificial neural nets

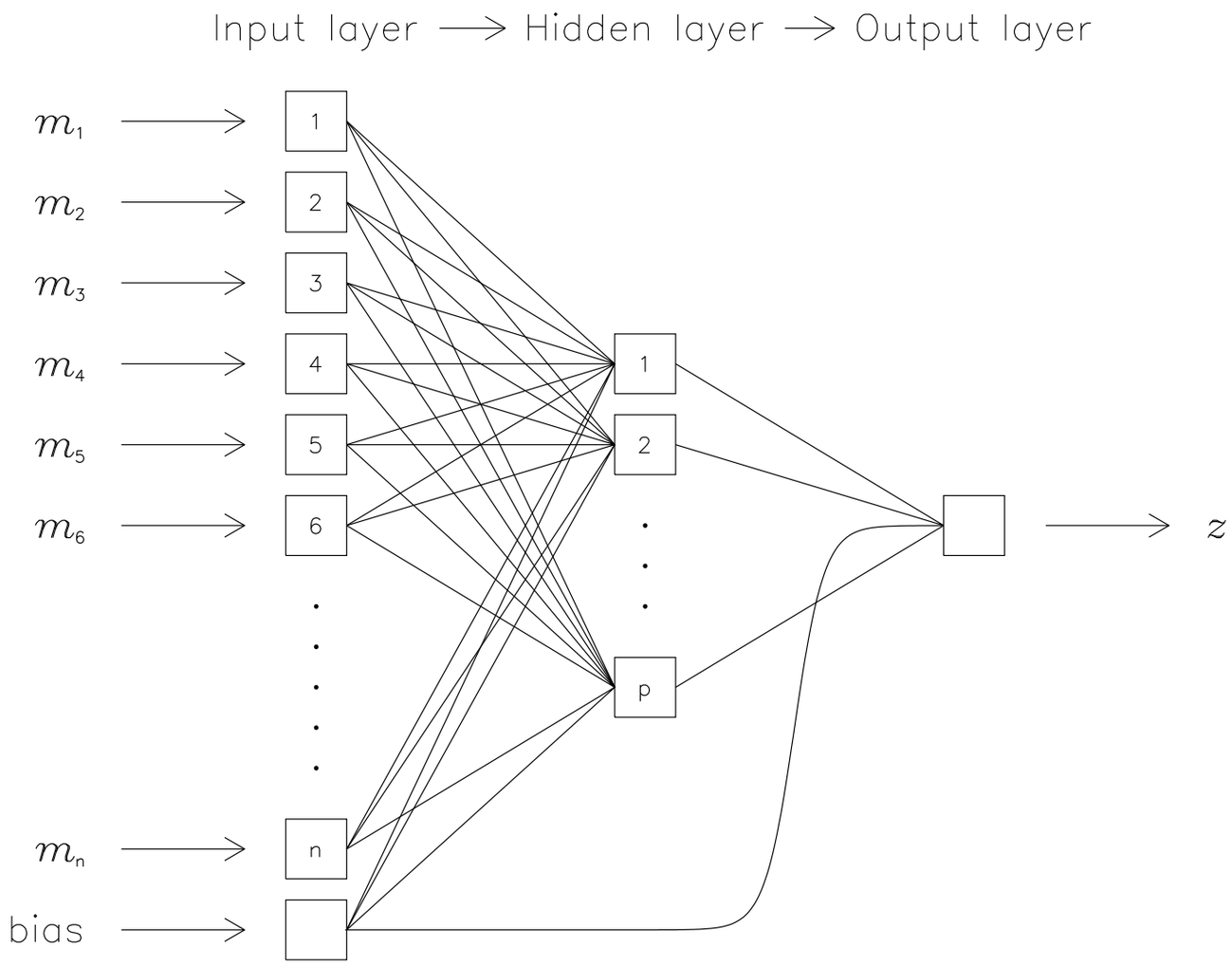
Bailer-Jones et al (1997)

Input: 821 fluxes
Neural net: 821 input units, 2 hidden layers with 5 weights each and 1 output unit

Combined 10 different neural nets (to check results, a kind of simplistic cross-validation).

Found that good estimates of T_{eff} can be found using this simple models but that metallicity introduces some uncertainty.

Redshifts from Neural Networks (ANNz)



ANNz is a freely available code that estimates a galaxy's redshift from input parameters.

Use a group of networks (a committee) and choose the best - needs to train on a comparable sample to what it will be applied to!

http://www.homepages.ucl.ac.uk/~ucapola/annz.html

Is it useful?

Appears to be useful for astronomical problems.
It does have a lot of strong believers and some strong opponents.

Advantages:

Very flexible - can approximate any function using two hidden layers.
Simple in structure and relatively easy to implement.
Widely used so plenty of literature and heuristic information.

Disadvantages:

Can be time-consuming to train and test - cross-validation can be extremely time-consuming.
Very hard to “understand” - do you get any insight from it?

From neural nets to deep learning

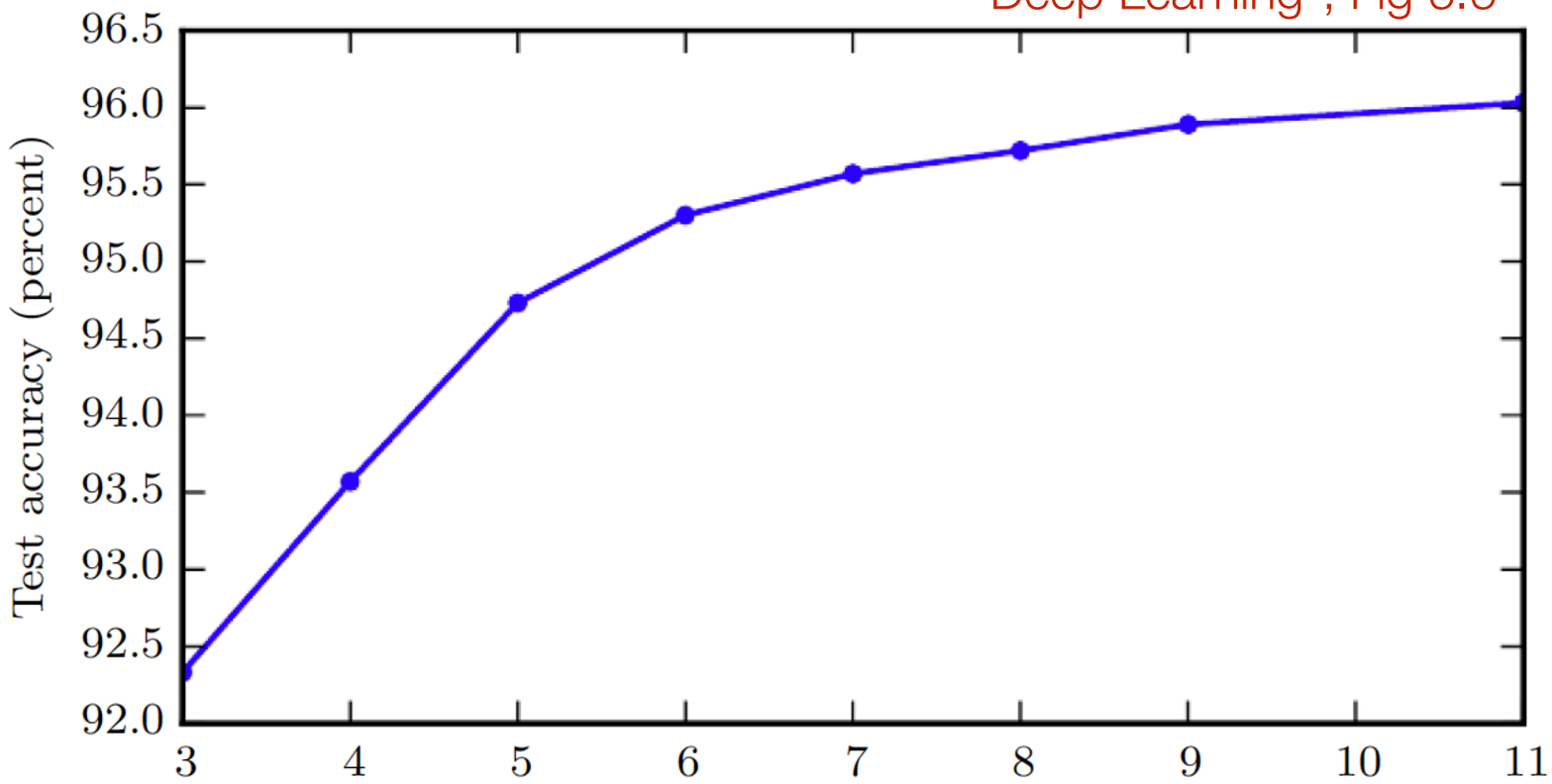
Deep learning is another name for neural networks.
So why the new name?

1. Much larger datasets enable more complex networks to be trained.
2. Improved computer speeds & better algorithms have made it easier to train many-layered networks.

Because these advances allows many more hidden layers to be used, it is called ‘deep’.

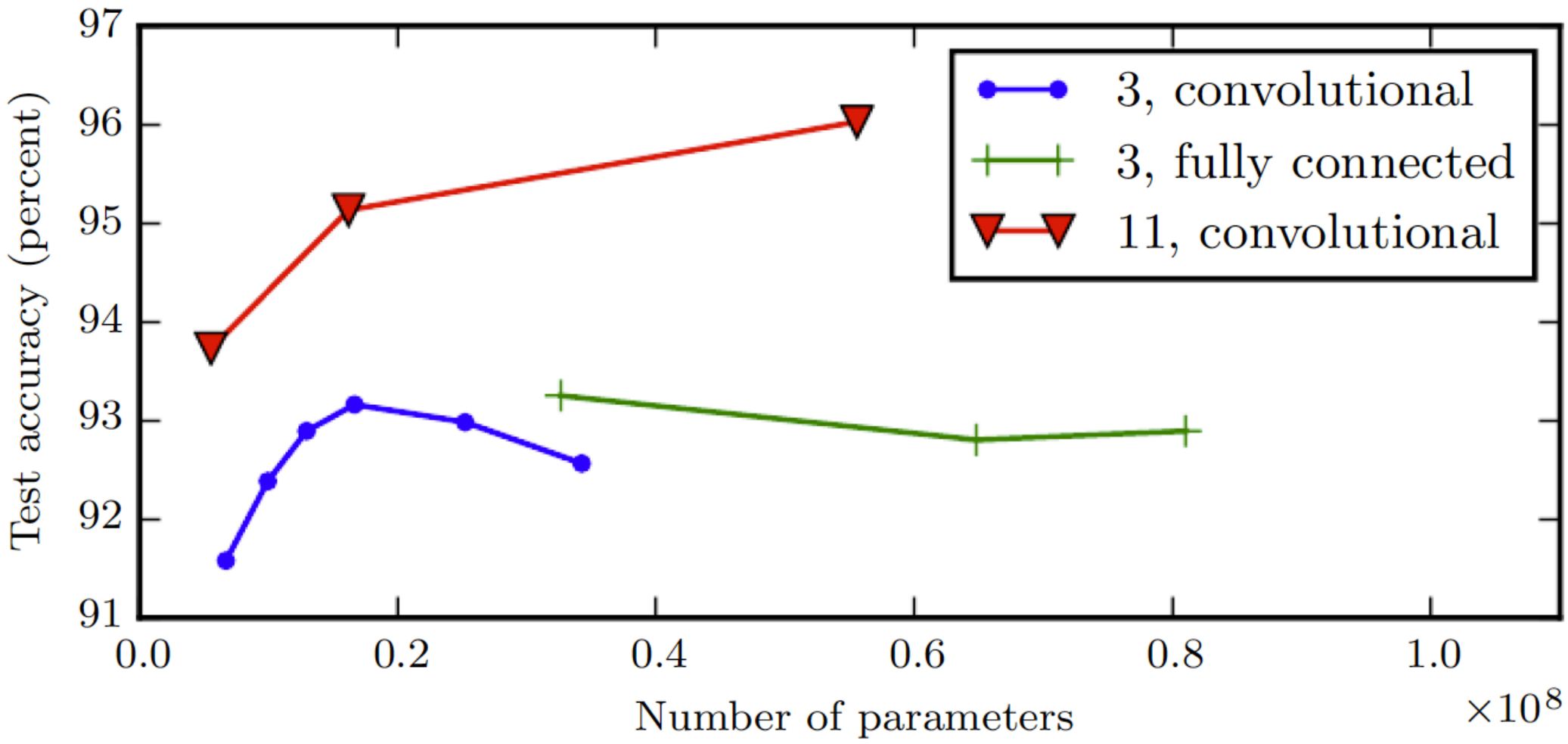
The advantage of depth

"Deep Learning", Fig 6.6



The performance of the neural networks tends to improve with depth.

The (dis?)advantage of depth



10^8 parameters is a lot… You need many training samples to avoid overfitting!

Handling overfitting -

Regularisation (very widely used): $J(\theta; \mathbf{X}, \mathbf{y}) + \lambda\Omega(\theta)$

e.g.:

$$J(\theta) + \lambda \sum \theta_i^2$$

(weight decay)

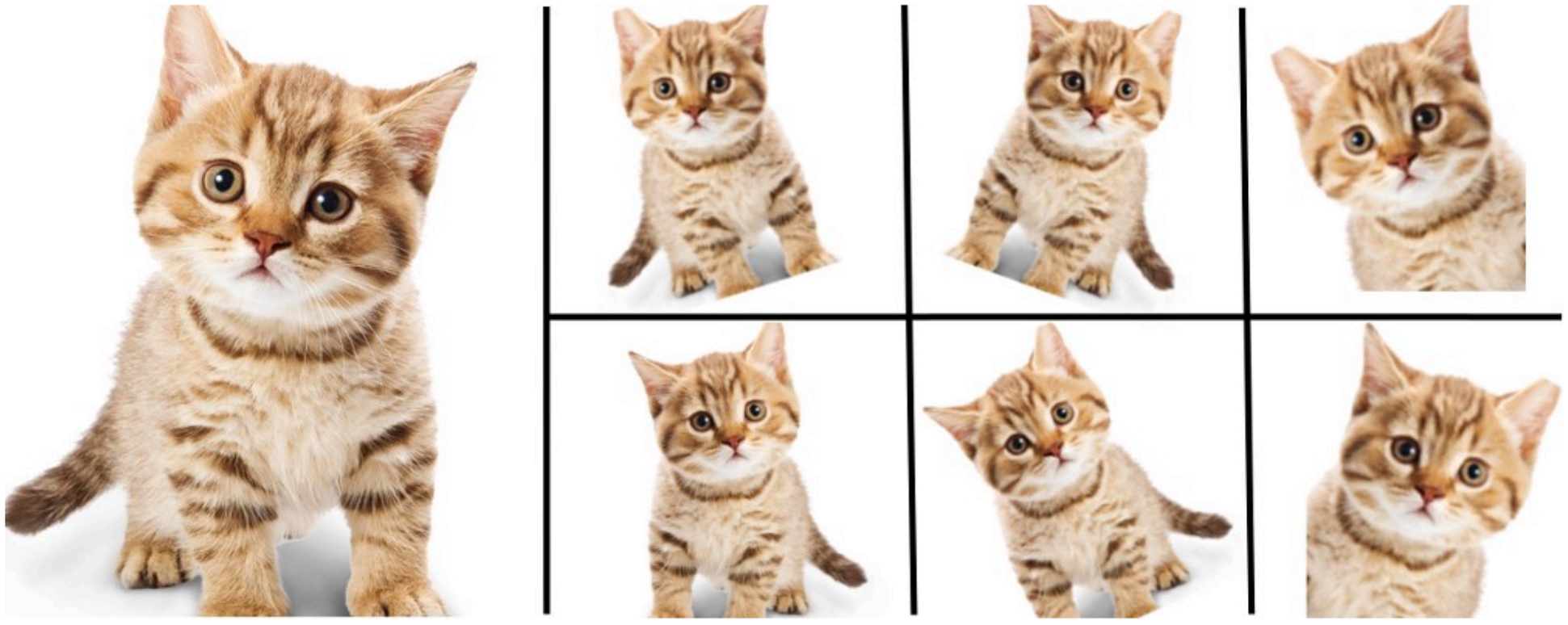
$$J(\theta) + \lambda \sum |\theta_i|$$

Note that the sum is over all i>0, thus the bias term is exempt

Very widely used - similarly to the case for regression, the absolute value regularisation leads to sparse solutions.

Handling overfitting: augmentation

Dataset augmentation: Enlarge your input data with modifications. Add rotated, shifted, noised, scaled examples.

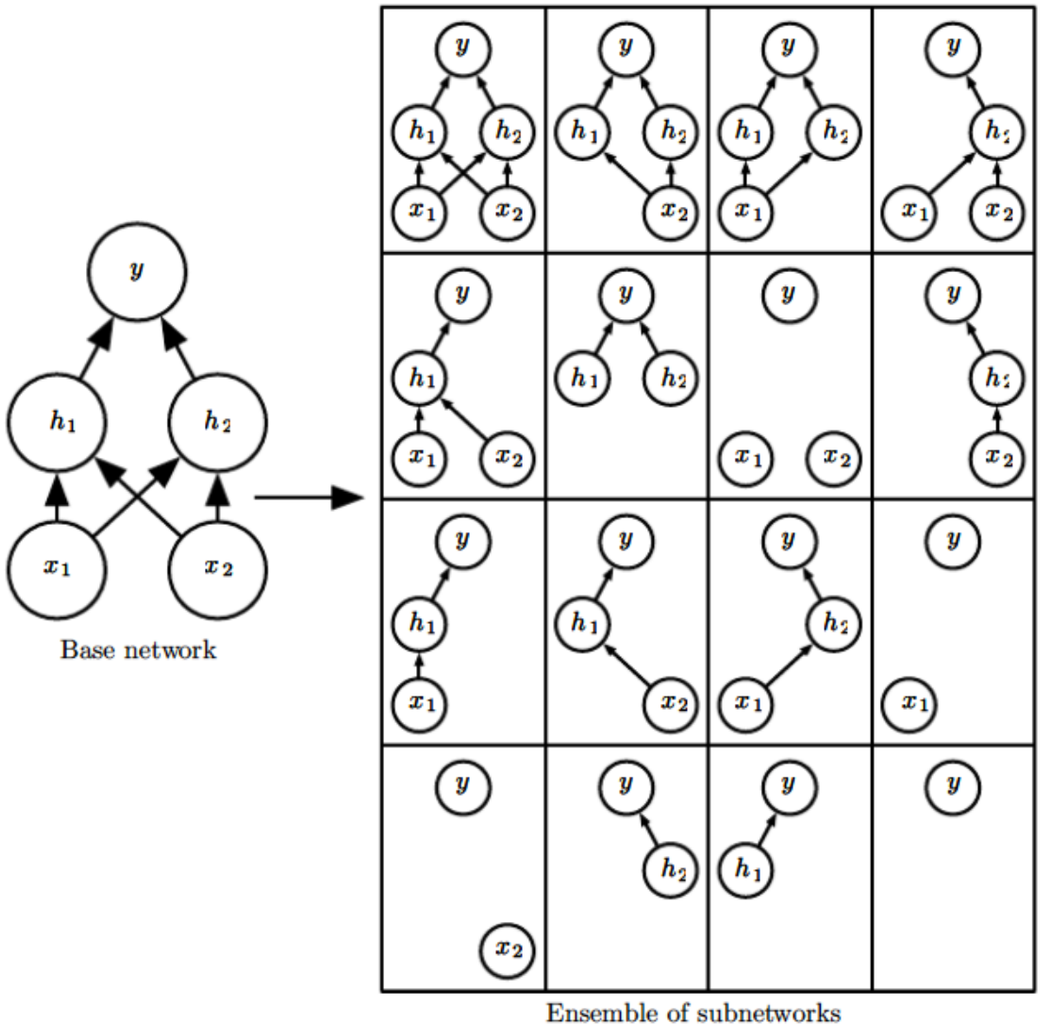


https://medium.com/nanonets/how-to-use-deep-learning-when-you-have-limited-data-part-2-data-augmentation-c26971dc8ced

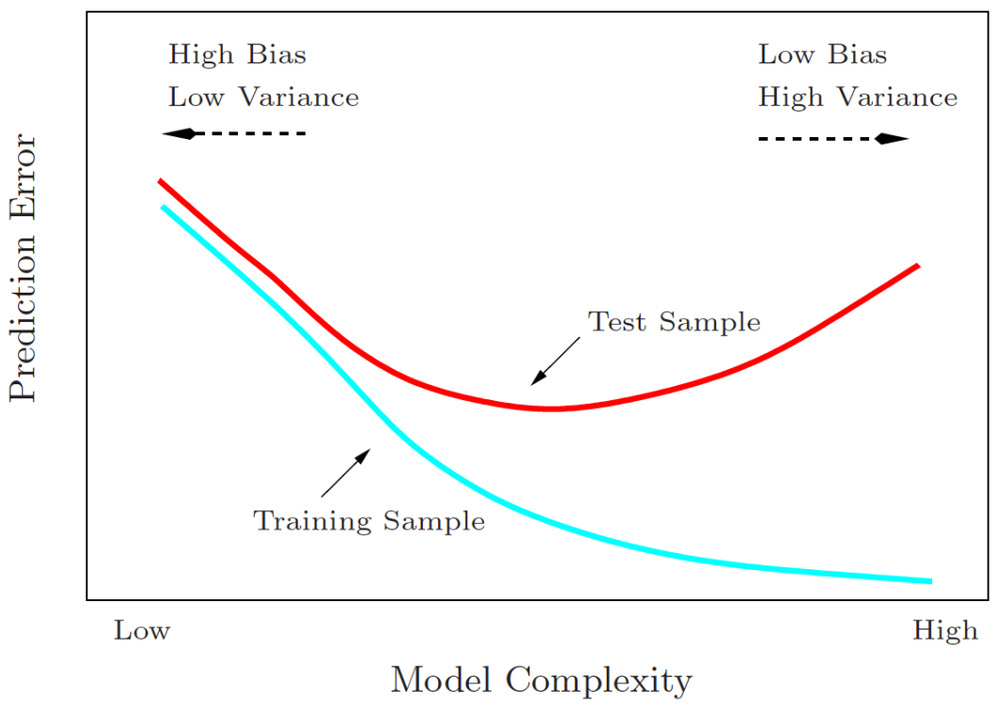
Handling overfitting: ensemble methods

Bagging: averaging multiple neural nets. Works, but can be extremely costly. However winners in competitions often use averaging.

Dropout: in practice consists of randomly dropping connections in the network and is computationally less expensive approach and achieves good performance, so is often used (for large datasets at least).

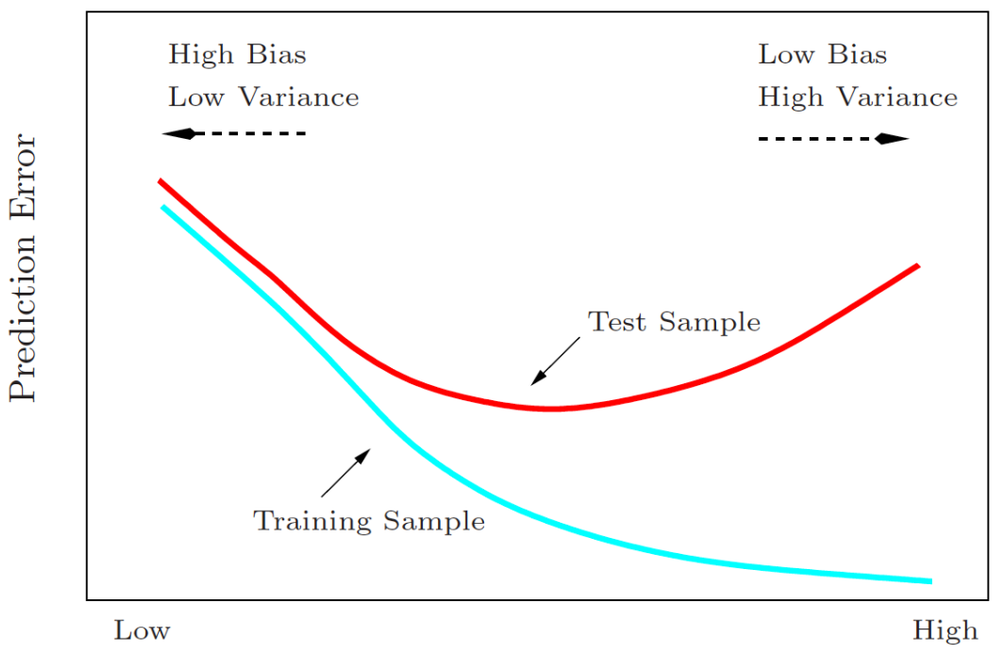


Bias-variance for deep neural networks



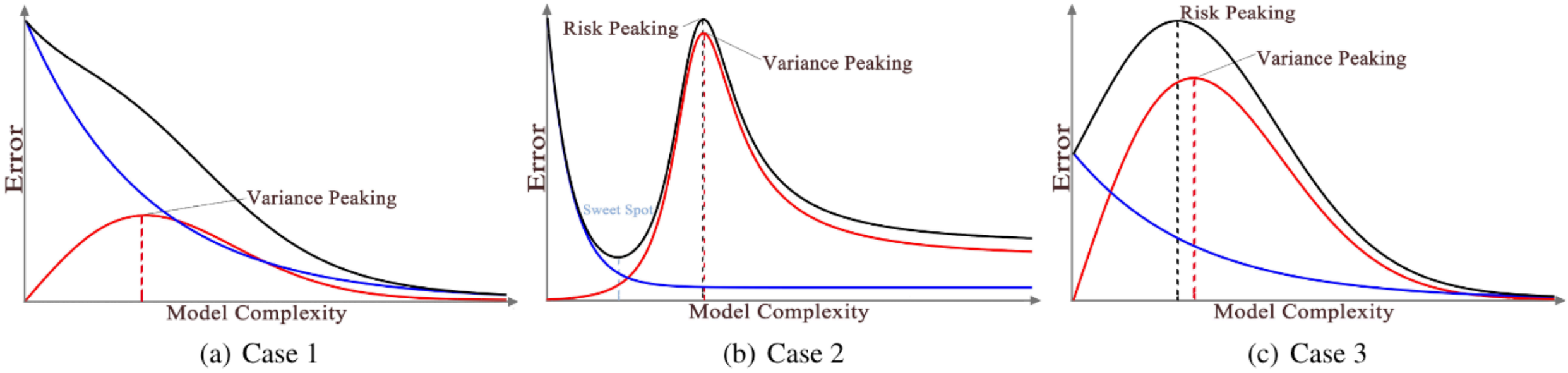
Seemed fine - right?

Bias-variance for deep neural networks



Seemed fine - right?

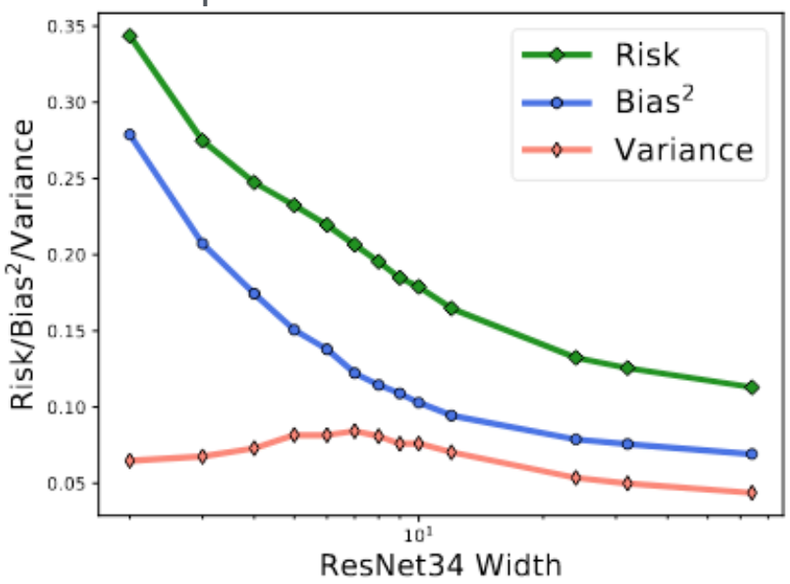
Yang et al (2020, arXiv:2002.11328v3)

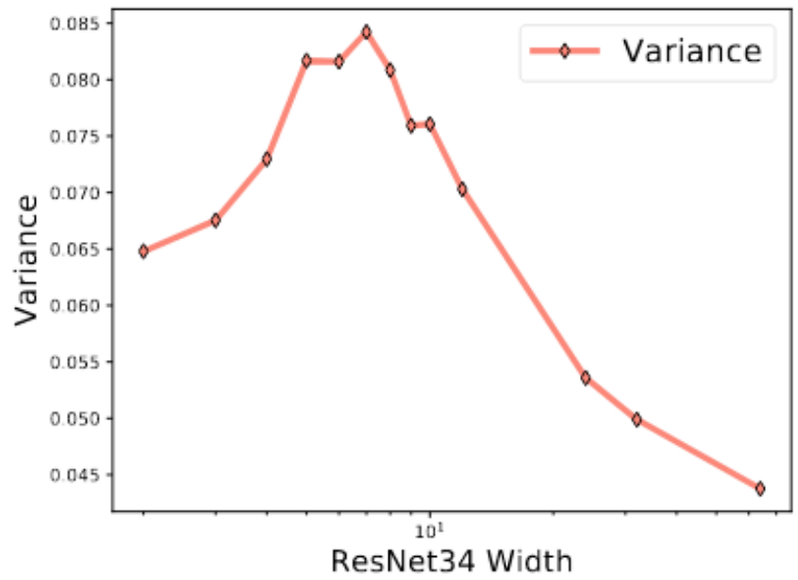


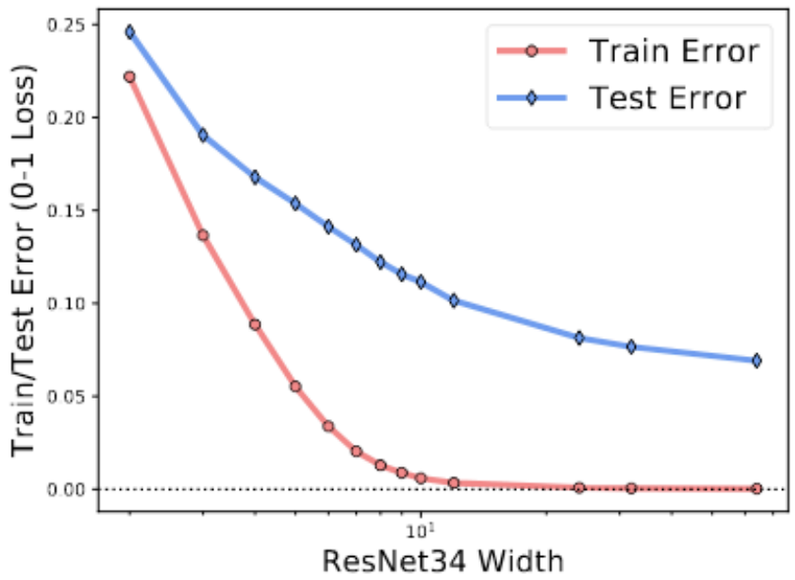
Bias-variance for deep neural networks

Yang et al (2020, arXiv:2002.11328v3)

Empirical runs:



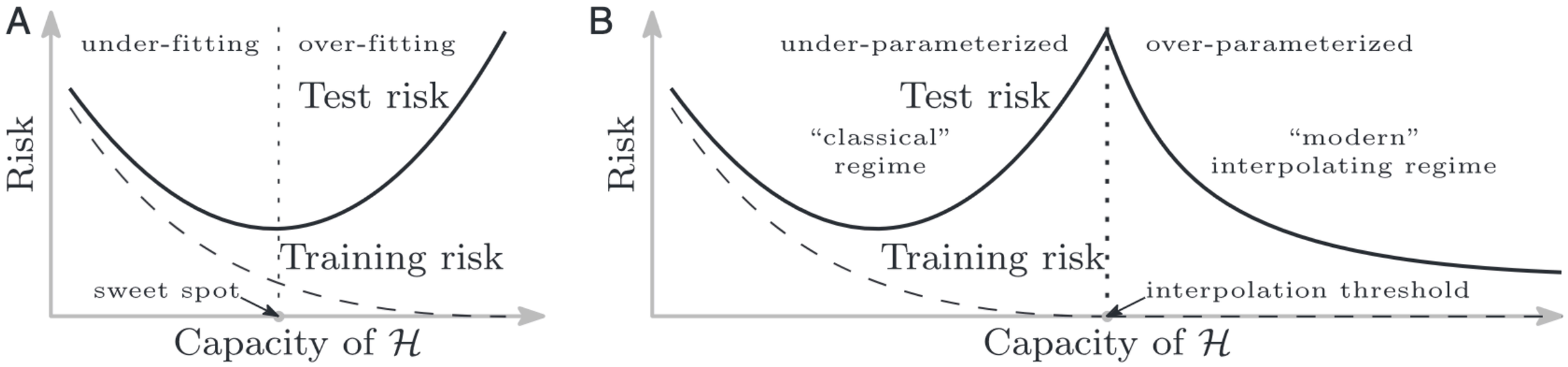




There seems to be fairly broad agreement that deep neural networks have reduced variance when the training sample gets above certain size/network sufficiently complex.

Is this a real, does it signify a change in nature of the learning?
Probably not, but active area of research - likely to be associated to regularisation in the network - see e.g. the Yang et al paper which uses ridge regression to illustrate where this might come from.

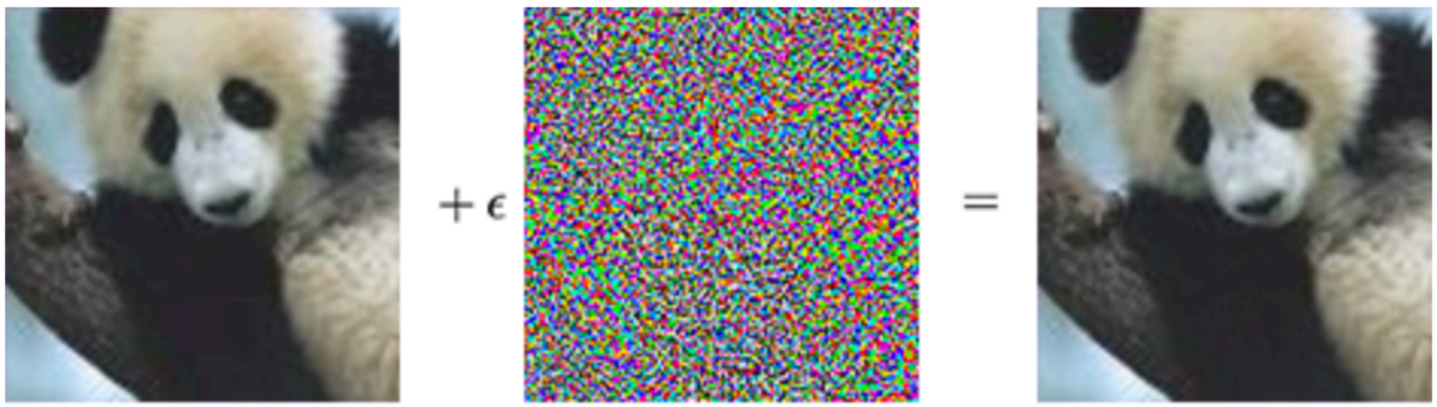
Deep learners ignore bias variance

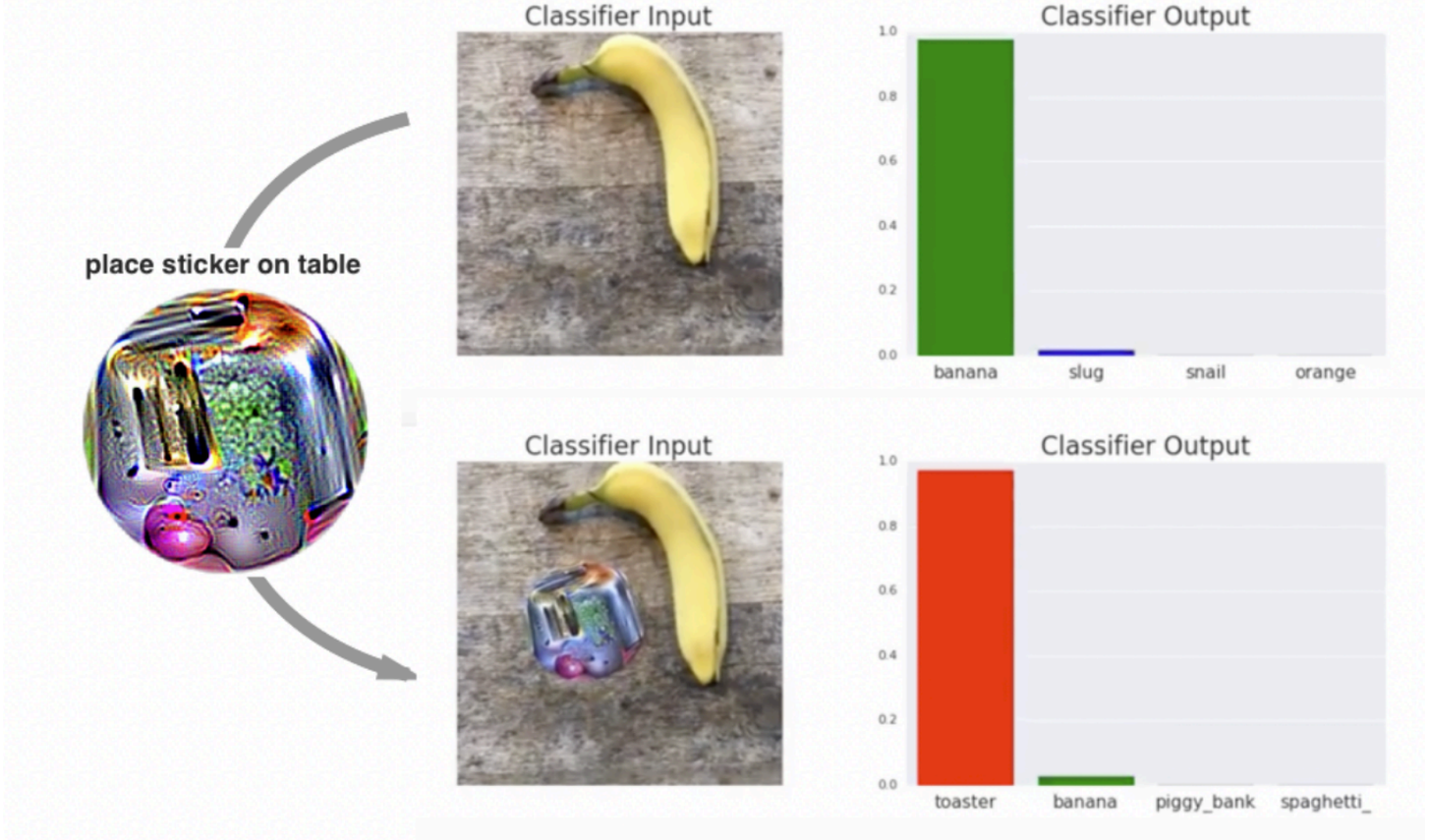


There are other similar phenomenons - e.g. *grokking*, where training performance improves, but test performance is for a long time poor - until after a while it improves a lot.

Belkin et al (2019) - https://www.pnas.org/cgi/doi/10.1073/pnas.1903070116

Sensitivity & adversarial examples





See: https://towardsdatascience.com/breaking-neural-networks-with-adversarial-attacks-f4290a9a45aa for refs

Sensitivity & adversarial examples

Impersonation glasses:

on images:

Reese Witherspoon

100% confidence

Russel Crowe

(using glasses)

Russel Crowe

really.

https://www.cs.cmu.edu/~sbhagava/papers/face-rec-ccs16.pdf

Sensitivity & adversarial examples

Impersonation glasses:

on images:

Reese Witherspoon

Russel Crowe

Russel Crowe

100% confidence (using glasses) really.

in real life:

(a) (b) (c) (d)

https://www.cs.cmu.edu/~sbhagava/papers/face-rec-ccs16.pdf

Sensitivity & adversarial examples

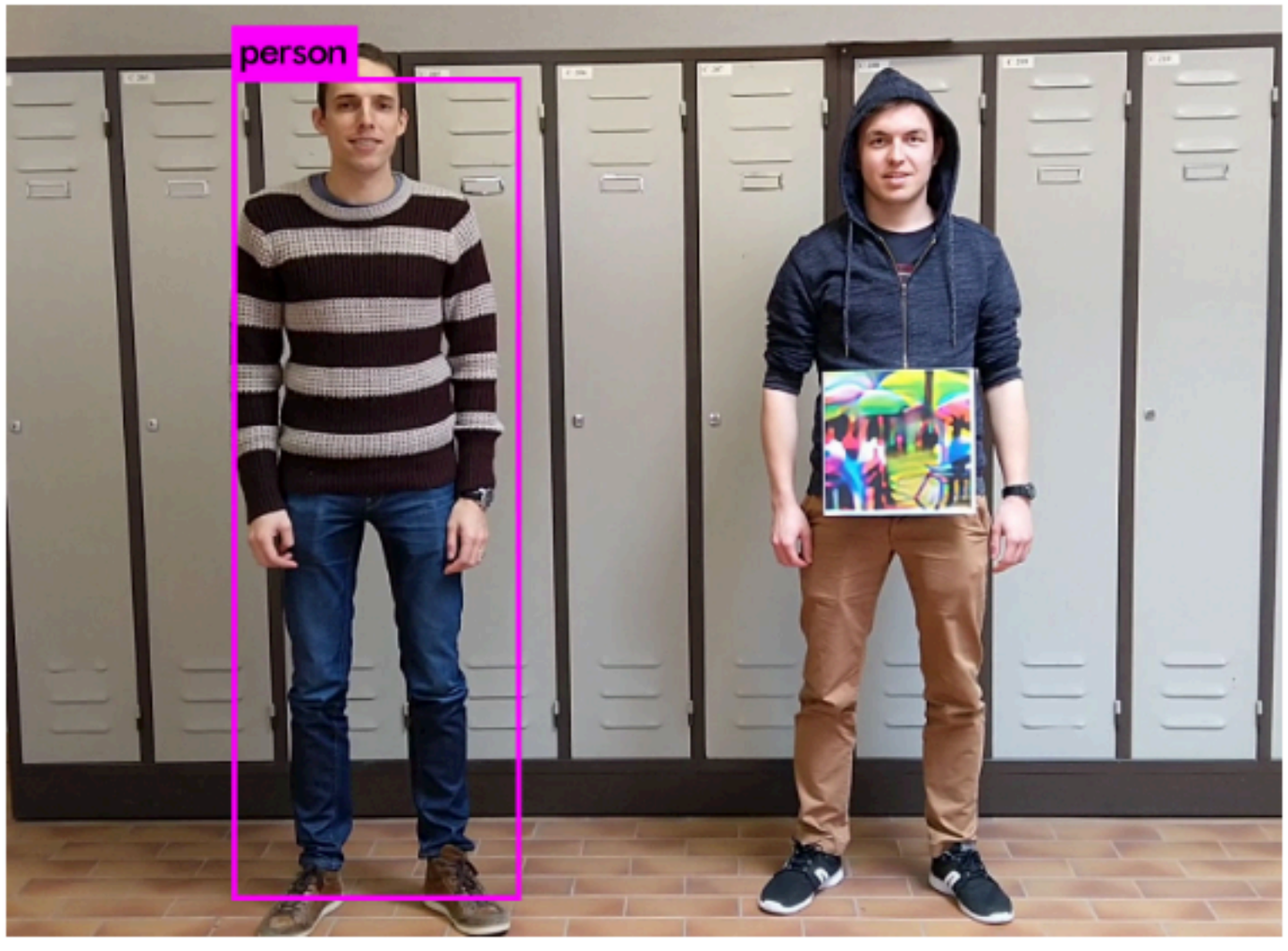
Camouflage Graffiti	Camouflage Art (LISA-CNN)	Camouflage Art (GTSRB-CNN)
		
		
67%	100%	80%

=

Putting small stickers on classifies this all as 45 mph signs

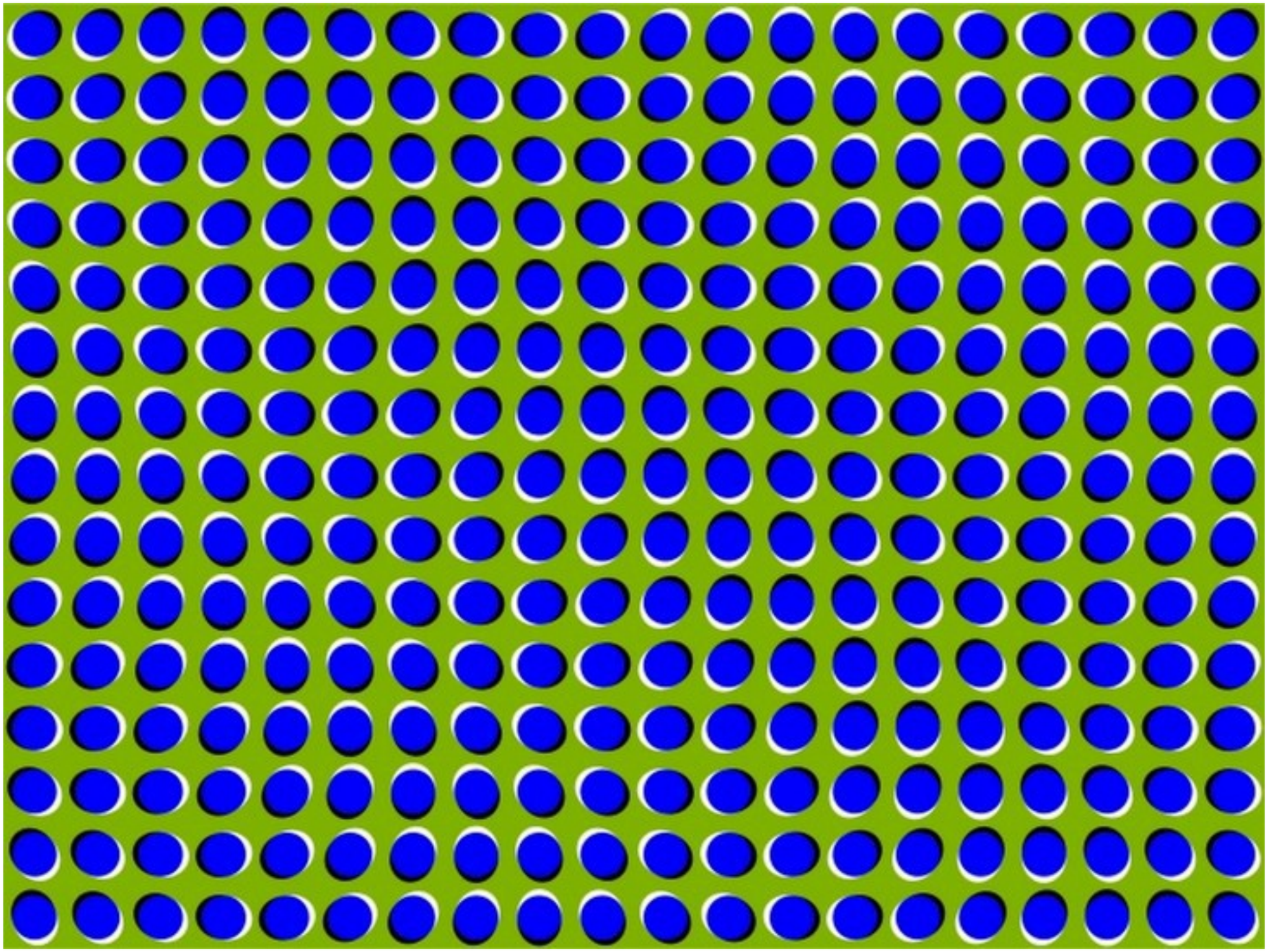
Eykholt et al (2017): https://arxiv.org/abs/1707.08945

“Fooling automated surveillance cameras”



Thys et al (2019, arXiv:1904.08653v1)

Adversarial examples = optical illusions?



How does this work?

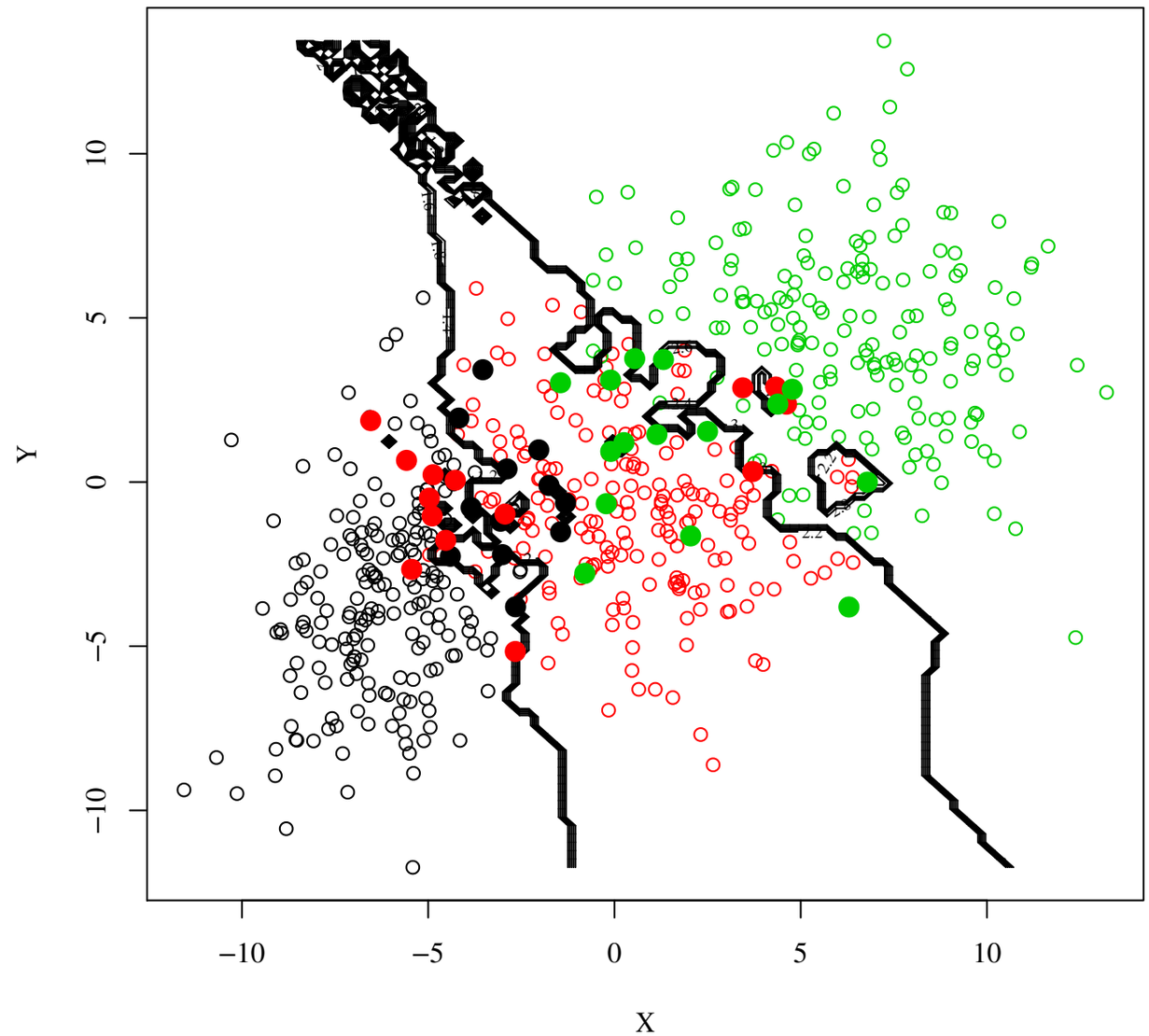
Tailored attacks: look at a part of the PDF for panda, calculate the gradient towards gibbon a place where they are closest and add the gradient:

Imagine something like:

How does this work?

Tailored attacks: look at a part of the PDF for panda, calculate the gradient towards gibbon a place where they are closest and add the gradient:

Imagine something like:

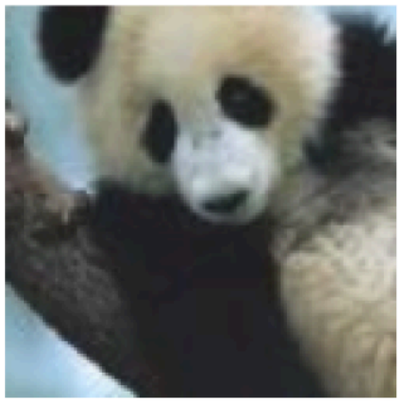


How does this work?

Tailored attacks: look at a part of the PDF for panda, calculate the gradient towards gibbon a place where they are closest and add the gradient:

How does this work?

Tailored attacks: look at a part of the PDF for panda, calculate the gradient towards gibbon a place where they are closest and add the gradient:



$\boldsymbol{x}$

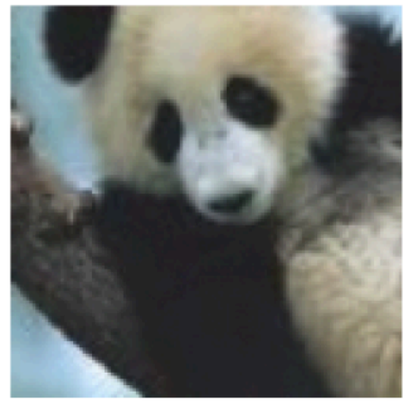
$y =$“panda”
w/ 57.7%
confidence

$+ .007 \times$

$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

“nematode”
w/ 8.2%
confidence

$=$



$\boldsymbol{x} +$
$\epsilon \, \text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

“gibbon”
w/ 99.3 %
confidence

Why deep learning

With the complex and flexible nature of deep learning models, we now have the possibility to skip the feature selection and instead move to a **data driven feature learning** model.

But why now? Many of the ideas go back to the 1950s!

Why deep learning

With the complex and flexible nature of deep learning models, we now have the possibility to skip the feature selection and instead move to a **data driven feature learning** model.

But why now? Many of the ideas go back to the 1950s!

- Massive data sets
- Powerful hardware advances (e.g. GPUs)
- Improved tools (tensorflow, Google Colabs, etc)

Types of deep neural networks

- CNN - Convolutional neural networks
- RNN - Recurrent neural networks
- Encoder/Decoders - systematic dimension reduction methods
- Autoencoders - unsupervised encoders
- Variational autoencoders - probabilistic versions of autoencoders
- GAN - Generative Adversarial Networks
- Reinforcement learning

Convolutional neural networks

These are neural networks that replace matrix multiplication with convolution in at least one of the layers.

These networks can work on images/spectra/time-series with no need to pre-select features.

For that reason they have quickly become very popular in astronomy.

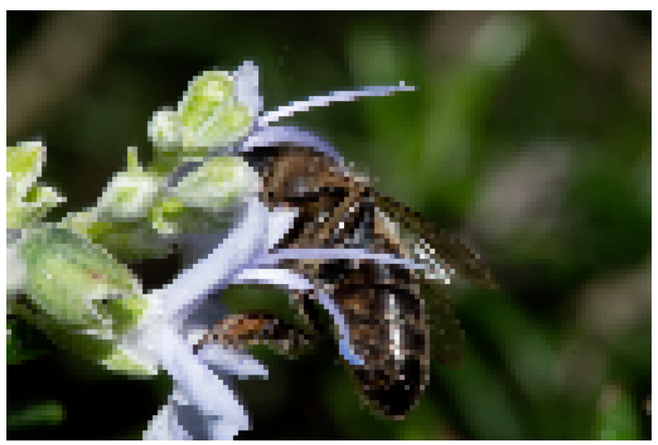
Convolutional layers are usually combined with pooling layers to reduce the dimensionality.

Sharing data - convolutional and recurrent networks

Sometimes you want units to be very similar or identical

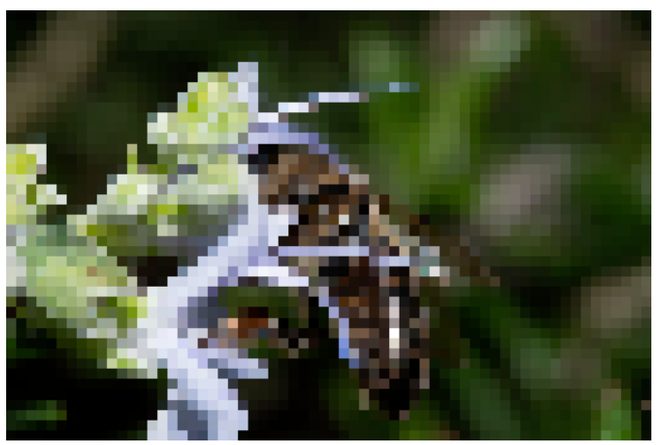
Convolutional layers do this by convolving images with a kernel - this can be used to detect edges, or simply to bin pixels together:

4x



10912 units

8x



2728 units

16x

628 units

Sharing data - convolutional and recurrent networks

Recurrent networks do the same thing in sequences

$$s(t) = f[x_t;\ s(t-1)]$$

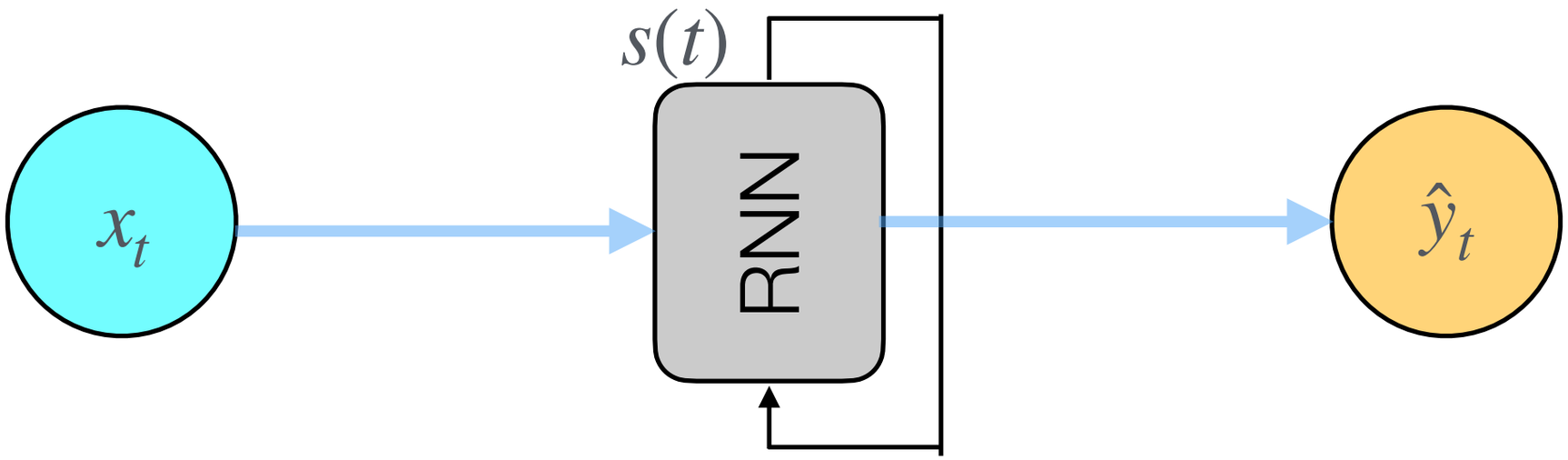
the state, s(t), is represented by a hidden layer - and more complex systems can be modelled

Sharing data - convolutional and recurrent networks

Recurrent networks do the same thing in sequences

$$s(t) = f[x_t;\ s(t-1)]$$

the state, s(t), is represented by a hidden layer - and more complex systems can be modelled

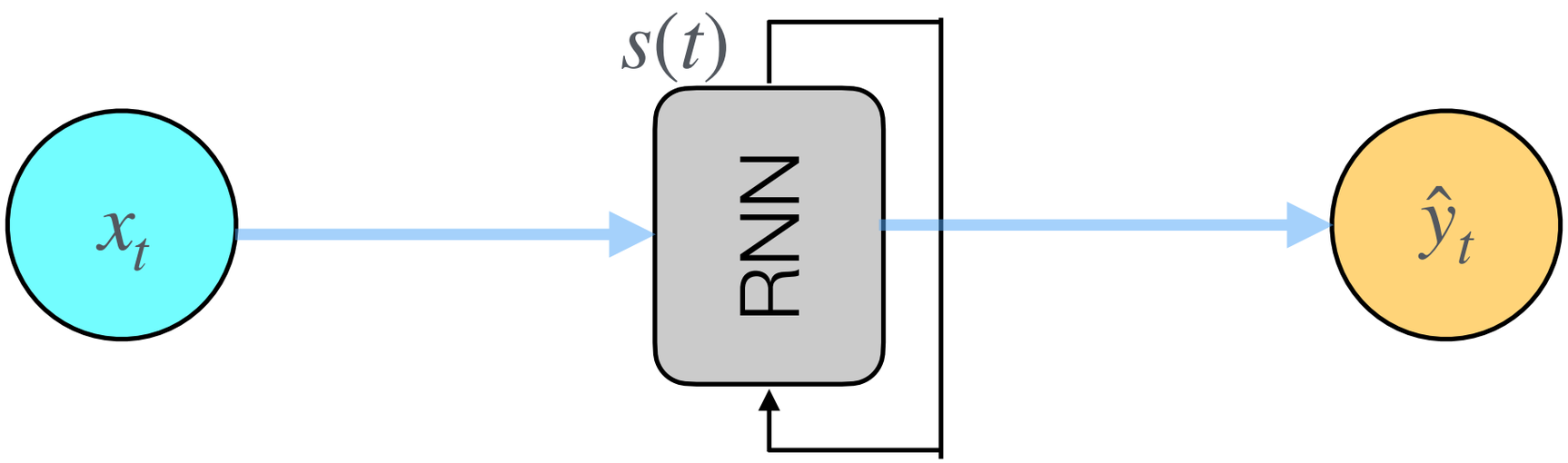


Sharing data - convolutional and recurrent networks

Recurrent networks do the same thing in sequences

$$s(t) = f[x_t;\ s(t-1)]$$

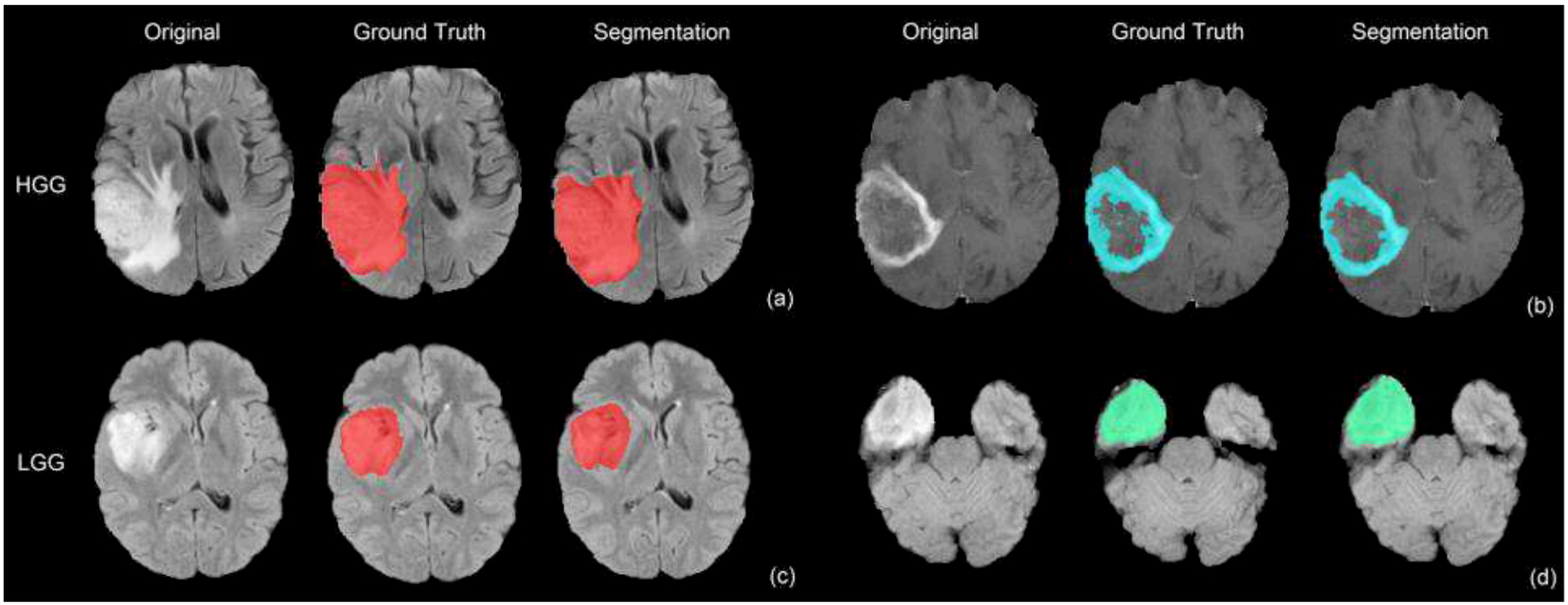
the state, s(t), is represented by a hidden layer - and more complex systems can be modelled



Very useful for e.g. speech recognition, handwriting recognition, translation etc.

Segmentation - a challenging problem

Example: Brain tumour detection, Dong et al (2017):



Achieved using a type of CNN architecture called a Unet

Segmentation - the Unet - an encoder/decoder

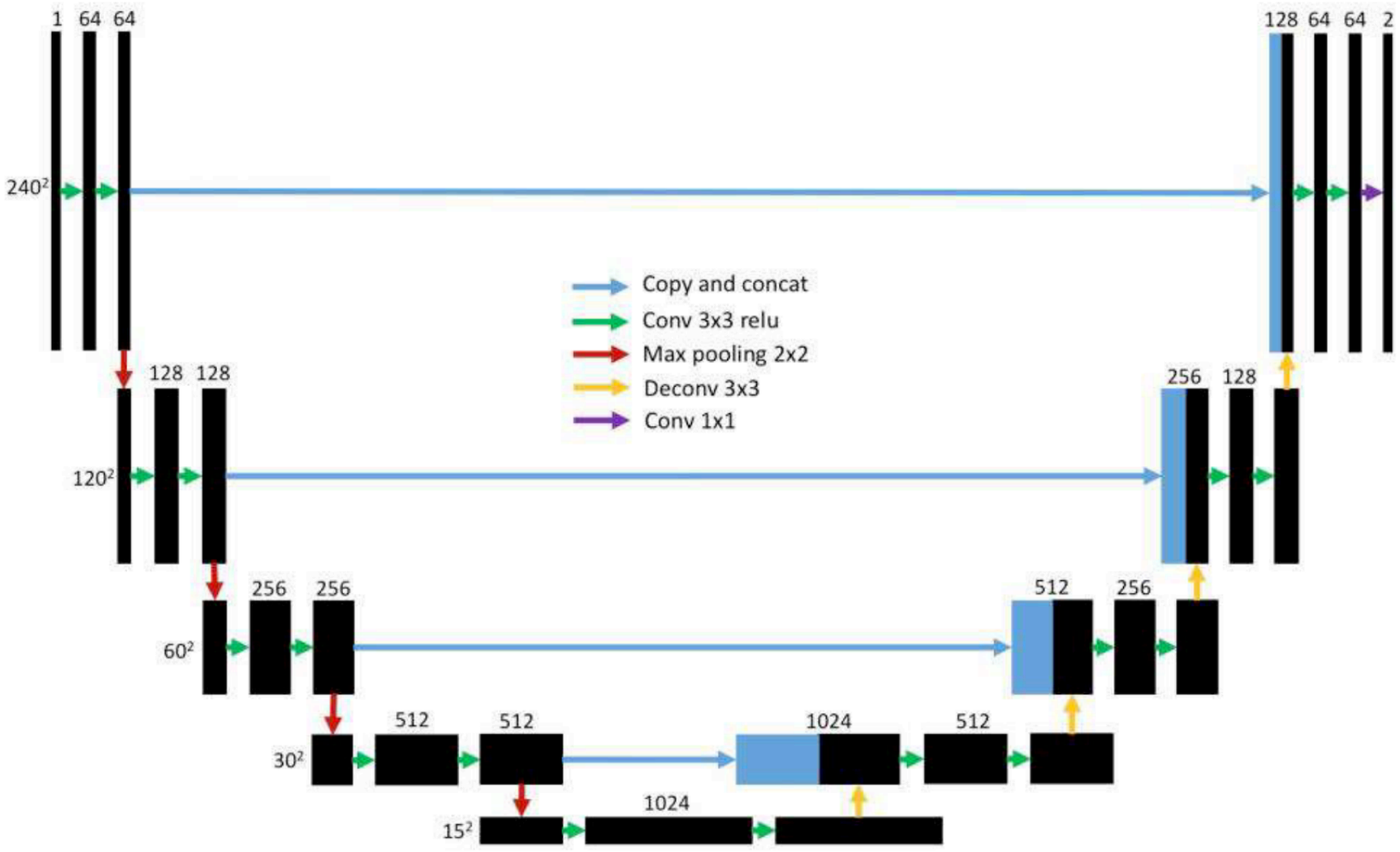


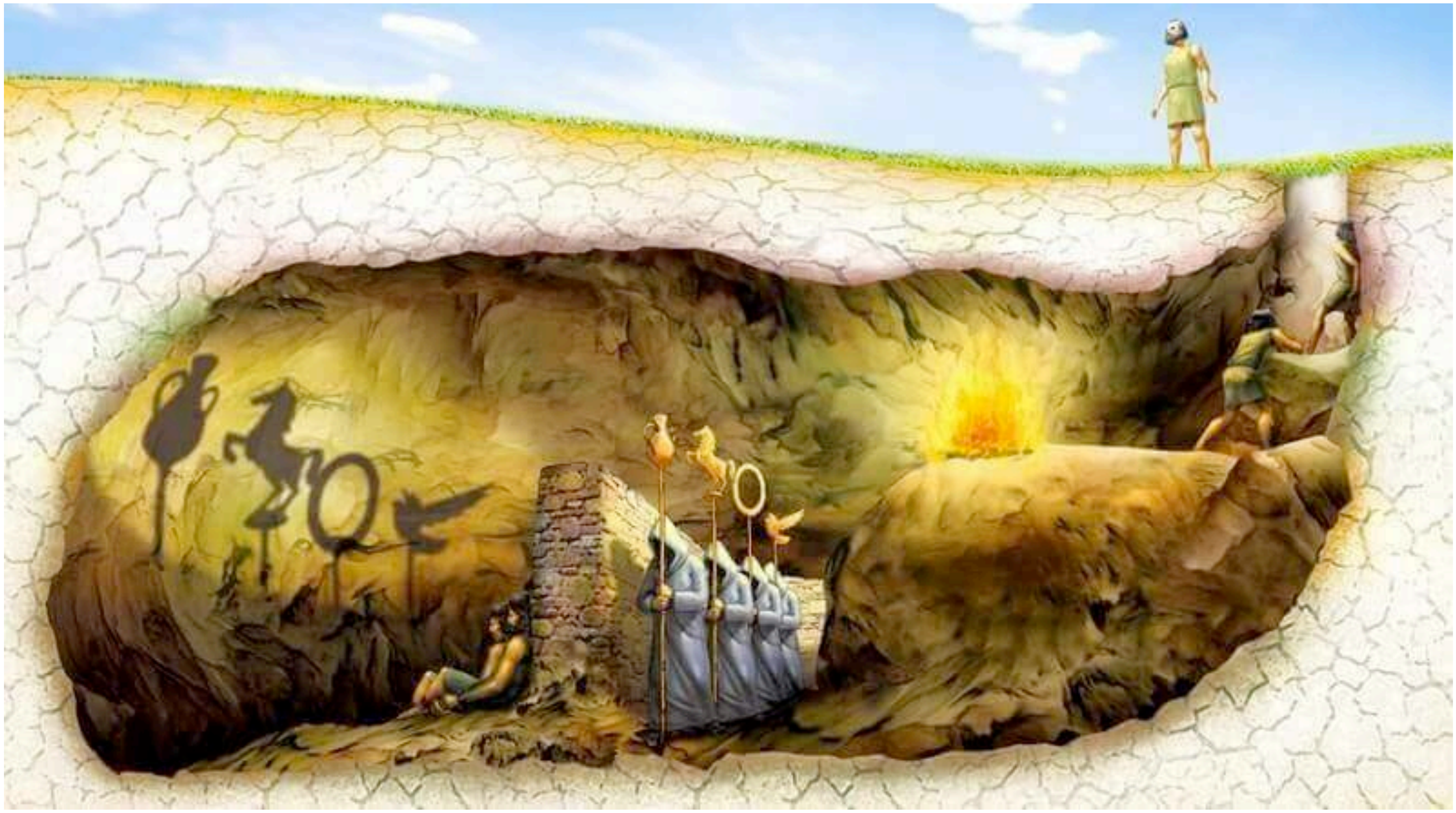
Fig. 1. Our developed U-Net architecture.

Table 2. Parameters setting for the developed U-Net.

Parameters	Value
Number of convolutional blocks	[4, 5, 6]
Number of deconvolutional blocks	[4, 5, 6]
Regularization	L1, L2, Dropout

Dong et al (2017)

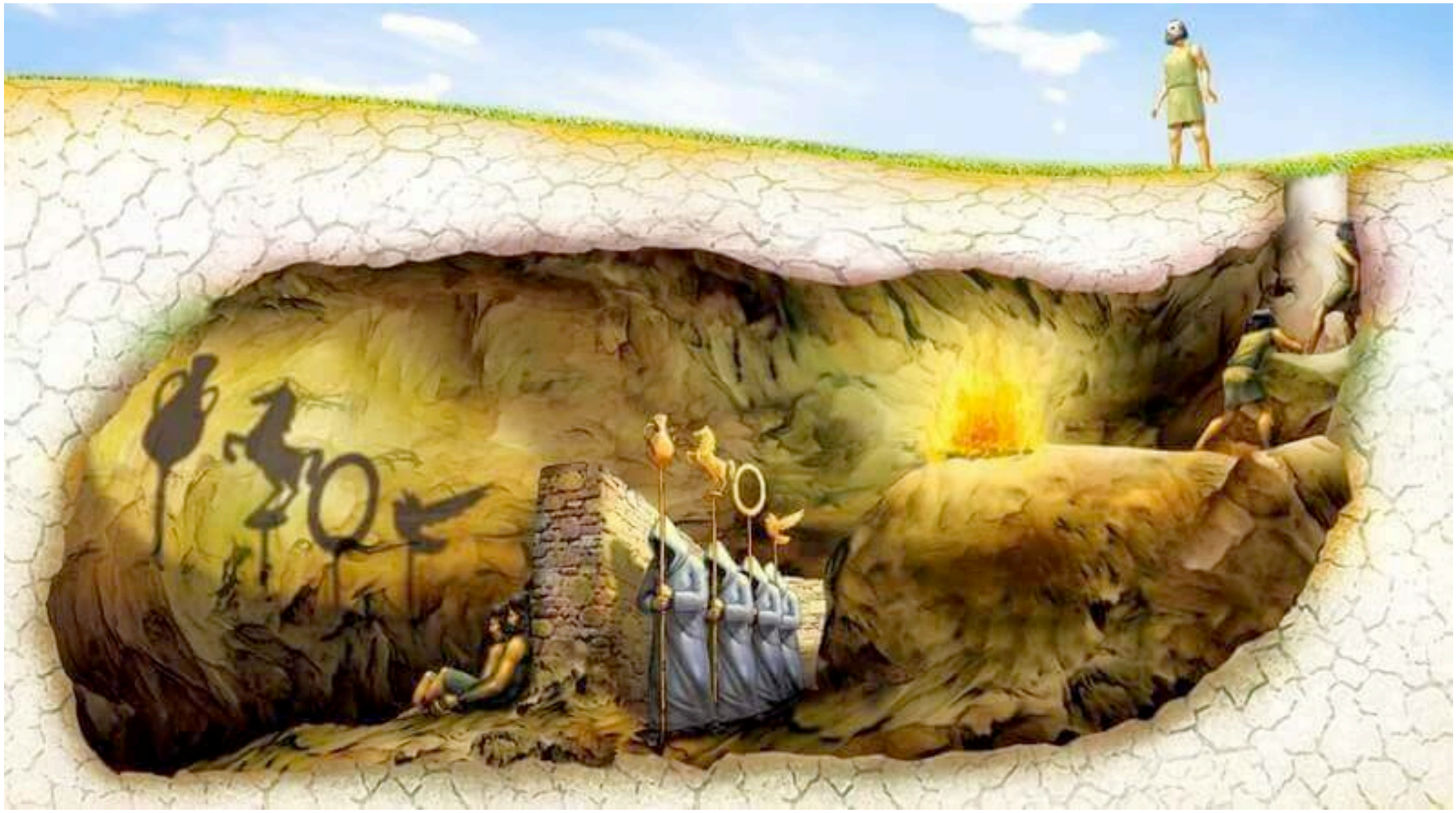
Latent variables



Plato's myth of the cave - image from https://www.studiobinder.com/blog/platos-allegory-of-the-cave/

How do you do it?

Latent variables

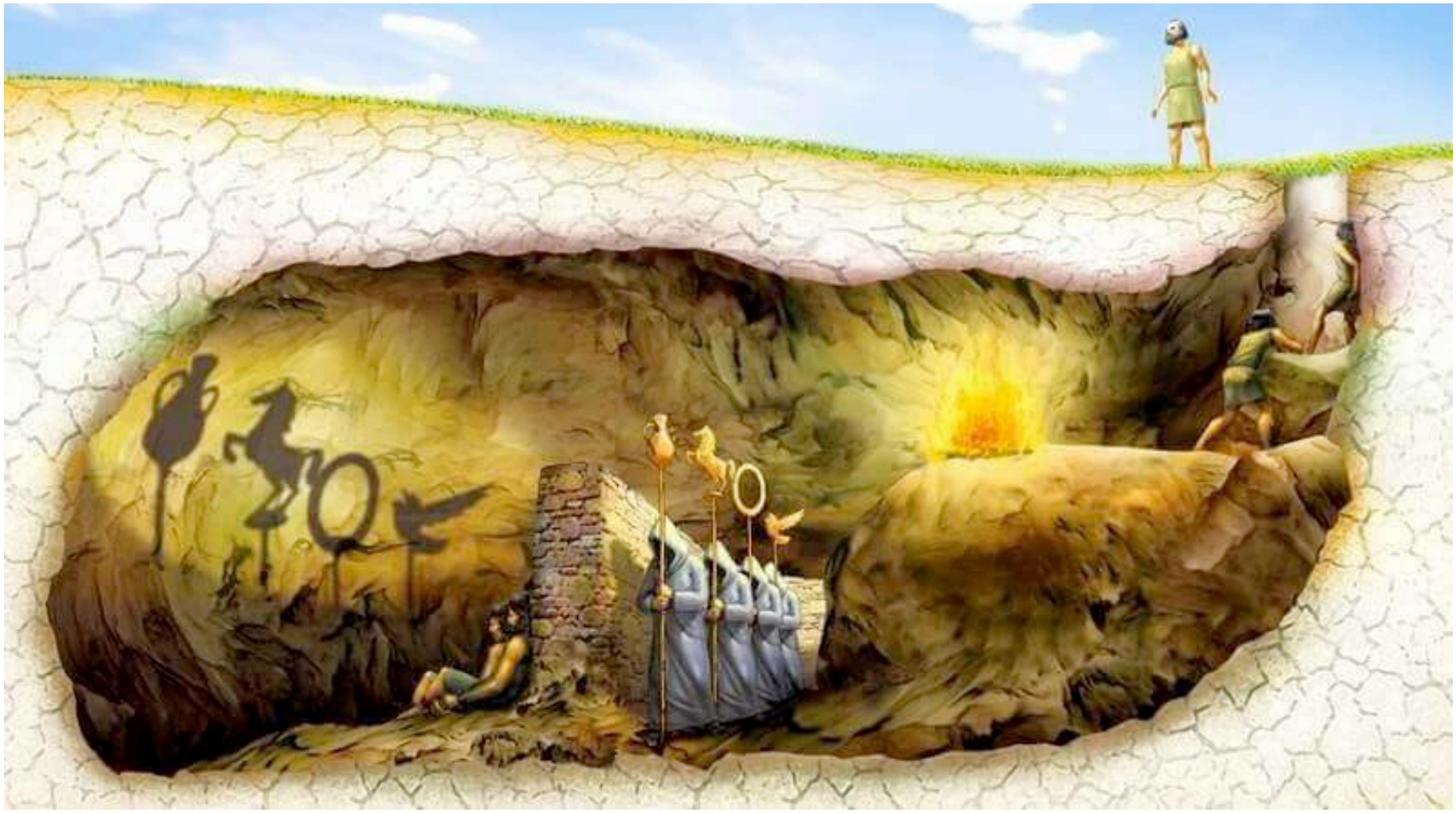


Observed

Plato's myth of the cave - image from https://www.studiobinder.com/blog/platos-allegory-of-the-cave/

How do you do it?

Latent variables



Observed Latent

Plato's myth of the cave - image from https://www.studiobinder.com/blog/platos-allegory-of-the-cave/

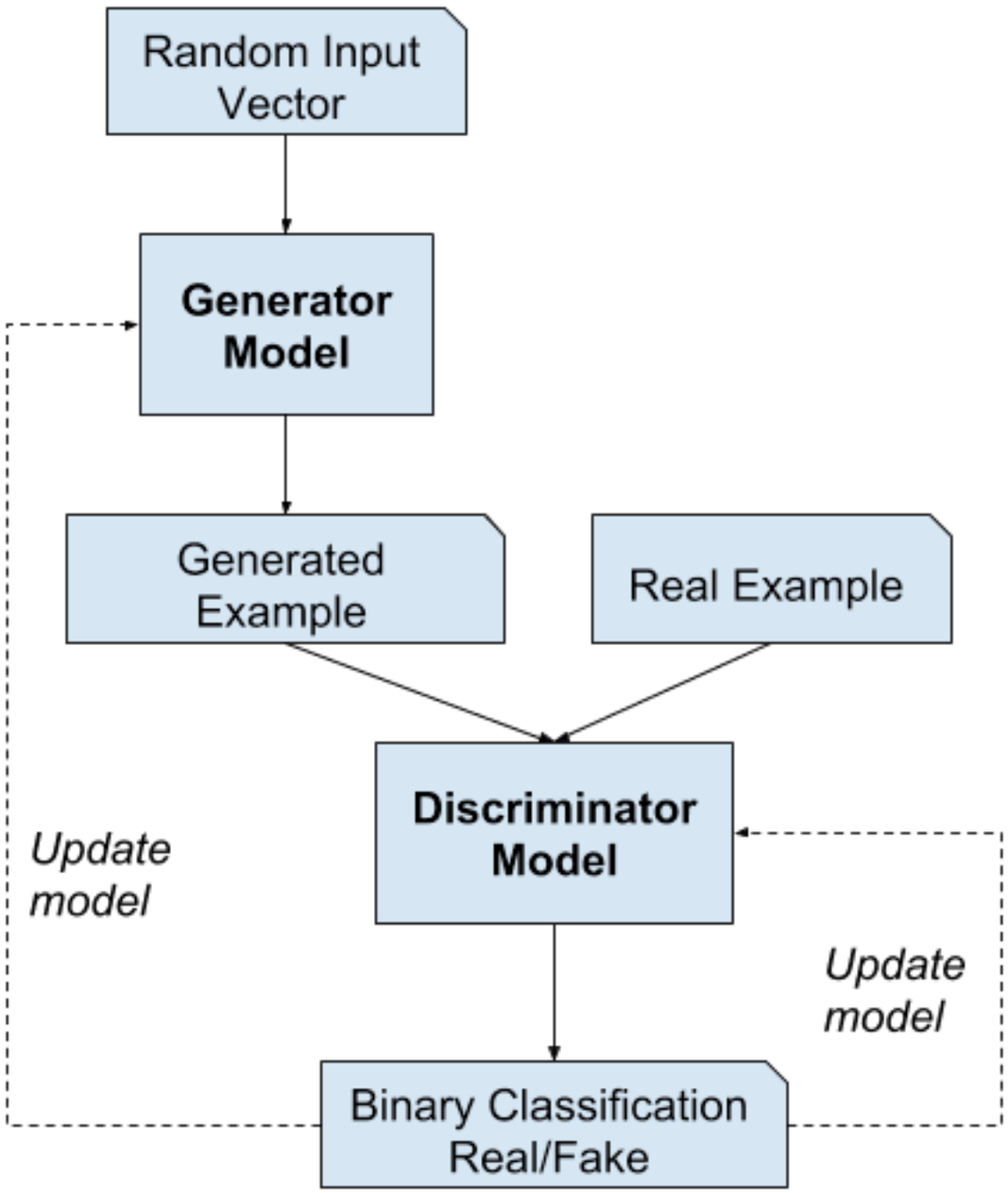
How do you do it?

DL for dimension reduction

From http://introtodeeplearning.com/

This can use e.g. encoder/decoder networks, autoencoders, variational autoencoders

GANs



1. Train a discriminator (D) on known data.
2. Then train the generator (G) to generate samples from noise input.
3. D tries to distinguish real and fake data.
4. G responds and tries to produce better fake data
5. And so it goes on….

From https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans/

Planning for the future - reinforcement learning

We have distinguished supervised and unsupervised learning. Reinforcement learning is slightly different:

Input: state & action to take

Goal: maximise long-term (future) reward = winning a game, surviving

Reinforcement learning

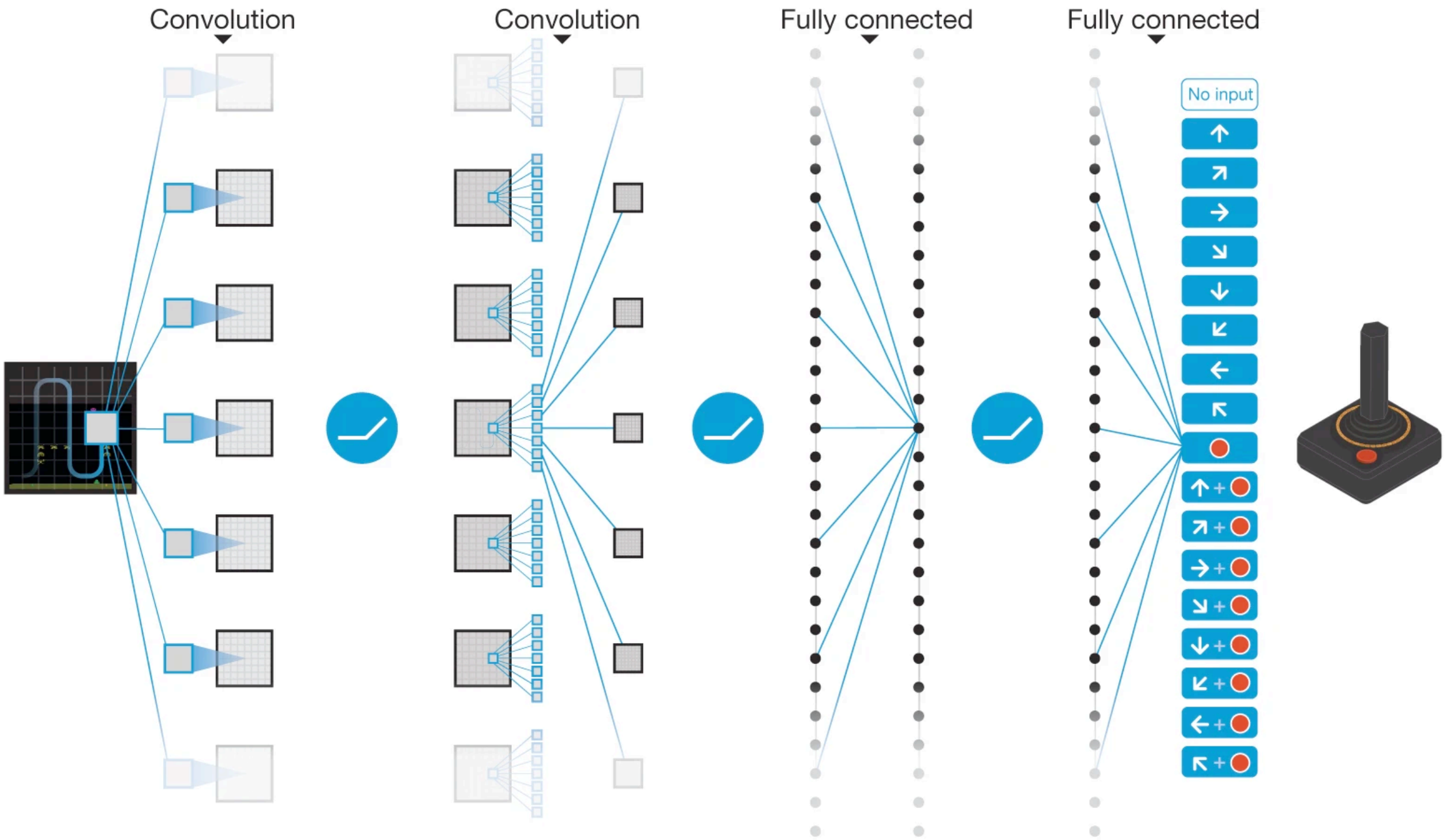
Have a set of actions a_t, and a set of states, s_{t+1}. From these you evolve a system, allowing it to interact with the surroundings and calculate a reward, r_t, which is used in the loss function: (c.f. Minh et al 2015, Nature):

$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots | s_t = s,\, a_t = a,\, \pi\right],$$

π is the policy to follow - basically what maximisation to use. This can also be learned in policy learning - typically this will be to maximise the reward at the end state.

Playing Atari games

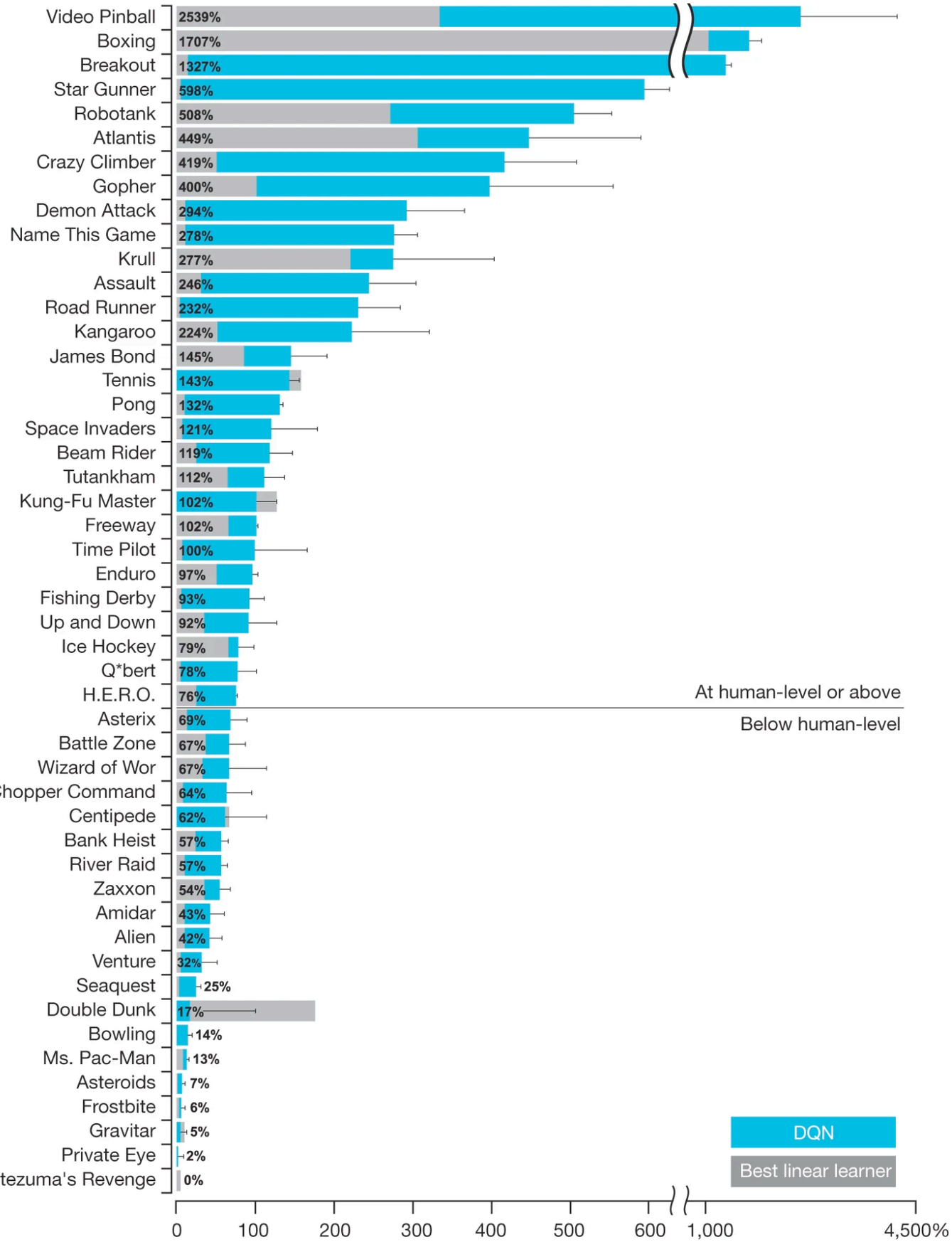
Minh et al (2015, Nature):



Playing Atari games

Minh et al (2015, Nature):

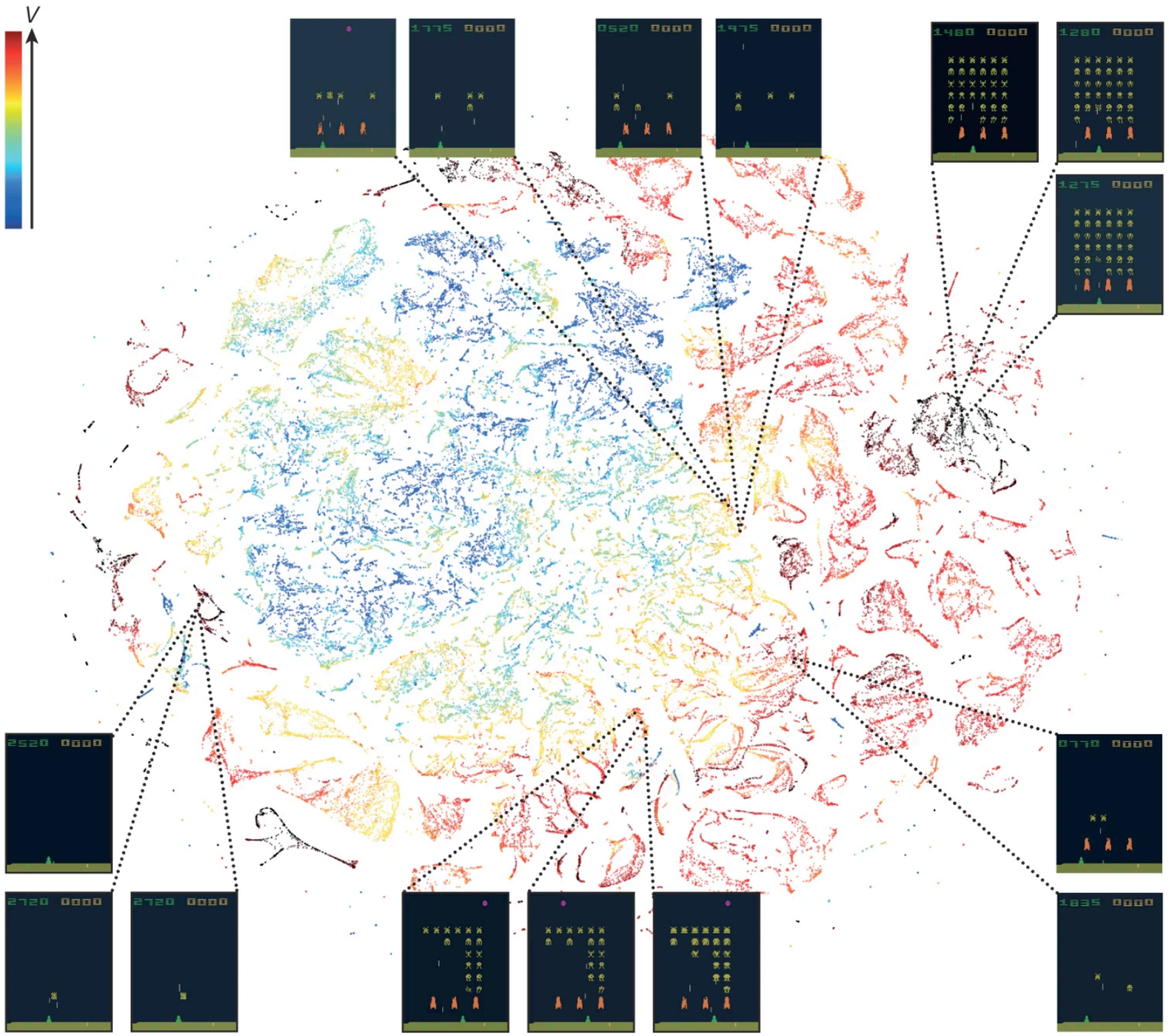
Very successful strategy and it has now >25000 citations…



Playing Atari games

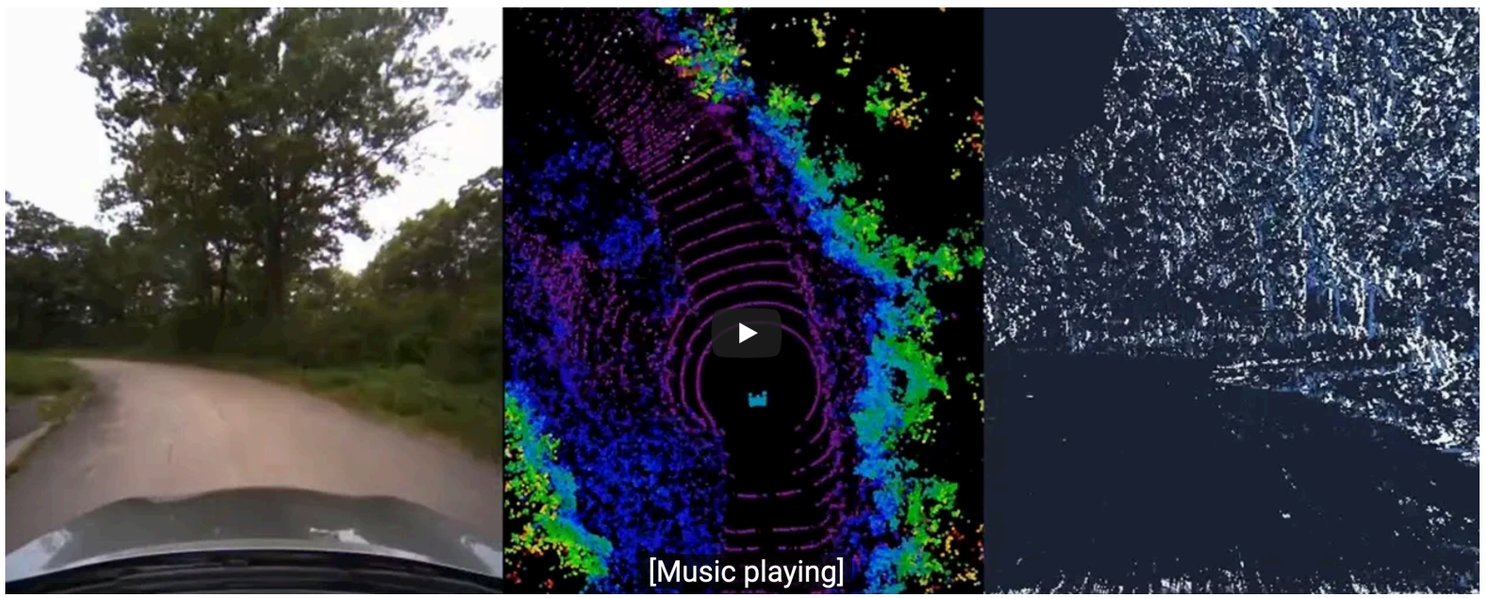
Minh et al (2015, Nature):

Using t-SNE to visualise the state of the network. The colour is the value of that position.



Reinforcement learning - widely used

Playing Go



Self-driving cars

Modern use of neural networks in astronomy

Huertas-Company & Lanousse (2023)

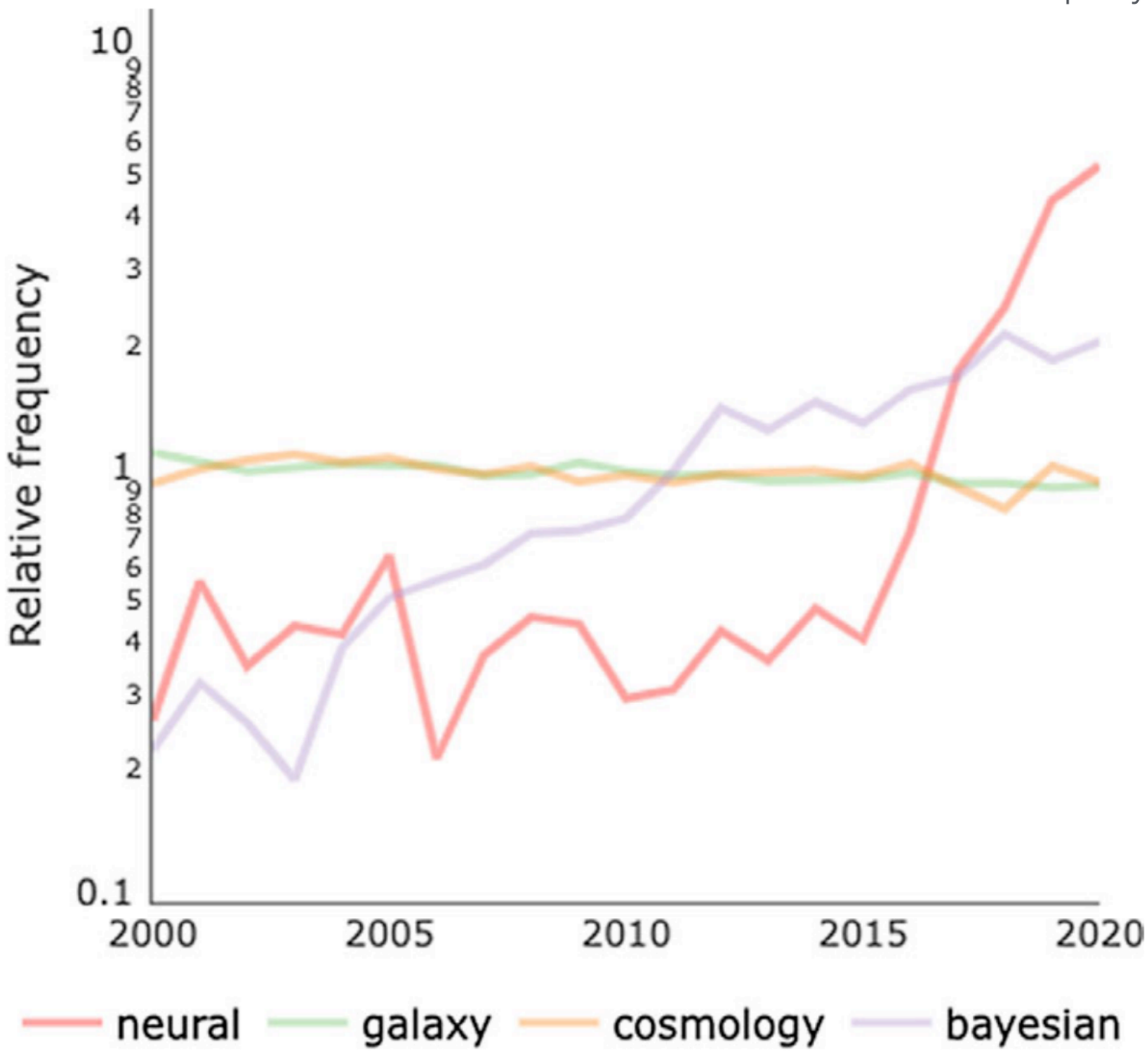


Figure 1. Relative change of the number of papers on arXiv:astro-ph with different keywords in the abstract as a function of time. The number of works mentioning neural networks in the abstract has experienced an unprecedented growth in the last ∼6 yr, significantly steeper than other topic in astrophysics. Source: ArXivSorter.

Write code in TensorFlow to train a neural network for image classification

Sure, here's an example code in TensorFlow to train a neural network for image classification using the CIFAR-10 dataset. This code uses a convolutional neural network (CNN) with three convolutional layers and two fully connected layers.

ork for image classification

rain a neural network for image
code uses a convolutional neural network
ully connected layers.

python Copy code

```python
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool

# Load CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = cifar10.load_data()

# Normalize pixel values between 0 and 1
x_train = x_train / 255.0
x_test = x_test / 255.0

# Convert labels to one-hot encoded vectors
y_train = tf.keras.utils.to_categorical(y_train, num_classes=10)
y_test = tf.keras.utils.to_categorical(y_test, num_classes=10)

# Define model architecture
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(32,
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))

# Compile model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['a

# Train model
model.fit(x_train, y_train, batch_size=128, epochs=10, verbose=1, validation

# Evaluate model on test set
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

In this code, we first load the CIFAR-10 dataset and normalize the pixel values between 0 and 1. We also convert the labels to one-hot encoded vectors.

Next, we define the model architecture using the `**Sequential**` API from Keras. The model has three convolutional layers with increasing number of filters, followed by max pooling and dropout layers to prevent overfitting. The output of the last convolutional layer is flattened and passed through two fully connected layers with ReLU activation and a final softmax layer for classification.

We compile the model with categorical cross-entropy loss and the Adam optimizer. We then train the model on the training set for 10 epochs with a batch size of 128, and validate the model on the test set after each epoch.

Finally, we evaluate the model on the test set and print the test loss and accuracy.

ork for image classification

rain a neural network for image
code uses a convolutional neural network
ully connected layers.

NB: Not tested!

```python
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool

# Load CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = cifar10.load_data()

# Normalize pixel values between 0 and 1
x_train = x_train / 255.0
x_test = x_test / 255.0

# Convert labels to one-hot encoded vectors
y_train = tf.keras.utils.to_categorical(y_train, num_classes=10)
y_test = tf.keras.utils.to_categorical(y_test, num_classes=10)

# Define model architecture
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(32,
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))

# Compile model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['a

# Train model
model.fit(x_train, y_train, batch_size=128, epochs=10, verbose=1, validation

# Evaluate model on test set
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

In this code, we first load the CIFAR-10 dataset and normalize the pixel values between 0 and 1. We also convert the labels to one-hot encoded vectors.

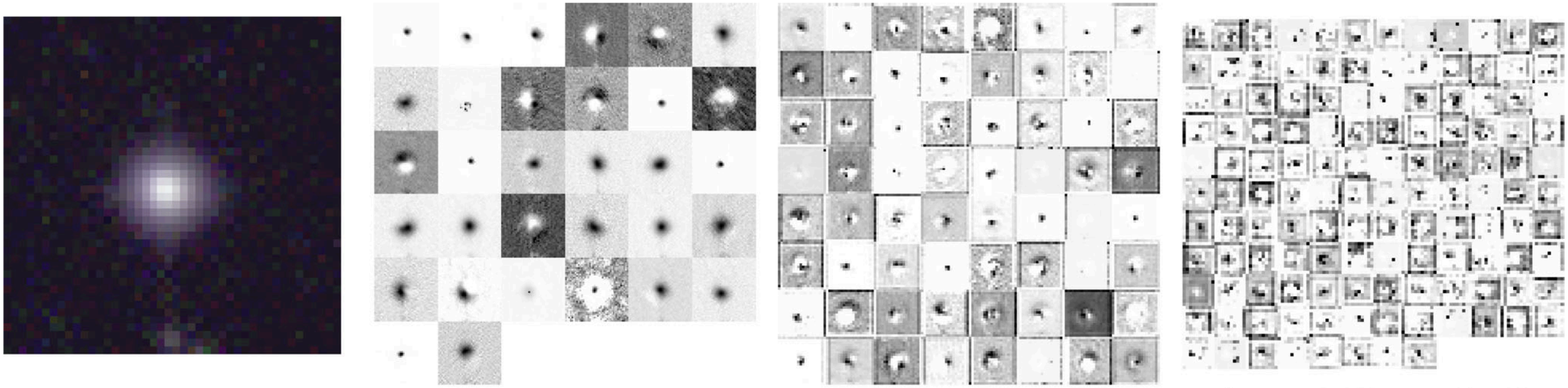
Next, we define the model architecture using the `Sequential` API from Keras. The model has three convolutional layers with increasing number of filters, followed by max pooling and dropout layers to prevent overfitting. The output of the last convolutional layer is flattened and passed through two fully connected layers with ReLU activation and a final softmax layer for classification.

We compile the model with categorical cross-entropy loss and the Adam optimizer. We then train the model on the training set for 10 epochs with a batch size of 128, and validate the model on the test set after each epoch.

Finally, we evaluate the model on the test set and print the test loss and accuracy.

Star-galaxy separation

Kim & Brunner (2015)



(a) Input (5 bands×44×44) (b) Layer 1 (32 maps×40×40) (c) Layer 3 (64 maps×20×20) (d) Layer 6 (128 maps×10×10)

11 layers in total: 8 convolutional, three fully connected.

Leaky ReLUs as their activation functions: $\sigma(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0.01x & \text{if } x < 0. \end{cases}$

10^7 parameters, but 4×10^4 images - overfitting!

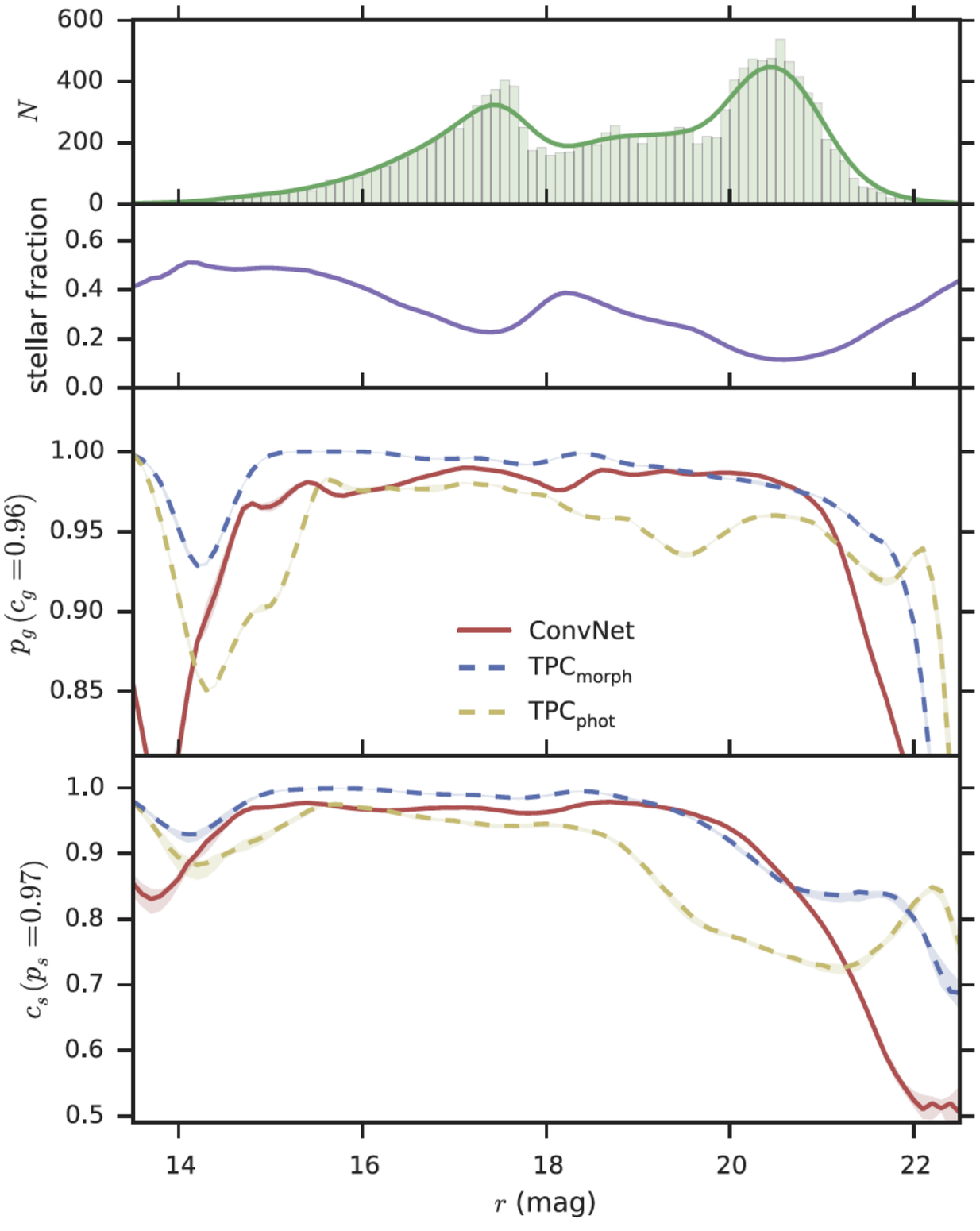
Star-galaxy separation - avoiding overfitting

Kim & Brunner (2015)

a) Data augmentation (also input rotated, reflected, translated, noised images).
b) Dropout (randomly dropping neurons) - this helps the network be more robust.

Grouping pixels together (“pooling”) so that less is kept as you go down in the network.

Star-galaxy separation - results



TPC is a random forest classifier.
As a general rule the convolutional neural network performs almost as well as this.

The advantage(?): TPC requires measurements as input, the ConvNet works straight on the images.

Transient detection

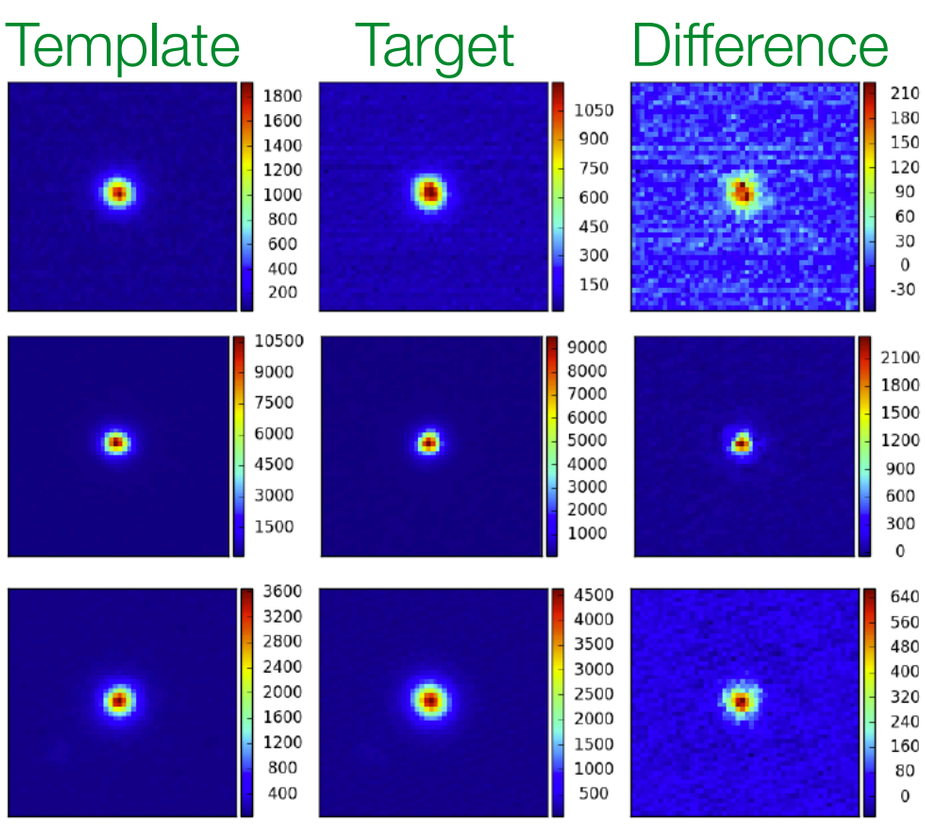
Future surveys like LSST will detect ~1 million transient events per night. Which ones should we follow up?

Analysis typically focuses on difference images & features are analysed using random forests typically.

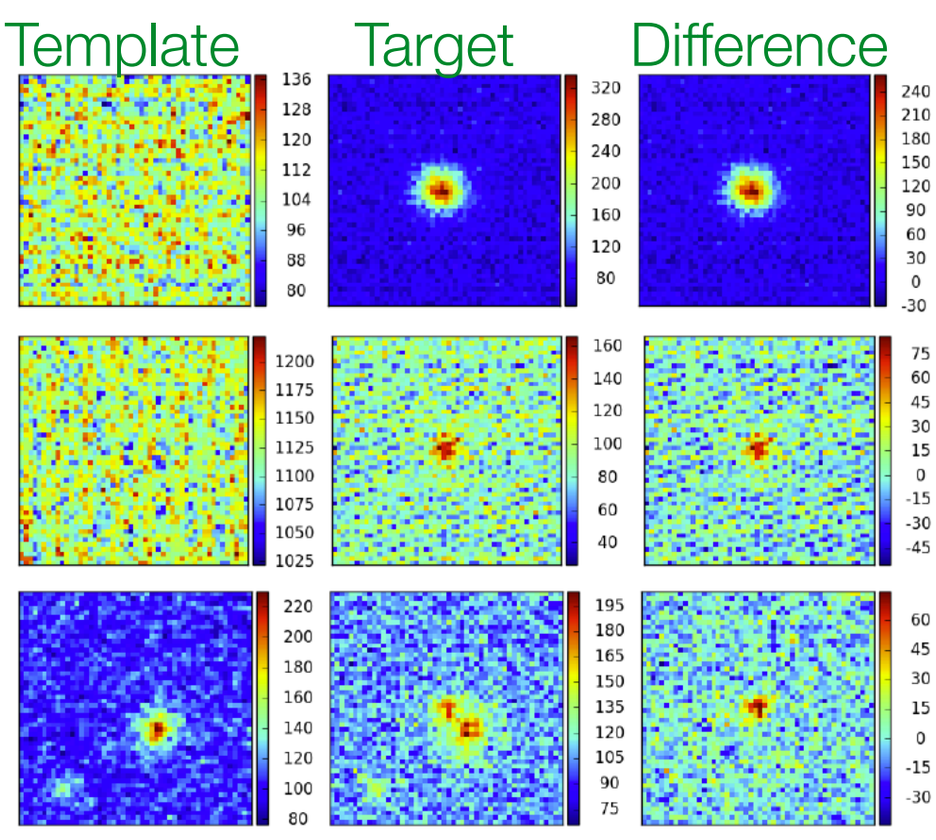
Gieseke et al (2017) uses a convolutional neural network to classify transients.

Transient detection

Gieseke et al (2017) uses a convolutional neural network to classify transients.



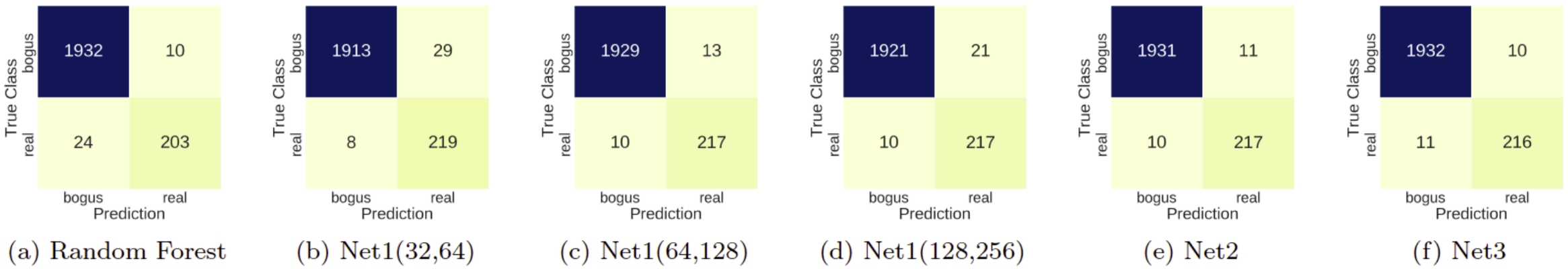
(a) Bogus



(b) Real

Transient detection

Gieseke et al (2017) uses a convolutional neural network to classify transients.

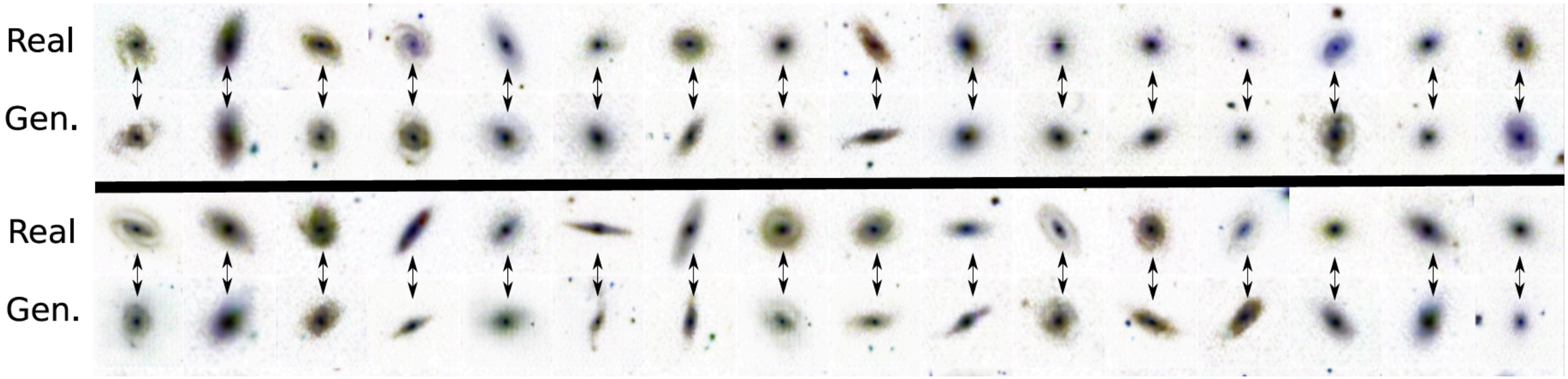


(a) Random Forest (b) Net1(32,64) (c) Net1(64,128) (d) Net1(128,256) (e) Net2 (f) Net3

They find again good performance relative to random forests, but not clearly better.

Generating new images

Ravanbakhsh et al (2016) -

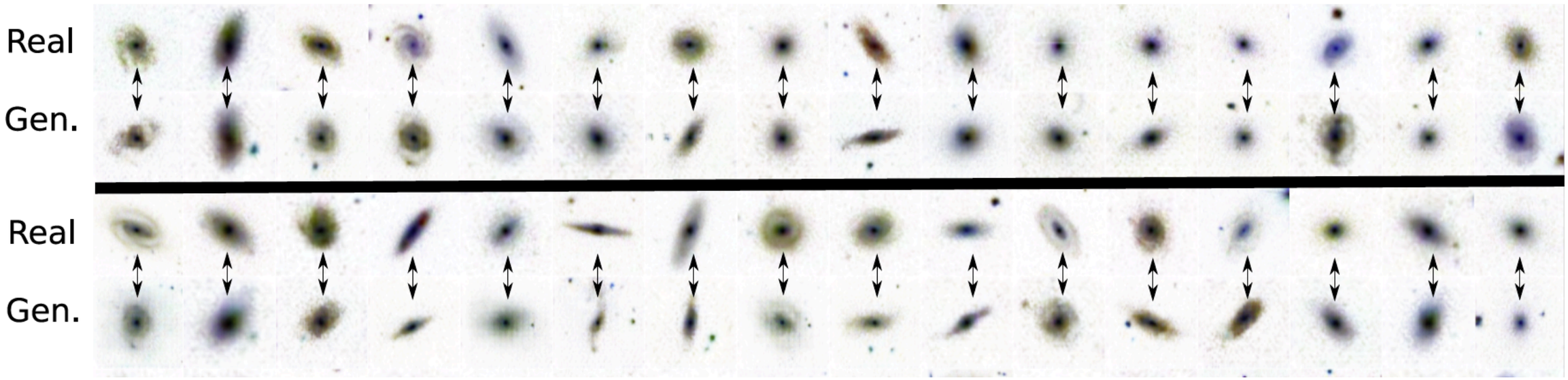


Use neural networks that can generate realistic images:

- Variational Autoencoder
- Generative Adversial Networks

Generating new images

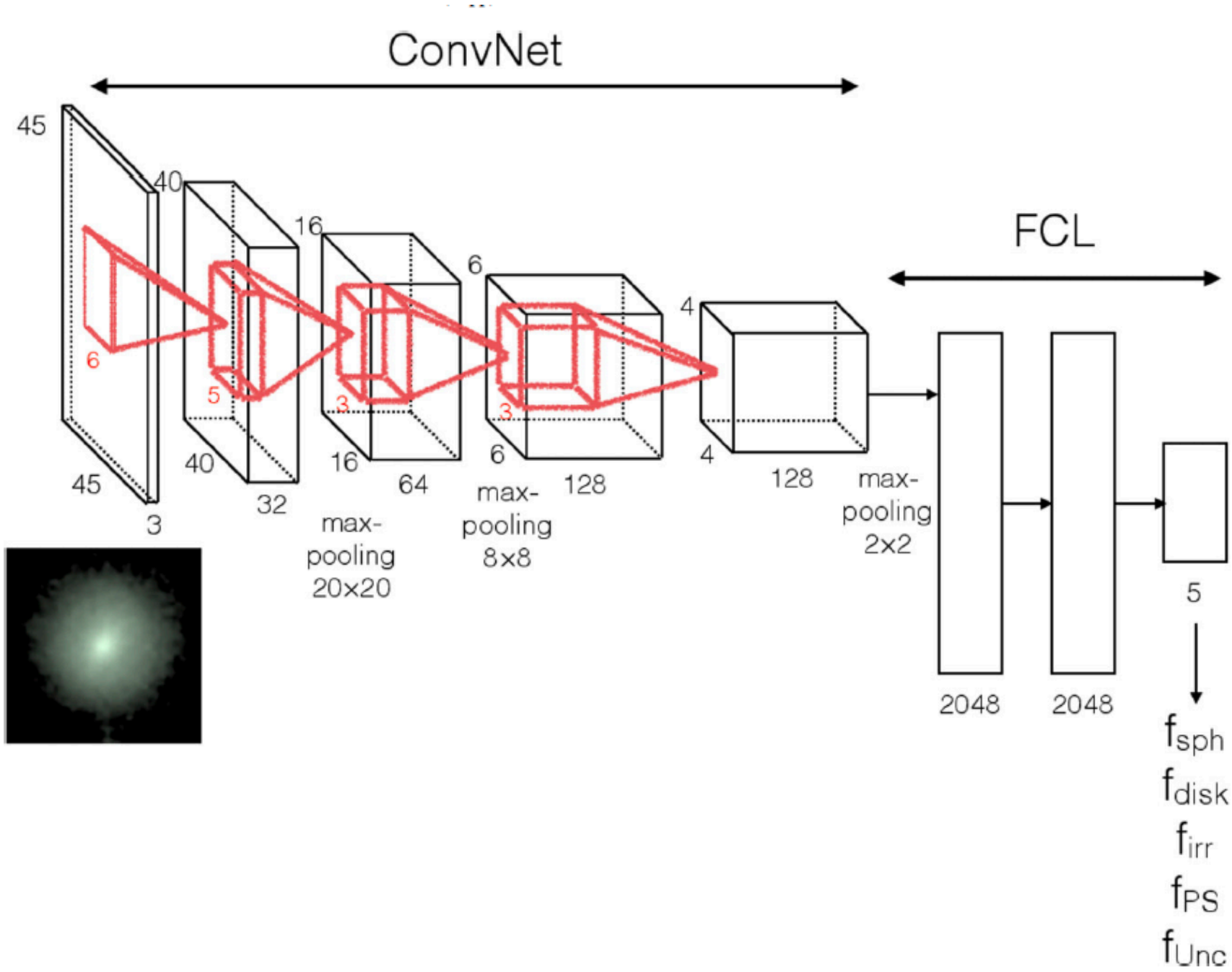
Ravanbakhsh et al (2016) -



aim: Create training samples for gravitation lensing methods etc.

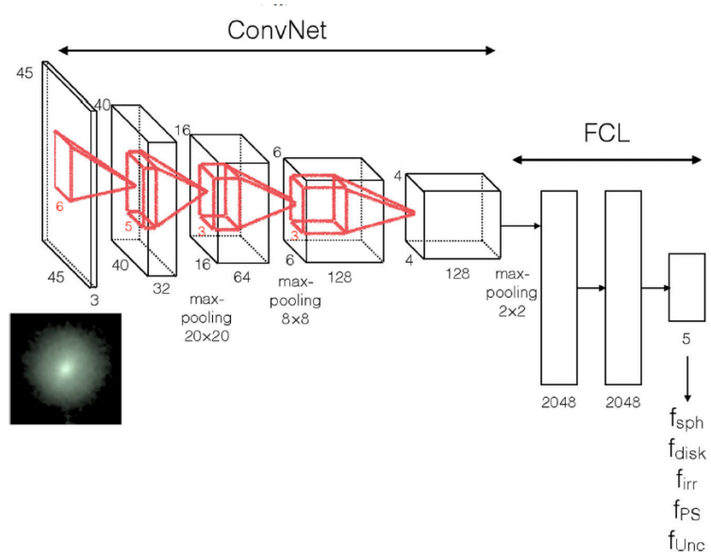
Galaxy morphology

Huertas-Company et al (2015):



Galaxy morphology

Huertas-Company et al (2015):



Convolutional neural network, ~1000 galaxies/hour on a Tesla M2090 GPU

Performance: Less than 10% error, comparable to human classifier.

Metallicity from 3-colour images

Figure 4.

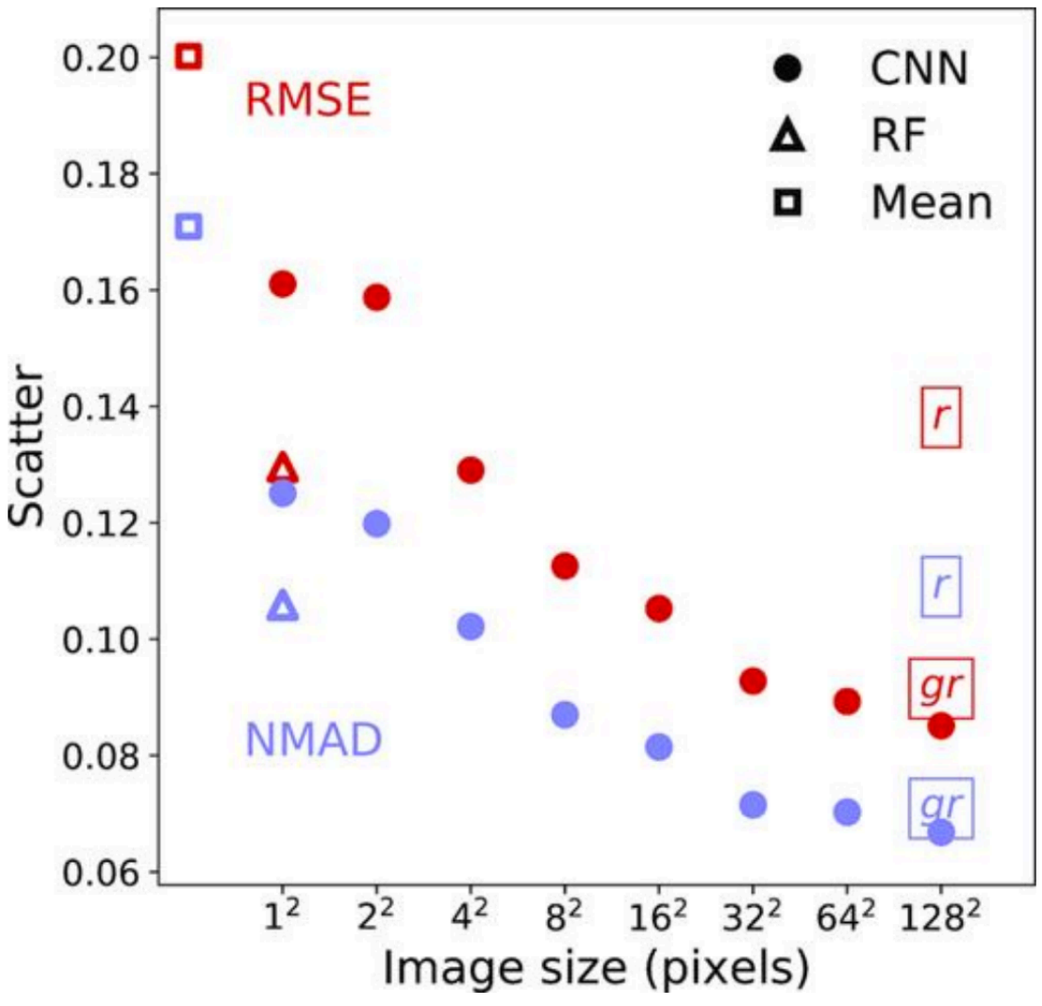
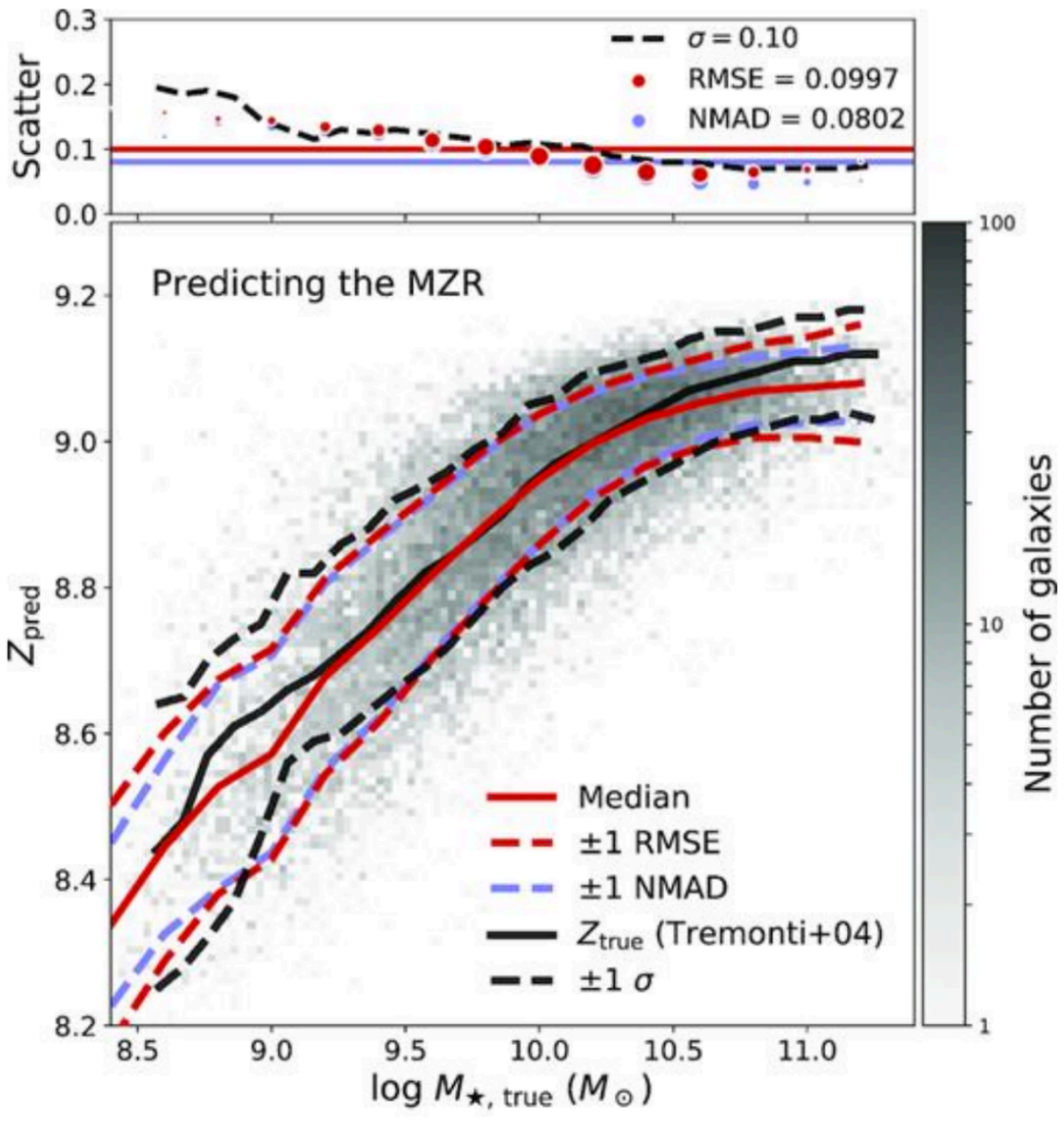


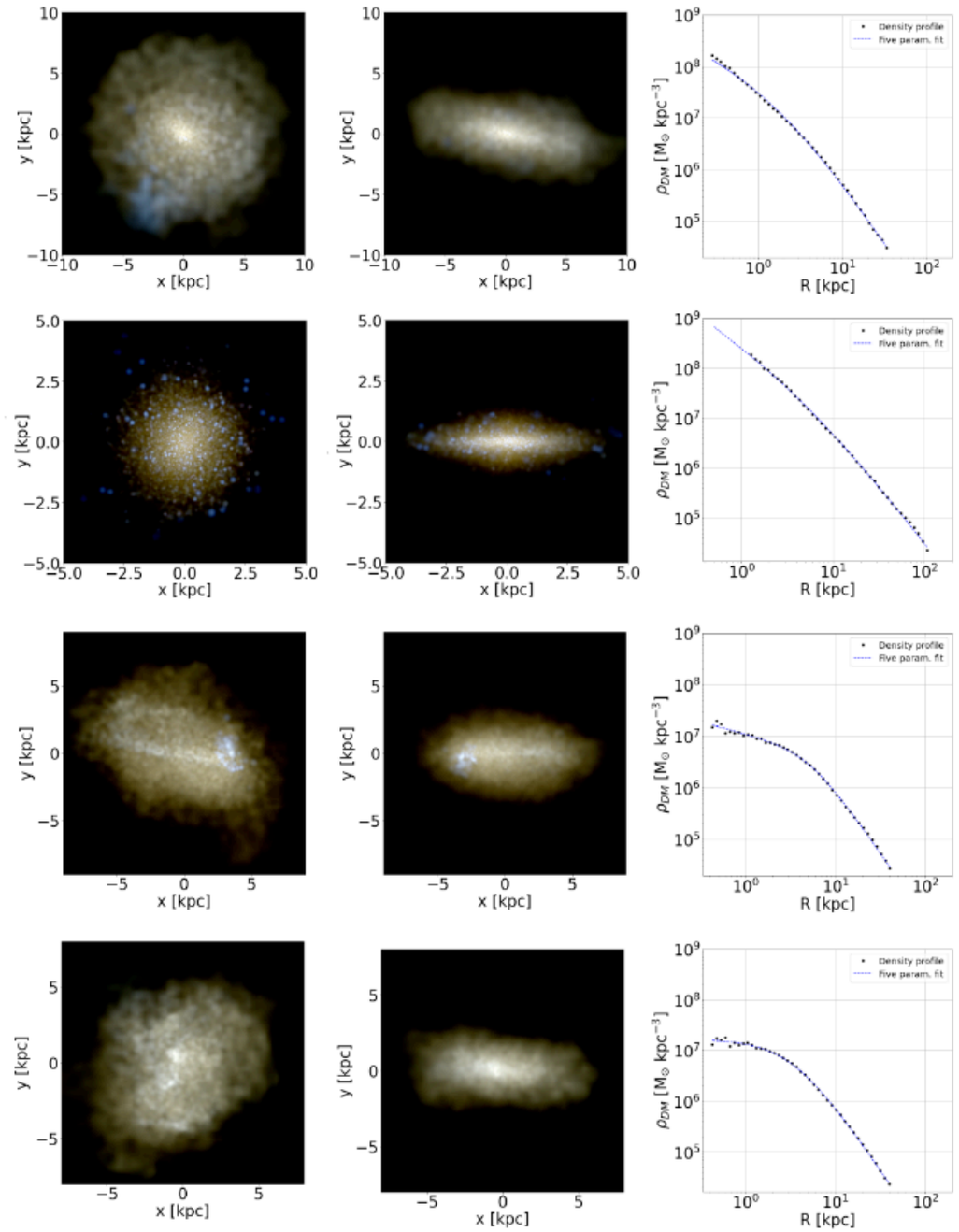
Figure 6.



Wu & Boada (2019)

Convolutional neural network. 34 layers - start with ImageNet.
Using pytorch and also using data augmentation.

Determining mass profiles using deep networks

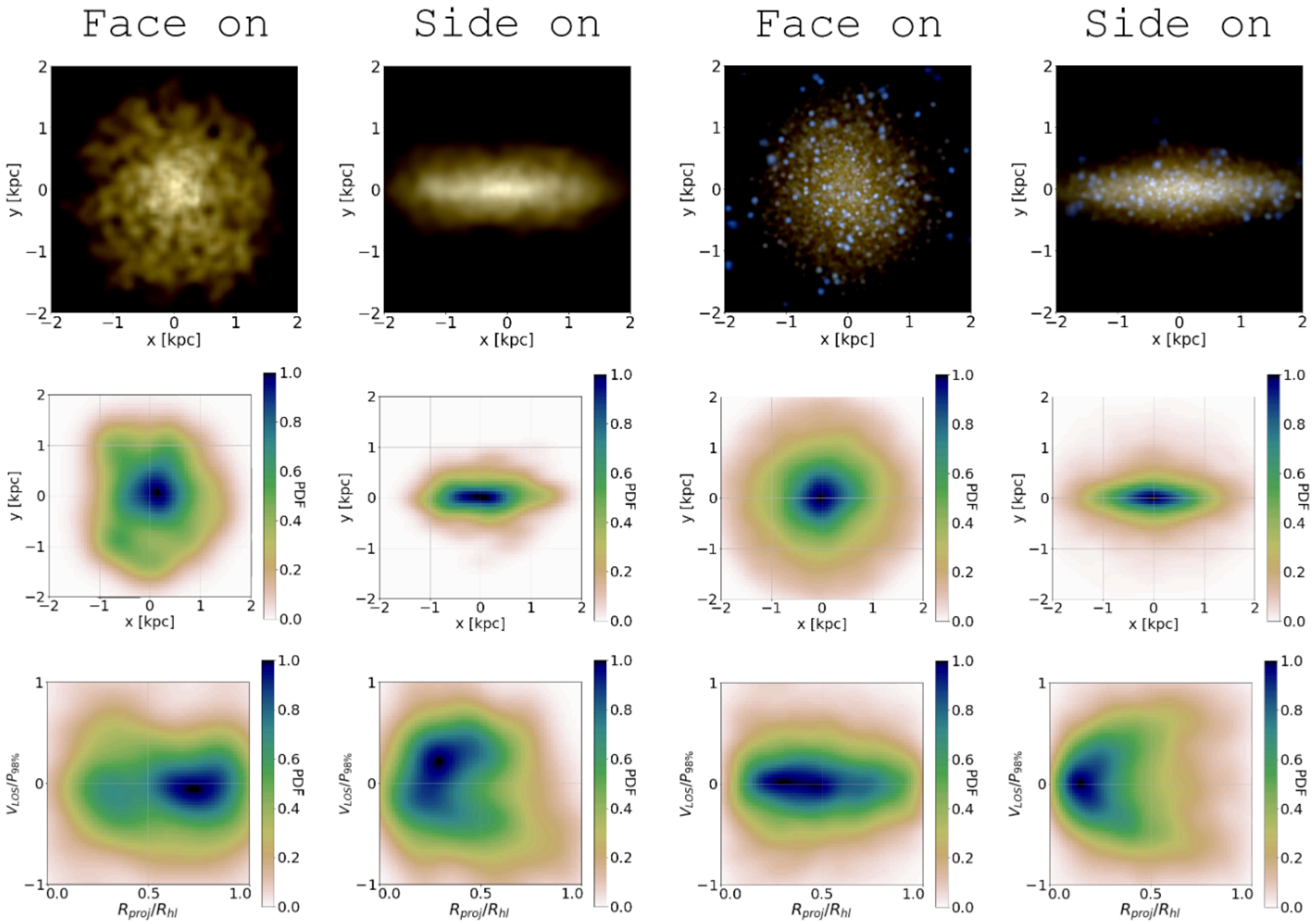


KDE → Input variables

Cored and cuspy galaxies from simulations (NIHAO, AURIGA)

Expósito-Márquez et al (2023)

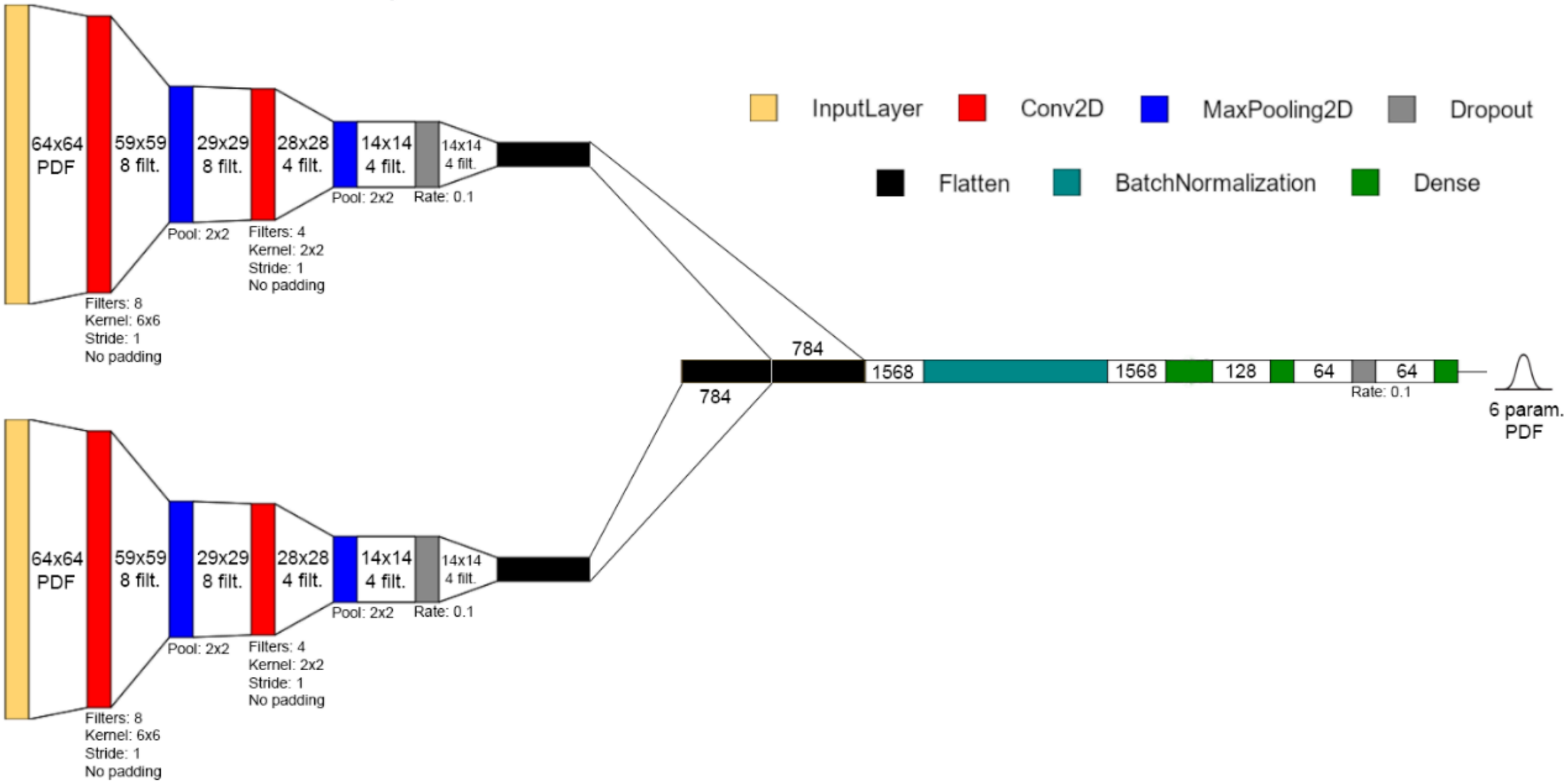
Determining mass profiles using deep networks



Expósito-Márquez et al (2023)

Determining mass profiles using deep networks

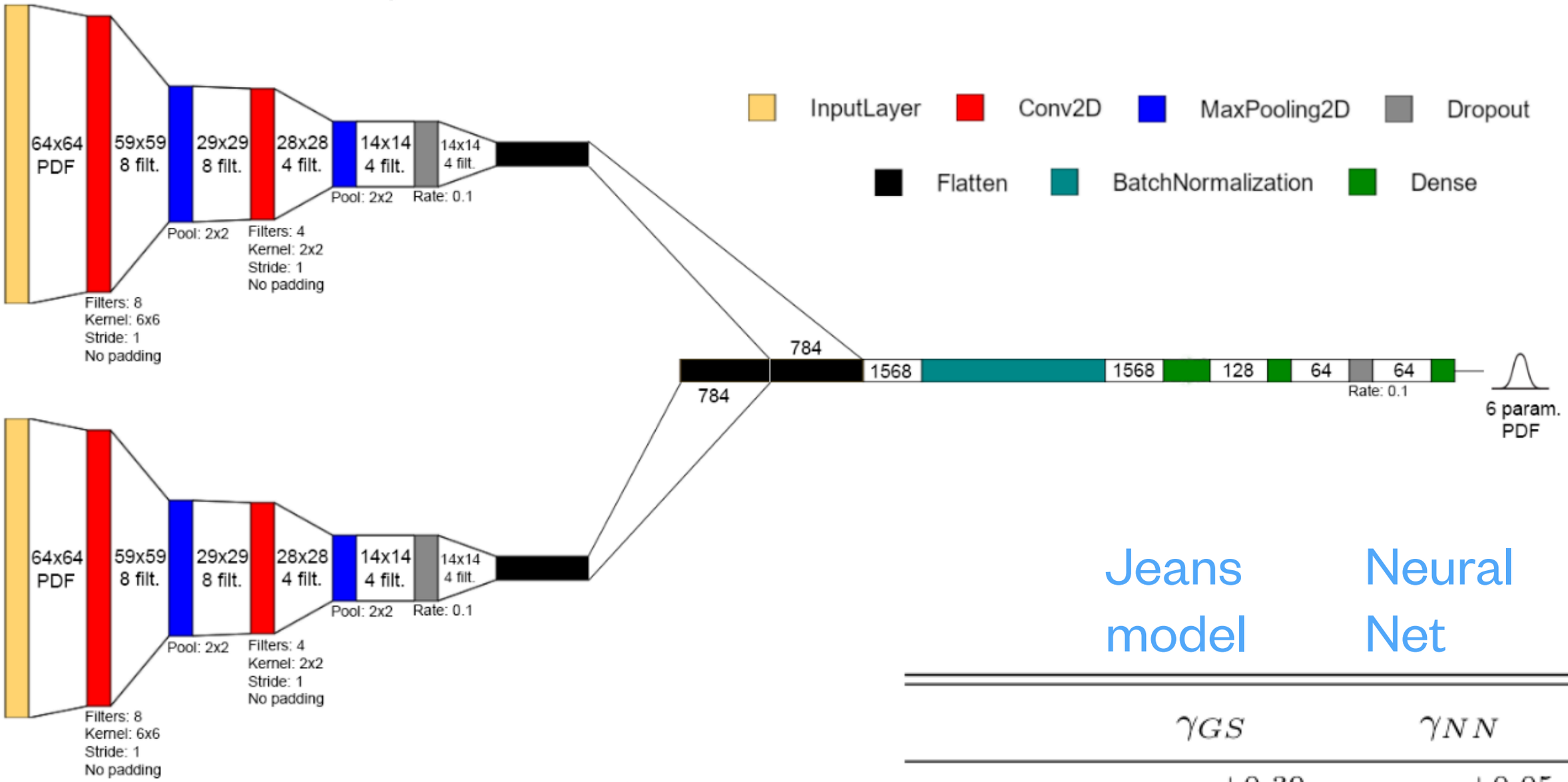
The network layout schematic:



Expósito-Márquez et al (2023)

Determining mass profiles using deep networks

The network layout schematic:

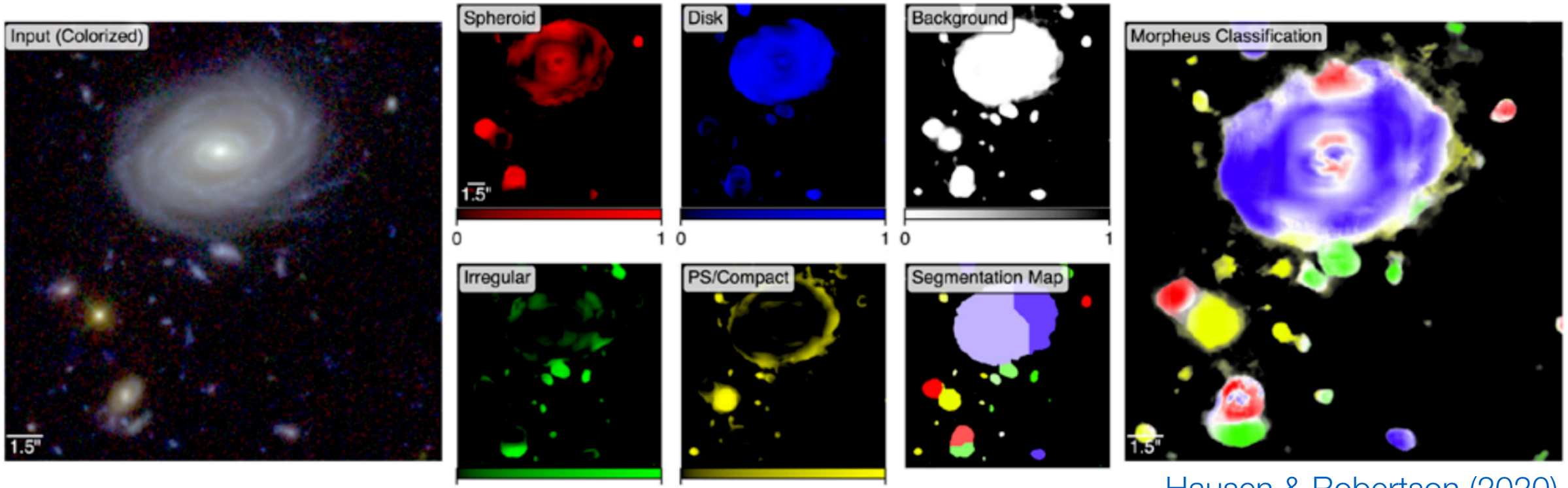


Results:

	Jeans model γ_{GS}	Neural Net γ_{NN}
Carina	$-1.23^{+0.39}_{-0.35}$	$-1.06^{+0.05}_{-0.04}$
Sextans	$-0.95^{+0.25}_{-0.25}$	$-1.25^{+0.25}_{-0.09}$
Fornax	$-0.30^{+0.21}_{-0.28}$	$-0.38^{+0.01}_{-0.02}$
Sculptor	$-0.83^{+0.30}_{-0.25}$	$-1.08^{+0.08}_{-0.04}$

Expósito-Márquez et al (2023)

Segmentation of images



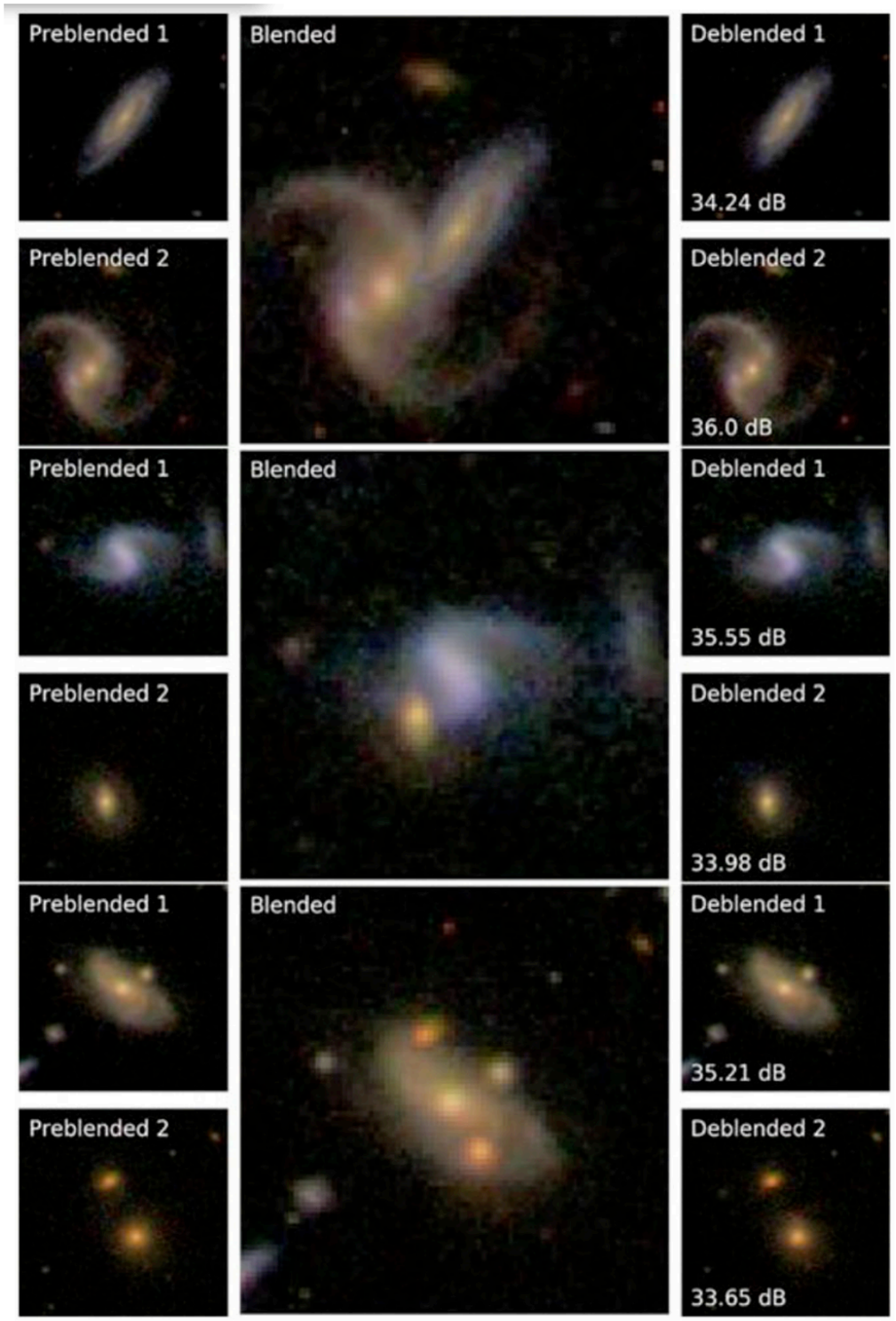
Hausen & Robertson (2020)

The most successful approach has used Unets. The lack of “truth” and training sets means this has not been widely used yet.

Deblending galaxies

This uses a GAN - in this case adding images from SDSS.

Challenge: knowing the truth.



Reiman & Göhre (2019)

Going from apples to oranges

The lack of a ground truth is frustrating and leads on to wonder if something else could be done.

Logical choice: train on simulations where you know the truth.

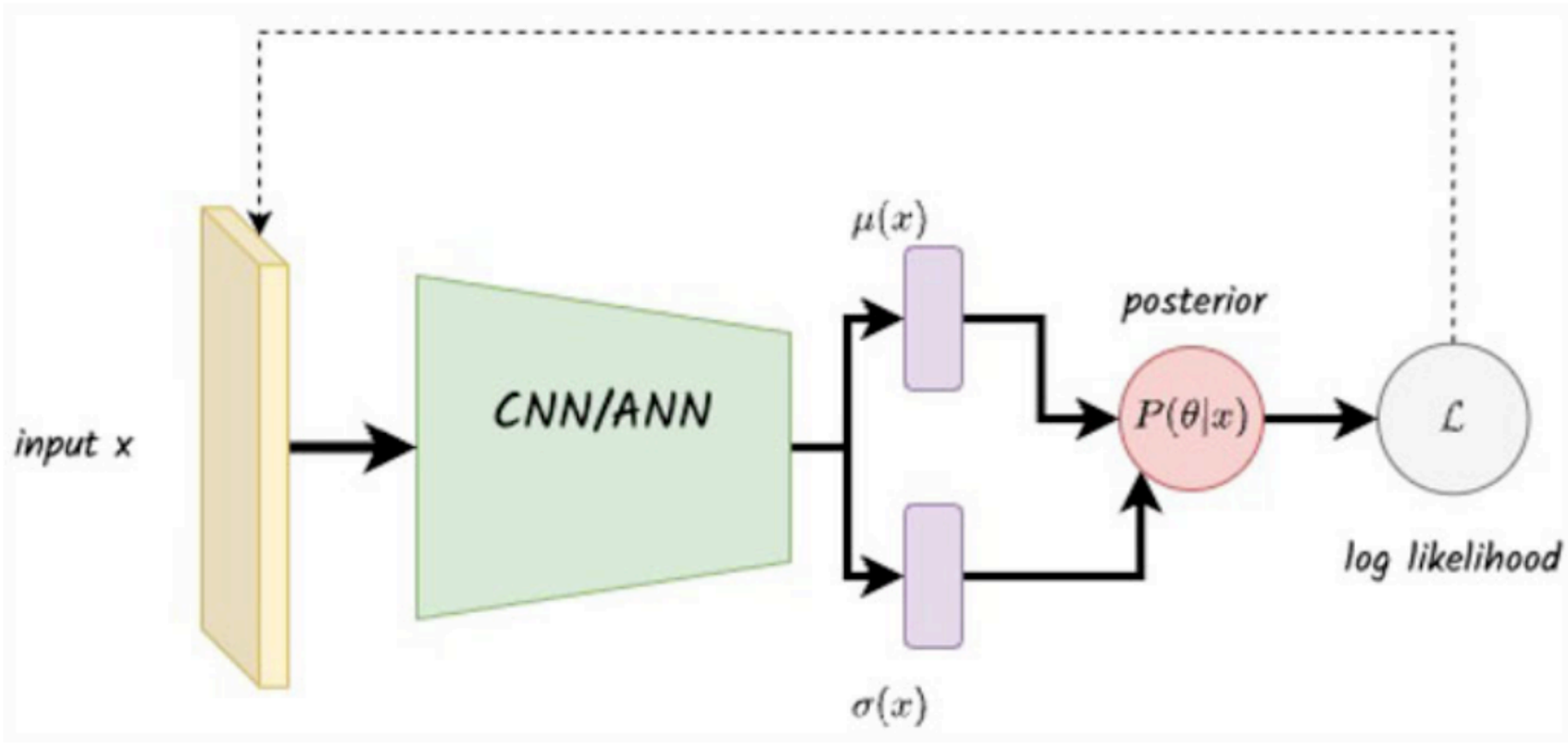
Problem: simulations are not the real world….

Enter: transfer learning

In this case we *reuse* a previously trained model when training on new data. Thus by training on simulations and then cleverly transfering this knowledge to real data we might get much better results.

Estimating parameters

We would really like to get likelihood distributions out - some probabilistic NN models are then good options



A Mixture Density Model is a kind of CNN that gives likelihoods out. In many ways it is similar to a mixture model (e.g. Gaussian Mixture Model) but within a neural network.

Tools for deep learning

Toolkits:

Theano (http://deeplearning.net/software/theano/)

Torch (http://torch.ch/) & **pyTorch** (http://pytorch.org/)

TensorFlow (https://pythonhosted.org/nolearn/)

MXNet (http://mxnet.incubator.apache.org/)

Interfaces:

Lasagne (http://lasagne.readthedocs.io/en/latest/) Used by Kim & Brunner (2015)

nolearn (https://pythonhosted.org/nolearn/) Used by Gieseke et al (2017)

Keras (https://keras.io/) Probably the simplest interface today

Example use - Keras

Sequential setup:

Sequential →

Decide on loss function, optimizer, activation functions and various parameters

```
model.compile()
model.fit()
model.predict()
```

Example use - Keras

Sequential setup:

```
import numpy as np
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dense
```

Example use - Keras

Using the library of stellar spectra from last lecture

```
model = Sequential()

model.add(Dense(n_data, input_dim=n_data,
                activation='relu'))
model.add(Dense(250, activation='relu'))
model.add(Dense(100, activation='relu'))
model.add(Dense(1, activation='linear'))
```

Example use - Keras

Using the library of stellar spectra from last lecture

```
model = Sequential()

model.add(Dense(n_data, input_dim=n_data,
                activation='relu'))
model.add(Dense(250, activation='relu'))
model.add(Dense(100, activation='relu'))
model.add(Dense(1, activation='linear'))
```

Predict MK class

Example use - Keras

Using the library of stellar spectra from last lecture

```
model.compile(optimizer='Adam',
              loss='mse')
```

(you can also add ‘metrics’ to get other evaluations of performance that are not used for training)

Example use - Keras

Using the library of stellar spectra from last lecture

```
model.compile(optimizer='Adam',
              loss='mse')
```

(you can also add ‘metrics’ to get other evaluations of performance that are not used for training)

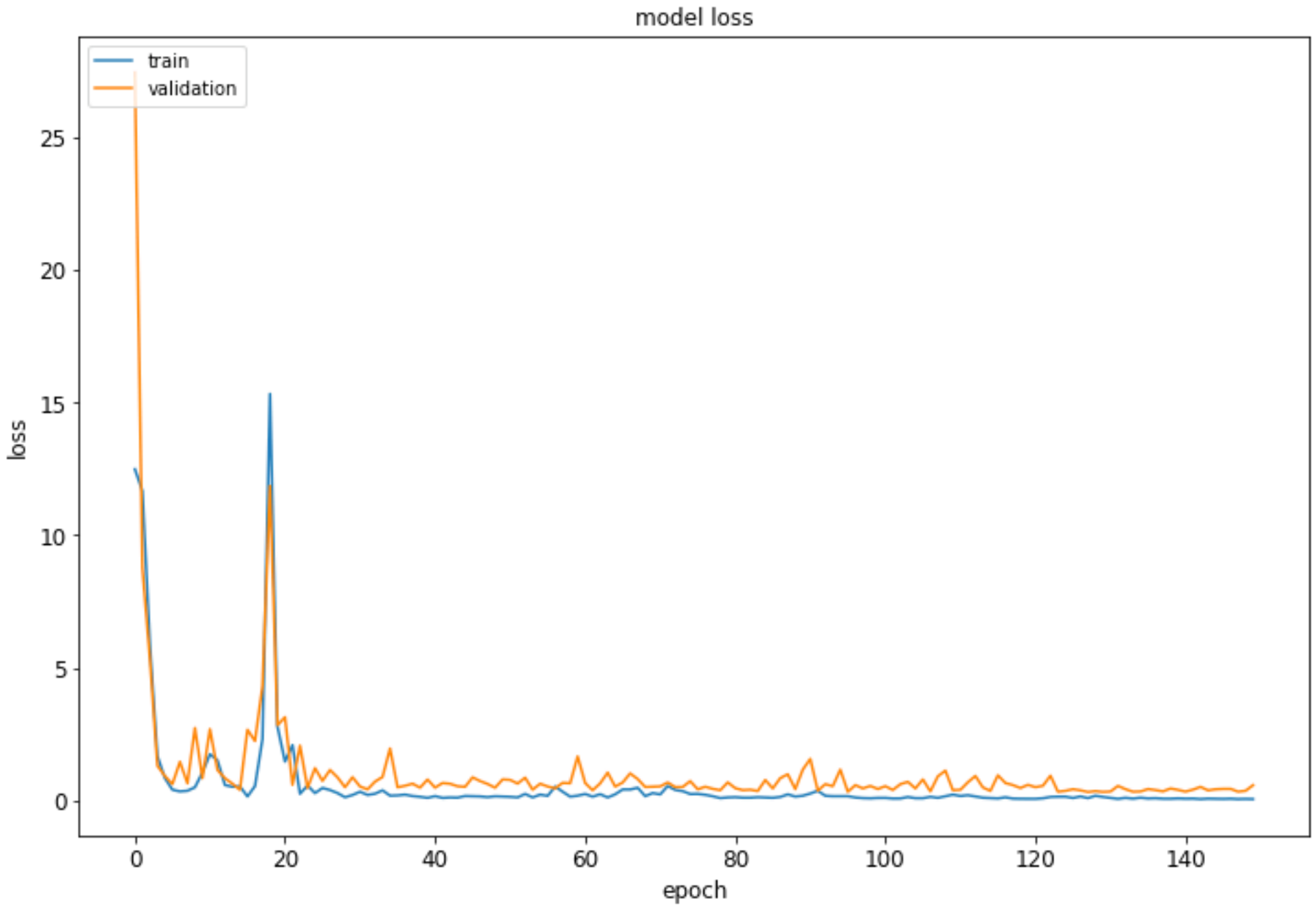
```
history = model.fit(train_X, train_y,
                    validation_data=(test_X, test_y),
                    epochs=150, verbose=0)
```

Example use - Keras

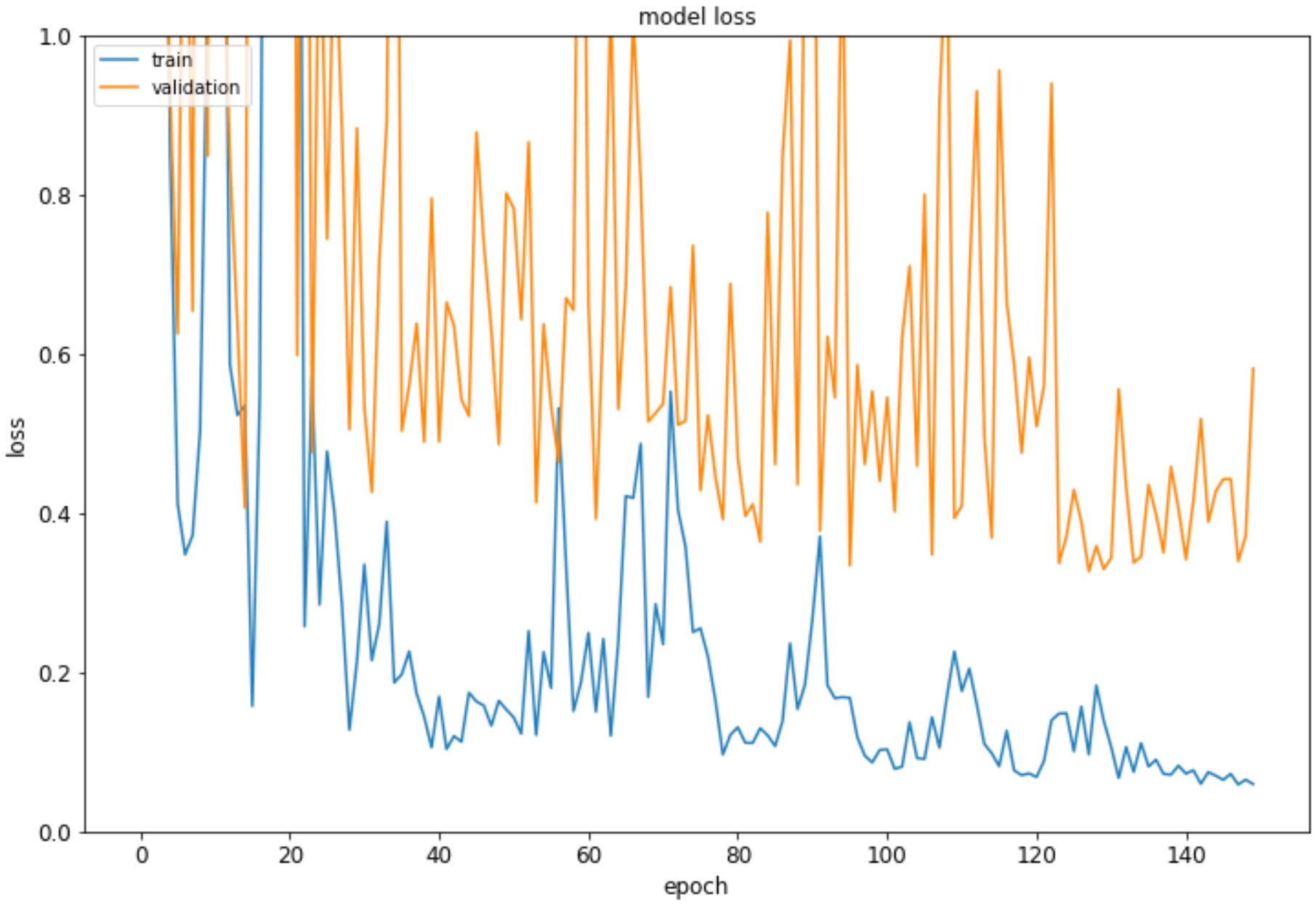
Plotting the training result:

```
fig, ax = plt.subplots(figsize=(12, 8))
ax.plot(history.history['loss'])
ax.plot(history.history['val_loss'])
ax.set_title('model loss')
ax.set_ylabel('loss')
ax.set_xlabel('epoch')
ax.legend(['train', 'validation'], loc='upper left')
```

Example use - Keras

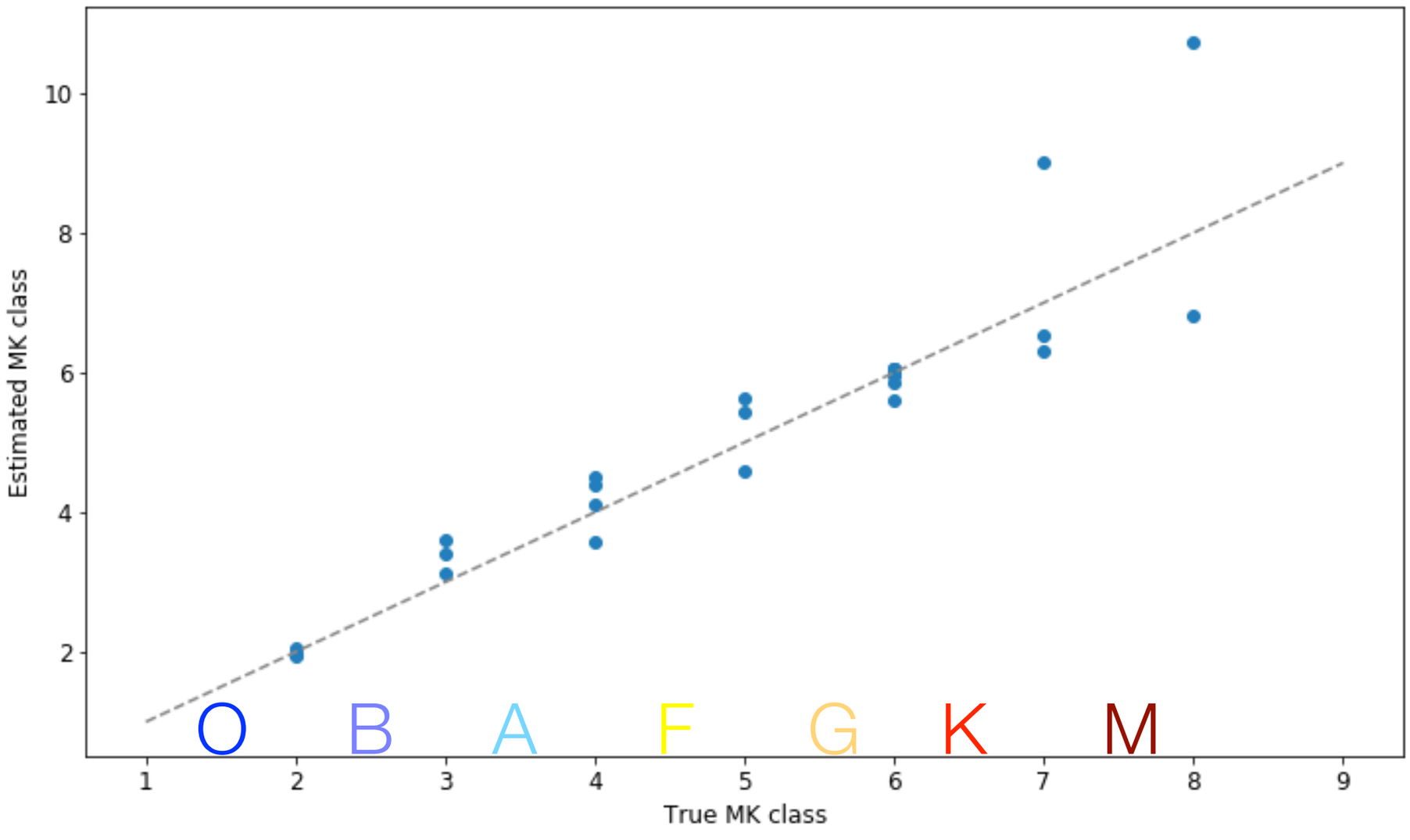


Example use - Keras



Final result on test set

```
y_pred = model.predict(test_X)
```



Scatter +/- 0.5 for most classes.

Looking back at the course

What should you use?

And when?

Looking back at the course

If you have an idea of a physical model behind, and want to learn something about this:

Regression
Bayesian/frequentist inference
Gaussian mixture models

and other techniques that are easily interpretable

Looking back at the course

If you do not have a (good) model, but plenty of empirical examples, or if you do not care about understanding why something works:

Ensemble methods, in particular random forest
kNNs
Deep Learning networks

among others


That's all Folks!