CS463/516

Lecture 17

Machine learning

For final project

Machine learning

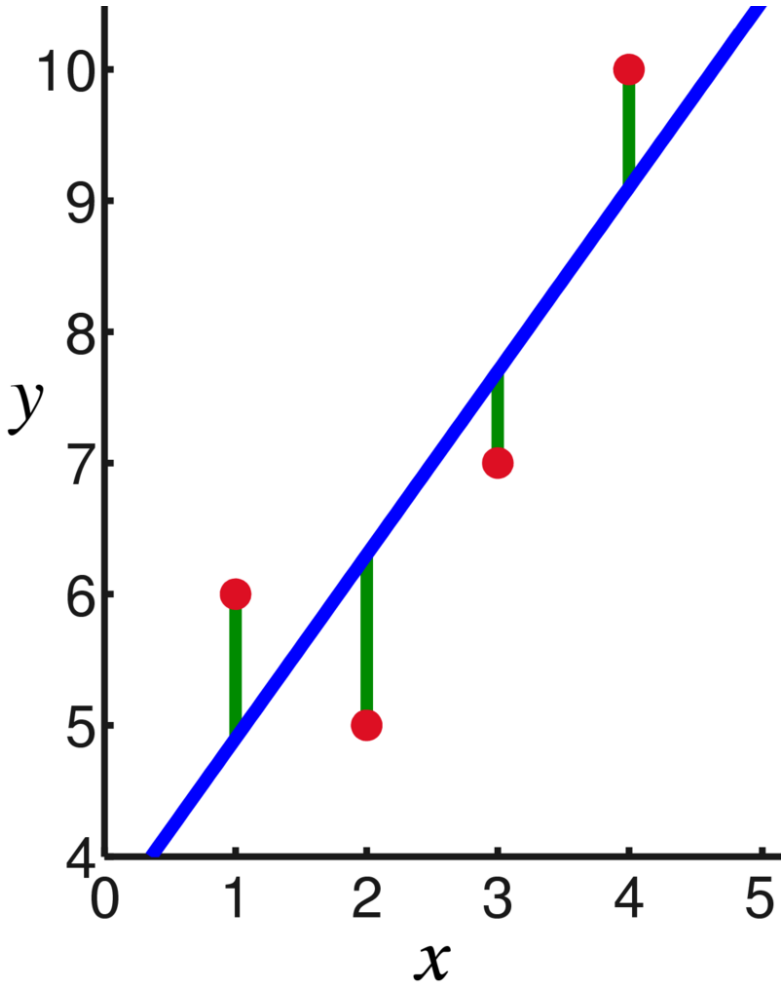
- Machine learning is the field of study that gives computers the ability to learn without being explicitly programmed
- A computer program is said to learn from experience E with respect to some task T and some performance measure P if its performance on T as measured by P improves with experience E
- Example: spam email filtering
- Your email has a spam filter that sends junk emails to ‘trash’
  - Given examples of spam emails (flagged by users) and ham emails (nonspam) the spam filter can learn to flag spam automatically
  - Examples that the system uses to learn is called the *training set*
    - Each training example is called a *training instance* or *sample*
  - Here, task T is to flag spam for new emails, experience E is the *training data* and performance measure P could be the ratio of correctly classified emails (accuracy)

Linear regression

- Linear regression models relationship between dependent and independent variable
- Given dataset $\{y_i, x_{i1}, \ldots x_{ip}\}_{i=1}^{n}$, linear regression assumes that relationship between dependent variable y and vector of regressors $\boldsymbol{x}$ is linear:
- $y_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \epsilon_i = \boldsymbol{x}_i^T \boldsymbol{\beta} + \epsilon_i$
- $i = 1, \ldots n$ and error variable ϵ_i models noise
- Can be written in matrix notation: $\boldsymbol{y} = \boldsymbol{X\beta} + \boldsymbol{\epsilon}$
  - Where $\boldsymbol{y}$ is a vector of *observed values,* $\boldsymbol{X}$ is the *design matrix* of row vectors $\boldsymbol{x}_i$, $\boldsymbol{\beta}$ is a $(p+1)$-dimensional *parameter vector* where β_0 is the intercept term
- Inference in linear regression focuses on $\boldsymbol{\beta}$:

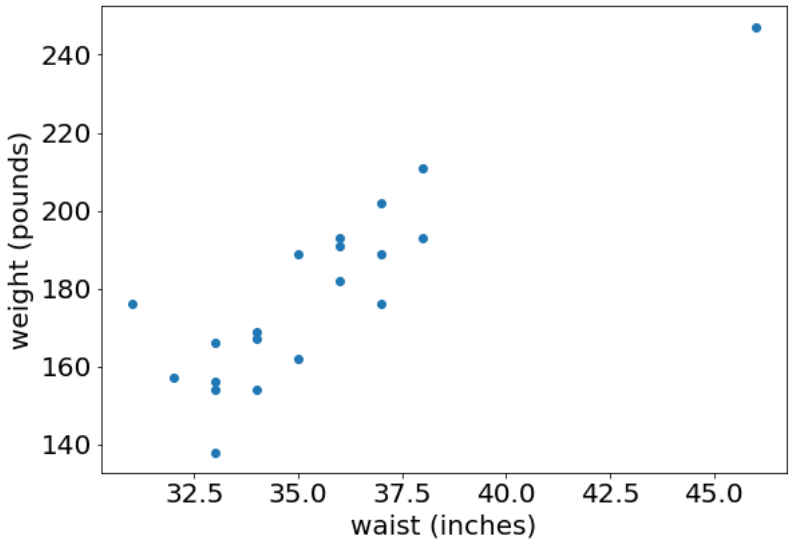
$$\boldsymbol{\beta} = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{y}$$

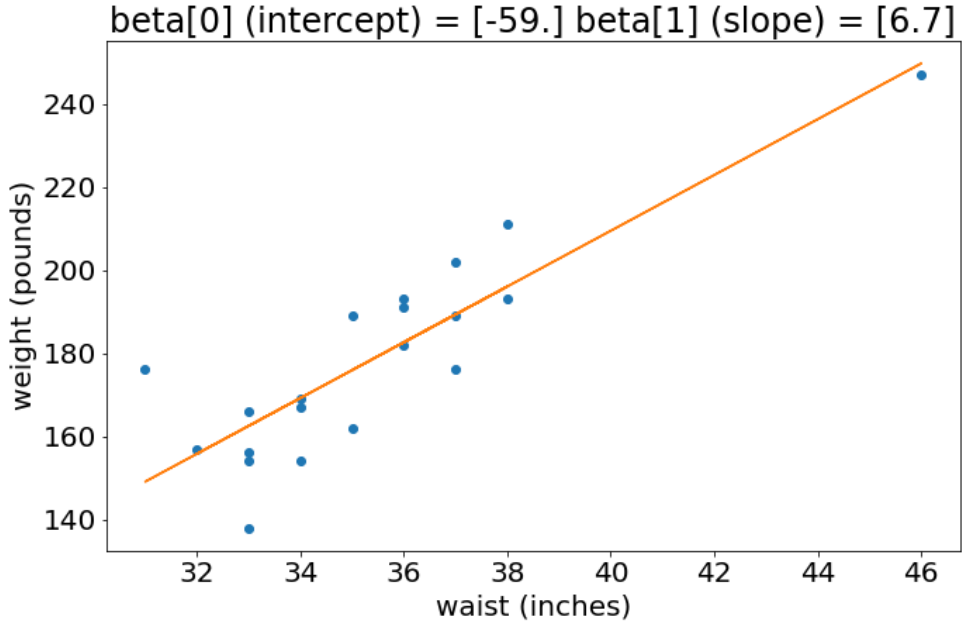
called *linear least squares,* find values for $\boldsymbol{\beta}$ that minimizes $\boldsymbol{\epsilon} = \boldsymbol{y} - \boldsymbol{X\beta}$



Linear regression example

- Given waist in inches. Want to predict weight
- Clear linear relationship between the two variables
- $\boldsymbol{\beta} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$
- In this cases, $\boldsymbol{\beta}$ is composed of two values, a slope and an intercept
  - intercept = -59
  - Slope 6.7
- Can now predict weight using $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta}$
  - Say waist is 42 inches. Then, weight = $-59 + 6.7 \cdot 42 = 222$
- Linear regression is the most basic of all prediction models



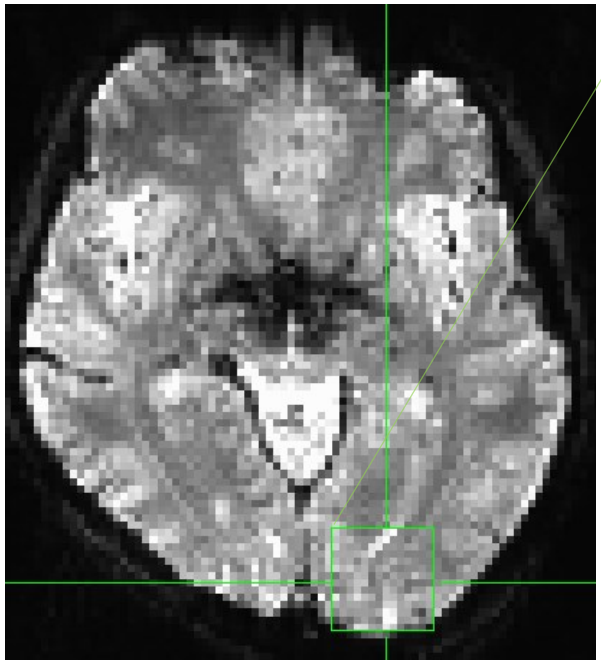


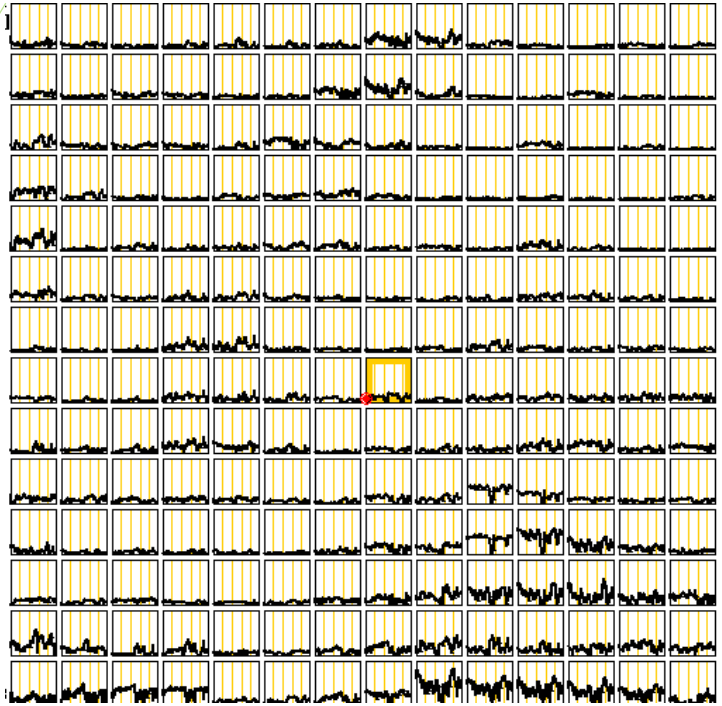
weight	waist
191.0	36.0
189.0	37.0
193.0	38.0
162.0	35.0
189.0	35.0
182.0	36.0
211.0	38.0
167.0	34.0
176.0	31.0
154.0	33.0
169.0	34.0
166.0	33.0
154.0	34.0
247.0	46.0
193.0	36.0
202.0	37.0
176.0	37.0
157.0	32.0
156.0	33.0
138.0	33.0

(n =20 people)

More complex problems

- In linear regression, we had only one dependent variable (y)
- What about our problem (final project)?
- Basically trying to predict an image based on BOLD signal in thousands of voxels
  - Each pixel is a dependent variable, 250,000 pixels





predict

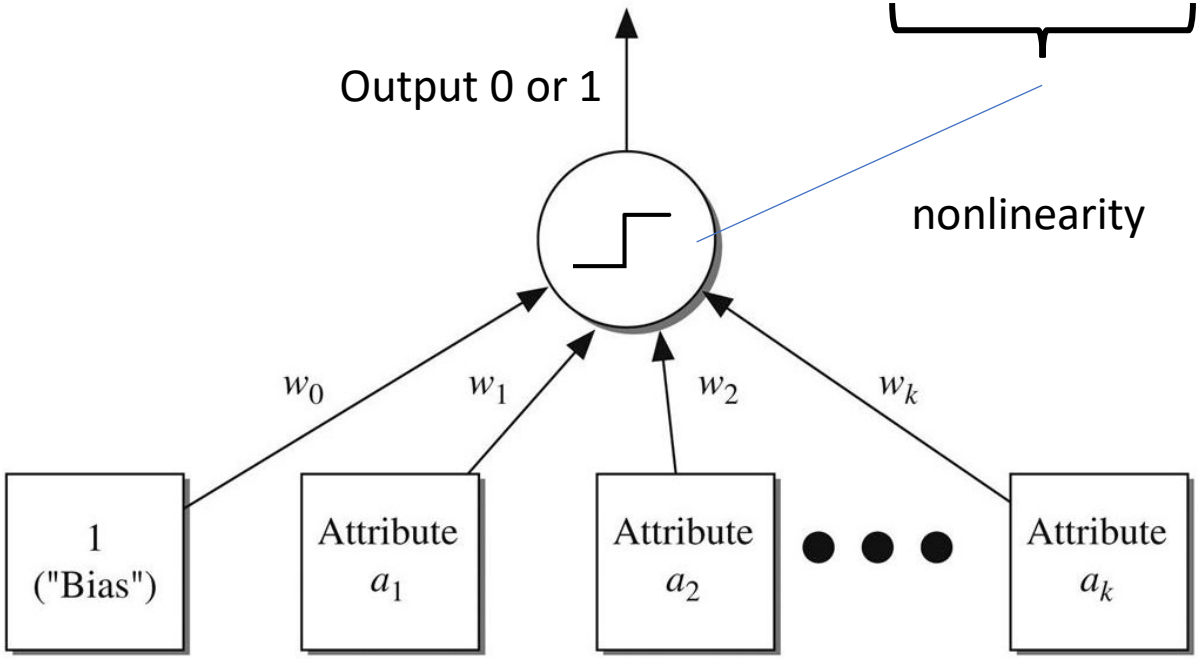
250,000 dependent variables

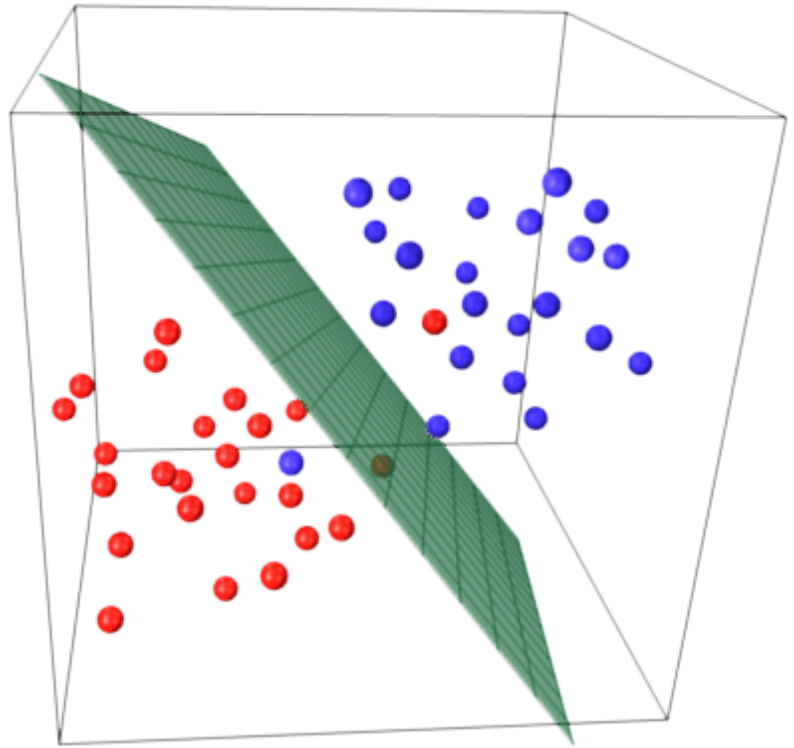
← 500 pixels →

← 500 pixels →

Neural networks

- Linear least squares ($\boldsymbol{\beta} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$) can only solve the simplest of problems
- use *neural networks* (and other methods) to solve more complicated problems
- If $w_0a_0 + w_1a_1 + \cdots w_ka_k > 0$, class=1
- If $w_0a_0 + w_1a_1 + \cdots w_ka_k < 0$, class=0

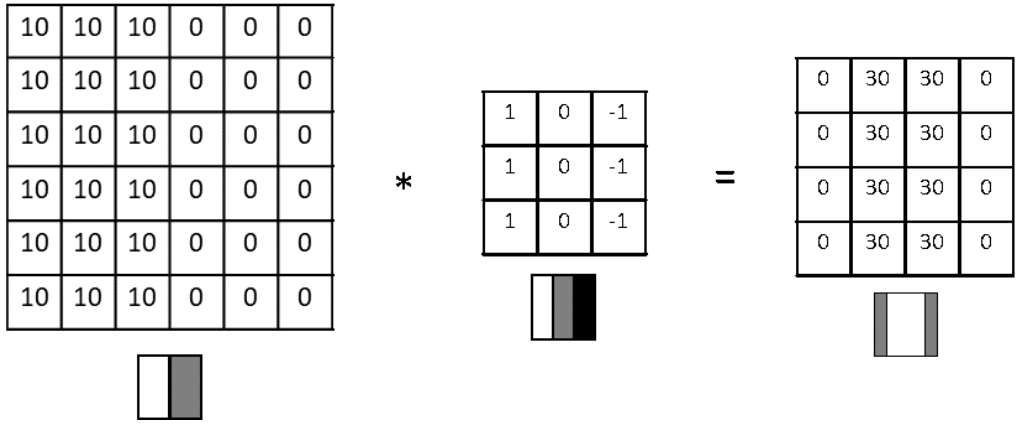




Convolutional neural network (CNN)

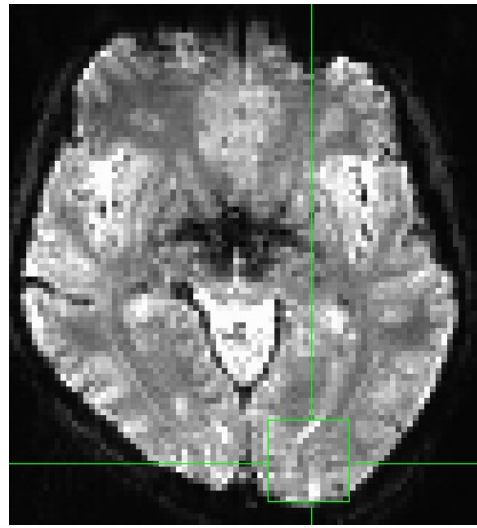
- Want to learn complex, high-dimensional, nonlinear mappings from x to y
- Predicting raw image rarely works. Too much variability between instances, cannot learn good mapping
- **idea!** extract *features* from image, and predict features instead
  - example:

Use edge-detection filter to create a feature map of the original image, highlighting edges



- How to know which features to use? CNNs can *learn* optimal features for a given task

x

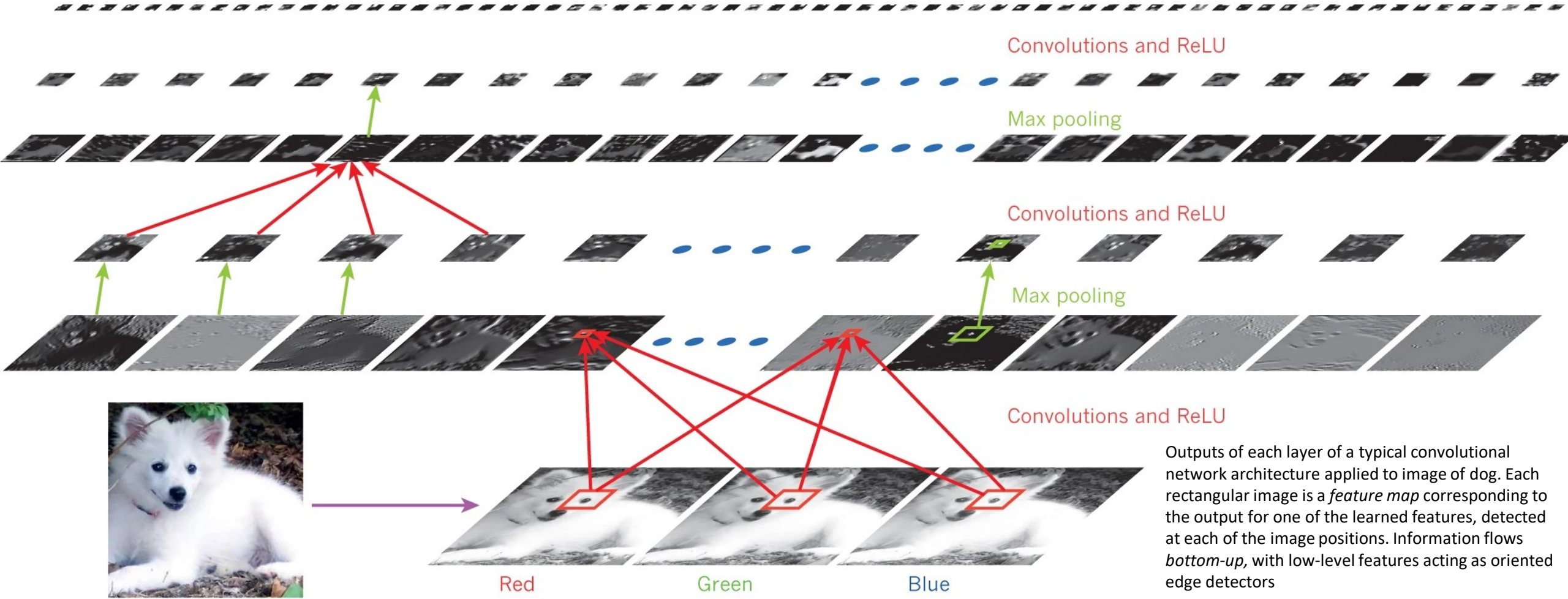


y

https://www.nature.com/articles/nature14539

CNNs and deep learning

- Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction
- Discovers intricate structure in large datasets by using backpropagation algorithm



Outputs of each layer of a typical convolutional network architecture applied to image of dog. Each rectangular image is a *feature map* corresponding to the output for one of the learned features, detected at each of the image positions. Information flows *bottom-up,* with low-level features acting as oriented edge detectors

Pre-trained CNNs

https://keras.io/api/applications/

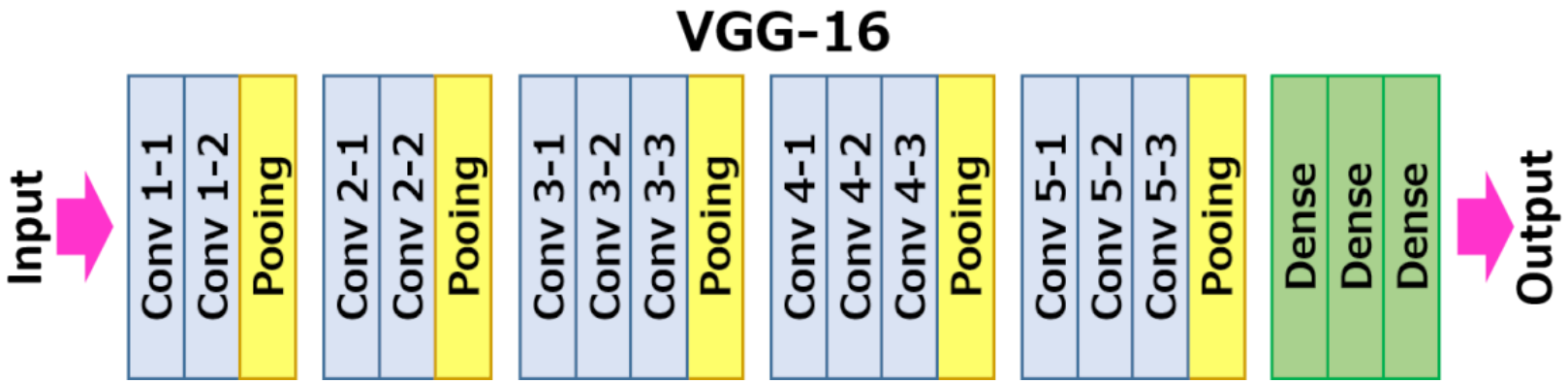
- Keras comes with several pre-trained models you can use to extract features from the image. these features will be the targets for your machine-learning algorithm

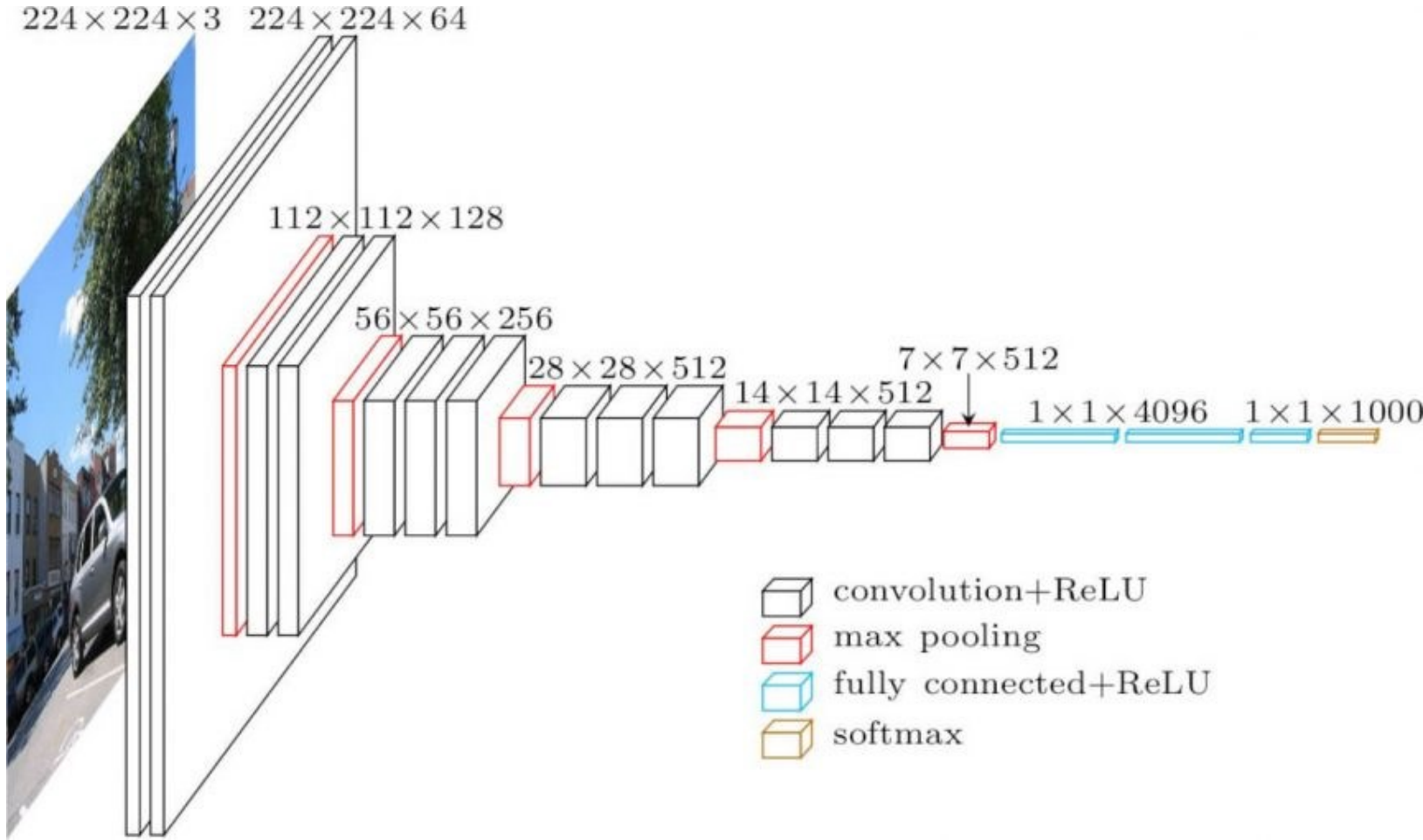
Available models

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
ResNet50	98 MB	0.749	0.921	25,636,712	-
ResNet101	171 MB	0.764	0.928	44,707,176	-
ResNet152	232 MB	0.766	0.931	60,419,944	-
ResNet50V2	98 MB	0.760	0.930	25,613,800	-
ResNet101V2	171 MB	0.772	0.938	44,675,560	-
ResNet152V2	232 MB	0.780	0.942	60,380,648	-
InceptionV3	92 MB	0.779	0.937	23,851,784	159

VGG16

- Designed for imagenet dataset (which has 1000 classes)
- Input is fixed-size 224x224 RGB image
- Image passed through stack of convolutional layers, where filters have size 3×3
- Spatial pooling carried out by 5 separate max-pooling layers, performed over a 2×2 pixel window with strid 2
- Three fully connected layers follow a stack of convolutional layers
  - First two have 4096 channels, third performs 1000-way classification (one channel for each class)





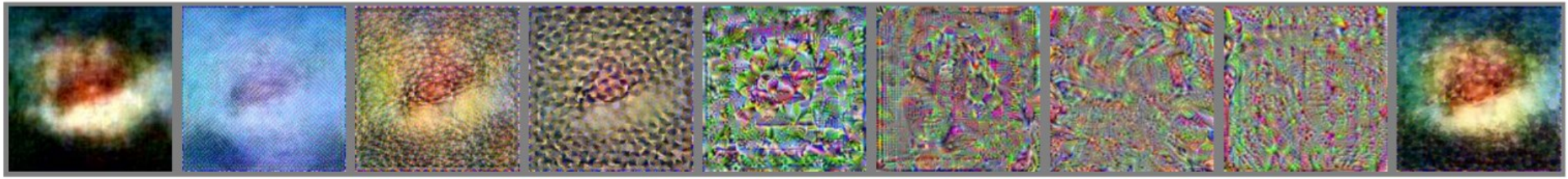
Getting features from images using VGG

- Want to get features from images so we can predict feature instead of full image
- Once we have the features, can reconstruct the original image using L-BFGS algorithm which iteratively reconstructs the image by making decoded features more and more similar to the feature vector decoded from fMRI

```
from tensorflow.keras.applications.vgg19 import VGG19
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.vgg19 import preprocess_input
from tensorflow.keras.models import Model
import numpy as np

base_model = VGG19(weights='imagenet')
model = Model(inputs=base_model.input, outputs=base_model.get_layer('block5_pool').output)

img_path = 'C:/shared/a5/images/training/n01518878_8432.JPEG'
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

features = model.predict(x)
```

Reconstructions from true features

Reconstructions from decoded features



Reconstructing images from features

- The image reconstruction using L-BFGS is implemented in icnn library (https://github.com/KamitaniLab/icnn).
- To switch the reconstruction algorithm, modify importing part of a "reconstruct_image" function in sample code of DeepImageReconstruction (https://github.com/KamitaniLab/DeepImageReconstruction), like:
  - `from icnn.icnn_lbfgs import reconstruct_image  # for L-BFSG without DGN`

    or

    `from icnn.icnn_dgn_lbfgs.py import reconstruct_image # for L-BFSG with DGN`

    (and also you may need to modify the inpuput arguments of the reconstruct_image function. See README.md and example codes of icnn for further information)
- **Reminder –** you do *not* need to use this approach. Can use any approach you want to reconstruct the image based on BOLD signal

Google colab, Google compute engine

- https://cloud.google.com/deep-learning-vm
  - ~$300 of free compute time on high-quality GPUs
- https://colab.research.google.com/notebooks/intro.ipynb
  - Be sure to select ‘GPU’ under ‘Hardware accelerator’ from ‘Runtime type’