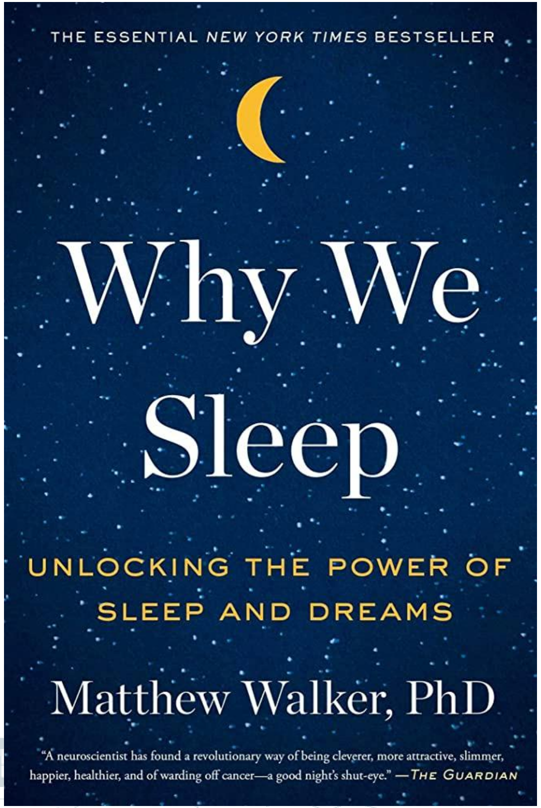
Data Science Initiative for Africa (DSI-A) - Deep Learning Lecture 8

Object Localization, Face Recognition
Advanced architectures

Santiago Romero-Brufau
Harvard T.H. Chan School of Public Health



THE ESSENTIAL *NEW YORK TIMES* BESTSELLER

Why We Sleep

UNLOCKING THE POWER OF SLEEP AND DREAMS

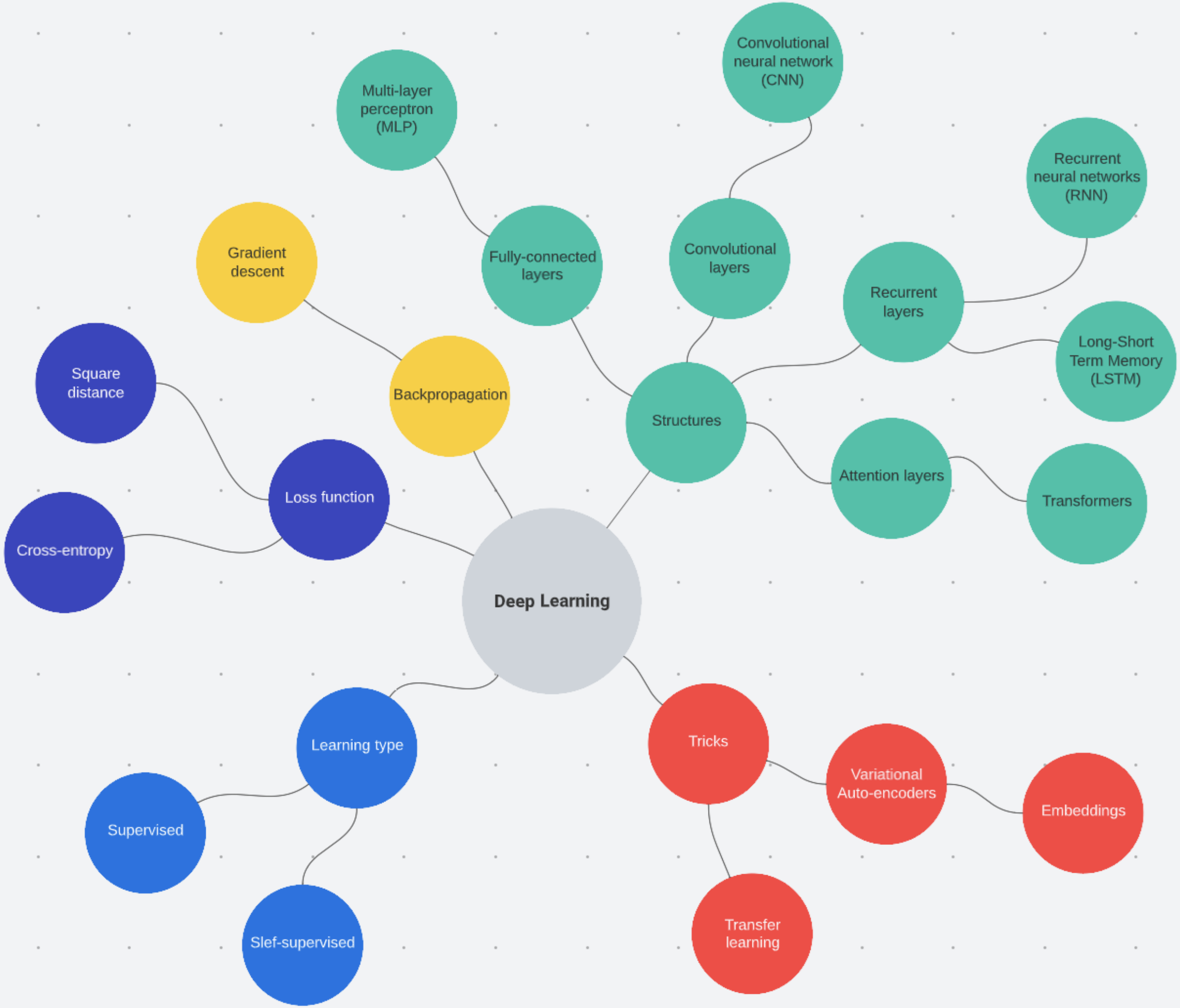
Matthew Walker, PhD

"A neuroscientist has found a revolutionary way of being cleverer, more attractive, slimmer, happier, healthier, and of warding off cancer—a good night's shut-eye." —THE GUARDIAN

"if you don't sleep the very first night after learning, you lose the chance to consolidate those memories"

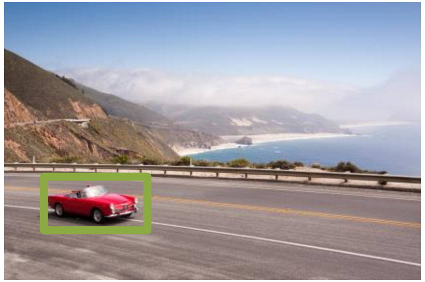
Matthew Walker, Why We Sleep

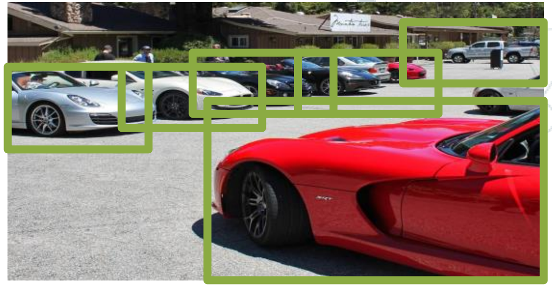
Mind Map


Deep Learning
Structures
Fully-connected layers
Multi-layer perceptron (MLP)
Convolutional layers
Convolutional neural network (CNN)
Recurrent layers
Recurrent neural networks (RNN)
Long-Short Term Memory (LSTM)
Attention layers
Transformers
Backpropagation
Gradient descent
Loss function
Square distance
Cross-entropy
Learning type
Supervised
Slef-supervised
Tricks
Variational Auto-encoders
Embeddings
Transfer learning

Object Detection and Location

Localization and Detection





Car

Car, but where?

Multiple cars

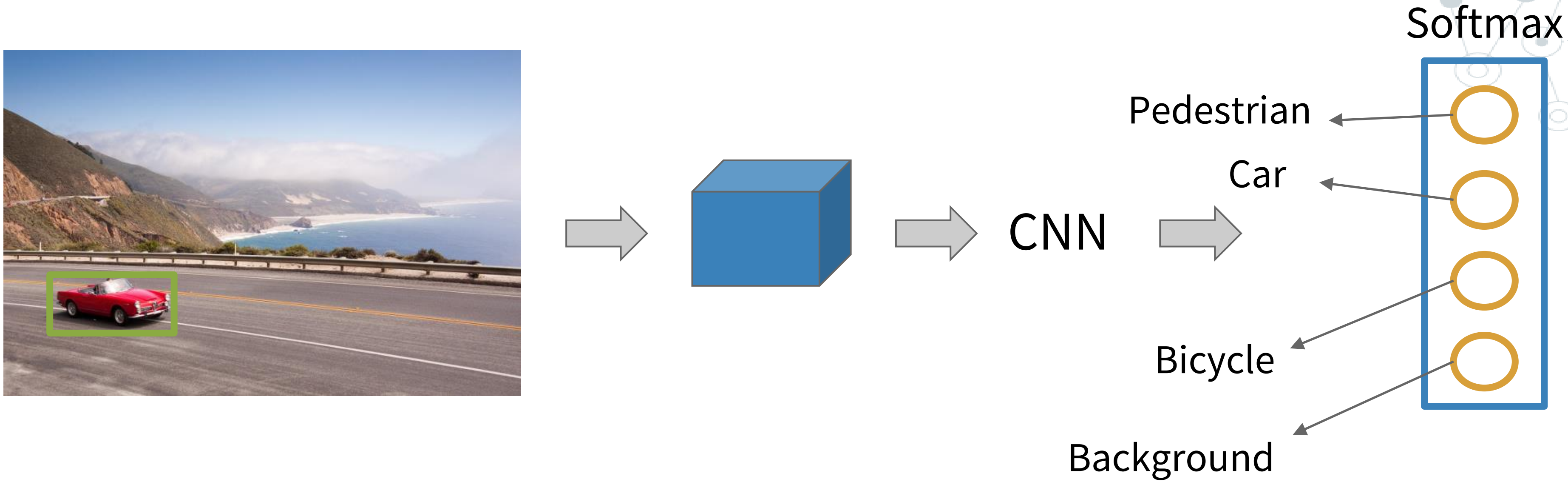
(Classification)
(Detection)

(Classification with localization)

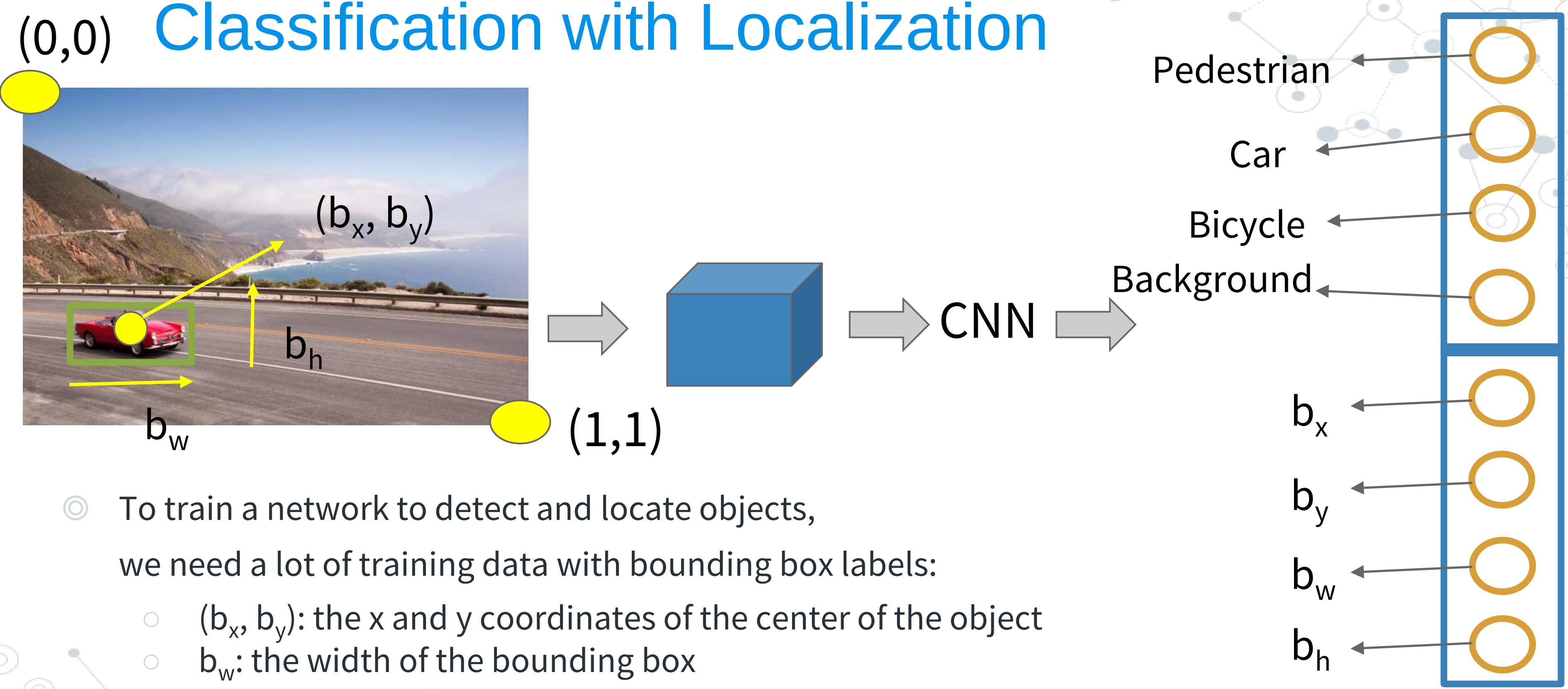
1 object

Multiple objects; could be from different classes

Classification



Classification with Localization



- To train a network to detect and locate objects,

  we need a lot of training data with bounding box labels:

  - (b_x, b_y): the x and y coordinates of the center of the object
  - b_w: the width of the bounding box
  - b_h: the height of the bounding box
  - New image label: [class, b_x, b_y, b_w, b_h]

Classification with Localization

Classes

Pedestrian (c_1)

Car (c_2)

Bicycle (c_3)

Background (no object)

New y label: [p_d, b_x, b_y, b_w, b_h, c_1, c_2, c_3]

Loss:

Note: assuming there is only 1 object in image

Probability there is an object in the image (and not just background)

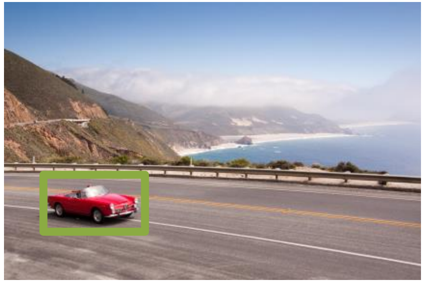
$$L(\hat{y}, y) = (\hat{y}_1, y_1)^2 + (\hat{y}_2, y_2)^2 \dots (\hat{y}_8, y_8)^2$$

if $p_d = 1$

$$L(\hat{y}, y) = (\hat{y}_1, y_1)^2$$

if $p_d = 0$

Localization and Detection



Classes

Pedestrian (c_1)

Car (c_2)

Bicycle (c_3)

Background (no object)

$y = [1, 0.25, 0.75, 0.2, 0.15, 0, 1, 0]$

New y label: $[p_d, b_x, b_y, b_w, b_h, c_1, c_2, c_3]$

Loss:

$$L(\hat{y}, y) = (\hat{y}_1, y_1)^2 + (\hat{y}_2, y_2)^2 \dots (\hat{y}_8, y_8)^2$$

if

$p_d = 1$

$$L(\hat{y}, y) = (\hat{y}_1, y_1)^2$$

if

Localization and Detection

Classes

Pedestrian (c_1)

Car (c_2)

Bicycle (c_3)

Background (no object)

$$y = [0, ?, ?, ?, ?, ?, ?, ?]$$

New y label: $[p_d, b_x, b_y, b_w, b_h, c_1, c_2, c_3]$

Loss: $L(\hat{y}, y) = (\hat{y}_1, y_1)^2 + (\hat{y}_2, y_2)^2 \dots (\hat{y}_8, y_8)^2$

$L(\hat{y}, y) = (\hat{y}_1, y_1)^2$

if

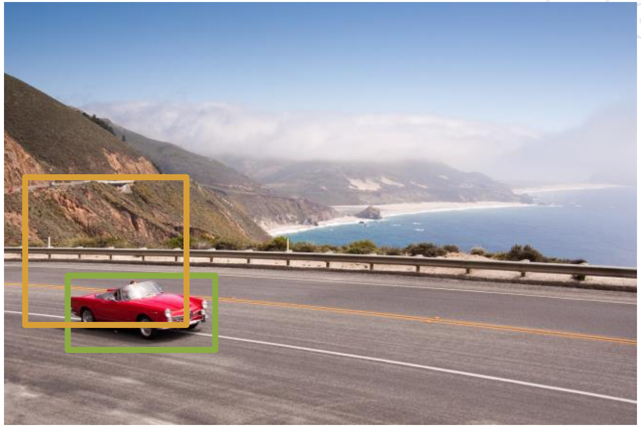
$p_d = 1$

Evaluating Object Localization

- How well is your algorithm working in terms of finding the bounding boxes?
- One metric to measure the performance of the algorithm is Intersection over Union

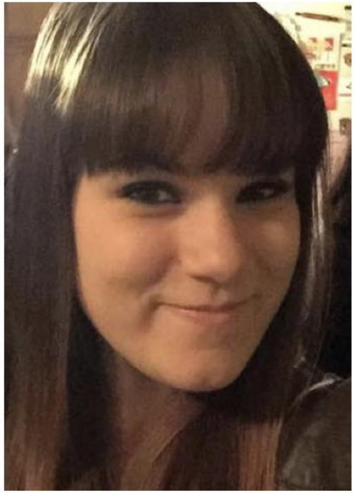
$$IoU = \frac{\text{size of intersection}}{\text{size of union}}$$

- “Correct” if $IoU \geq 0.5$ or some other threshold
- Basically measures the overlap of the predicted bounding box with the ground truth bounding box - more overlap is better

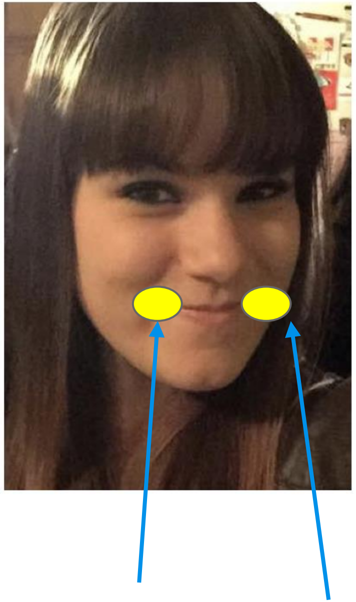


Landmark Detection

Landmark Detection

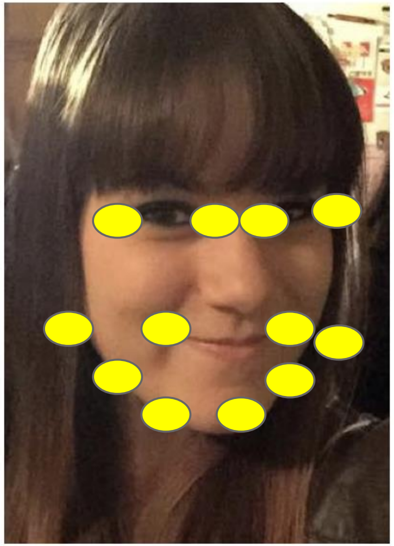


Landmark Detection



(l_{x1}, l_{y1}) (l_{x2}, l_{y2})

Landmark Detection



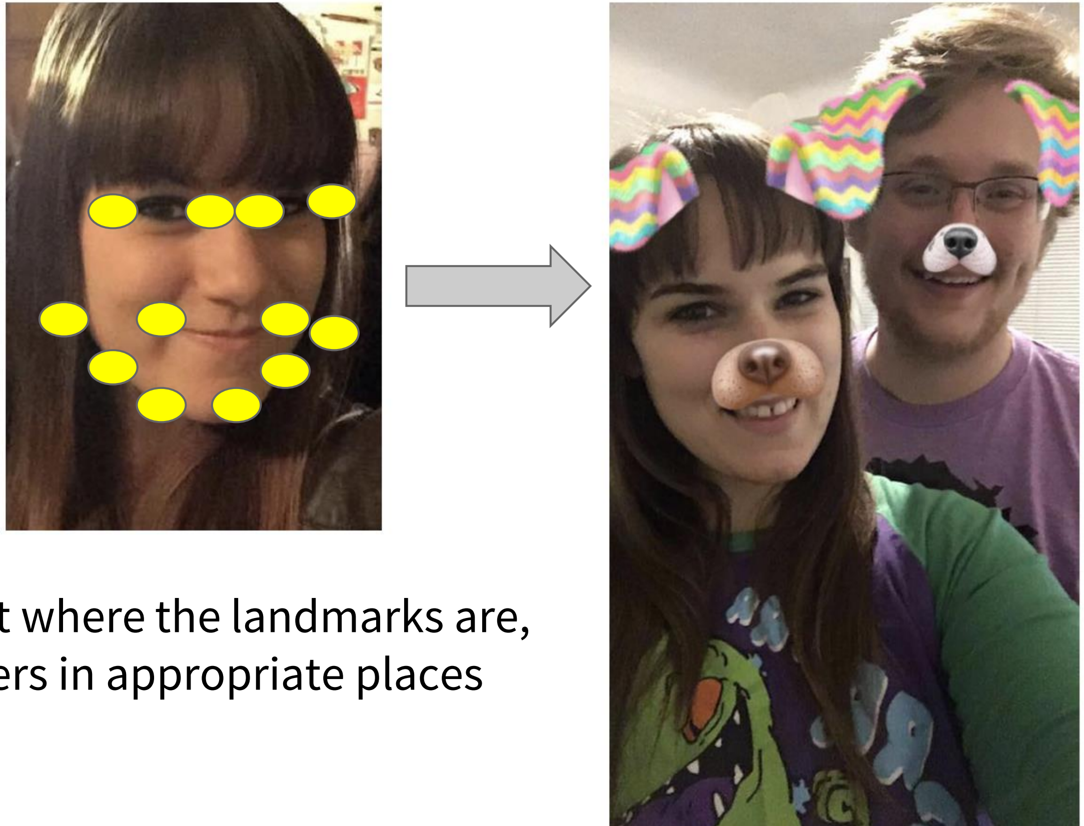
Label every point

Input: image with n landmarks
Output:
$[p_{face}, l_{x1}, l_{y1}, l_{x2}, l_{y2}, \ldots, l_{xn}, l_{yn}]$

Fit CNN to output if the image is of a face and the locations of the landmarks if it is a face

Note: landmarks have to be consistent across all training images, i.e. landmark # 1 is the left corner of the right eye, for example

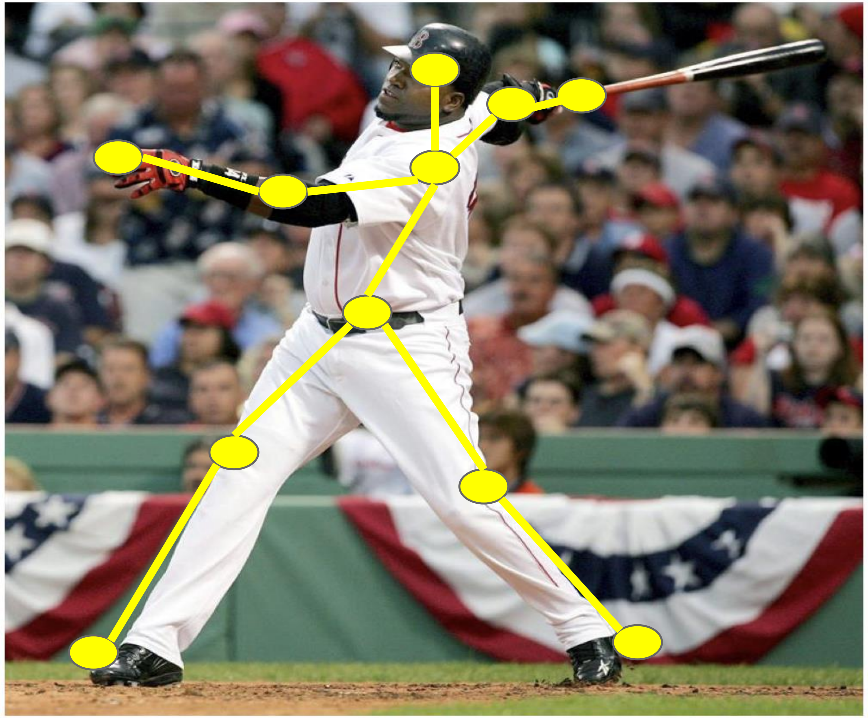
Landmark Detection



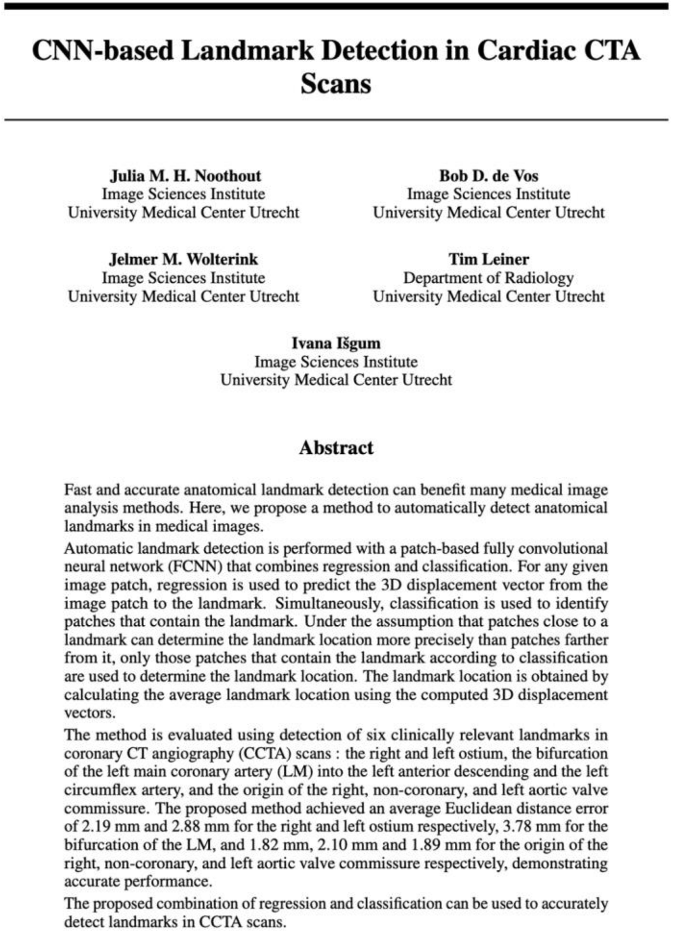
If I can detect where the landmarks are,
I can add filters in appropriate places

Landmark Detection

Landmark Detection



“Pose” detection



CNN-based Landmark Detection in Cardiac CTA Scans

Julia M. H. Noothout
Image Sciences Institute
University Medical Center Utrecht

Bob D. de Vos
Image Sciences Institute
University Medical Center Utrecht

Jelmer M. Wolterink
Image Sciences Institute
University Medical Center Utrecht

Tim Leiner
Department of Radiology
University Medical Center Utrecht

Ivana Išgum
Image Sciences Institute
University Medical Center Utrecht

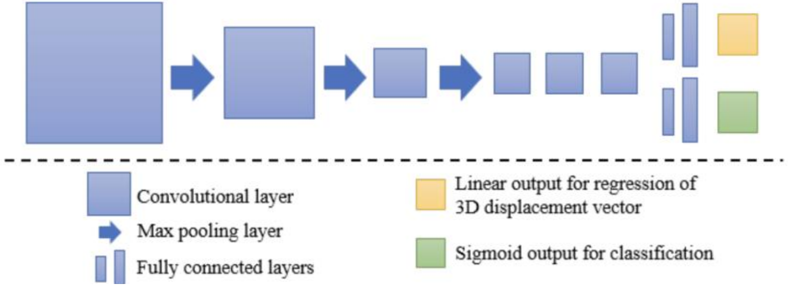
Abstract

Fast and accurate anatomical landmark detection can benefit many medical image analysis methods. Here, we propose a method to automatically detect anatomical landmarks in medical images.

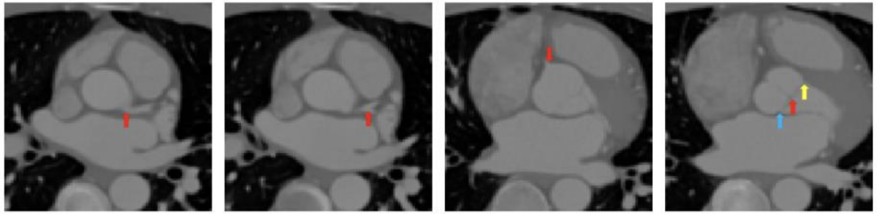
Automatic landmark detection is performed with a patch-based fully convolutional neural network (FCNN) that combines regression and classification. For any given image patch, regression is used to predict the 3D displacement vector from the image patch to the landmark. Simultaneously, classification is used to identify patches that contain the landmark. Under the assumption that patches close to a landmark can determine the landmark location more precisely than patches farther from it, only those patches that contain the landmark according to classification are used to determine the landmark location. The landmark location is obtained by calculating the average landmark location using the computed 3D displacement vectors.

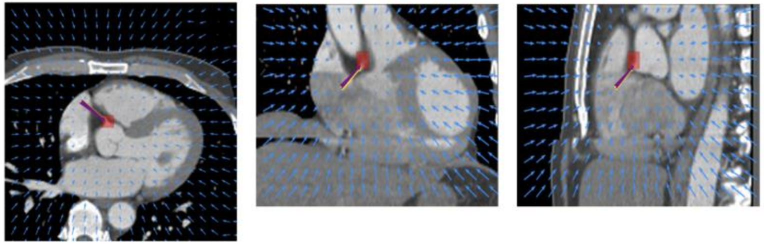
The method is evaluated using detection of six clinically relevant landmarks in coronary CT angiography (CCTA) scans : the right and left ostium, the bifurcation of the left main coronary artery (LM) into the left anterior descending and the left circumflex artery, and the origin of the right, non-coronary, and left aortic valve commissure. The proposed method achieved an average Euclidean distance error of 2.19 mm and 2.88 mm for the right and left ostium respectively, 3.78 mm for the bifurcation of the LM, and 1.82 mm, 2.10 mm and 1.89 mm for the origin of the right, non-coronary, and left aortic valve commissure respectively, demonstrating accurate performance.

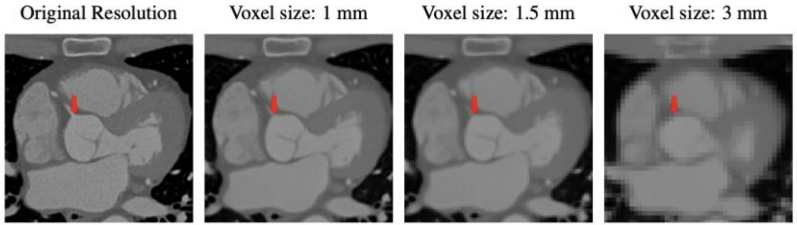
The proposed combination of regression and classification can be used to accurately detect landmarks in CCTA scans.



Convolutional layer
Max pooling layer
Fully connected layers
Linear output for regression of 3D displacement vector
Sigmoid output for classification







https://openreview.net/pdf?id=r1malb3jz

Face Recognition

Terminology

- Recognition
  - Have a database of K persons
  - Get an input image
  - Output ID if the image is any of the K persons, or “not recognized” if not like any of the K persons

- Verification
  - Input image and name/ID
  - Output whether the input image is that of the claimed person

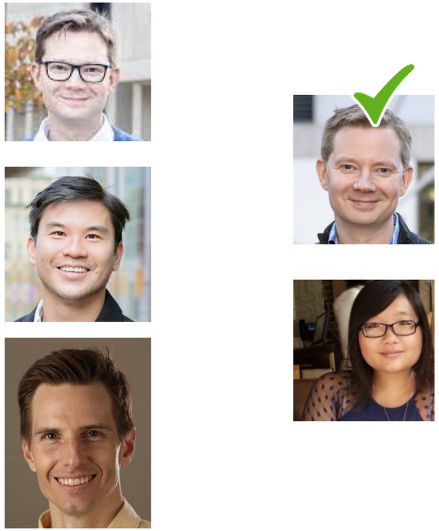
- Andrew Ng demo video

Face Recognition

- One-shot Learning: learning from 1 example to recognize that person again
- Major downside: needs to be re-trained every time another person is added to group, only 1 example to learn from

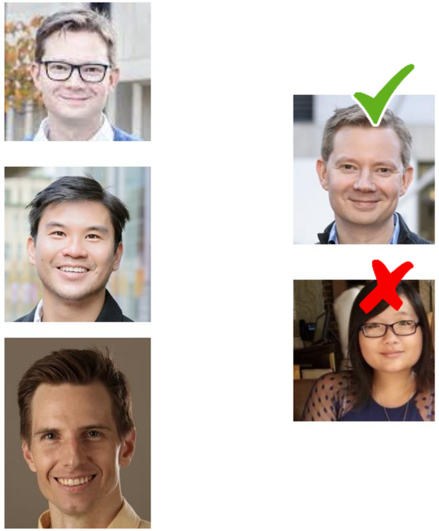
Face Recognition

- One-shot Learning: learning from 1 example to recognize that person again
- Major downside: needs to be re-trained every time another person is added to group, only 1 example to learn from



Face Recognition

- One-shot Learning: learning from 1 example to recognize that person again
- Major downside: needs to be re-trained every time another person is added to group, only 1 example to learn from



Similarity Function

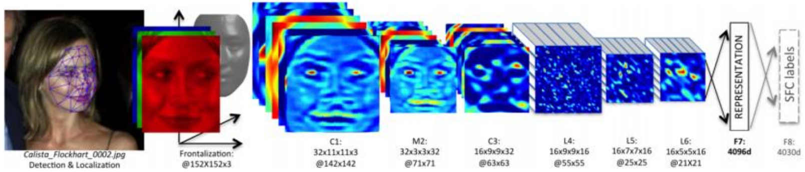


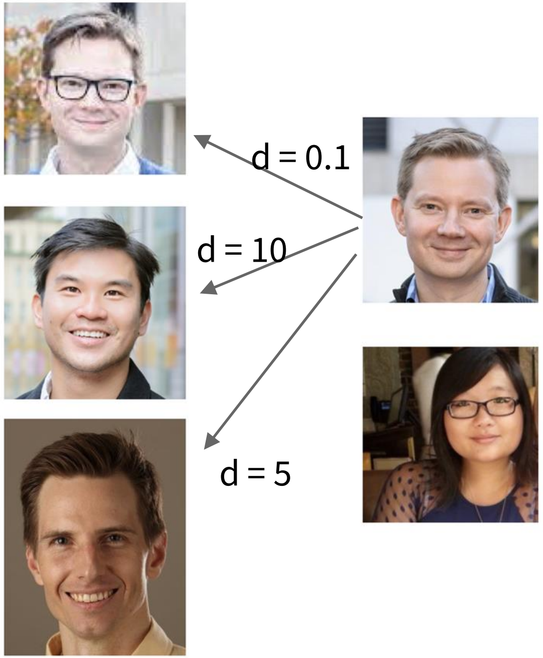
Figure 2. **Outline of the *DeepFace* architecture**. A front-end of a single convolution-pooling-convolution filtering on the rectified input, followed by three locally-connected layers and two fully-connected layers. Colors illustrate feature maps produced at each layer. The net includes more than 120 million parameters, where more than 95% come from the local and fully connected layers.

- Similarity function: quantify how similar or different two images are
- If difference is large, the images are of two different people
- If difference is small, the images are of the same person
- DeepFace by Taigman et al 2014
- Define the similarity function as

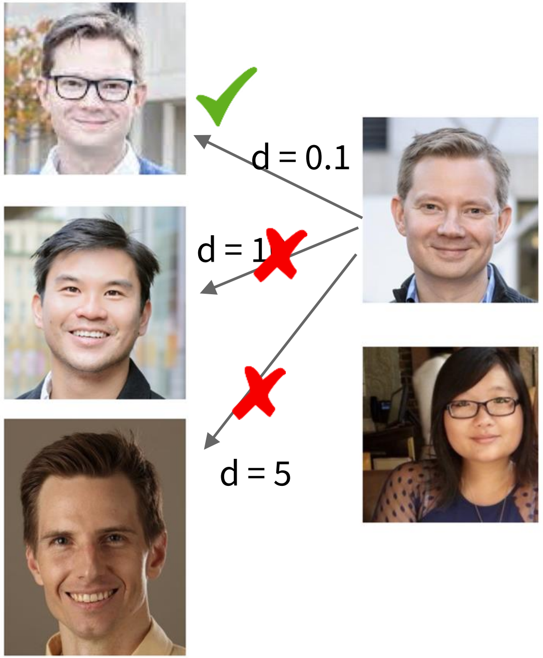
$$d(x^{(1)}, x^{(2)}) = \|f(x^{(1)}) - f(x^{(2)})\|_2^2$$

- Learn parameters such that if x$^{(i)}$ and x$^{(j)}$ are the same person, d is small, and if x$^{(i)}$ and x$^{(j)}$ are different people, d is large
- Come up with threshold of what is “small”

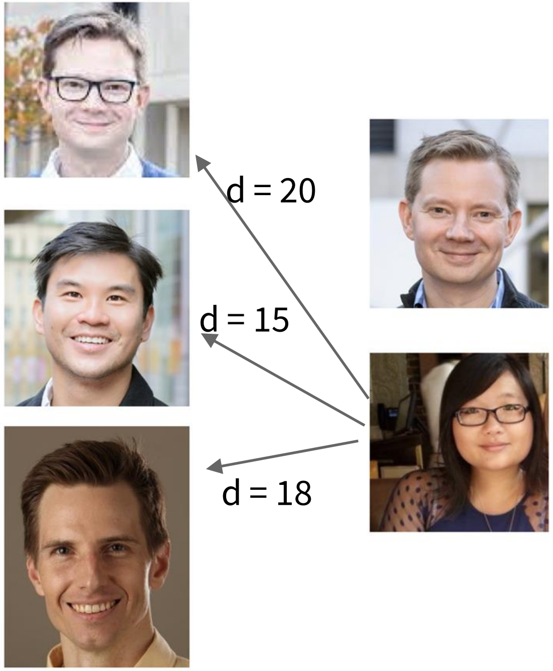
Similarity Function



Similarity Function



Similarity Function

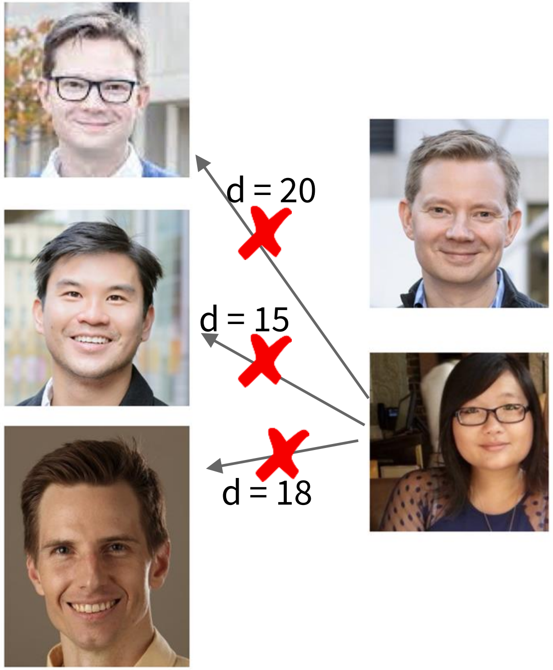


d = 20

d = 15

d = 18

Similarity Function



Not in database

Advanced Architectures

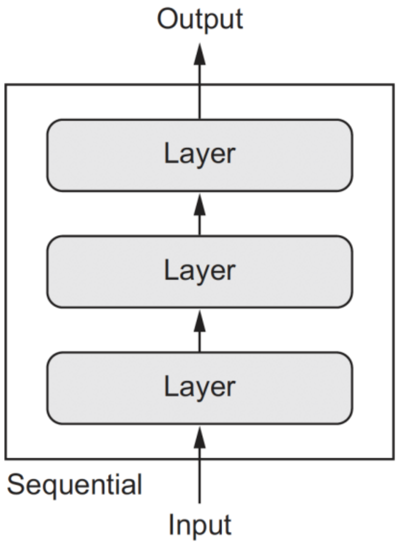
(Or, putting pieces together differently)

Beyond the Sequential Model

- Throughout the course we have assumed each network has exactly one input and exactly one output, and that it consists of a linear stack of layers

- But what if we have **multiple types** of inputs? Or multiple types of outputs?

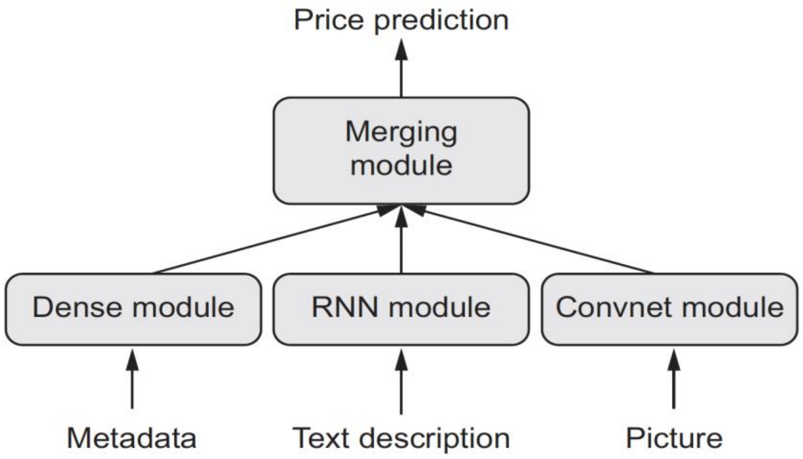
- We can change the network structure - Keras makes this easy to do



Multimodal (Multi-inputs) Model

- Multimodal inputs merge data coming from different input sources, processing each type of data using different kinds of neural layers

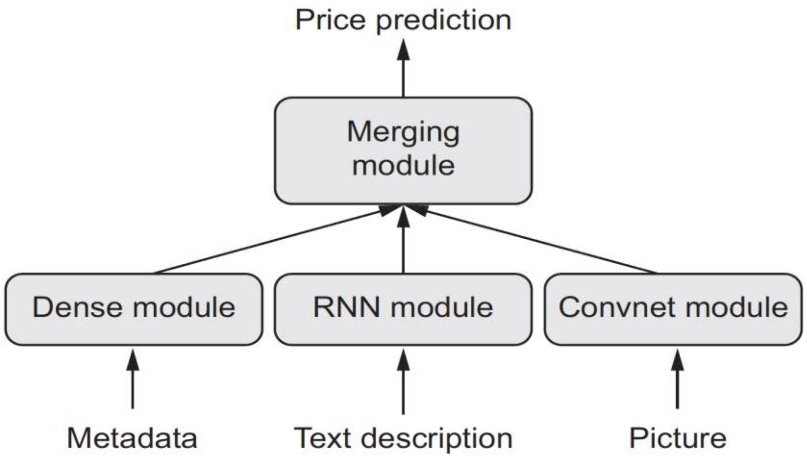
- Example: predict the most likely market price of a second-hand piece of clothing, using the following inputs:
  - User-provided **metadata** (brand, age, etc.)
  - User-provided **text** description
  - **Picture** of the item



Multimodal (Multi-inputs) Model

- Suboptimal approach: **train three separate models** and then do a weighted average of their predictions
  - Information may be redundant

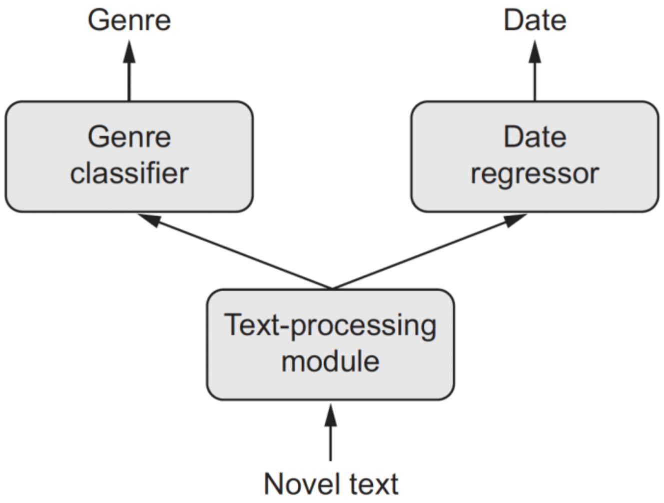
- Better approach: **jointly learn** a more accurate model of the data by using a model that can see all available input modalities simultaneously: a model with three input branches



Multi-output (multihead) Model

- Predict multiple target attributes (outputs) of input data

- Example: predict the genre and date of a novel using the novel’s text

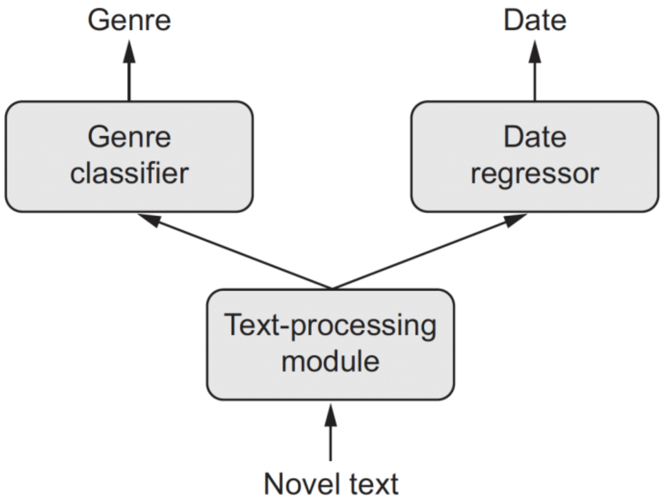


Multi-output (multihead) Model

- Suboptimal approach: **train two separate models**: one for the genre and one for the date
  - But these attributes aren't statistically independent

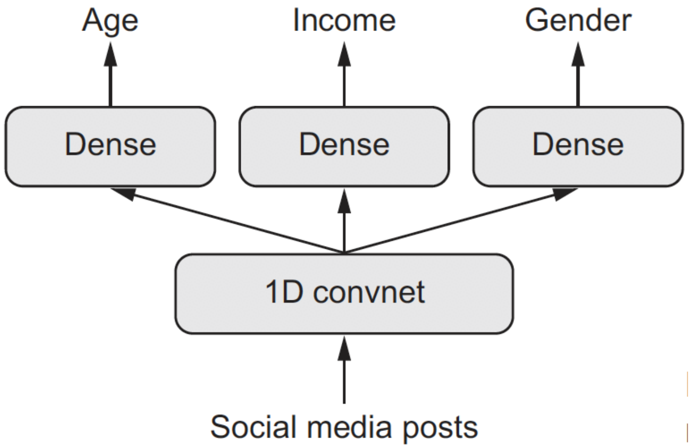
- Better approach: **jointly predict** both genre and date at the same time
  - Correlations between date and genre of the novel helps training

- Thesis work on predicting the genre of novels using machine learning and deep learning



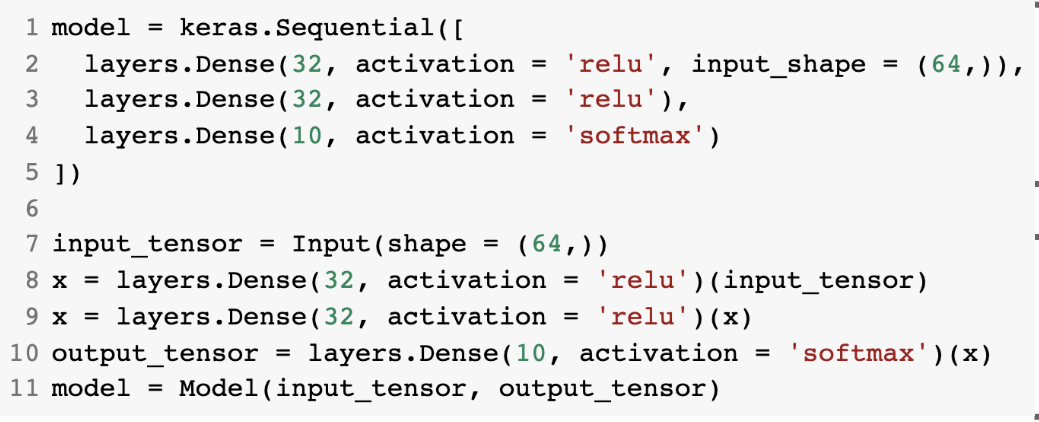
Multi-output models

- Example: a network that attempts tc simultaneously predict different properties of the data, such as a network that takes as input a series of social media posts from a single anonymous person and tries to predict attributes of that person, such as age, gender, and income level



The Functional API in Keras

- Directly manipulate tensors
- Use layers as **functions** that take tensors and return tensors



```
model = keras.Sequential([
  layers.Dense(32, activation = 'relu', input_shape = (64,)),
  layers.Dense(32, activation = 'relu'),
  layers.Dense(10, activation = 'softmax')
])

input_tensor = Input(shape = (64,))
x = layers.Dense(32, activation = 'relu')(input_tensor)
x = layers.Dense(32, activation = 'relu')(x)
output_tensor = layers.Dense(10, activation = 'softmax')(x)
model = Model(input_tensor, output_tensor)
```

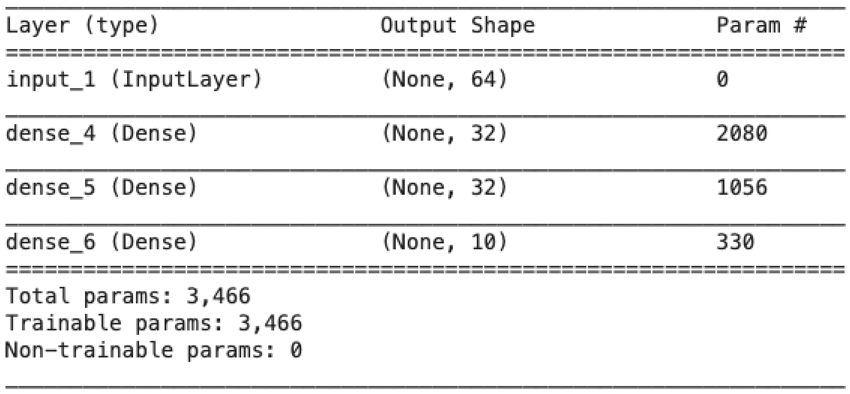
Sequential model (what we’ve seen before)

Functional equivalent to the above model

The Model class turns an input tensor and output tensor into a model

The Functional API in Keras

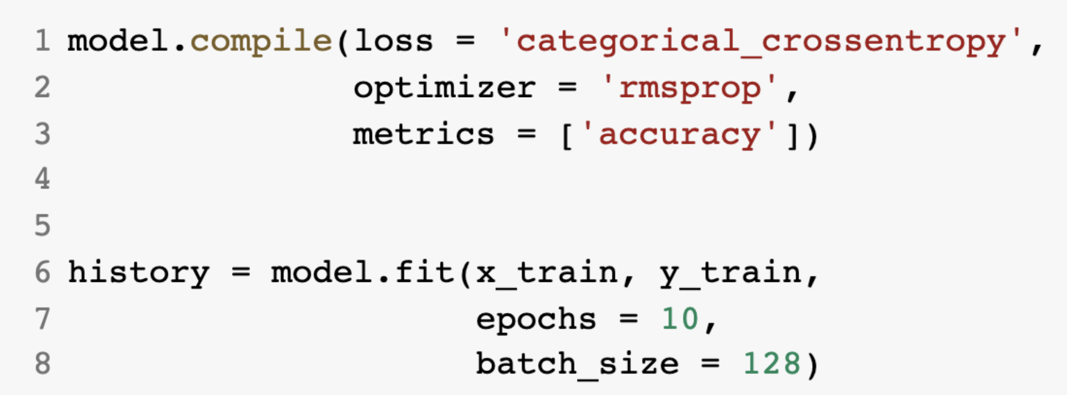
Keras retrieves every layer involved in going from the input tensor to the output tensor and brings them together into a graph-like structure (a Model)



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_1 (InputLayer)         (None, 64)                0         
_________________________________________________________________
dense_4 (Dense)              (None, 32)                2080      
_________________________________________________________________
dense_5 (Dense)              (None, 32)                1056      
_________________________________________________________________
dense_6 (Dense)              (None, 10)                330       
=================================================================
Total params: 3,466
Trainable params: 3,466
Non-trainable params: 0
_________________________________________________________________
```

The Functional API in Keras

Everything is the same when you compile, train and evaluate an instance of Model:

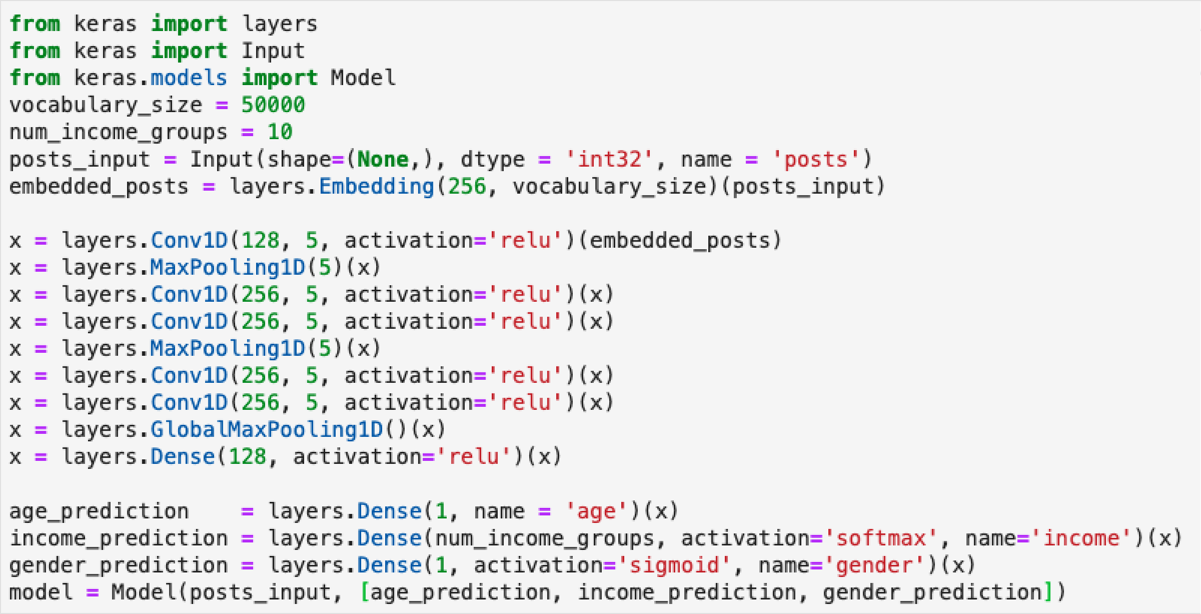


```
model.compile(loss = 'categorical_crossentropy',
              optimizer = 'rmsprop',
              metrics = ['accuracy'])


history = model.fit(x_train, y_train,
                    epochs = 10,
                    batch_size = 128)
```

https://keras.io/guides/functional_api/

Multi-output Models

Can specify different functions for different outcomes

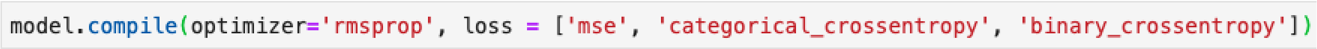


```
from keras import layers
from keras import Input
from keras.models import Model
vocabulary_size = 50000
num_income_groups = 10
posts_input = Input(shape=(None,), dtype = 'int32', name = 'posts')
embedded_posts = layers.Embedding(256, vocabulary_size)(posts_input)

x = layers.Conv1D(128, 5, activation='relu')(embedded_posts)
x = layers.MaxPooling1D(5)(x)
x = layers.Conv1D(256, 5, activation='relu')(x)
x = layers.Conv1D(256, 5, activation='relu')(x)
x = layers.MaxPooling1D(5)(x)
x = layers.Conv1D(256, 5, activation='relu')(x)
x = layers.Conv1D(256, 5, activation='relu')(x)
x = layers.GlobalMaxPooling1D()(x)
x = layers.Dense(128, activation='relu')(x)

age_prediction    = layers.Dense(1, name = 'age')(x)
income_prediction = layers.Dense(num_income_groups, activation='softmax', name='income')(x)
gender_prediction = layers.Dense(1, activation='sigmoid', name='gender')(x)
model = Model(posts_input, [age_prediction, income_prediction, gender_prediction])
```

So, how do we train this?

Multi-output Models

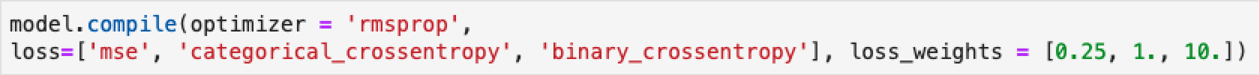
- This model requires the ability to specify different loss functions for different heads of the network:
  - Age prediction is a scalar regression task
  - Gender prediction is a binary classification task
  - Income prediction in this example is a multiclass classification task
    - Income categories
- But because gradient descent requires you to minimize a scalar, you must combine these losses into a single value in order to train the model.
- The simplest way to combine different losses is to sum them all.
  - In Keras, you can use either a list or a dictionary of losses in **compile** to specify different objects for different outputs; the resulting loss values are summed into a global loss, which is minimized during training.



```
model.compile(optimizer='rmsprop', loss = ['mse', 'categorical_crossentropy', 'binary_crossentropy'])
```

Multi-output Models

- A note on imbalanced loss contributions
  - Imbalances will cause the model representations to be optimized preferentially for the task with the largest individual loss, at the expense of the other tasks
  - To remedy this, you can assign different levels of importance to the loss values in their contribution to the final loss
  - This is particularly useful if the losses' values use different scales
  - Example:
    - MSE takes values around 3-5
    - Cross-entropy loss can be as low as 0.1
    - Here we could assign a weight of 10 for the cross-entropy loss and a weight of 0.25 to the MSE loss



```
model.compile(optimizer = 'rmsprop',
loss=['mse', 'categorical_crossentropy', 'binary_crossentropy'], loss_weights = [0.25, 1., 10.])
```

Hyperparameter Optimization

- There is no way of knowing which values are the optimal ones before building and training your model
- Even with experience and intuition, your first pass at the values will be suboptimal
- There are no formal rules to tell you which values are the best ones for your task
- You can (and we have in this course) tweak the value of each hyperparameter by hand
  - But this is inefficient
- It’s better to let the machine do this, and there is a whole field of research around this
  - Bayesian optimization
  - Genetic algorithms
  - Simple random search
  - Grid search
  - Etc.
- The decision between manual and automated comes down to a balance between understanding your model and computational cost

Hyperparameter Optimization

- The process:

  1. Choose a set of hyperparameters (automatically)
  2. Build the corresponding model
  3. Fit it to your training data and measure the final performance on validation data
  4. Choose the next set of hyperparameters to try (automatically)
  5. Repeat
  6. Eventually, measure performance on your test data

Hyperparameter Optimization

- More tools available each year
  - One option is Hyperopt
    - A Python library for hyperparameter optimization that internally uses trees of Parzen estimators to predict sets of hyperparameters that are likely to work well
  - Another option is Hperas
    - Another library that integrates Hyperopt for use with Keras
  - Weights and Biases
  - SageMaker
  - Comet.ml
  - This great post

Hyperparameter Optimization

- CAUTION!!
  - Can easily overfit to the validation data
  - Can be very computationally expensive