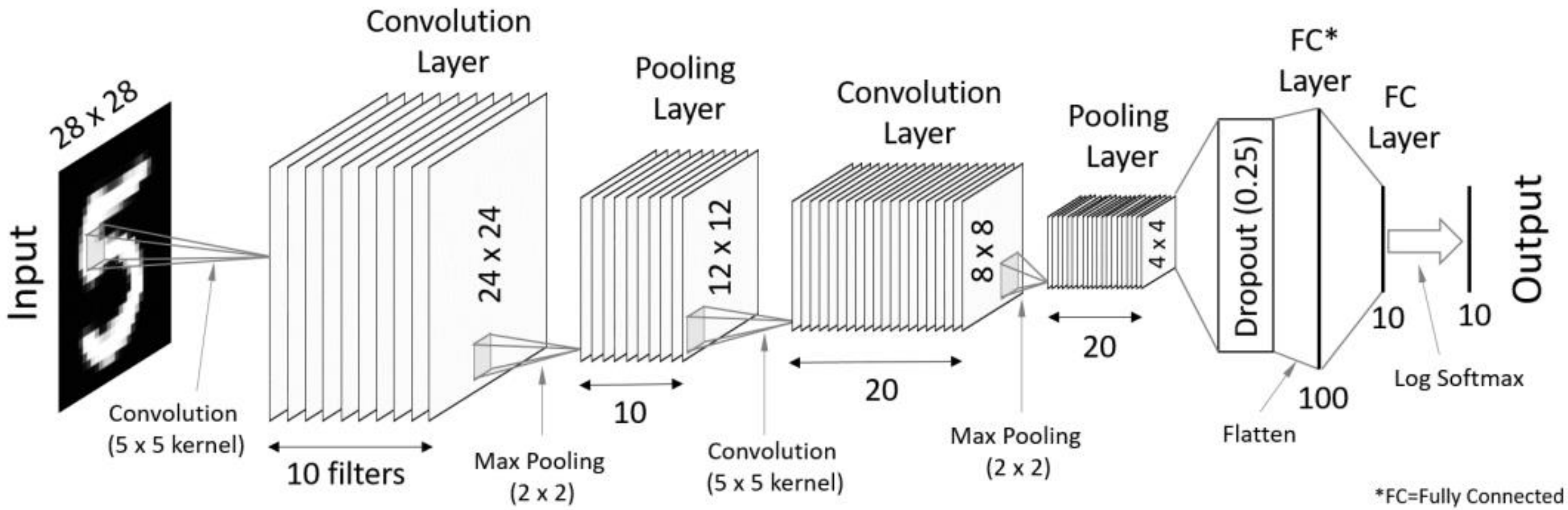
Image Processing

Jeff Prosise

@jprosise

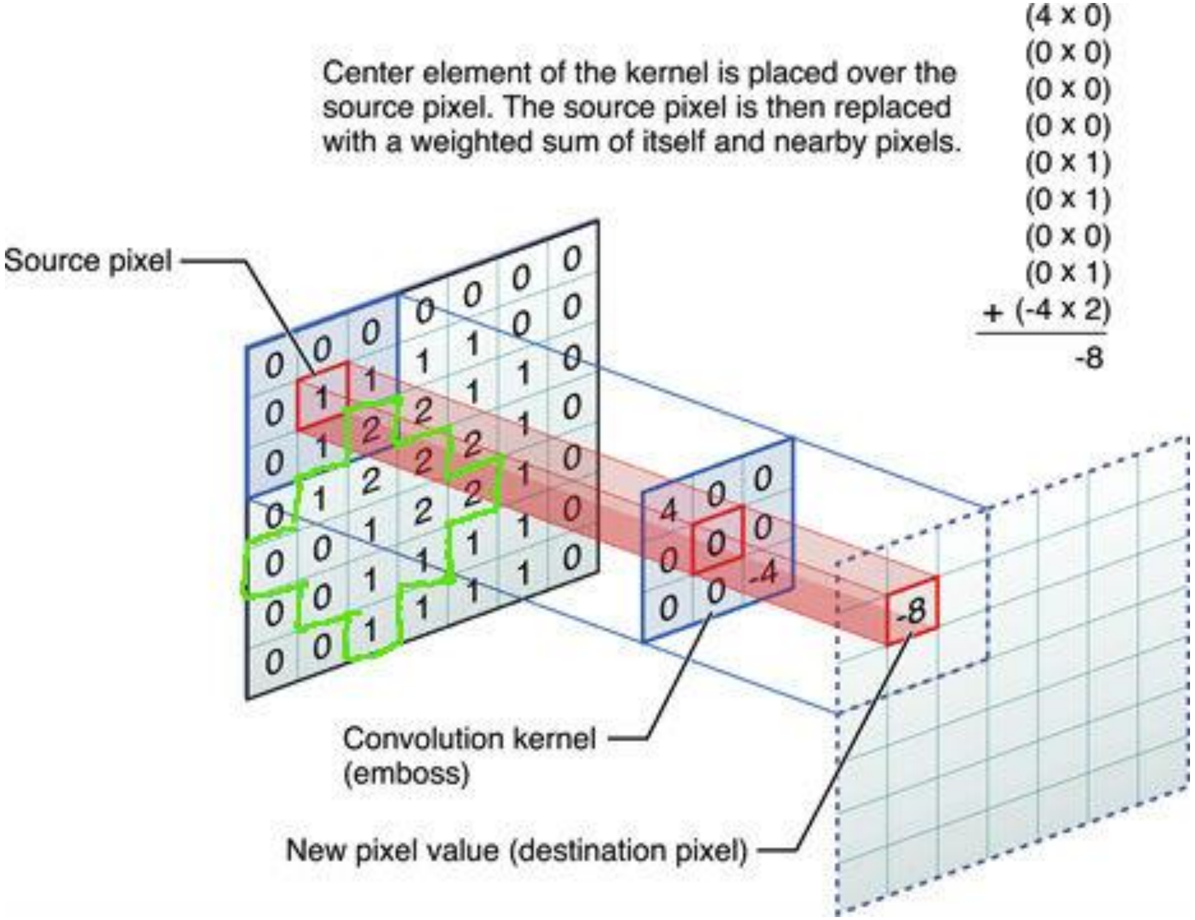
Convolutional Neural Networks (CNNs)

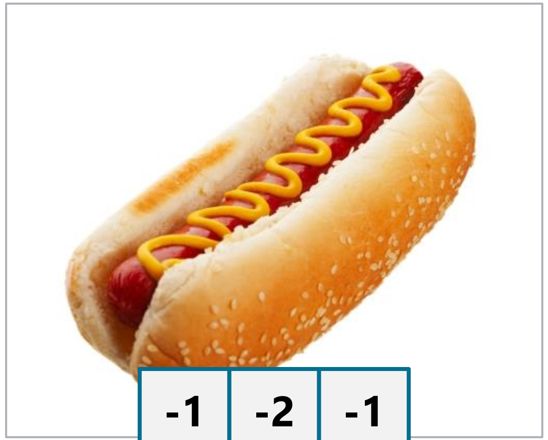
- Excel at computer-vision tasks such as image classification
- Use convolution layers and convolution kernels to create feature maps
- Use pooling layers to subsample feature maps and generalize features



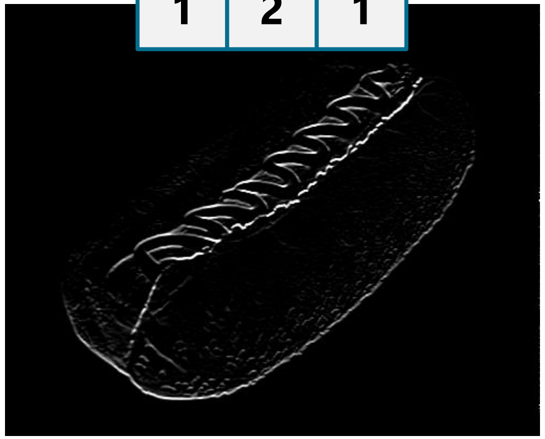
Source: https://codetolight.wordpress.com/2017/11/29/getting-started-with-pytorch-for-deep-learning-part-3-neural-network-basics/

Convolution Kernels





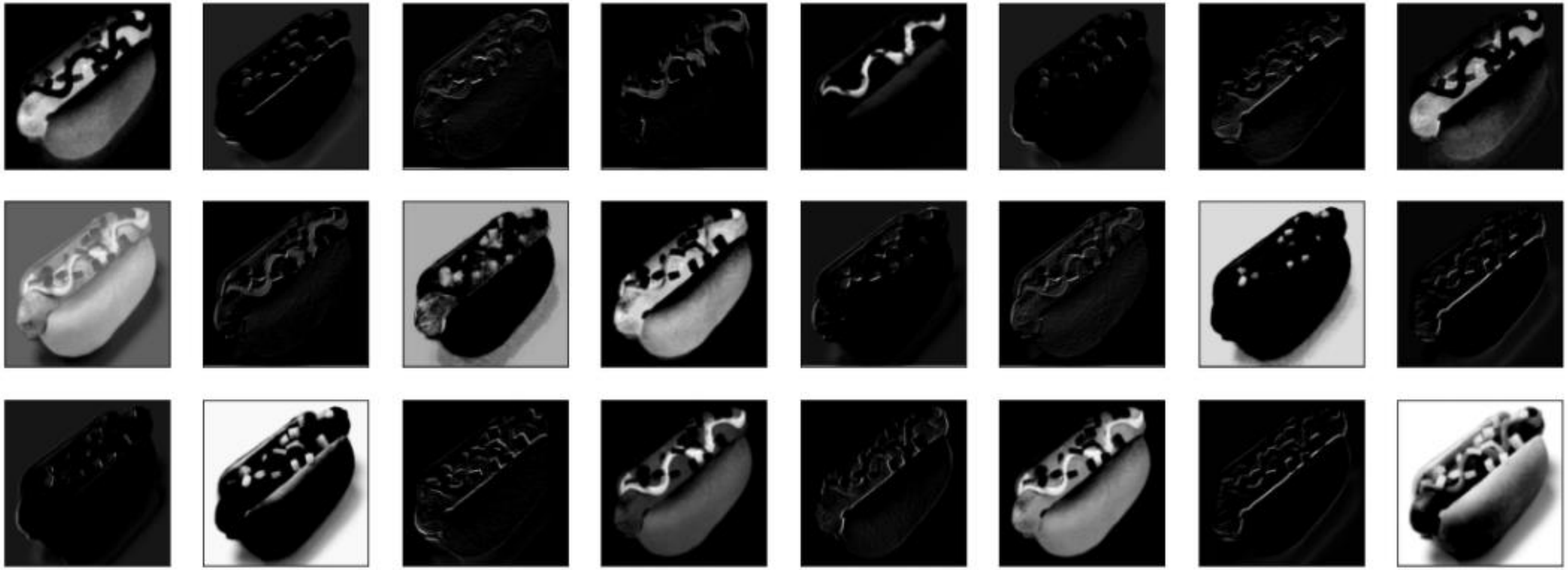
-1	-2	-1
0	0	0
1	2	1



Source: https://stats.stackexchange.com/questions/235032/any-use-of-non-rectangular-shaped-kernels-in-convolutional-neural-networks-espe

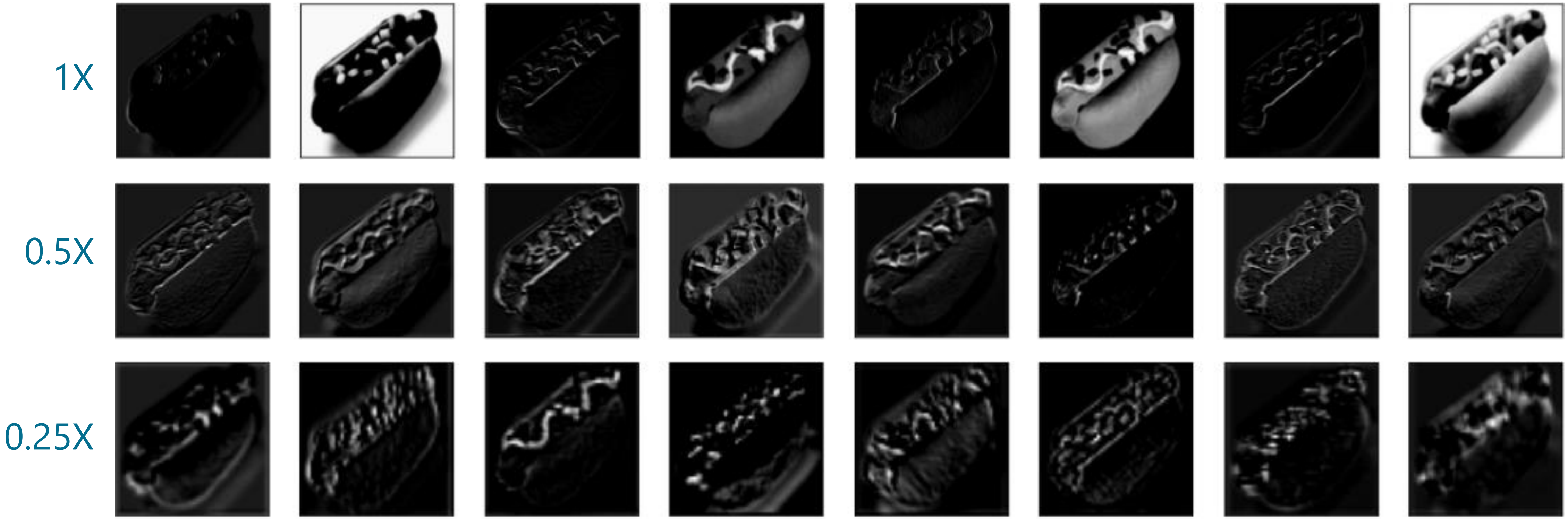
Convolution Layers

- Use convolution kernels to extract features from images
- Use multiple kernels per layer, with values "learned" during training



Pooling Layers

- Successively reduce images to half their original size
- Reduce positional sensitivity and extract features at various resolutions



Evolution of CNNs

2015 ResNet (ILSVRC'15) 3.57

Year	Codename	Error (percent)	99.9% Conf Int
2014	**GoogLeNet**	**6.66**	**6.40 - 6.92**
2014	VGG	7.32	7.05 - 7.60
2014	MSRA	8.06	7.78 - 8.34
2014	AHoward	8.11	7.83 - 8.39
2014	DeeperVision	9.51	9.21 - 9.82
2013	Clarifai†	11.20	10.87 - 11.53
2014	CASIAWS†	11.36	11.03 - 11.69
2014	Trimps†	11.46	11.13 - 11.80
2014	Adobe†	11.58	11.25 - 11.91
2013	**Clarifai**	**11.74**	**11.41 - 12.08**
2013	NUS	12.95	12.60 - 13.30
2013	ZF	13.51	13.14 - 13.87
2013	AHoward	13.55	13.20 - 13.91
2013	OverFeat	14.18	13.83 - 14.54
2014	Orange†	14.80	14.43 - 15.17
2012	SuperVision†	15.32	14.94 - 15.69
2012	**SuperVision**	**16.42**	**16.04 - 16.80**
2012	ISI	26.17	25.71 - 26.65
2012	VGG	26.98	26.53 - 27.43
2012	XRCE	27.06	26.60 - 27.52
2012	UvA	29.58	29.09 - 30.04

Microsoft ResNet, a 152 layers network

GoogLeNet, 22 layers network

U. of Toronto, SuperVision, a 7 layers network

human error is around *5.1%* on a subset

Building and Training a CNN

```
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten()) # Reshape output from previous layer for input to next layer
model.add(Dense(128, activation='relu'))
model.add(Dense(10, activation='softmax'))
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam',
              metrics=['accuracy'])
model.fit(x, y, validation_split=0.2, epochs=10, batch_size=50)
```

Loading and Preparing Training Images

- **keras.preprocessing.image** has methods for loading images
  - **load_img** loads an image from the file system and resizes it if needed
  - **img_to_array** converts image returned by **load_img** into a NumPy array
- Divide pixel values by 255 before using them to train a CNN

```
# Load all images from a specified directory and prepare them for training
images = []
for file in os.listdir(path):
    img = image.load_img(os.path.join(path, file), target_size=(224, 224, 3))
    img = image.img_to_array(img) / 255
    images.append(img)
x = np.array(images) # Shape is (n, 224, 224, 3)
```

Demo

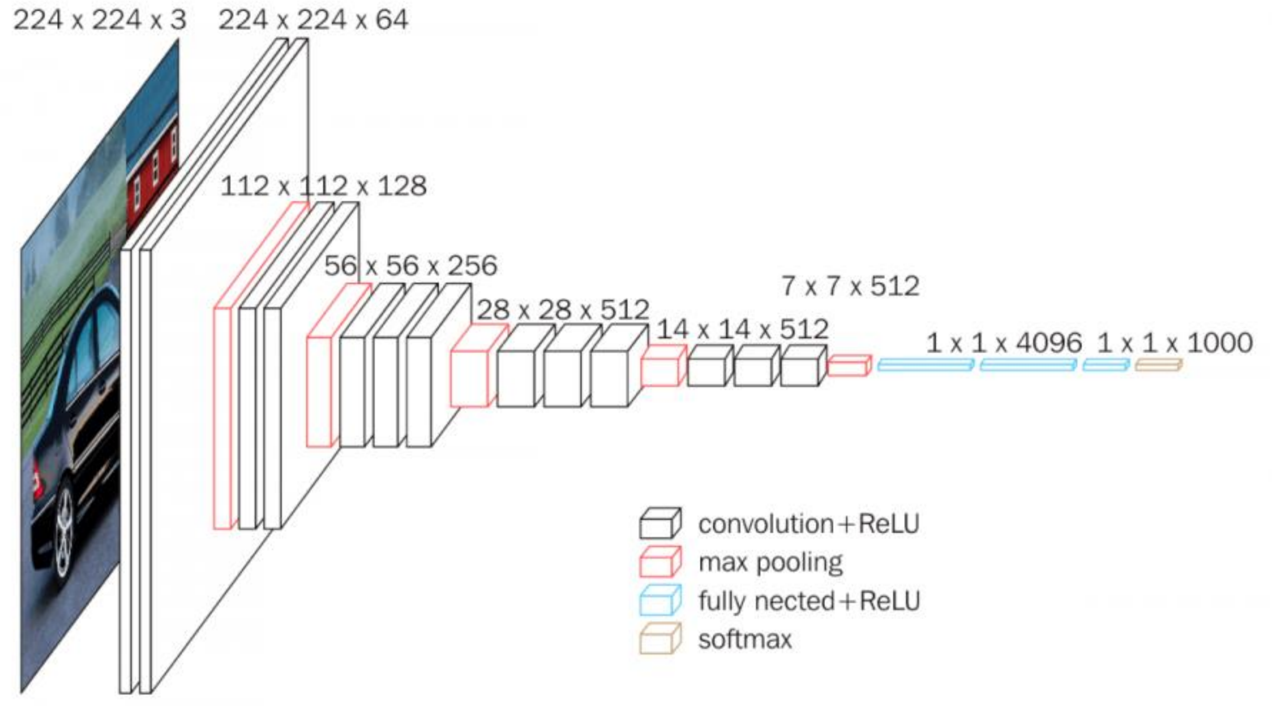
Convolutional Neural Networks

Pretrained CNNs

- Sophisticated CNNs built by Microsoft, Google, and others
- Trained on ImageNet dataset and published for anyone to use

VGG-16 convolutional neural network proposed by K. Simonyan and A. Zisserman of the University of Oxford in the paper “Very Deep Convolutional Networks for Large-Scale Image Recognition." The model achieved **92.7% top-5 test accuracy** on a subset of the ImageNet dataset containing almost **1.3 million images** and **1,000 classes**.

VGG-16 required **weeks of training using NVIDIA Titan GPUs** and is freely available to researchers.



Source: https://neurohive.io/en/popular-networks/vgg16/

Pretrained CNNs Included with Keras

Model	Accuracy	Versions
DenseNet	Up to 93.6%	DenseNet121, DenseNet169, and DenseNet201
EfficientNet	N/A	EfficientNetB0, EfficientNetB1, EfficientNetB2, EfficientNetB3, EfficientNetB4, EfficientNetB5, EfficientNetB6, and EfficientNetB7
Inception	Up to 95.3%	InceptionV3 and InceptionResNetV2
MobileNet	Up to 90.1%	MobileNet and MobileNetV2
NASNet	Up to 96.0%	NASNetMobile and NASNetLarge
ResNet	Up to 94.2%	ResNet50, ResNet50V2, ResNet101, ResNet101V2, ResNet152, and ResNet152V2
VGG	Up to 92.7%	VGG16 and VGG19
Xception	94.5%	Xception

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

Using VGG-16 to Classify Images

```
# Instantiate the model
model = VGG16(weights='imagenet')

# Load and preprocess the image to be classified
x = image.load_img('IMAGE_PATH', target_size=(224, 224))
x = image.img_to_array(x) # Converts image into (224, 224, 3) NumPy array
x = np.expand_dims(x, axis=0) # Converts (224, 224, 3) to (1, 224, 224, 3)
x = preprocess_input(x) # Performs network-specific preprocessing

# Use the model to classify the image
predictions = model.predict(x)
print(decode_predictions(predictions, top=5)[0])
```

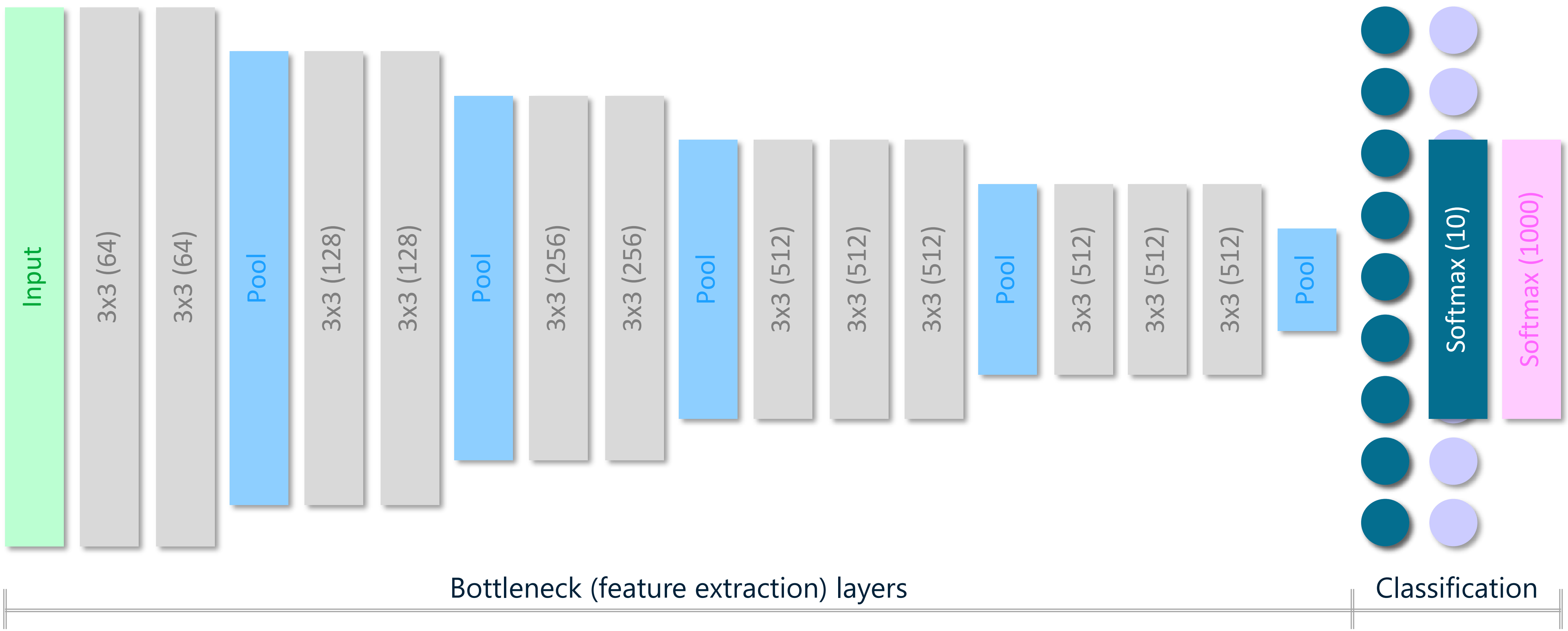
Demo

Pretrained CNNs

Transfer Learning

- Leverages pretrained CNNs to achieve acceptable accuracy with exponentially less data, compute power, and training time
  - Replaces fully connected classification layers in pretrained model with new layers, reusing pretrained model's feature-extraction layers
  - Allows image-classification models to be trained with as few as 50-100 images
  - Lessens need for GPUs (train on a PC or laptop)
- Repurposes pretrained CNNs to solve domain-specific problems
  - Train network to recognize classes it wasn't originally trained to recognize

How Transfer Learning Works



"Retraining" a Pretrained CNN

```
# Instantiate the model (minus the classification layers) and freeze the layers
base_model = VGG16(weights='imagenet', include_top=False)

for layer in base_model.layers:
    layer.trainable = False

# Add and train new classification layers
model = Sequential()
model.add(base_model)
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(10, activation='softmax'))
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.fit(x, y, validation_split=0.2, epochs=10, batch_size=10)
```

Making a Prediction

```
# Load and preprocess the image to be classified
x = image.load_img('IMAGE_PATH', target_size=(224, 224))
x = image.img_to_array(x)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

# Pass the image to the model's predict() method
y = model.predict(x)
```

Fast Transfer Learning

```
# Instantiate the model (minus the classification layers)
base_model = VGG16(weights='imagenet', include_top=False)

# Run the images through the base model
x = base_model.predict(x)

# Build a network for classification and train it with the output
model = Sequential()
model.add(Flatten(input_shape=x.shape[1:]))
model.add(Dense(128, activation='relu'))
model.add(Dense(10, activation='softmax'))
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.fit(x, y, validation_split=0.2, epochs=10, batch_size=10)
```

Making a Prediction

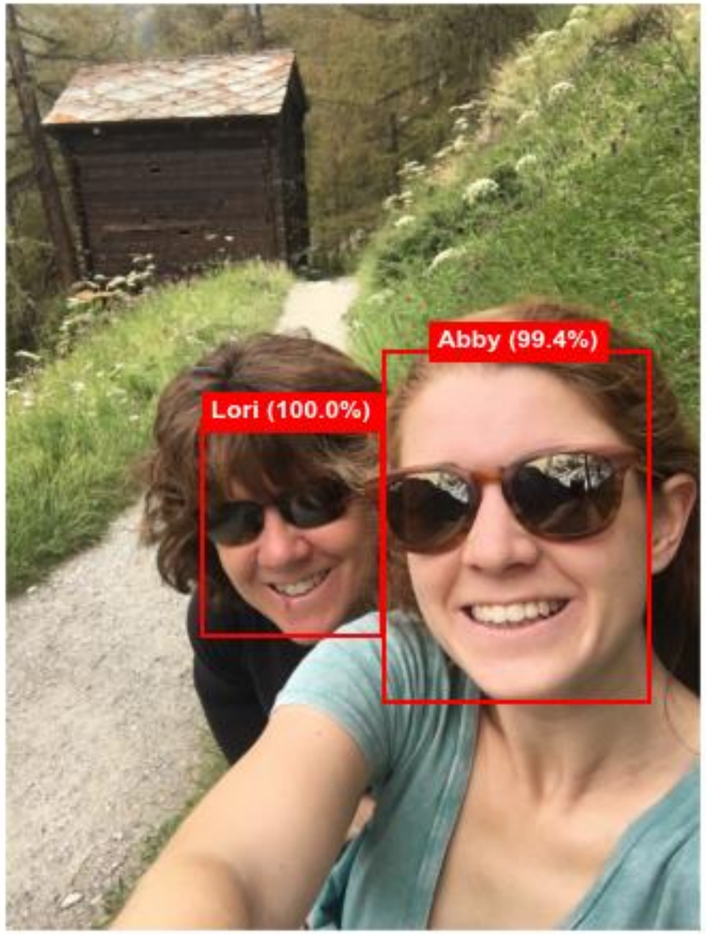
```python
# Load and preprocess the image to be classified
x = image.load_img('IMAGE_PATH', target_size=(224, 224))
x = image.img_to_array(x)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

# Pass the image to the base model's predict() method for feature extraction, and
# then pass the extracted features to the model's predict() method for classification
features = base_model.predict(x)
y = model.predict(features)
```

Demo

Transfer Learning

Face Detection

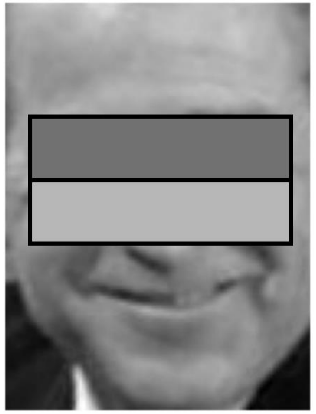
- Facial recognition is a 2-step challenge
  - Find (detect) the faces in an image
  - Identify (recognize) the faces in an image
- Detection can be performed in many ways:
  - Cascade classifier (Viola-Jones)
  - Histogram of Oriented Gradients (HoG)
  - Multitask Cascaded CNNs (MTCNNs)
- Use Viola-Jones for speed, MTCNN for accuracy



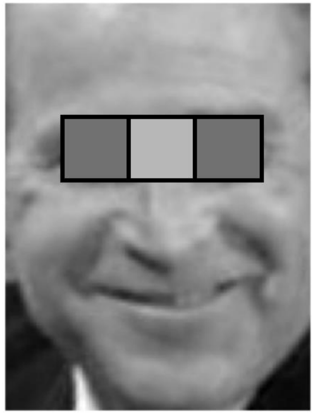
Viola-Jones

- Detects faces by examining photos for Haar-like features
- Uses *integral images* to quickly calculate differences in intensity between arbitrary adjacent blocks of pixels

Photo of George W. Bush from the **Labeled Faces in the Wild** dataset



Two-rectangle Haar-like feature possibly indicative of **eyes**, **brow**, and **cheeks**



Three-rectangle Haar-like feature possibly indicative of **eyes** and **bridge of nose**

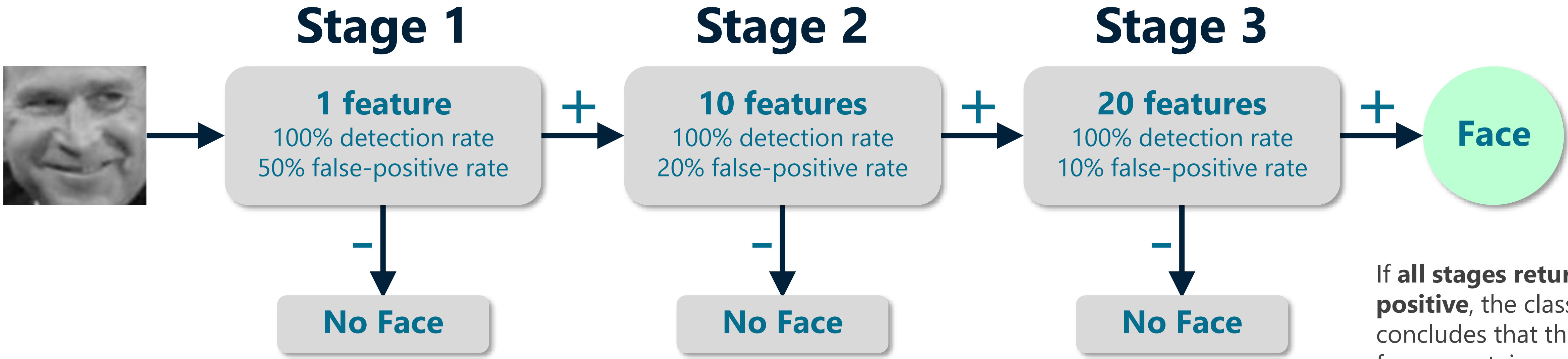
Original Image

2	4	1	2
1	1	2	2
3	2	2	1

Integral Image

2	6	7	9
3	8	11	15
6	13	18	23

Cascade Classifiers



Stage 1 uses **one feature** to determine whether the frame input to the classifier contains a face. A positive response means it **might**, while a negative response means it **conclusively does not** and ends the cascade.

Stage 2 uses **10 features** to determine whether the frame contains a face. A positive response means it **might**, while a negative response means it **does not**. The cumulative false-positive rate on output is 20% of 50%, or **10%**.

Stage 3 uses **20 features** to determine whether the frame contains a face. A positive response means it **might**, while a negative response means it **does not**. The cumulative false-positive rate on output is 10% of 10%, or just **1%**.

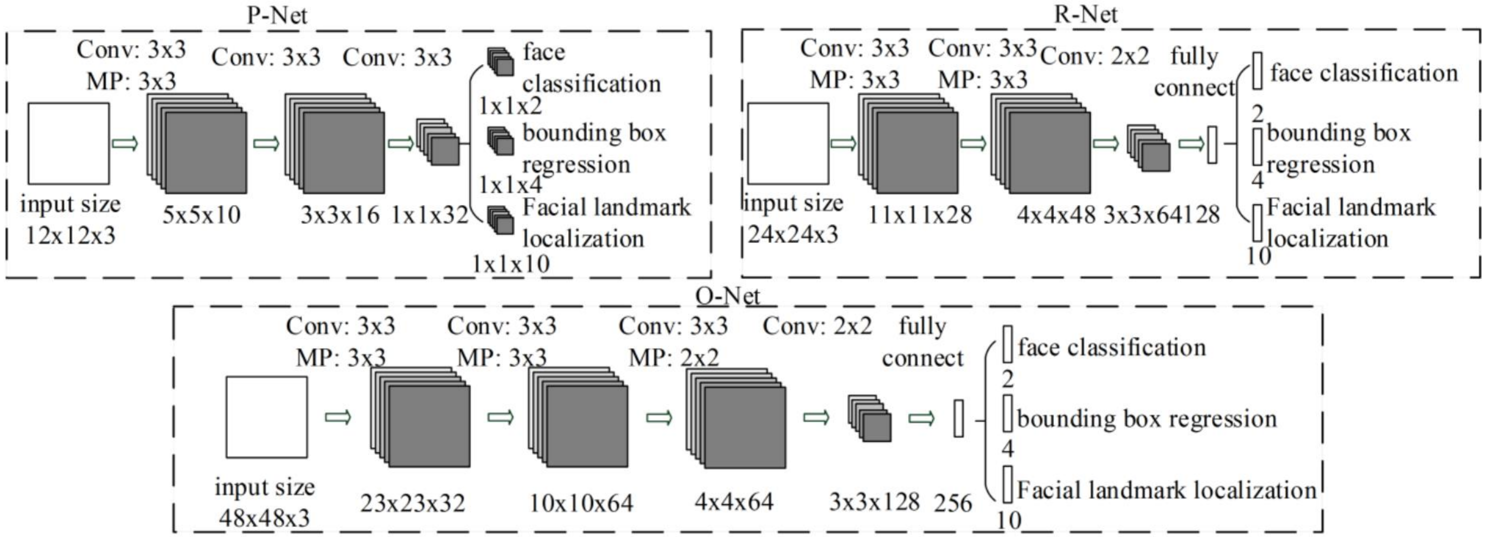
If **all stages return positive**, the classifier concludes that the frame contains a face. With just **three stages**, the error rate is 1%. More stages **reduce the error rate exponentially**.

Using OpenCV's *CascadeClassifier* Class

```python
import cv2
from cv2 import CascadeClassifier
import matplotlib.pyplot as plt

image = plt.imread('PATH_TO_IMAGE_FILE')
model = CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
faces = model.detectMultiScale(image)

for face in faces:
    x, y, w, h = face
    print(f'Upper left: ({x}, {y}), Lower right: ({x + w}, {y + h})')
```

MTCNN

- Multitask cascaded convolutional neural networks (MTCNNs) use CNNs arranged in three stages to identify and refine bounding boxes
- Implementation available in Python package named **MTCNN**



P-Net - Shallow CNN that searches at various resolutions for features indicative of faces

R-Net - Deeper CNN that examines candidate rectangles more closely and rejects those that lack faces

O-Net - Filters candidate rectangles and identifies facial landmarks

Using the *MTCNN* Class

```
from mtcnn.mtcnn import MTCNN
import matplotlib.pyplot as plt

detector = MTCNN()
image = plt.imread('PATH_TO_IMAGE_FILE')
faces = detector.detect_faces(image)

for face in faces:
    x, y, w, h = face['box']
    print(f'Upper left: ({x}, {y}), Lower right: ({x + w}, {y + h})')
```

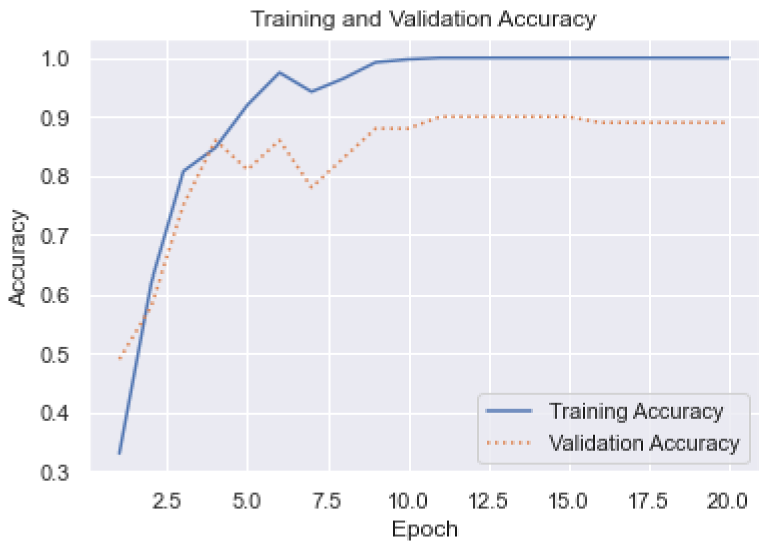
Demo

Face Detection

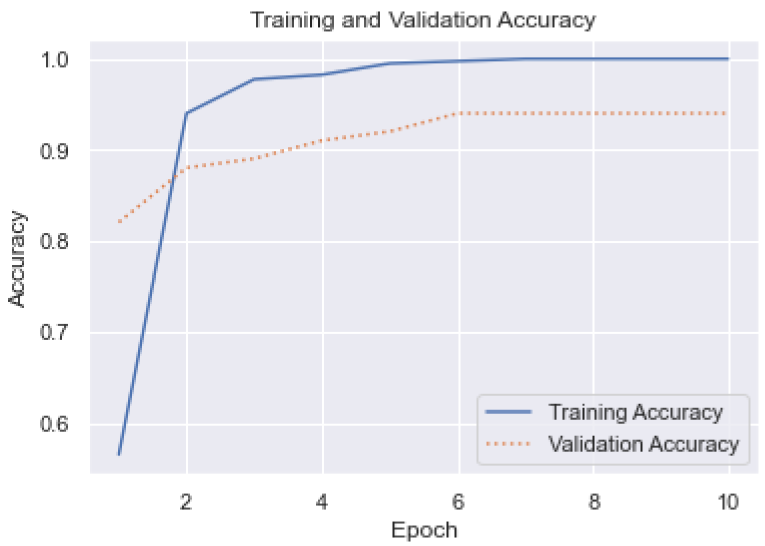
Using CNNs to Recognize Faces

- CNN trained from scratch on the LFW dataset achieves 90% accuracy
- Applying **ResNet50** with transfer learning boosts accuracy to 93%

CNN trained from scratch on LFW dataset



Transfer learning with ResNet50



VGGFace2

- Version of **ResNet50** trained on more than 3 million facial images by University of Oxford's Visual Geometry Group (VGG)
  - Trained to recognize thousands of celebrities
  - Excels at extracting features from facial images
- Weights published for anyone to use
- Python package **keras-vggface** contains trained model with TensorFlow-compatible weights and **VGGFace** class encapsulating those weights

arXiv:1710.08092v2 [cs.CV] 13 May 2018

VGGFace2: A dataset for recognising faces across pose and age

Qiong Cao, Li Shen, Weidi Xie, Omkar M. Parkhi and Andrew Zisserman
Visual Geometry Group, Department of Engineering Science, University of Oxford
{qiong,lishen,weidi,omkar,az}@robots.ox.ac.uk

***Abstract*—In this paper, we introduce a new large-scale face dataset named VGGFace2. The dataset contains 3.31 million images of 9131 subjects, with an average of 362.6 images for each subject. Images are downloaded from Google Image Search and have large variations in pose, age, illumination, ethnicity and profession (e.g. actors, athletes, politicians).**

The dataset was collected with three goals in mind: (i) to have both a large number of identities and also a large number of images for each identity; (ii) to cover a large range of pose, age and ethnicity; and (iii) to minimise the label noise. We describe how the dataset was collected, in particular the automated and manual filtering stages to ensure a high accuracy for the images of each identity.

To assess face recognition performance using the new dataset, we train ResNet-50 (with and without Squeeze-and-Excitation blocks) Convolutional Neural Networks on VGGFace2, on MS-Celeb-1M, and on their union, and show that training on VGGFace2 leads to improved recognition performance over pose and age. Finally, using the models trained on these datasets, we demonstrate state-of-the-art performance on the face recognition of IJB datasets, exceeding the previous state-of-the-art by a large margin. The dataset and models are publicly available[1].

***Keywords*-face dataset; face recognition; convolutional neural networks**

I. INTRODUCTION

Concurrent with the rapid development of deep Convolutional Neural Networks (CNNs), there has been much recent effort in collecting large scale datasets to feed these data-hungry models. In general, recent datasets (see Table I) have explored the importance of intra- and inter-class variations. The former focuses on depth (many images of one subject) and the latter on breadth (many subjects with limited images per subject). However, none of these datasets was specifically designed to explore pose and age variation. We address that here by designing a dataset generation pipeline to explicitly collect images with a wide range of pose, age, illumination and ethnicity variations of human faces.

We make the following *four* contributions: first, we have collected a new large scale dataset, VGGFace2, for public release. It includes over nine thousand identities with between 80 and 800 images for each identity, and more than 3M images in total; second, a dataset generation pipeline is proposed that encourages pose and age diversity for each subject, and also involves multiple stages of automatic and manual filtering in order to minimise label noise; third, we provide template annotation for the test set to explicitly explore pose and age recognition performance; and, finally, we show that training deep CNNs on the new dataset substantially exceeds the state-of-the-art performance on the IJB benchmark datasets [13], [14], [23]. In particular, we experiment with the recent Squeeze and Excitation network [9], and also investigate the benefits of first pre-training on a dataset with breadth (MS-Celeb-1M [7]) and then fine tuning on VGGFace2.

The rest of the paper is organised as follows: We review previous dataset in Section II, and give a summary of existing public dataset in Table I. Section III gives an overview of the new dataset, and describes the template annotation for recognition over pose and age. Section IV describes the dataset collection process. Section V reports state-of-the-art performance of several different architectures on the IJB-A [13], IJB-B [23] and IJB-C [14] benchmarks.

II. DATASET REVIEW

In this section we briefly review the principal "in the wild" datasets that have appeared recently. In 2007, the Labelled Faces in the Wild (LFW) dataset [10] was released, containing 5,749 identities with 13,000 images.

The CelebFaces+ dataset [21] was released in 2014, with 202,599 images of 10,177 celebrities. The CASIA-WebFace dataset [26] released the same year that has 494,414 images of 10,575 people. The VGGFace dataset [17] released in 2015 has 2.6 million images covering 2,622 people, making it amongst the largest publicly available datasets. The curated version, where label noise is removed by human annotators, has 800,000 images with approximately 305 images per identity. Both the CASIA-WebFace and VGGFace datasets were released for training purposes only.

MegaFace dataset [12] was released in 2016 to evaluate face recognition methods with up to a million distractors in the gallery image set. It contains 4.7 million images of 672,057 identities as the training set. However, an average of only 7 images per identity makes it restricted in its per identity face variation. In order to study the effect of pose and age variations in recognising faces, the MegaFace challenge [12] uses the subsets of FaceScrub [15] containing 4,000 images from 80 identities and FG-NET [16] containing 975 images from 82 identities for evaluation.

Microsoft released the large Ms-Celeb-1M dataset [7] in 2016 with 10 million images from 100k celebrities for

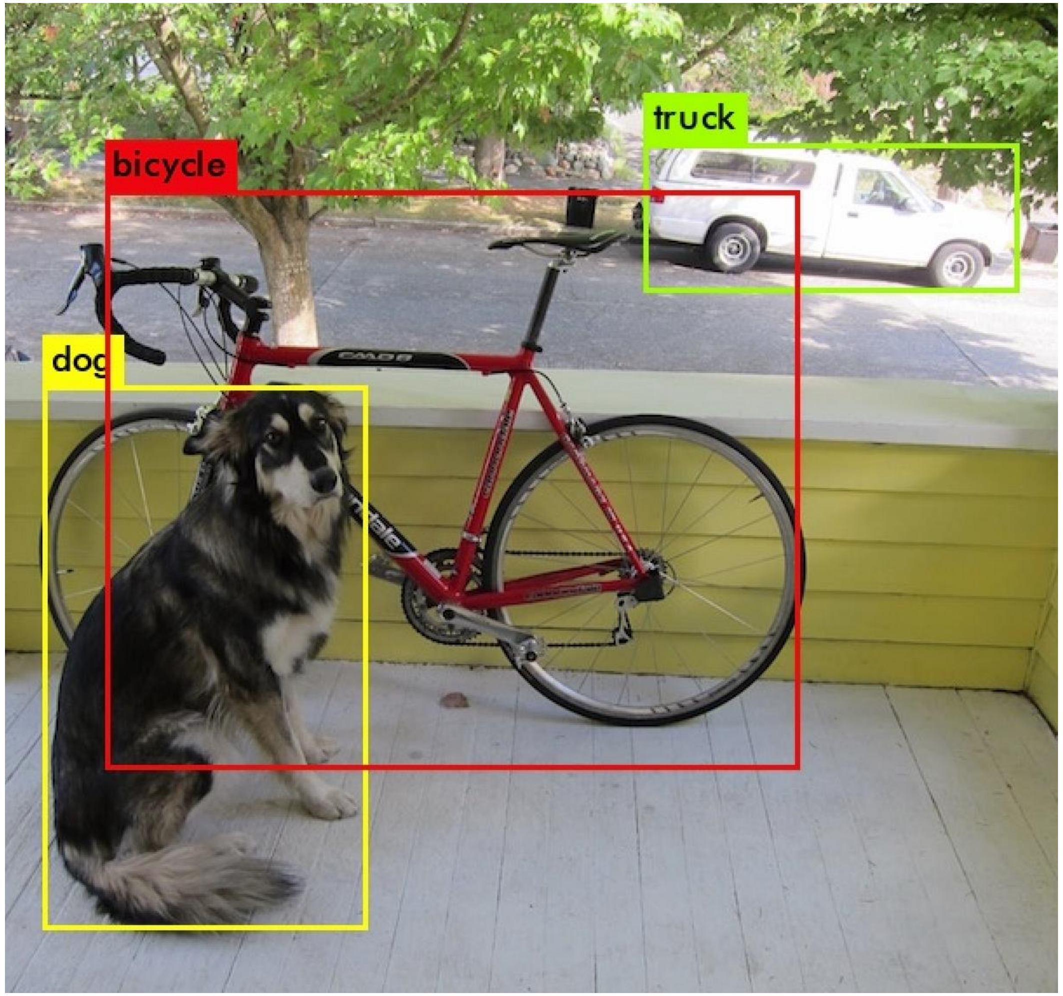
[1] http://www.robots.ox.ac.uk/~vgg/data/vgg_face2/

Demo

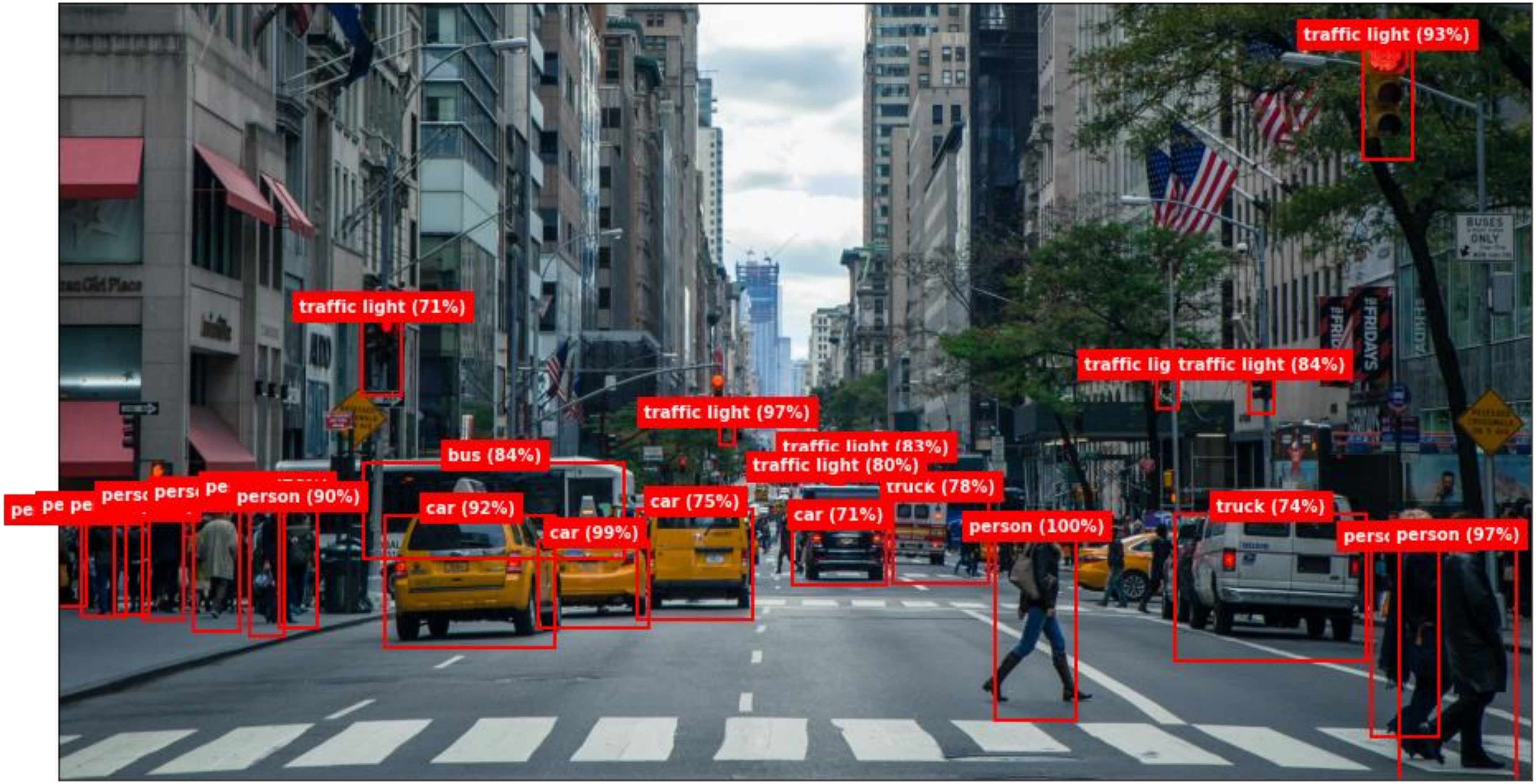
Facial Recognition

Object Detection

- How do self-driving cars find objects in video frames and identify them in real time?
- State-of-the-art object-detection systems rely on CNNs
  - Region-based CNNs (R-CNNs)
  - You Only Look Once (YOLO)
- Trained on popular labeled datasets such as COCO and Open Images

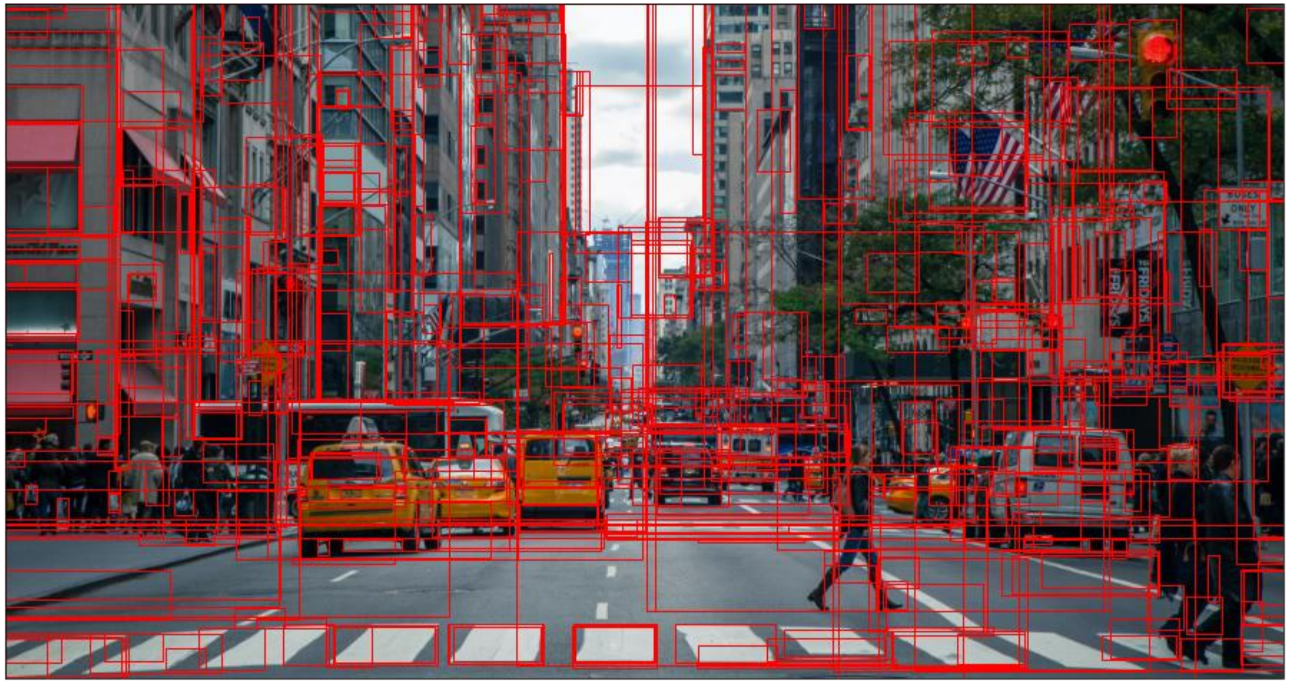


What a Self-Driving Car Sees



Selective Search

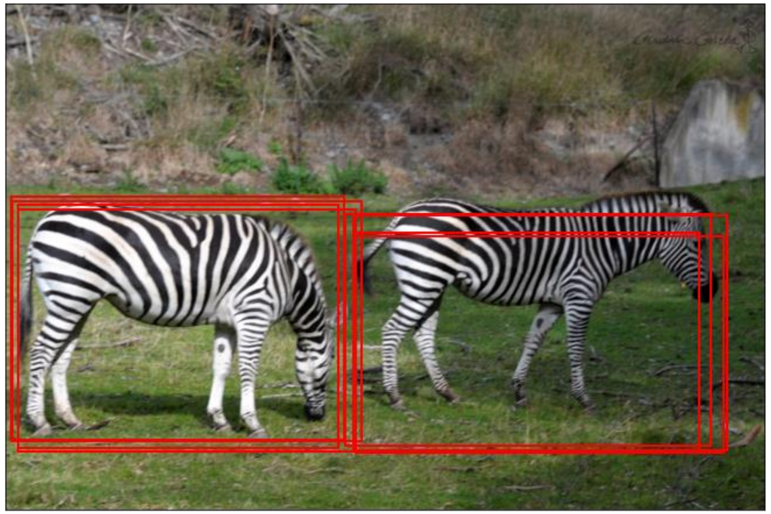
- Used by some region-based CNNs to identify regions of interest by keying on similarities in color, texture, shape, and size
- Implemented in OpenCV's **SelectiveSearchSegmentation** class

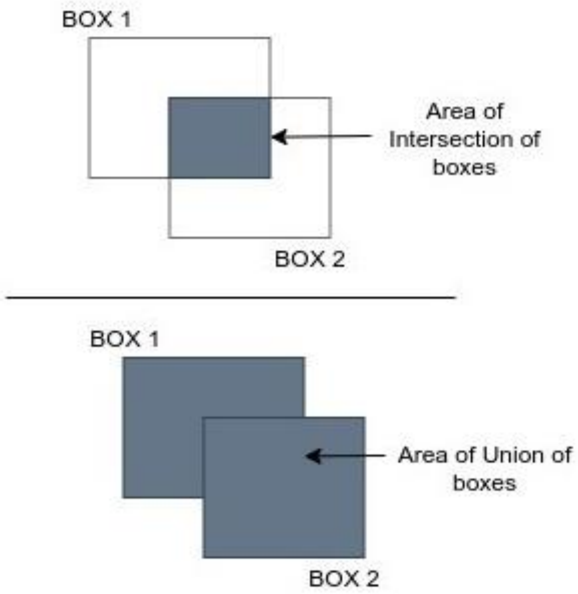


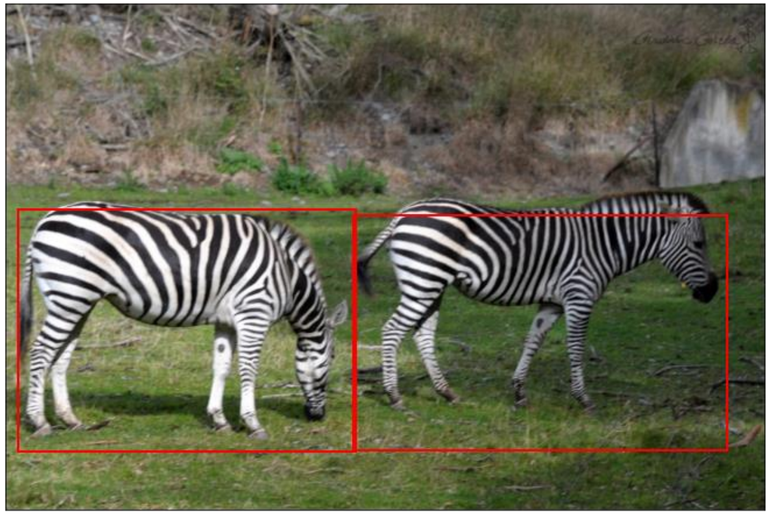
Non-Maximum Suppression (NMS)

- Candidate objects are usually identified by multiple bounding boxes
- NMS picks the best bounding box for each object using IoU algorithm

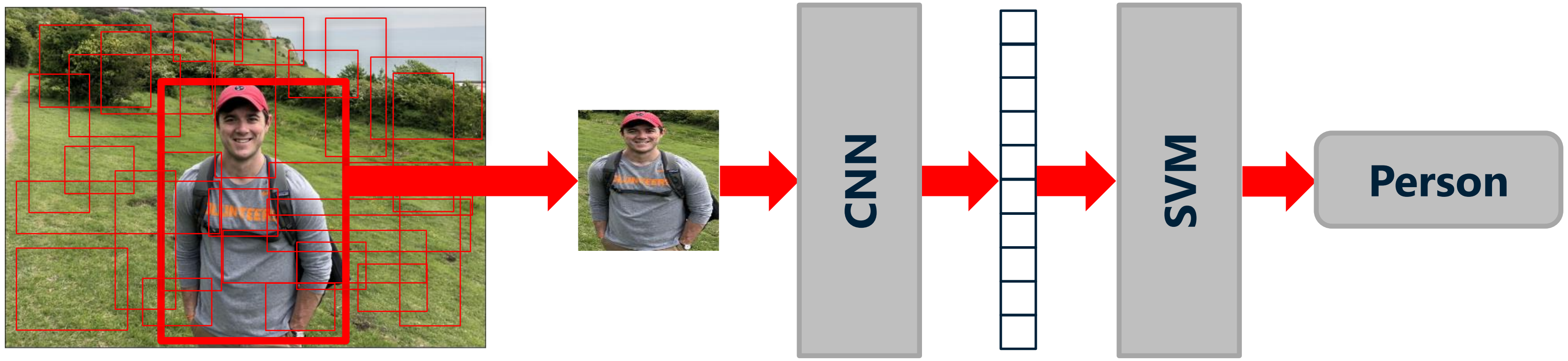
Intersection over Union (IoU)







R-CNN



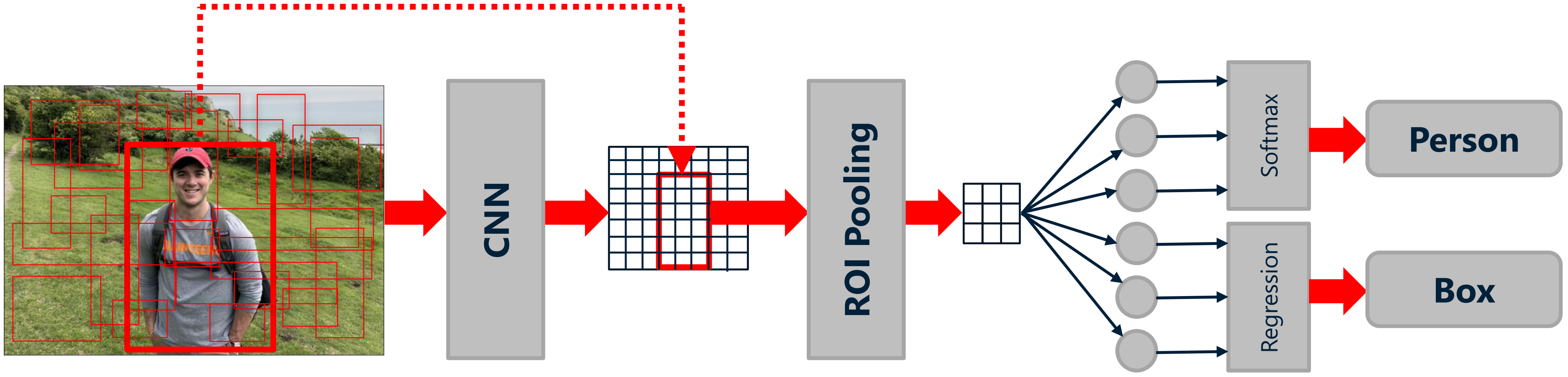
Regions of interest are identified using **selective search** or a similar algorithm.

Each region of interest is **scaled** and **input to a deep CNN** for feature extraction. The output is a feature vector uniquely characterizing the region.

The feature vector is input to a **support-vector machine** for classification. The SVM yields a **class label** and a **confidence score**. NMS identifies the best bounding box for each object.

Fast R-CNN

Regions of interest are **projected to the feature map** generated by the CNN.



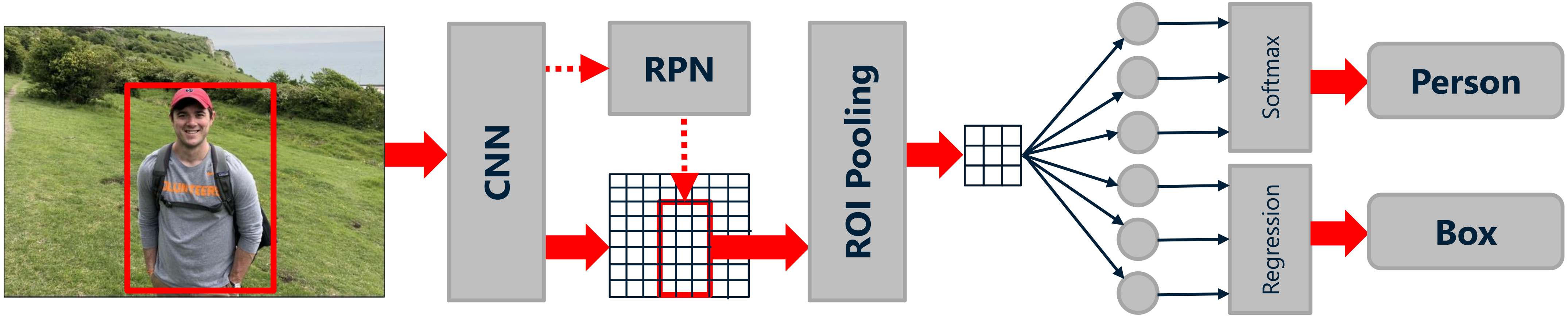
Regions of interest are identified using **selective search** or a similar algorithm. The **entire image** is passed to a CNN for feature extraction.

Each region projected to the feature map is reduced to a fixed-size feature vector using **ROI pooling**.

Feature vectors are flattened and input to **fully connected layers** for classification and regression. Output is split to predict a **class and confidence level** and a **bounding box**. NMS picks the best bounding box for each object.

Faster R-CNN

Features from the first few layers of the CNN are input to a **Region Proposal Network** to identify regions of interest. The RPN slides a window over the feature map to evaluate candidate regions defined by **anchor boxes** — typically 9 boxes of different sizes and aspect ratios.



The **entire image** is passed to a CNN for feature extraction.

Each region proposed by the RPN is reduced to a fixed-size feature vector using **ROI pooling**.

Feature vectors are flattened and input to **fully connected layers** for classification and regression. Output is split to predict a **class and confidence level** and a **bounding box**. NMS picks the best bounding box for each object.

Mask R-CNN

- Adds *instance segmentation* to Faster R-CNN
  - Identifies individual pixels belonging to objects
  - Provides additional context regarding those objects
- Used by Zoom to display custom backgrounds
- ONNX implementation available from Facebook Research

Instance segmentation provides more detail about objects in a scene – for example, whether a person's **arms are extended** or whether that person is **standing up** or **lying down**

Demo

Object Detection with Mask R-CNN