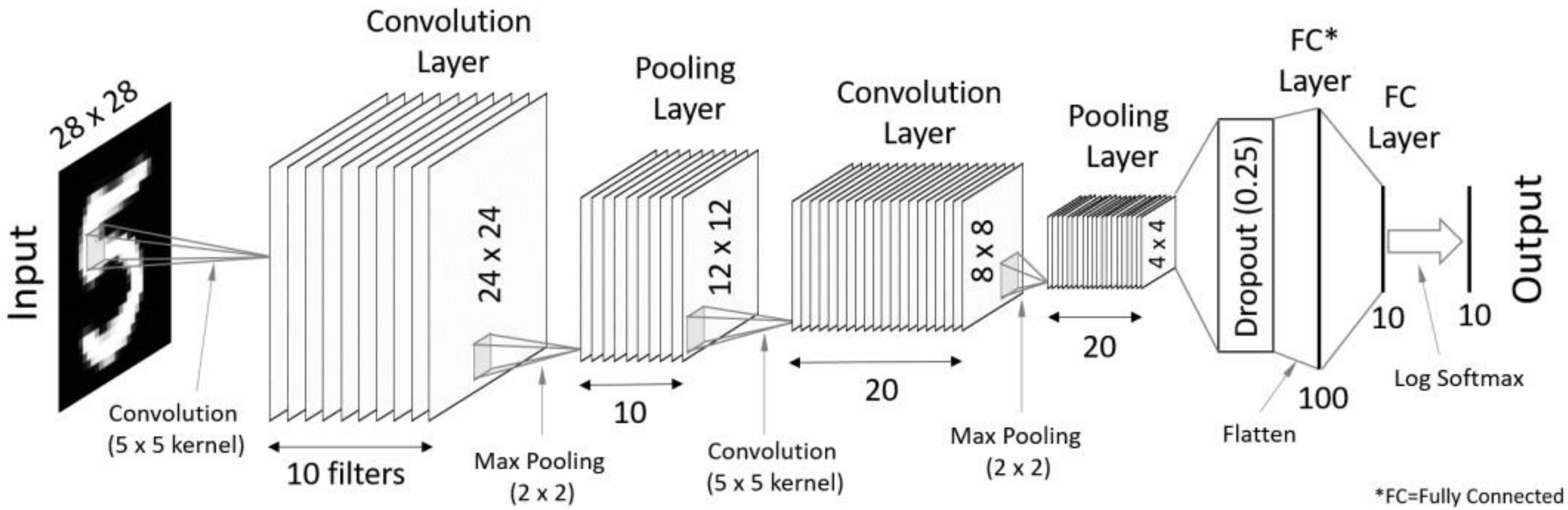
IMAGE PROCESSING WITH DEEP NEURAL NETWORKS

Jeff Prosise

Gold Microsoft Partner Microsoft | Azure Expert MSP

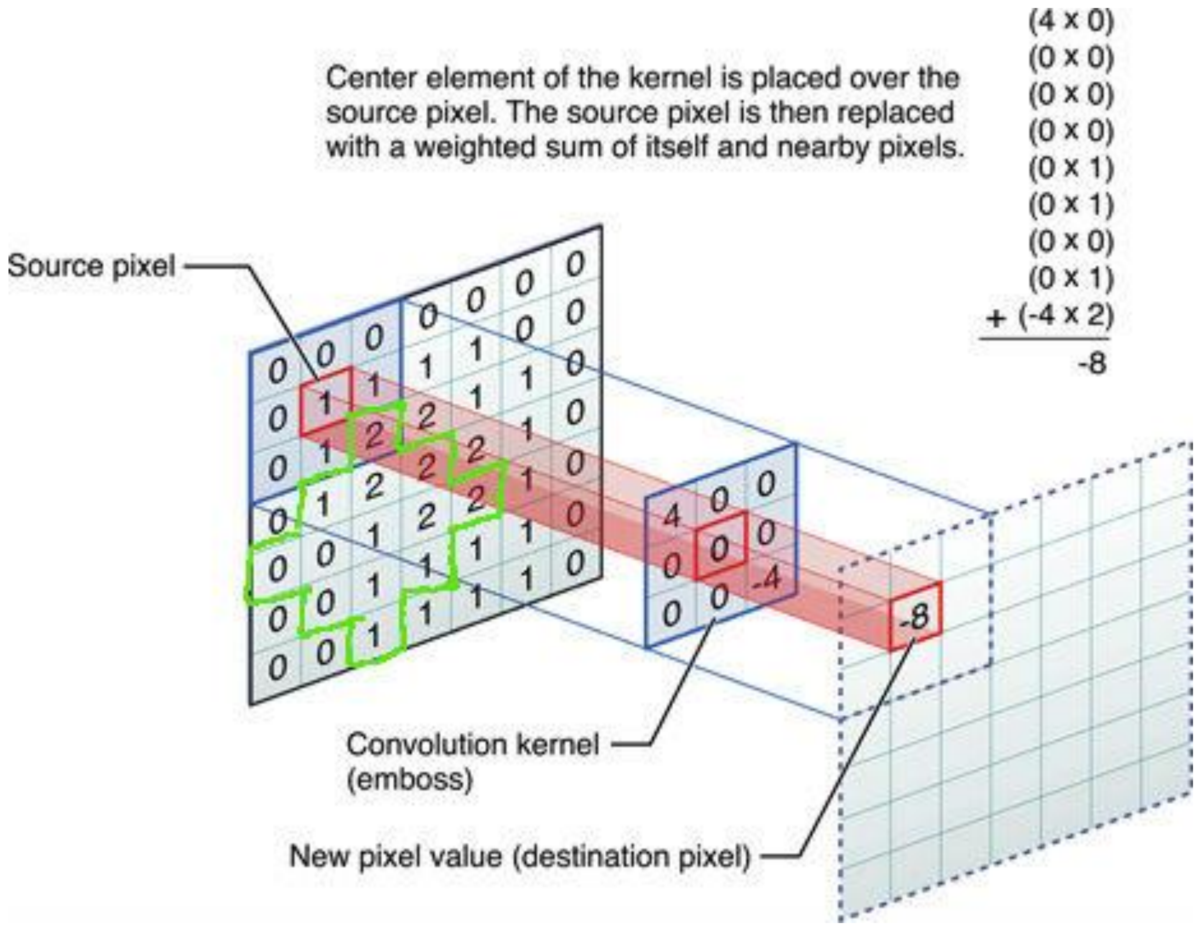
Convolutional Neural Networks (CNNs)

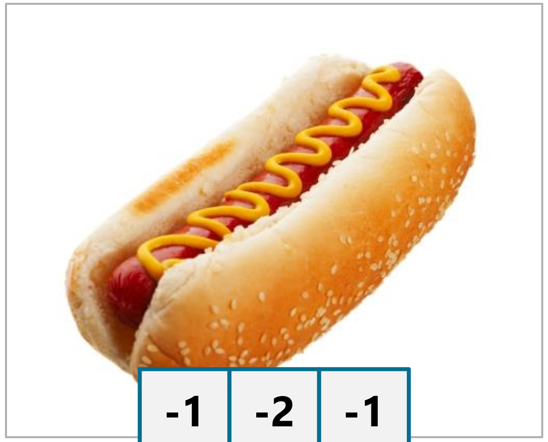
- Excel at computer-vision tasks such as image classification
- Use convolution layers and convolution kernels to create feature maps
- Use pooling layers to subsample feature maps and generalize features



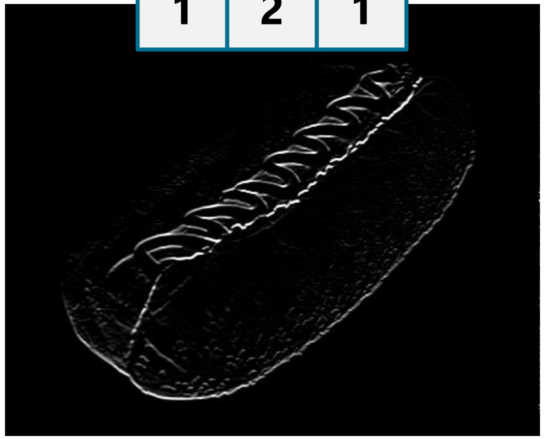
Source: https://codetolight.wordpress.com/2017/11/29/getting-started-with-pytorch-for-deep-learning-part-3-neural-network-basics/

Convolution Kernels





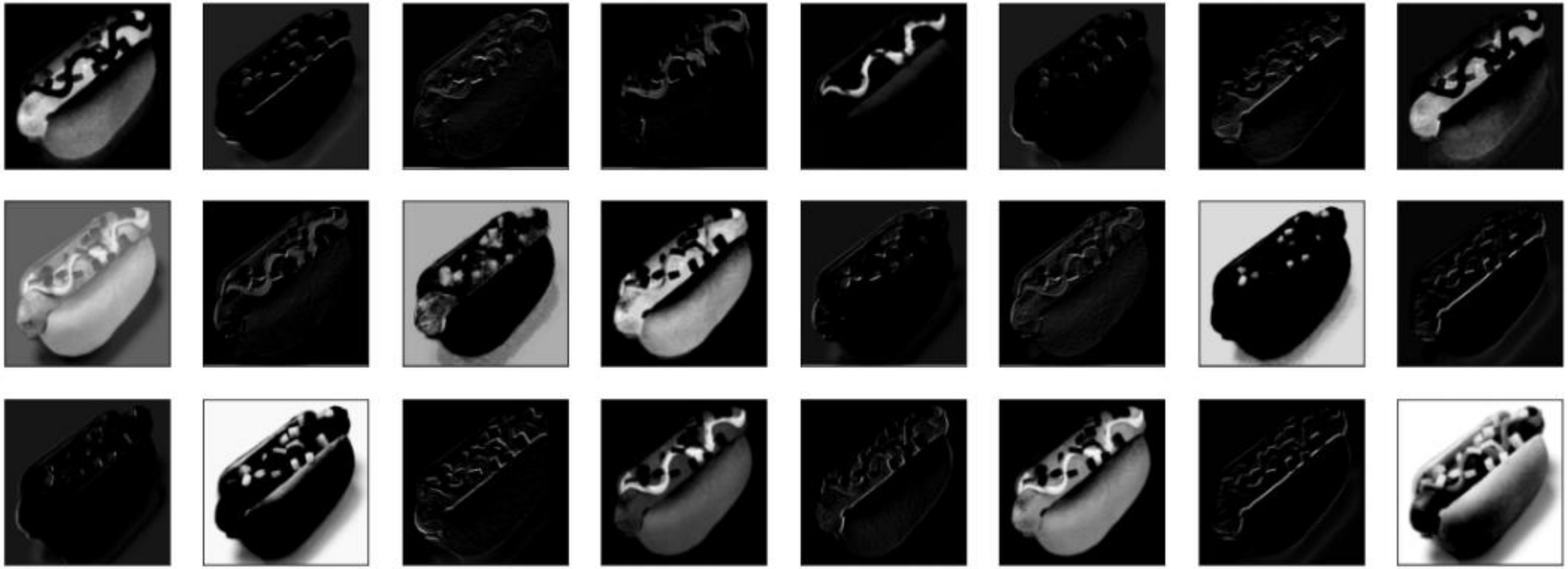
-1	-2	-1
0	0	0
1	2	1



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

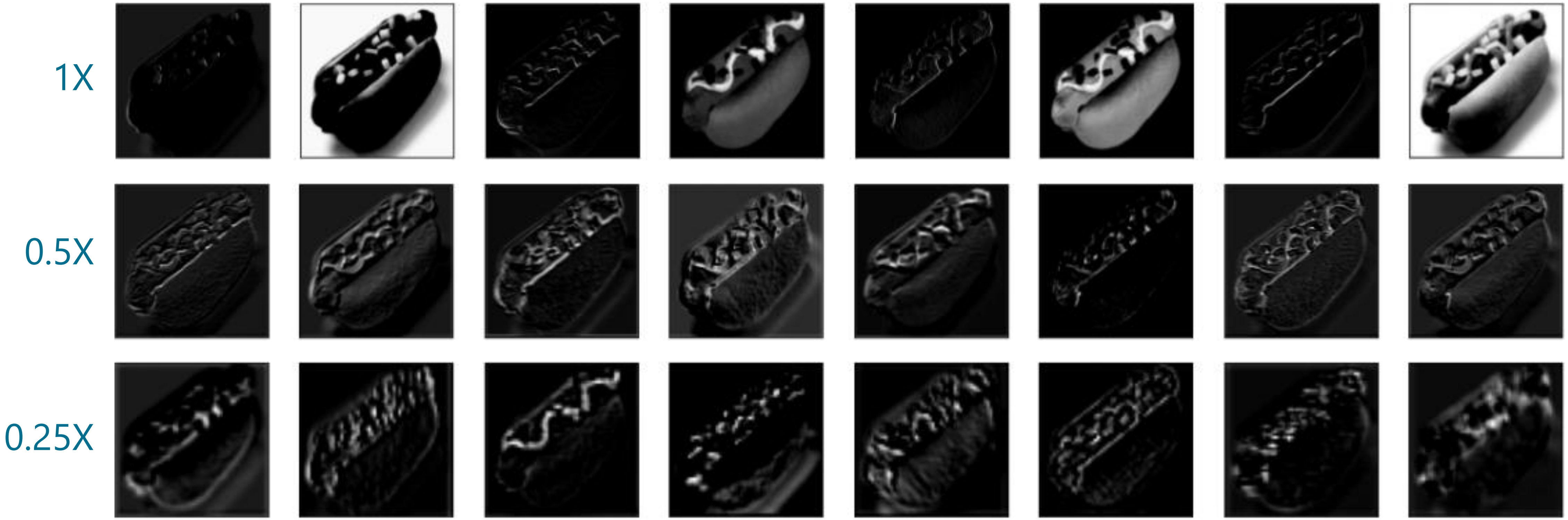
Convolution Layers

- 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)
```

Demo

Convolutional Neural Networks

atmosera®

Gold
Microsoft
Partner
Microsoft

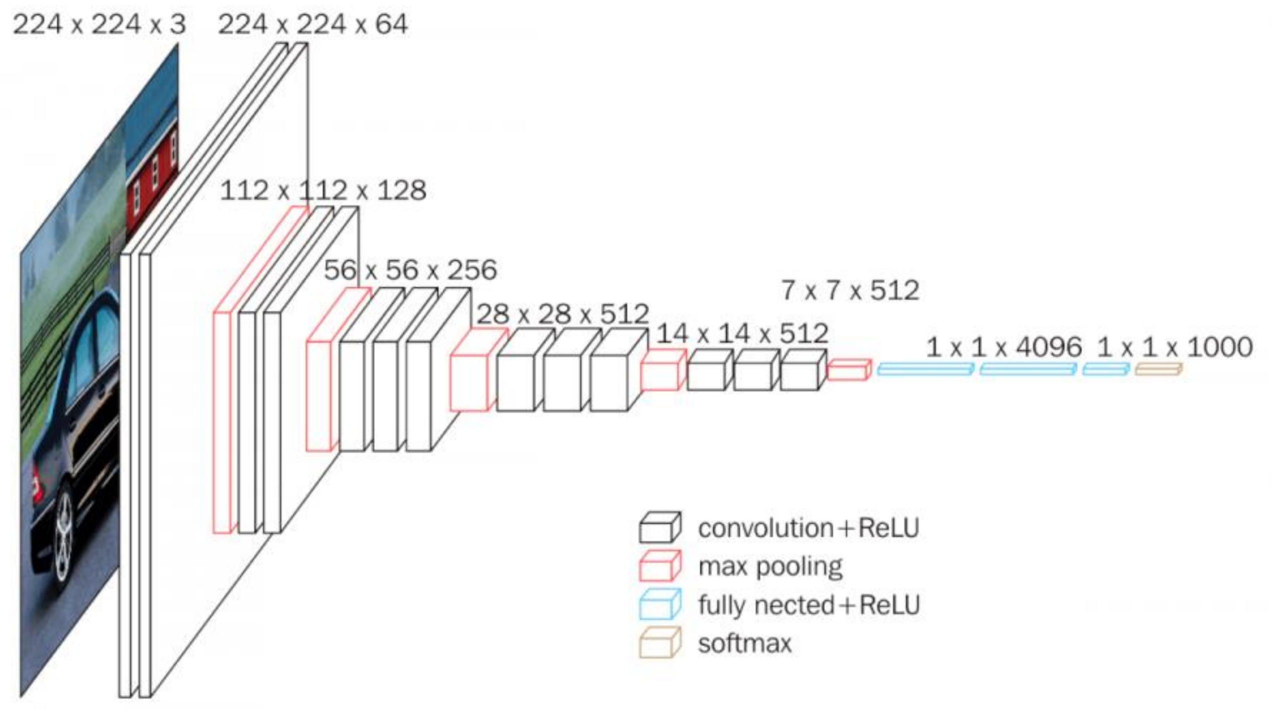
Azure
Expert
MSP

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/

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])
```

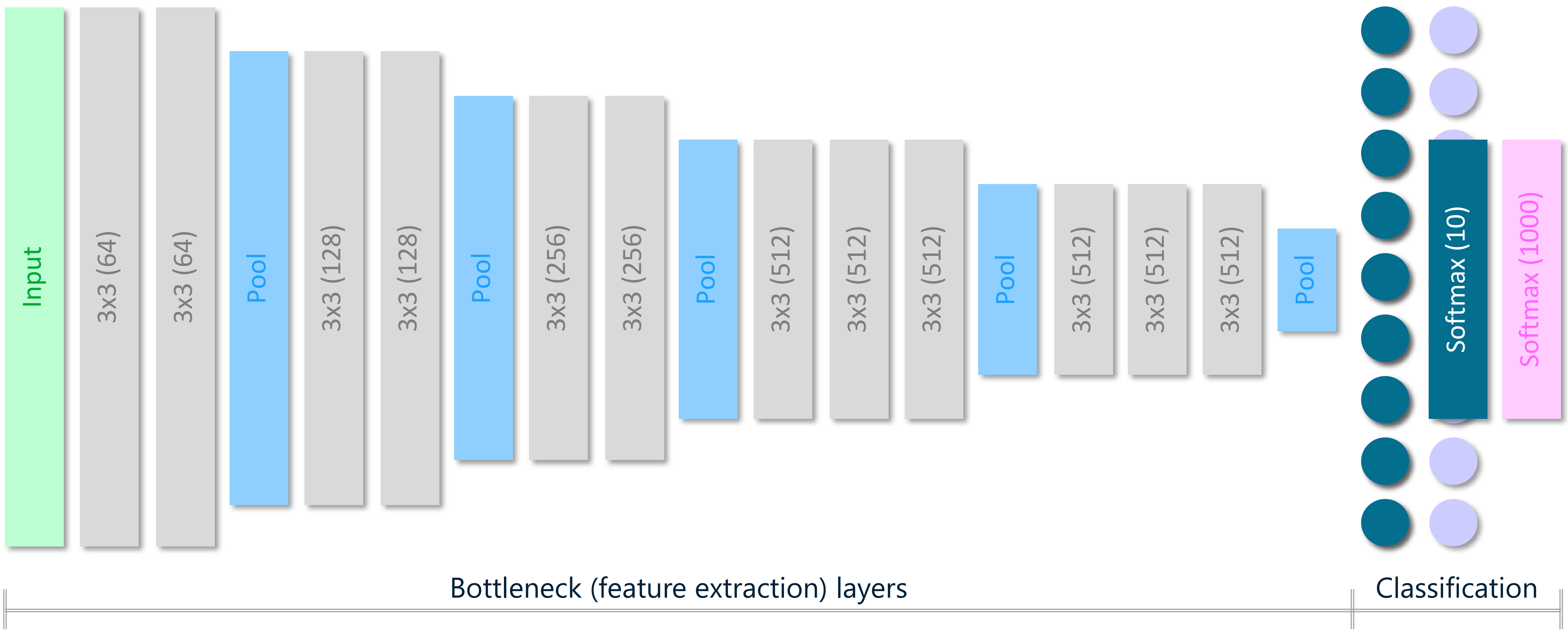
Demo

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

```
# 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

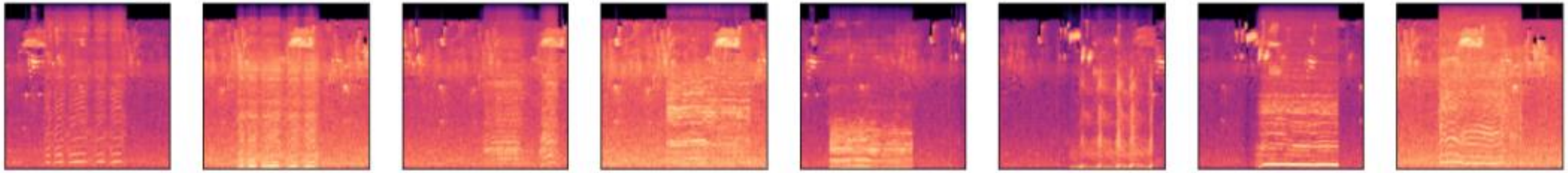
Transfer Learning

atmosera®

Gold Microsoft Partner Microsoft | Azure Expert MSP

Audio Classification

- Audio classification is often performed by converting audio samples into spectrogram images and using a CNN to classify images



- Python's **Librosa** package generates spectrograms from audio samples
- Rainforest Connection uses this technique to combat illegal logging and illegal poaching in the Amazon rainforest

Demo

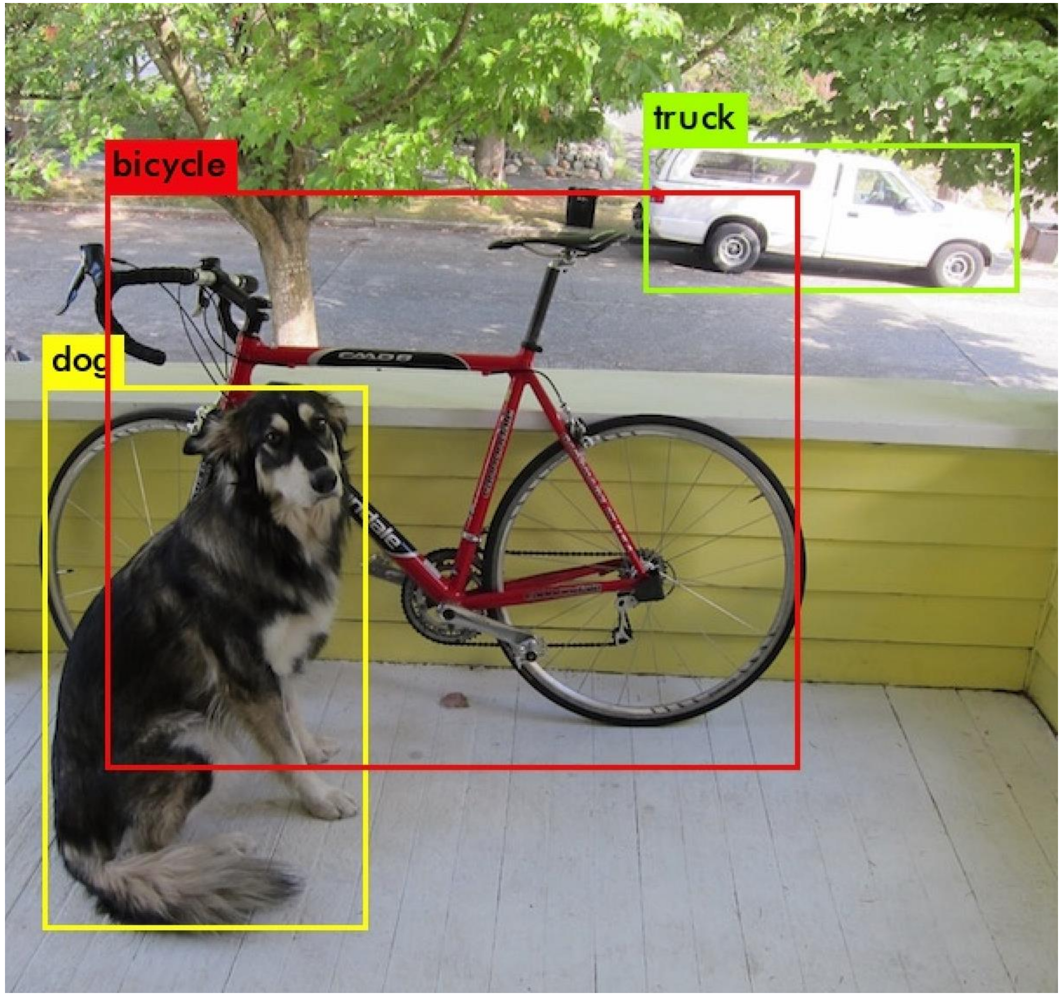
Audio Classification

atmosera®

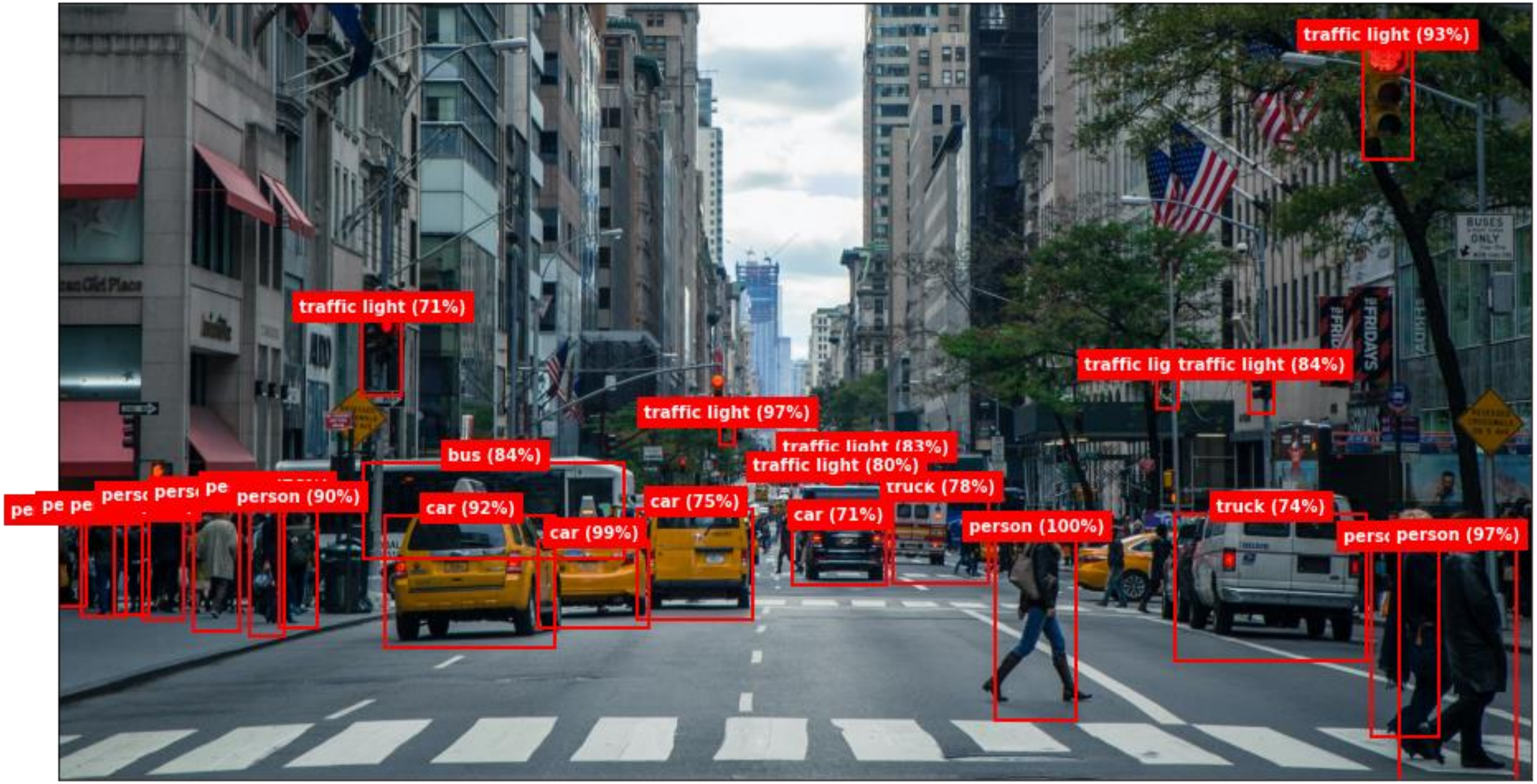
Gold Microsoft Partner Microsoft | Azure Expert MSP

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

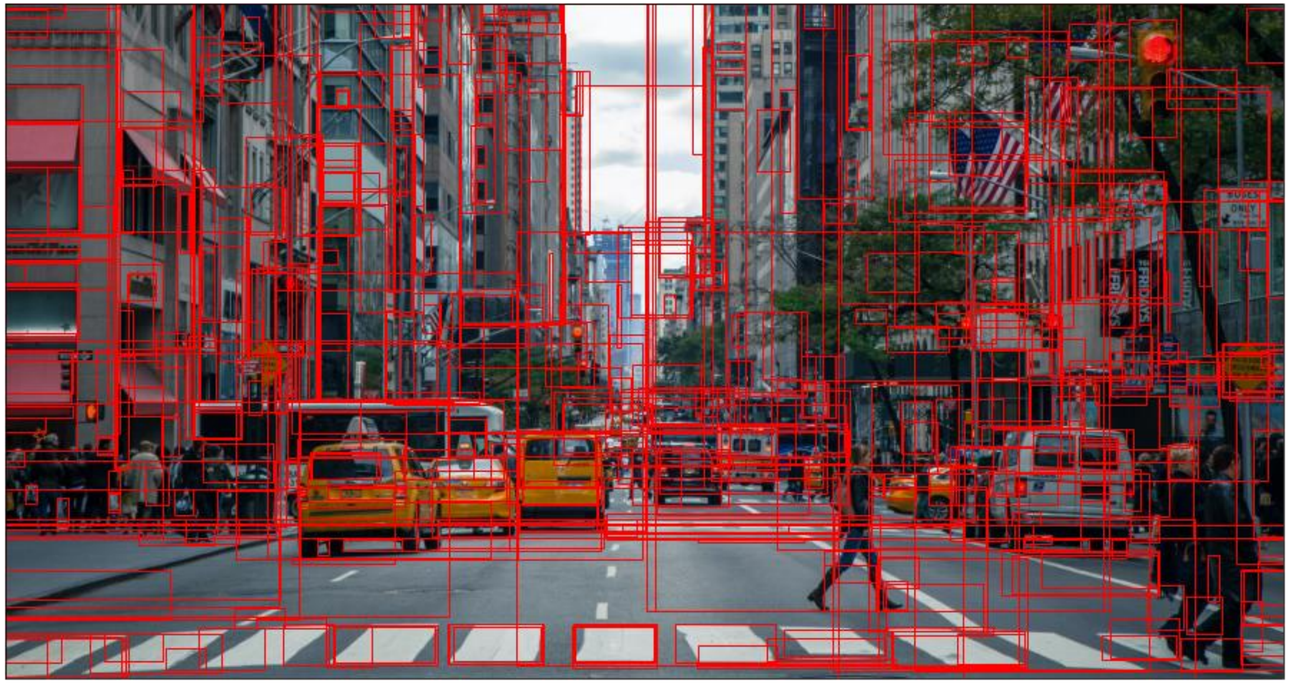


What a Self-Driving Car Sees



Selective Search

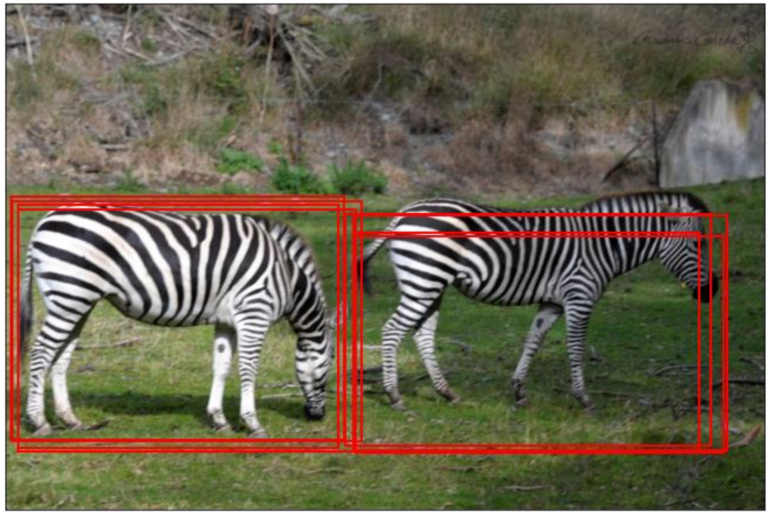
- Used by some region-based CNNs to identify regions of interest by keying on similarities in color, texture, shape, and size
- Implemented in OpenCV's **SelectiveSearchSegmentation** class

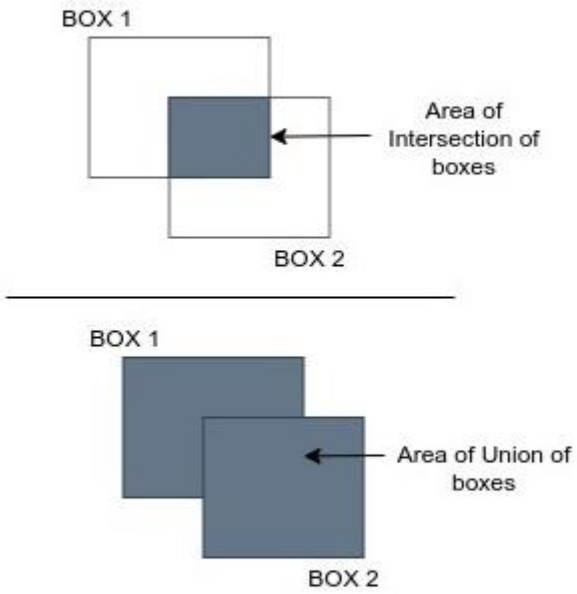


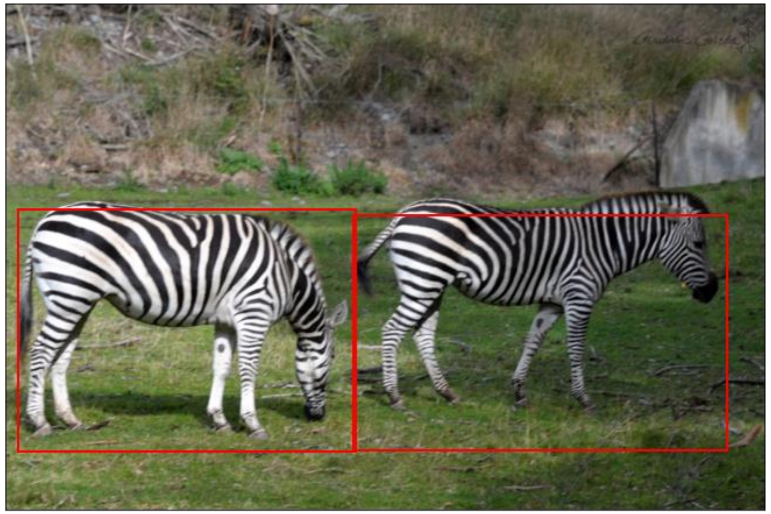
Non-Maximum Suppression (NMS)

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

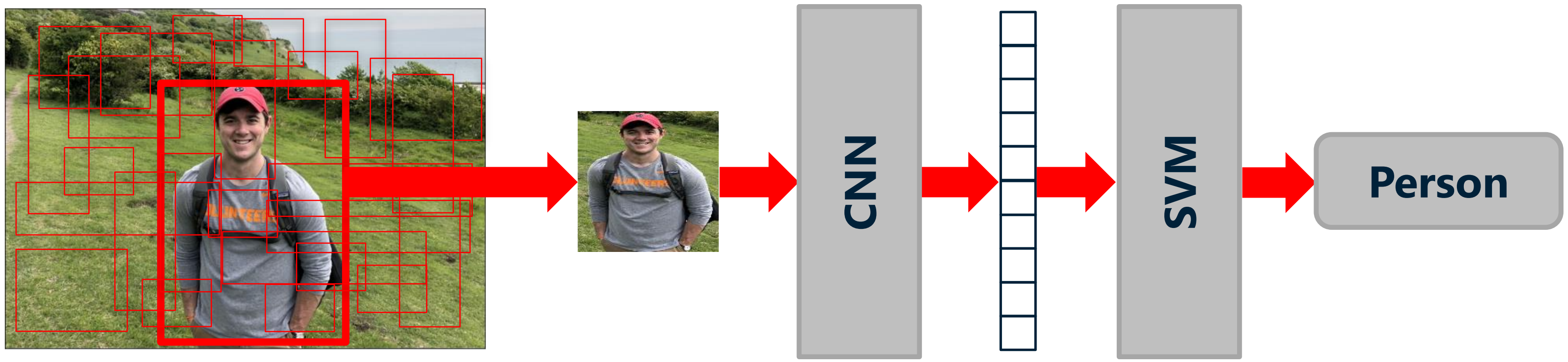
Intersection over Union (IoU)







R-CNN (2014)



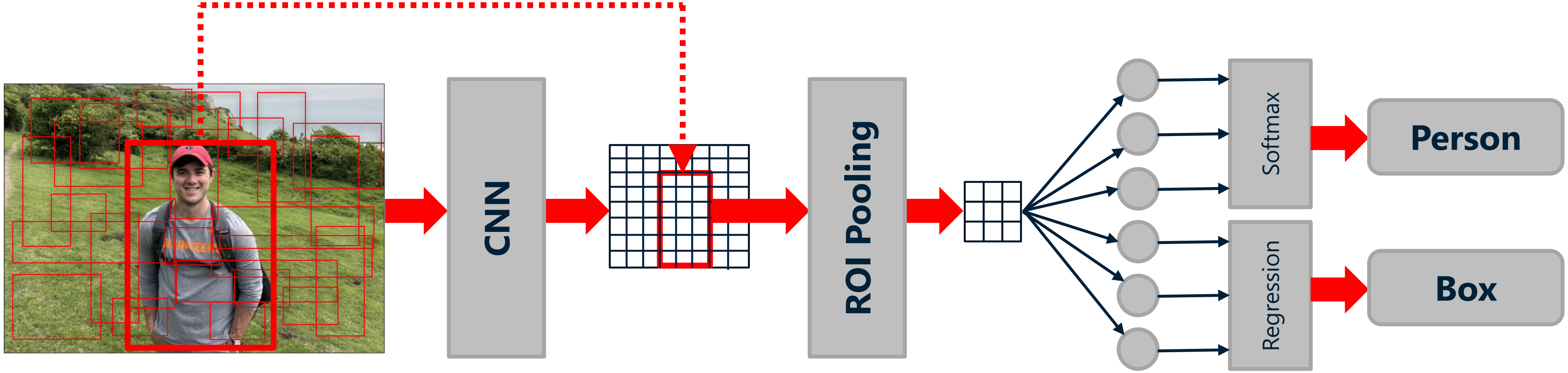
Regions of interest are identified using **selective search** or a similar algorithm.

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 (2015)

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



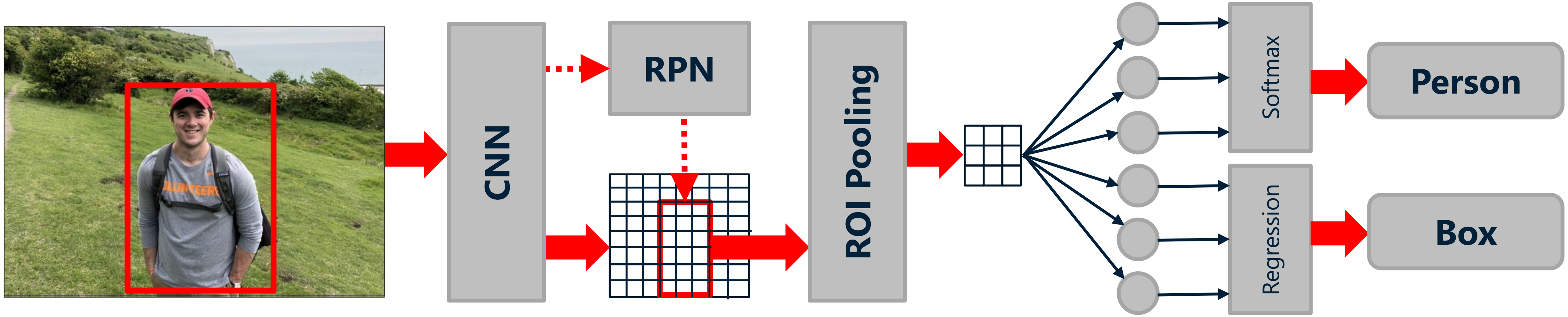
Regions of interest are identified using **selective search** or a similar algorithm. The **entire image** is passed to a CNN for feature extraction.

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 (2016)

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 (2017)

- 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