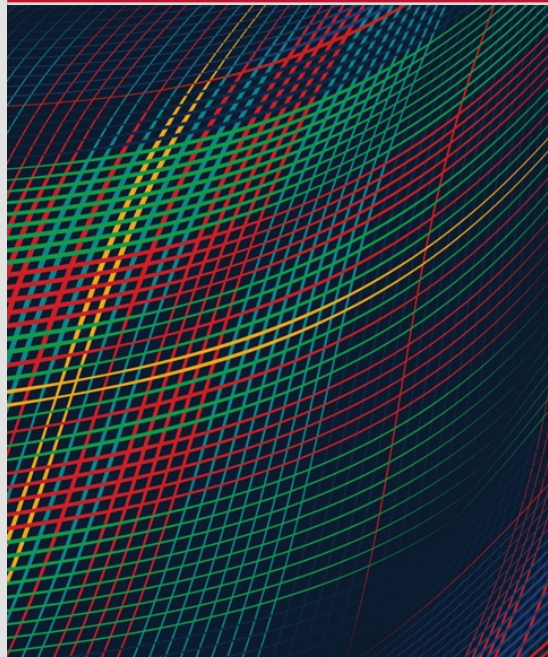
Hands-On Computer Vison

JUNE 2024

AI Division

Carnegie Mellon University
Software Engineering Institute



Welcome to Hands-On Computer Vision!

Carnegie Mellon University Software Engineering Institute

- Introduction to the Software Engineering Institute
- What we do

Cole Frank

Carnegie Mellon University Software Engineering Institute

- Associate AI Workforce Development Engineer, CMU – Software Engineering Institute

Eric Keylor, PhD

Carnegie Mellon University Software Engineering Institute

- AI Workforce Development Engineer
- Software Engineering

Institute,

University

- Background in
  - Instructional Design
  - Educational Technology
  - Educational Video Games
  - Data Science
  - Other things

Course Agenda

- Lesson 1: Introduction and Hardware Setup
- Lesson 2: Introduction to Computer Vision
- Lesson 3: Computer Images
- Lesson 4: Neural Networks for Computer Vision
- Lesson 5: Object Detection
- Lesson 6: You Only Look Once (YOLO)
- Lesson 7: Inference on Images
- Lesson 8: Next Steps

Course Learning Objectives

- Describe computer vision applications
- Define image data representation
- Preprocess images for a pretrained computer vision model
- Describe how a neural network performs computer vision
- Explain the steps in a computer vision pipeline
- Execute a computer vision model for an edge computing environment
- Use a computer vision model for object detection
- Postprocess computer vision algorithm (YOLO) results

Lecture 1

Setting Up the Raspberry Pi

Carnegie Mellon University
Software Engineering Institute

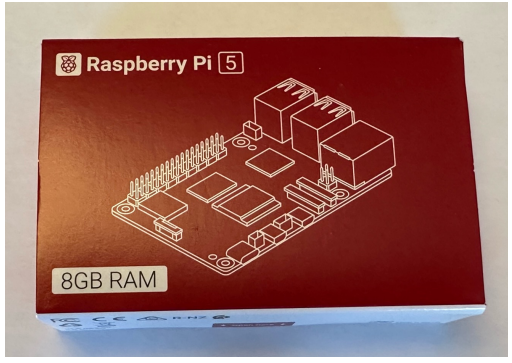
We're going to do real-time object detection!

- You can do computer vision on a microprocessor.
- Microprocessors are small but powerful.
- We'll demonstrate an important CV algorithm capable of real-time object detection.
- It's an example of sophisticated edge computing capability.
- It's a good example of what's possible!
- [INSERT IMAGE PUT THESE IN NOTES; Show what they'll do]

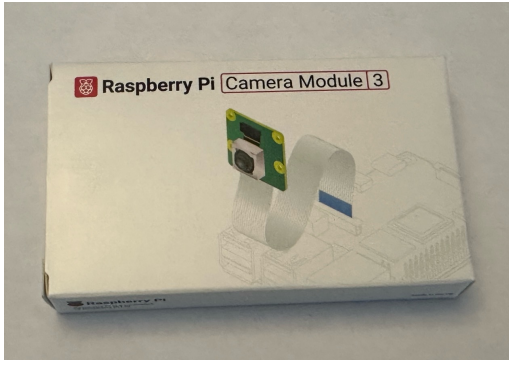
You'll need the following hardware:

Carnegie Mellon University Software Engineering Institute

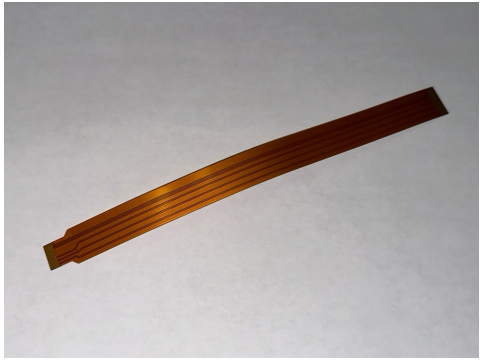
Raspberry Pi 5



Camera Module 3



Camera Cable



You will also need:

- A keyboard
- A mouse
- A monitor

We've prepared (almost) everything for you, but . . .
Some WARNINGS!

MAKE SURE YOU ARE GROUNDED WHEN HANDLING THE HARDWARE.

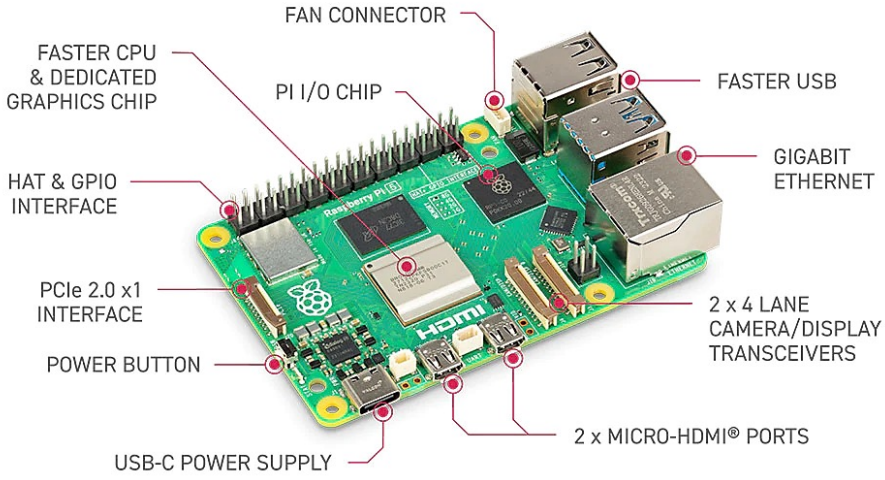
Touch something metal before handling the Raspberry Pi, camera, or camera cable.

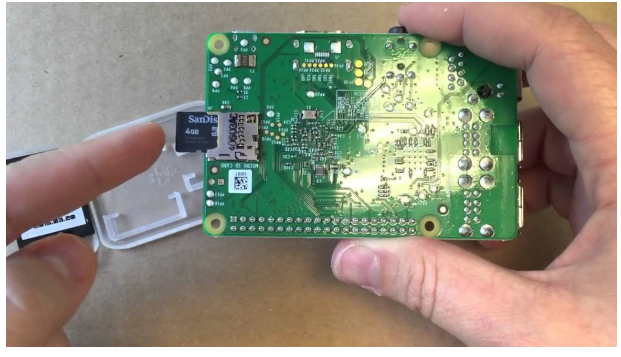
A STATIC DISCHARGE CAN DESTROY THE HARDWARE.

Don't pinch the camera cable. It can damage the wiring.

The Raspberry Pi is a popular microprocessor.

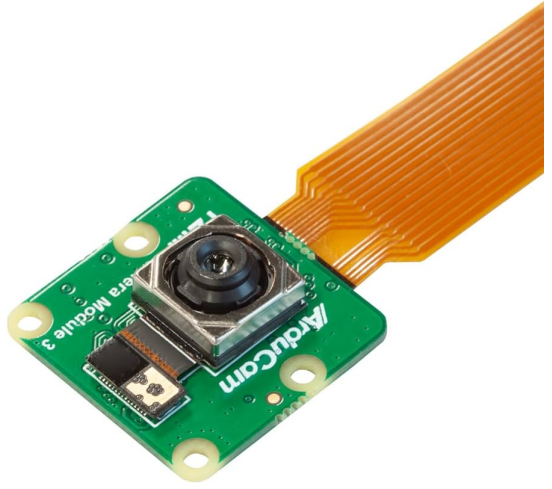
- Relatively cheap but still powerful.
- Doesn't come with an operating system. The OS is usually loaded with a microSD card.
- Typically use Debian, a flavor of Linux, for the operating system
- Designed to be extended with sensors and peripheral modules. (All the pins and interfaces.)



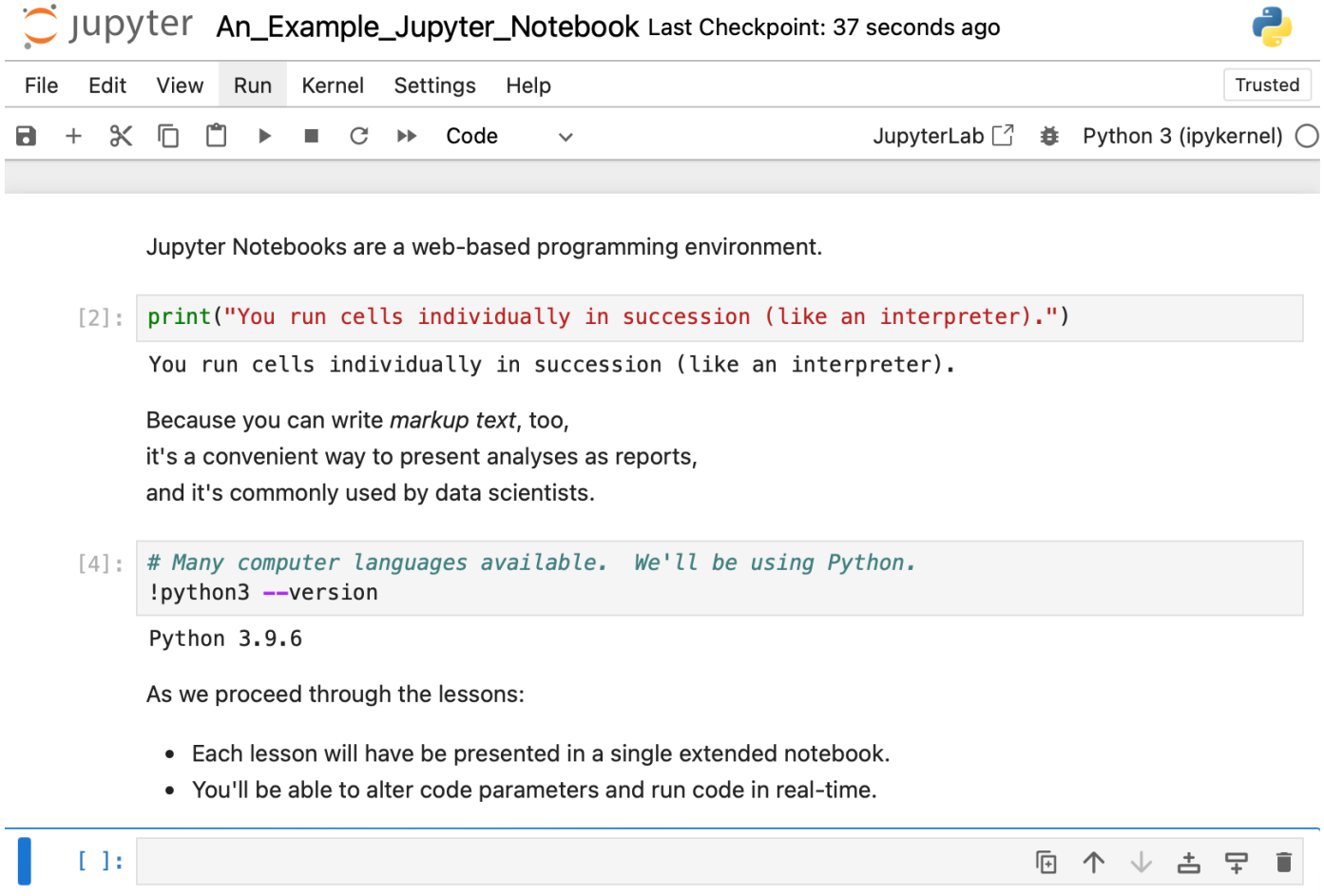


We're using the Raspberry Pi Camera Module 3.

- 2MP IMX708 Quad Bayer Sensor
- High Dynamic Range
- 75 Degree View (Wide Version has 120 Degrees)
- HD Video at 50 frames per second (fps)
- It's a powerful little camera!



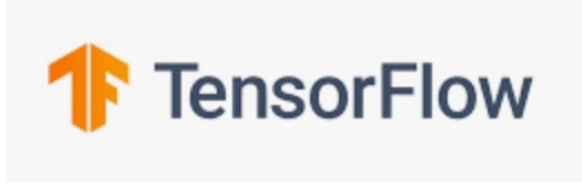
We'll use Jupyter Notebooks to run the software.



Python is the primary language for machine learning.

- Python offers access to state-of-the-art data science/AI packages.

- Commonly used frameworks for neural networks are available in Python.



- Easy to run in interactive environments.
- For data science or machine learning project, Python is a reasonable choice.

Here are the main Python packages we'll use.

NumPy provides high performance array (list) support.

Open Neural Network Exchange supports the creation of deep learning models and supports converting models between PyTorch and TensorFlow, popular neural network packages.

Open Computer Vision supports image processing.

Exercise 1

Setting Up the Raspberry Pi

Carnegie Mellon University
Software Engineering Institute

Remaining setup steps

Carnegie Mellon University Software Engineering Institute

- TODO: Add setup steps from freshly imaged SD card to Exercise-ready
  - Connect to Guest Wifi
  - Download visual studio?
  - Follow readme instructions to set up and activate virtual environment properly
  - git pull just to be safe
  - Clear outputs in

Exercise 1: Checking Access and Pi Setup

- Navigate to the Terminal program.
- ~~Check the wifi connection.~~
- Check the camera cable connection to the Pi.
- Check the camera with the libcamera-hello command.
- Navigate to the Jupyter notebooks folder.
- Open the notebook for lesson 2.
- [We'll add more detail when we have this set up.]

WiFi and Github

- Connecting to CMU-GUEST Network

Purpose	Start date	End date	Access code	Expire
rpi-cv-class	06/17/2024	06/30/2024	LTBCKQMU	expire

Clone repo onto Pi

- cfrank$ git clone https://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@github.com/sei-cfrank/HandsOnCV.git
- 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

Debrief

- **What questions or observations do you have?**
- **Does anything need clarification?**

Lesson 2

Introduction to Computer Vision

Carnegie Mellon University
Software Engineering Institute

Lesson 2: Learning Objectives

- Discuss applications of computer vision.
- Define what machine learning is.
- Describe computer vision's place within machine learning.
- Test camera settings with code.
- Adjust camera settings to improve image capture.

Lecture 2

Computer Vision Applications

Carnegie Mellon University
Software Engineering Institute

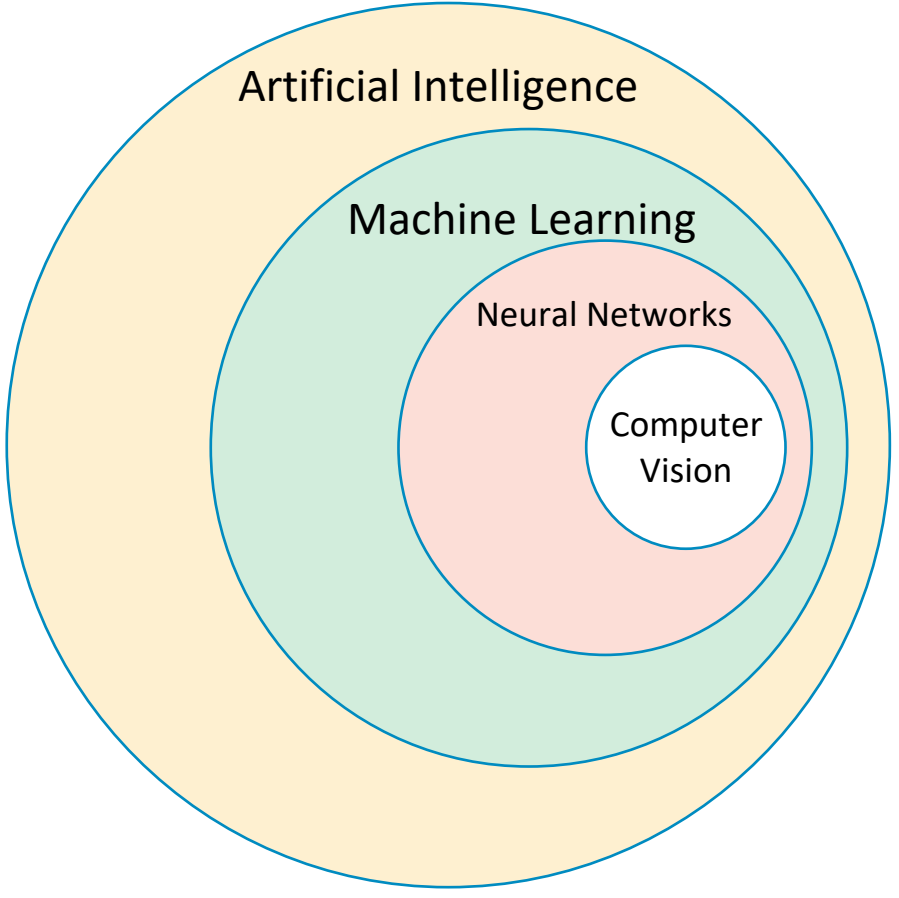
What do you think computer vision is used for? Why might you want to use it?

- List your ideas for applications and uses of computer vision.
- How many uses can you think of?
- How would you use computer vision in your work or work group?

What is “computer vision”?

- ***Enabling computers to understand visual content (images, videos, etc.)***
- Early computer vision systems used handcrafted features and *rule-based systems* to classify objects in images
- Since the 1990s, computer vision has been dominated by *machine learning*.
- Since the 2010s, computer vision has been dominated by *neural networks*.

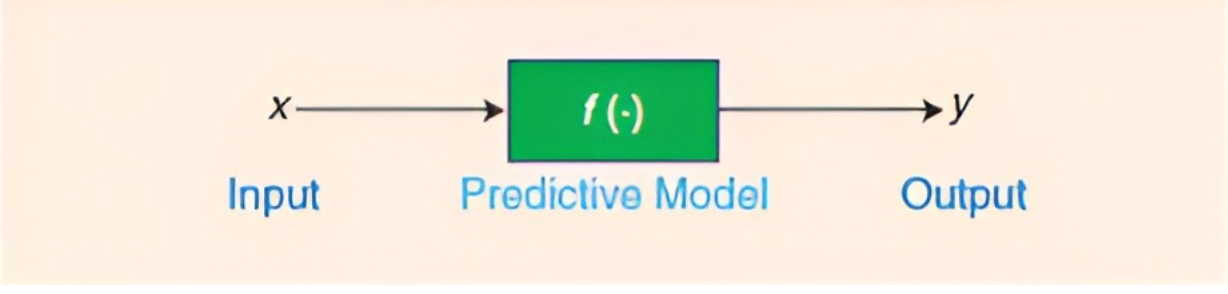
Computer vision is now a part of machine learning.



Topics are not at scale!

Machine learning performs actions (prediction and classification) **without** explicit programming.

- Machine learning models are programs that take some input **x** and produce some result, prediction, or action **y = f(x)**
- The rules for transforming input to output are not directly programmed.
- The rules are learned from analyzing lots of data.



For our purposes, machine learning can be thought of as implementing (learning) complex functions.

$$y = f(x) = 2x^2 - 4x + 5$$

- At their core, ML models are functions!*
- They map numerical **inputs** to numerical **outputs** based on some **transformation** specified by their **parameters**
- Parameters are learned during training
- These are very complicated functions
  - Inputs and outputs are high-dimensional vectors (lists) of numbers rather than scalar values
  - Many more parameters (millions, billions, even trillions)
  - Complicated transformations

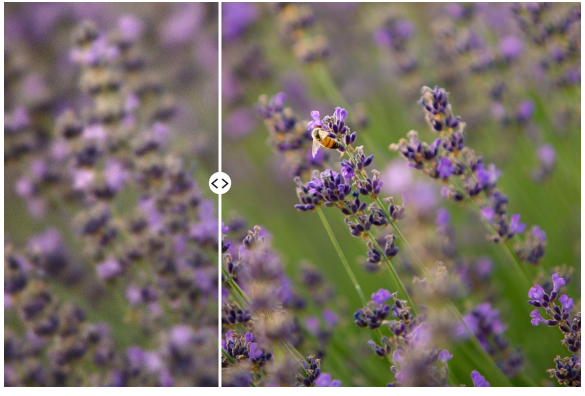
* Functions map each of their inputs to a unique outputs, whereas some ML models might map the same input to different outputs due to stochasticity in the model

The machine learning function metaphor describes computer vision (CV).

- The **inputs** to CV models are numerical representations of images
- The **output** of a CV model depends on the application (for example, object detection, object classification, identification, object tracking, etc.)

- **Example:** a model designed to detect whether an input image depicts a car might output a single number: the estimated probability that the image contains a car

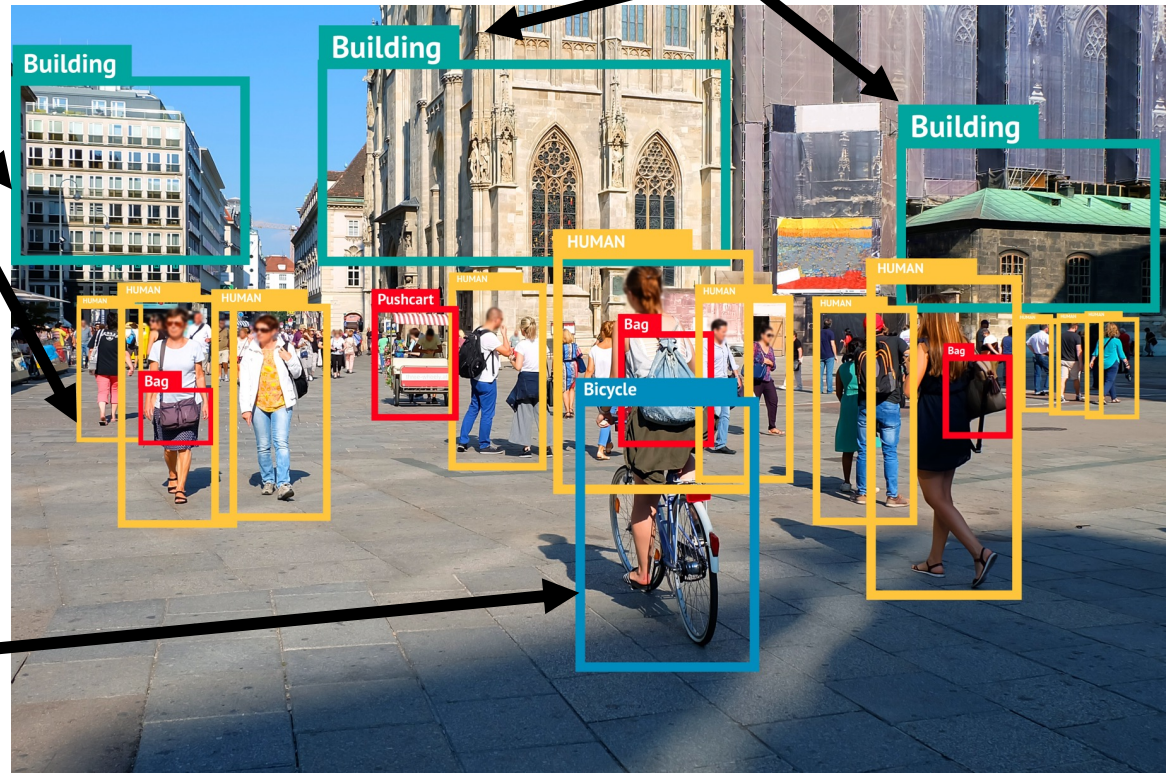
Computer vision has many useful applications.



Computer vision falls into three broad categories.

1. Object Segmentation

3. Object Classification/Identification



2. Object Dection

Exercise 2

Image Capture Using the Camera Module

Carnegie Mellon University
Software Engineering Institute

Exercise 2: Image Capture

- Review the code imports.
- Create and configure the camera.
- Start a viewing window.
- Resize and view real-time image data.

Exercise 2: Image Capture

Carnegie Mellon University Software Engineering Institute

Preliminary Imports and Connect to Camera

```
# Important that this code block is only run once!
# Otherwise will need to restart kernel
from picamera2 import Picamera2
import timeit [DO WE USE TIMEIT FOR ANYTHING?]
import cv2

# instantiate camera instance
picam2 = Picamera2()
```

Exercise 2: Image Capture

Set Camera Settings (Configuration) and Start Camera

```
# create a config with desired attributes: format, size, framerate
# NOTE: camera resolution 4608x2464, downsamples at 2304x1296 (56.03 fps)
# NOTE: XRGB8888 => shape: (height, width, 4); pixel value: [B, G, R, A]
config = picam2.create_preview_configuration(
    main={'format': 'XRGB8888', 'size': (2304, 1296)})  # 16:9 aspect ratio

# set camera configuration, start camera
picam2.configure(config)
picam2.start()
```

Exercise 2: Image Capture

Create the Viewing Window

```
# start opencv window thread
cv2.startWindowThread()
wnd_name = 'foo'
cv2.namedWindow(wnd_name, cv2.WINDOW_KEEPRATIO)
cv2.resizeWindow(wnd_name, 640, 480)                    # 4:3 aspect ratio
```

Exercise 2: Image Capture

Capture and Resize Real-Time Images

```
while True:
    # get current image data from 'main' camera stream
    arr1 = picam2.capture_array('main')

    # resize the image data using bi-linear interpolation
    arr2 = cv2.resize(arr1, (640, 480), 0, 0, cv2.INTER_LINEAR)

    # if window closed, break loop before imshow creates new window
    # if cv2.getWindowProperty(wnd_name, cv2.WND_PROP_AUTOSIZE) == -1:
    #     break

    # show resized image
    cv2.imshow(wnd_name, arr2)
    key = cv2.waitKey(1)

    if key == ord("q"):
        break
```

Exercise 2: Image Capture

Stop Camera and Close Viewing Window

```
# stop camera
cv2.destroyWindow(wnd_name)
picam2.stop()
```

Bonus Tasks?

- Change the following camera parameters <need specific instruction, outlining here>
- Resolution: <set low enough to see pixelation, what other useful values?>
- Framerate: <show frame rate values, adjust to different values>
- Exposure time <if possible, demonstrate relationship among resolution, framerate, exposure time>
- Autoexposure

Debrief

- What questions or observations do you have?
- Does anything need clarification?
- Anything else you'd like to do in this notebook?

Lesson 3

Computer Images

Carnegie Mellon University
Software Engineering Institute

Lesson 3: Learning Objectives

- Explain RGB color space.
- Define pixels and their integer representations.
- Describe image preprocessing steps.
- Preprocess images using provided code.

Lecture 3

The Anatomy of an Image

Carnegie Mellon University
Software Engineering Institute

Think About It: How do computers represent image data?

- How do you think computers store image data?
- What data needs to be stored?
- What type of values are needed?

Computer vision is not like human vision.

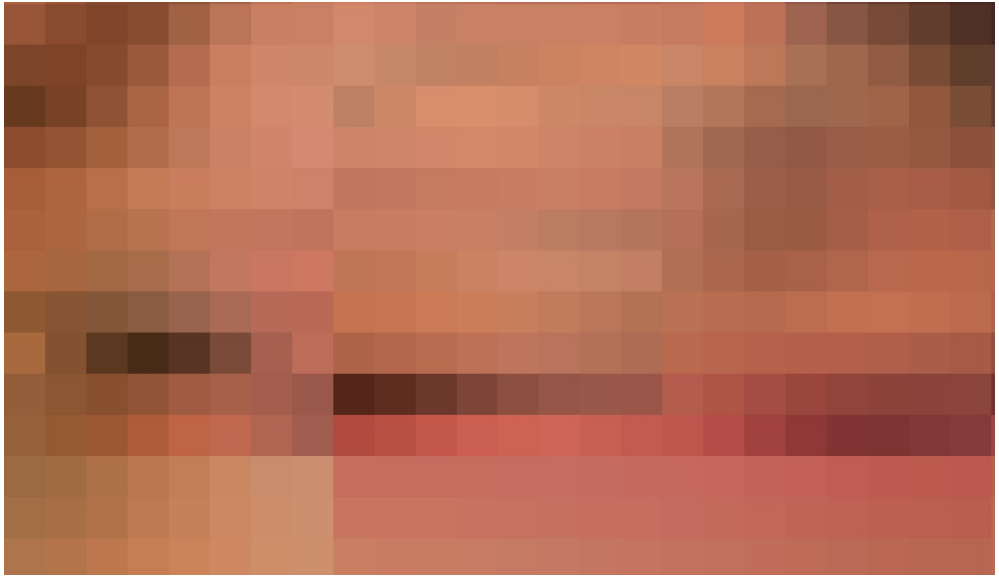


Image data is made of arrays (lists) of pixels.

```
[[[ 50  58  57]
  [ 51  59  58]
  [ 53  61  60]
  ...
  [ 47  89 142]
  [ 41  50  88]
  [ 47  71  63]]

 [[ 51  59  58]
  [ 51  59  58]
  [ 52  60  59]
  ...
  [ 37  74 124]
  [ 41  50  84]
  [ 46  70  58]]

 [[ 51  59  58]
  [ 51  59  58]
  [ 52  60  59]
  ...
  [ 25  54  98]
  [ 48  54  77]
  [ 43  62  45]]

 ...

 [[179 168 160]
  [179 168 160]
  [182 171 163]
  ...
  [ 64  64  80]
  [ 36  39  53]
  [ 48  53  62]]
```

Pixel (Each Row of 3 Numbers)

Columns (Image Width)

Row (Image Height)

Pixels are represented as channels of numbers. The channels represent a color space.

Red-Blue-Green (RGB)
is a common color space.
It reflects human color vision.

The example is BGR
due to a software default
in the notebook example.

There are other color spaces
used for video, printing, etc.

```
[[[ 50  58  57]
  [ 51  59  58]
  [ 53  61  60]
  ...
  [ 47  89 142]
  [ 41  50  88]
  [ 47  71  63]]

 [[ 51  59  58]
  [ 51  59  58]
  [ 52  60  59]
  ...
  [ 37  74 124]
  [ 41  50  84]
  [ 46  70  58]]

 [[ 51  59  58]
  [ 51  59  58]
  [ 52  60  59]
  ...
  [ 25  54  98]
  [ 48  54  77]
  [ 43  62  45]]

 ...

 [[179 168 160]
  [179 168 160]
  [182 171 163]
  ...
  [ 64  64  80]
  [ 36  39  53]
  [ 48  53  62]]
```

All colors can be created from red, green, and blue (RGB).

Carnegie Mellon University Software Engineering Institute

[0 0 0] is 

[255 255. 255] is 

[255 0 0] is 

[0 255 0] is 

[0 0 255] is 

[50 58 57] is 

[47 71 63] is 

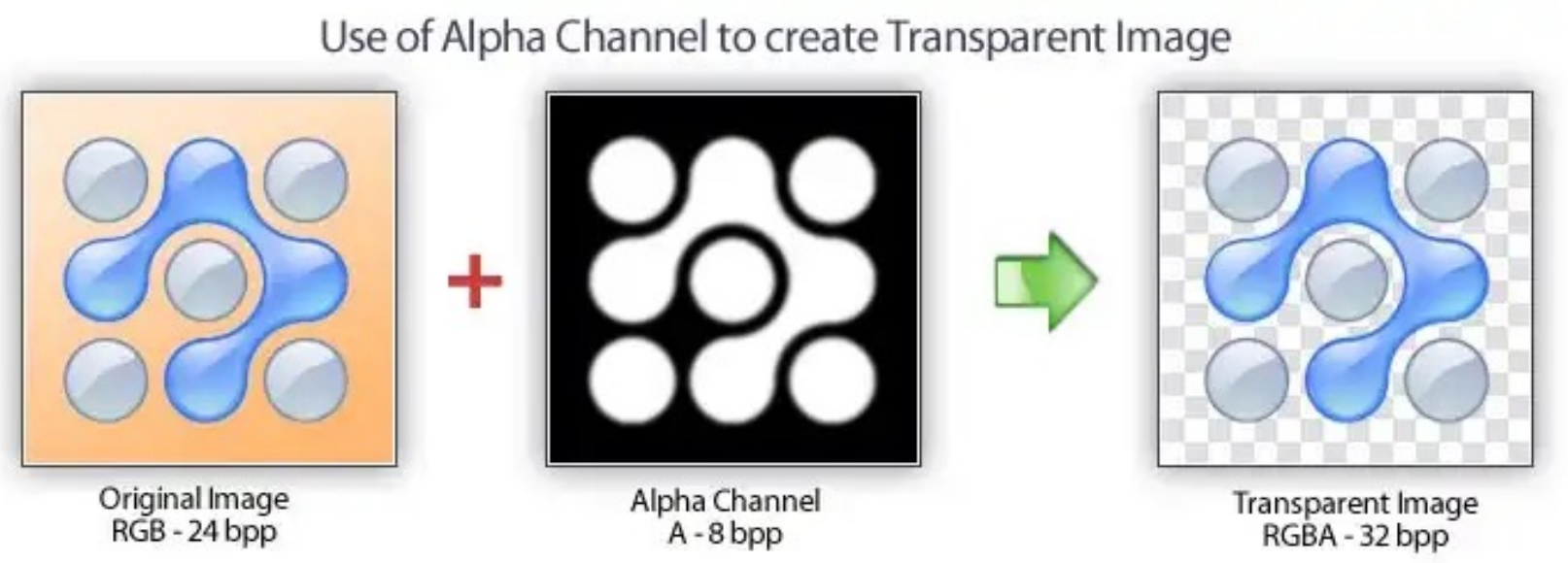
[37 74 124] is 

[179 168 160] is 

[127 127 127] is 

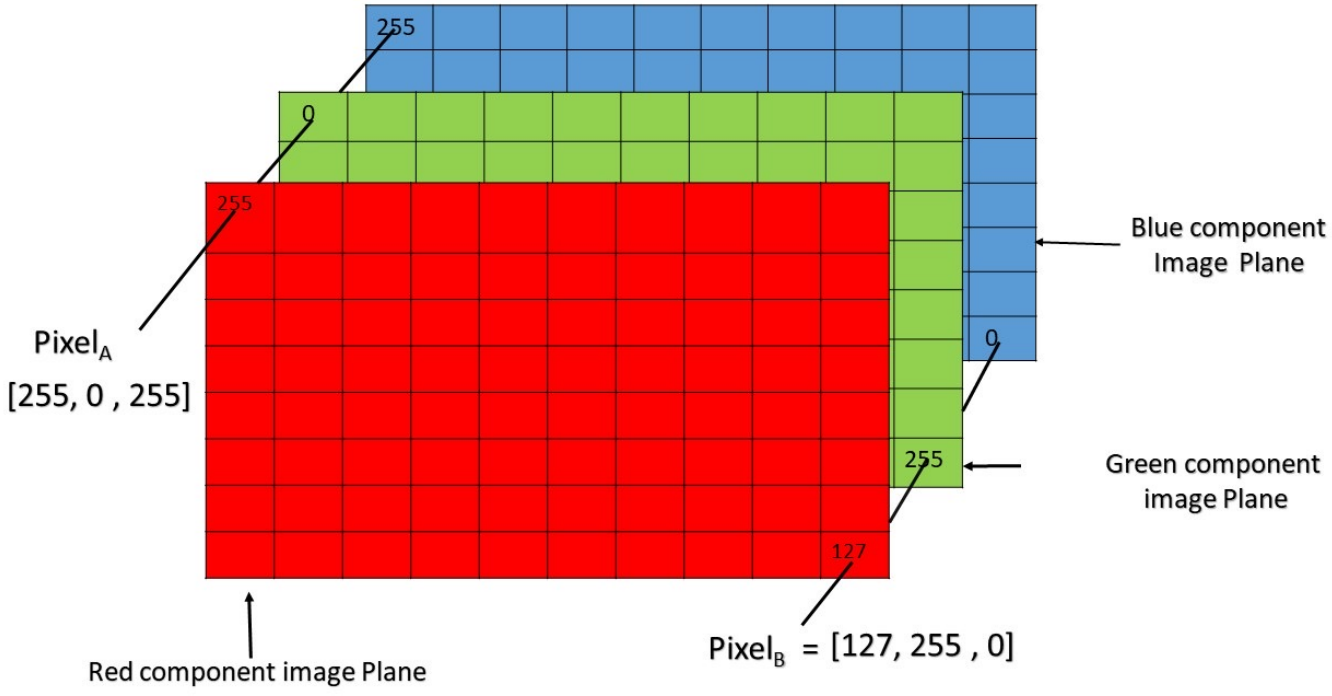
Transparency is referred to as alpha.

- Alpha values have their own channel like colors.
- For our purposes alpha is ignored.



Images are arrays!

- Three dimensional array:
  - Height
  - Width
  - Color channel
- (H x W x C)



Pixel of an RGB image are formed from the corresponding pixel of the three component images

Images usually must be transformed for model imput.

- Images may not be captured in the format necessary for input into the computer vision model
- If this happens, it's necessary to transform the image:
  - Scaling: changing the size of the image
  - Cropping: cutting out part of the image
  - Letter boxing: reduce image but keep aspect ratio
  - Flipping: flipping the image
- Transformations are achieved by altering the numerical representations of the images
- Pixel values may be normalized, which means divided by a value to fit in [0, 1].

Exercise 3

Transforming Images

Carnegie Mellon University
Software Engineering Institute

Exercise 3: Transforming Images for Reference

- Review setup.
- Define letterbox creation.
- Transform images for the model.
- Show annotated image.

Exercise 3: Transforming Images for Reference

Preliminary Imports and Connect to Camera

```
# Important that this code block is only run once!
# Otherwise will need to restart kernel
from picamera2 import Picamera2
import cv2

# instantiate camera instance
picam2 = Picamera2()

import numpy as np
```

Exercise 3: Transforming Images for Reference

Letterbox Function:

```
# letterbox procedure
def letterbox(src, dest_shape):
    # get src dims
    src_width = src.shape[1]    # img.shape returns tuple (rows, cols, chan)
    src_height = src.shape[0]   # NOTE: rows => height; cols => width

    # cons dest array (filled with gray), get dest dims
    # NOTE: each 32-bit [B, G, R, A] pixel value is [128, 128, 128, 255]
    dest = np.full(dest_shape, np.uint8(128))
    dest[:, :, 3] = np.uint8(255)
    dest_width = dest.shape[1]
    dest_height = dest.shape[0]
```

Exercise 3: Transforming Images for Reference

Letterbox Function:

```
# calculate width and height ratios
  width_ratio = dest_width / src_width        # NOTE: ratios are float values
  height_ratio = dest_height / src_height

  # init resized image width and height with max values (dest dims)
  rsz_width = dest_width
  rsz_height = dest_height

  # smallest scale factor will scale other dimension as well
  if width_ratio < height_ratio:
      rsz_height = int(src_height * width_ratio)  # NOTE: integer truncation
  else:
      rsz_width = int(src_width * height_ratio)
```

Exercise 3: Transforming Images for Reference

Preliminary Imports and Connect to Camera

```
    # resize the image data using bi-linear interpolation
      rsz_dims = (rsz_width, rsz_height)
      rsz = cv2.resize(src, rsz_dims, 0, 0, cv2.INTER_LINEAR)

      # embed rsz into the center of dest
      dx = int((dest_width - rsz_width) / 2)          # NOTE: integer truncation
      dy = int((dest_height - rsz_height) / 2)
      dest[dy:dy+rsz_height, dx:dx+rsz_width, :] = rsz

      # letterboxing complete, return dest
      return dest
```

Exercise 3: Transforming Images for Reference

Carnegie Mellon University Software Engineering Institute

Preliminary Imports and Connect to Camera

```
# pack_buffer procedure, ONNX model expects normalized float32 NCHW tensor
def pack_buffer(src):
    dest = np.array(src, dtype='float32')      # cons dest array via copy
    dest = dest[:, :, :3]                      # remove alpha channel
    dest = dest[..., ::-1]                     # reorder channels: BGR -> RGB
    dest /= 255.0                              # normalize vals
    dest = np.transpose(dest, [2, 0, 1])       # make channel first dim
    dest = np.expand_dims(dest, 0)             # ins batch dim before chan dim
    return dest
```

Exercise 3: Transforming Images for Reference

Preliminary Imports and Connect to Camera

```
# create a config with desired attributes: format, size, framerate
# NOTE: camera resolution 4608x2464, downsamples at 2304x1296 (56.03 fps)
# NOTE: XRGB8888 => shape: (height, width, 4); pixel value: [B, G, R, A]
config = picam2.create_preview_configuration(
    main={'format': 'XRGB8888', 'size': (2304, 1296)})  # 16:9 aspect ratio

# set camera configuration, start camera
picam2.configure(config)
picam2.start()

# start opencv window thread
cv2.startWindowThread()
wnd_name = 'foo'
cv2.namedWindow(wnd_name, cv2.WINDOW_KEEPRATIO)
cv2.resizeWindow(wnd_name, 416, 416)                    # 1:1 aspect ratio
```

Exercise 3: Transforming Images for Reference

Preliminary Imports and Connect to Camera

```
while True:
    # get current image data from 'main' camera stream
    arr1 = picam2.capture_array('main')

    # letterbox the image to resize for NN input (size: (height, width, chan))
    arr2 = letterbox(arr1, (416, 416, 4))

    # cons packed input buffer for ONNX model inference
    arr3 = pack_buffer(arr2)
    dim3 = np.array([arr2.shape[1],arr2.shape[0]],dtype=np.float32).reshape(1,2)

    # if window closed, break loop before imshow creates new window
    if cv2.getWindowProperty(wnd_name, cv2.WND_PROP_AUTOSIZE) == -1:
        break
```

Exercise 3: Transforming Images for Reference

Preliminary Imports and Connect to Camera

```
# stop camera

cv2.destroyWindow(wnd_name)

picam2.stop()
```

Debrief

- What questions or observations do you have?
- Does anything need clarification?
- Anything else you'd like to do in this notebook?

Lesson 4

Implementing Computer Vision

Carnegie Mellon University
Software Engineering Institute

Lesson 4: Learning Objectives

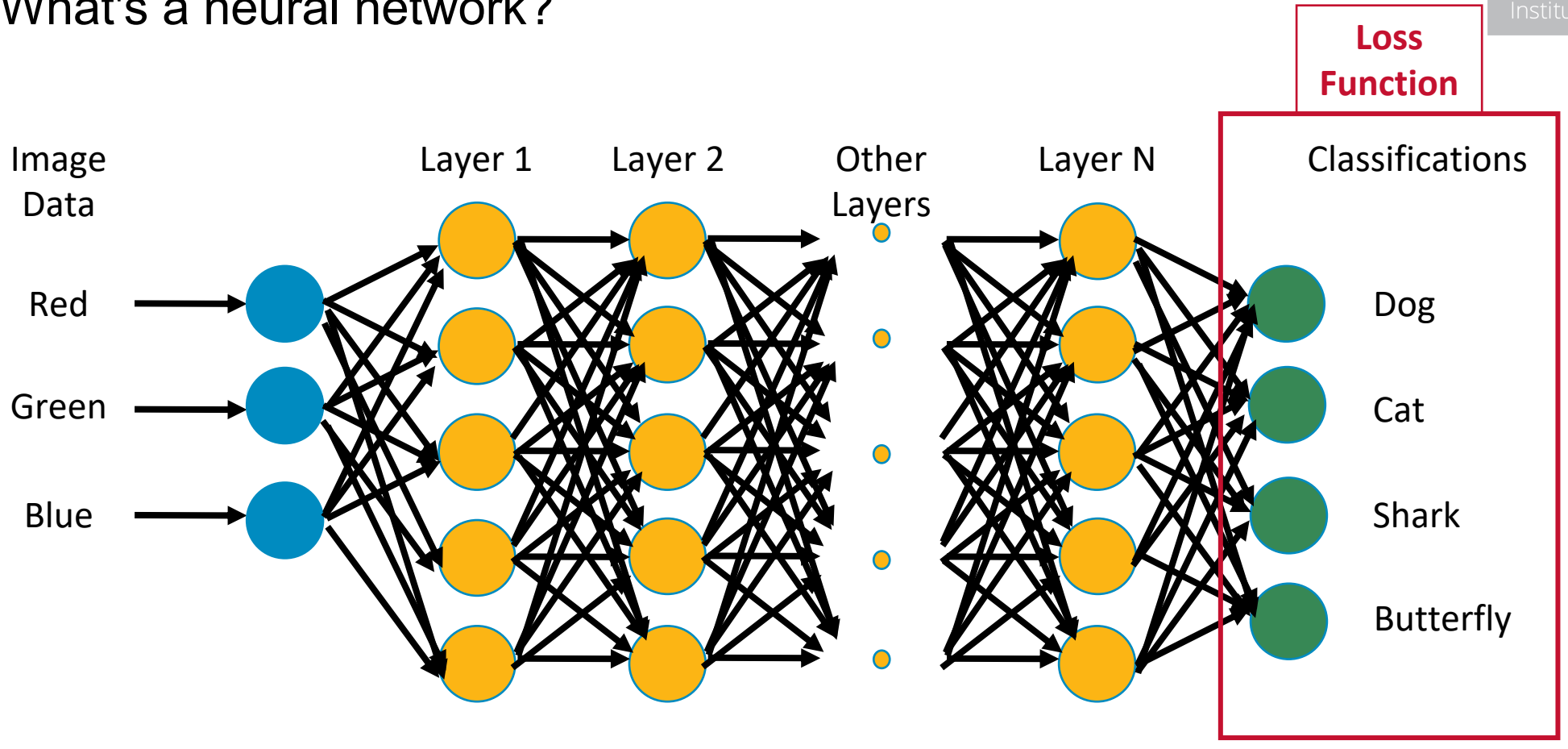
- Describe neural network structure.
- Describe what the layers of a neural network do.
- Explain the benefits of using pretrained models.
- Explain the machine learning pipeline for computer vision application development.

Lecture 4

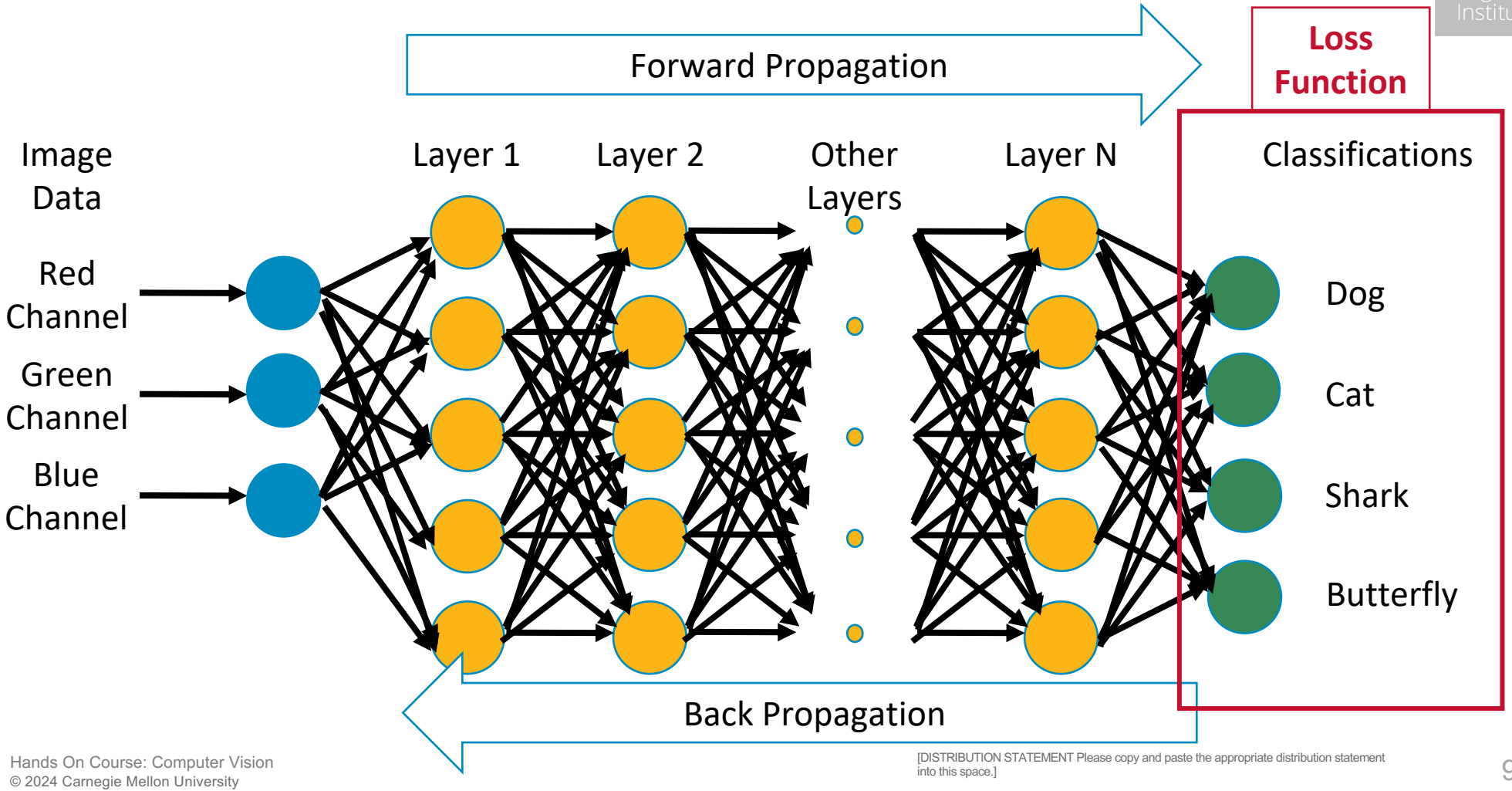
Neural Networks for Computer Vision

Carnegie Mellon University
Software Engineering Institute

Neural networks are used to implement current computer vision. What's a neural network?



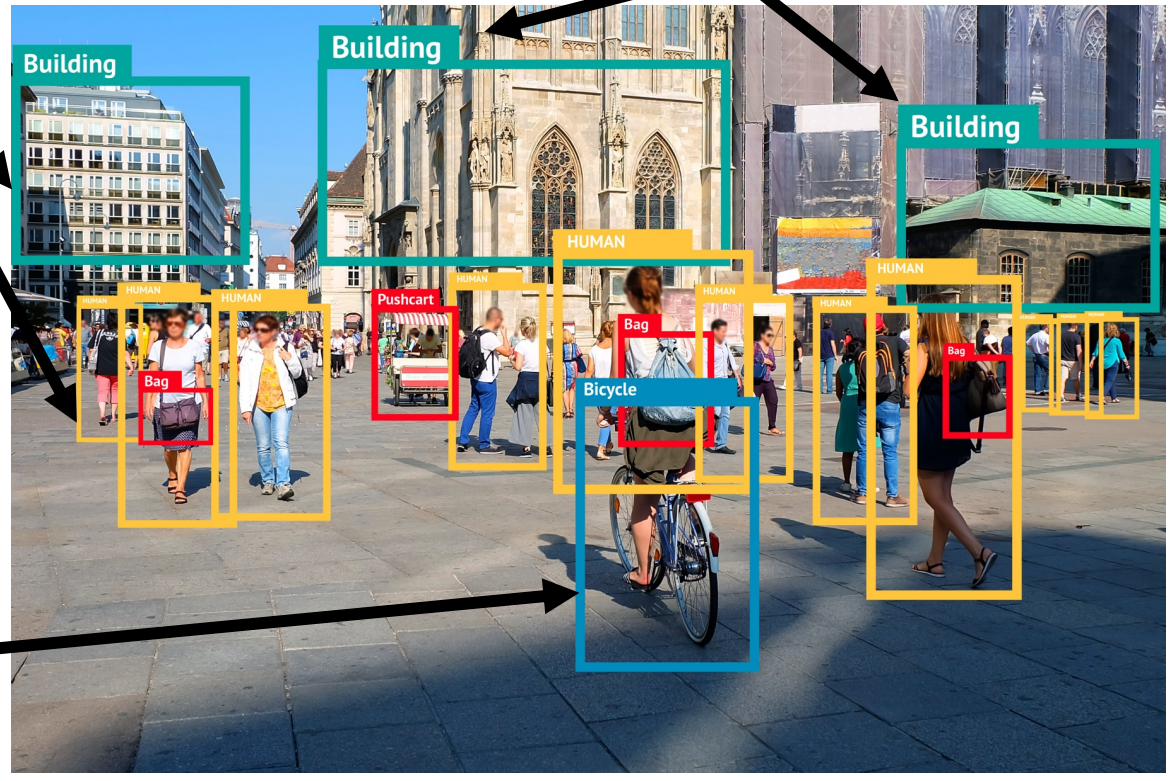
Neural networks are trained using forward and back propagation.



Neural networks can do different computer vision tasks.

1. Object Segmentation

3. Object Classification

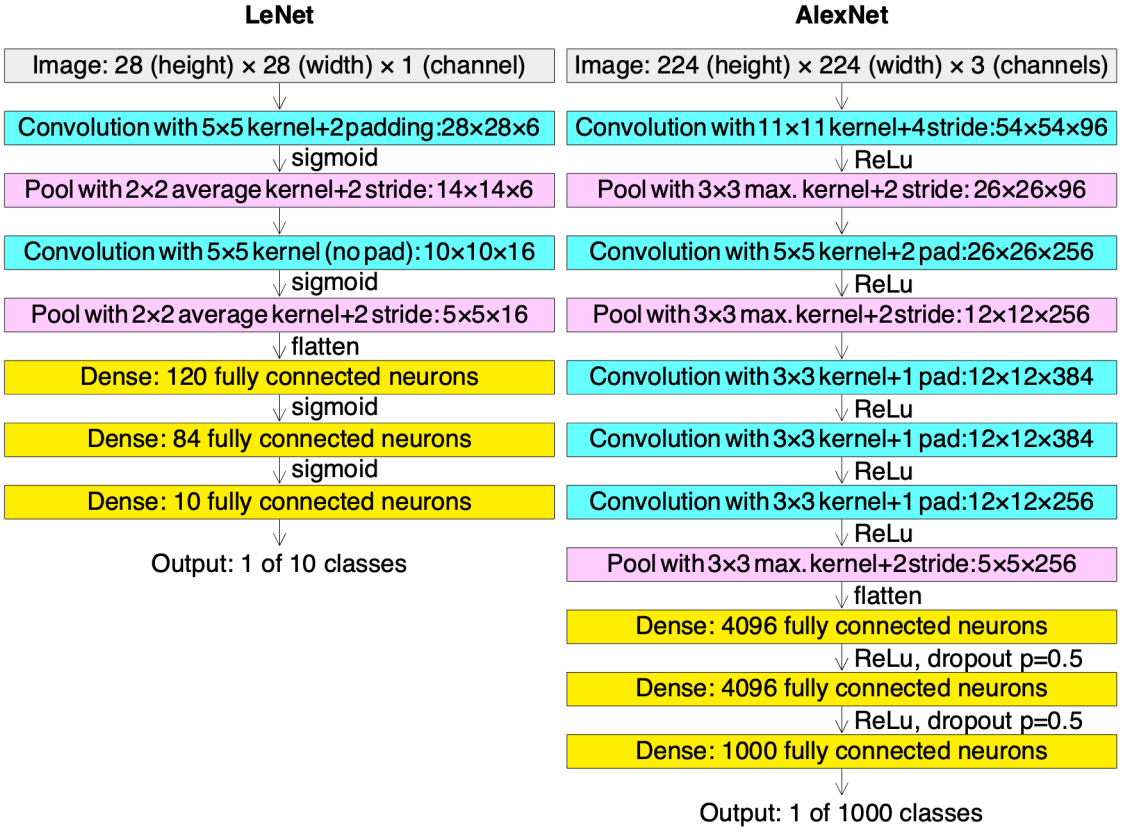


2. Object Detection

Deep learning revolutionized computer vision. Convolutional neural networks are especially important.

LeNet

- Created by LeCun in 1998
- Early convolutional network



By Cmglee - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=104937230

AlexNet

- The start of modern CV
- Classification model
- Won ImageNet Large Scale Recognition Challenge in 2012
- Top-5 error 15.3%
- 10.8% better than next model

The layers have different functions.

- Convolutional Layer: Identifies image characteristics.
- Pooling Layer: Reduces the data while keeping the information.
- Dense Layer: Does the feature (image characteristics) extraction and classification.

- Kernels perform calculations on layer (image) data.
- Different types of functions affect calculations between layers.

Let's take a closer look at convolution and kernels

Convolutional layers identify image features like edges. A kernel slides across the image to make the feature data.

Carnegie Mellon University Software Engineering Institute

Image Data

0	**1**	**2**
3	**4**	**5**
6	**7**	**8**

Kernel

0	1
2	3

Output

19	**25**
37	**43**

In this (very simplified) example, a 2 x 2 kernel matrix slides across a 3 x 3 image data matrix.
The kernel does a component-by-component multiplication and adds the values together.

For example: 0 * 0 + 1 * 1 + 3 * 2 + 4 * 3 = 0 + 1 + 6 + 12 = 19 (the top left output)

Here's how the next output cell is calculated. The *stride* is 1 because the kernel moves by 1 cell.

Image Data

0	**1**	**2**
3	**4**	**5**
6	**7**	**8**

Kernel

0	1
2	3

Output

19	**25**
37	**43**

For example: 1 * 0 + 2 * 1 + 4 * 2 + 5 * 3 = 0 + 2 + 8 + 15 = 25 (the top right output)

The calculations follow the same procedure for the bottom row.

Stride describes how the kernel moves over the data.
In this case the stride is 1 since it moves to the right and down by 1 cell.

Stride can be used to downsample (reduce) the data. Here's an example where stride = 2.

Image Data

0	**1**	**2**	**3**
1	**1**	**2**	**3**
2	**2**	**2**	**3**
3	**3**	**3**	**3**

Kernel

0	1
2	3

Output

6	**16**
17	**21**

When reducing the size of the data, it's important to keep the feature information.

Using larger kernel sizes reduces the size (dimensionality) of the next layer.
This is usually more computationally efficient, too.

Padding can be used to increase or keep the output the same size as the input data.

Image Data

0	0	0	0	0
0	**0**	**1**	**2**	0
0	**3**	**4**	**5**	0
0	**6**	**7**	**8**	0
0	0	0	0	0

Kernel

0	1
2	3

Output

0	**3**	**8**	**4**
9	**19**	**25**	**10**
21	**37**	**43**	**16**
6	**7**	**8**	**0**

For these shapes, padding the image data by 1 **increases** the output size assuming stride = 1.

A 3 x 3 kernel would have produced an output the same size as the input.

Kernels are important because kernel values capture specific features.

Identity Kernel

0	0	0
0	1	0
0	0	0

Edge Detection

0	-1	0
-1	4	-1
0	-1	0

Image Sharpening

0	-1	0
-1	5	-1
0	-1	0

Pooling (usually max pooling) reduces the data and preserves the features from the convolutional layers.

Carnegie Mellon University Software Engineering Institute

Image Data

0	**1**	**2**
3	**4**	**5**
6	**7**	**8**

Max Pooling
2 x 2 Matrix

Output

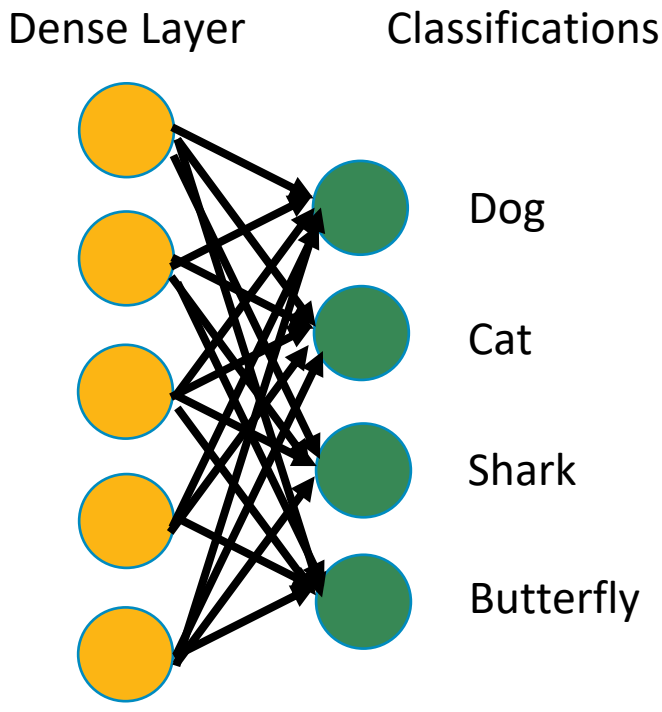
4	**5**
7	**8**

Max pooling takes the maximum value in the pooling window.
There are other pooling techniques (like taking the average).

Strides and padding can be adjusted like a kernel for a convolutional layer.

Dense (fully connected) layers have all input and output nodes are connected

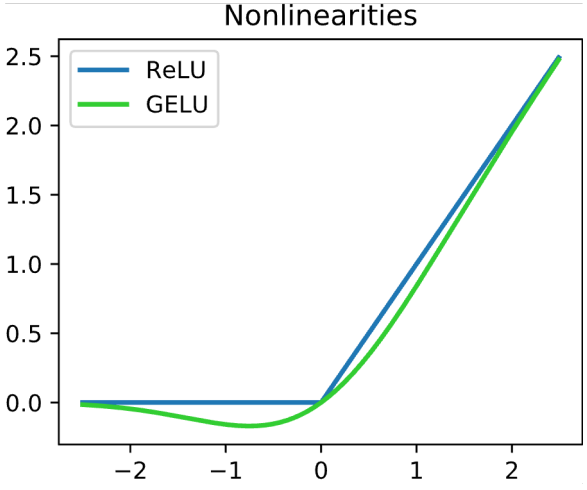
- Dense layers are another way of reducing the data.
- Eventually, the layers must be reduced to the correct number of possible answers to make a prediction.



Why convolutions?

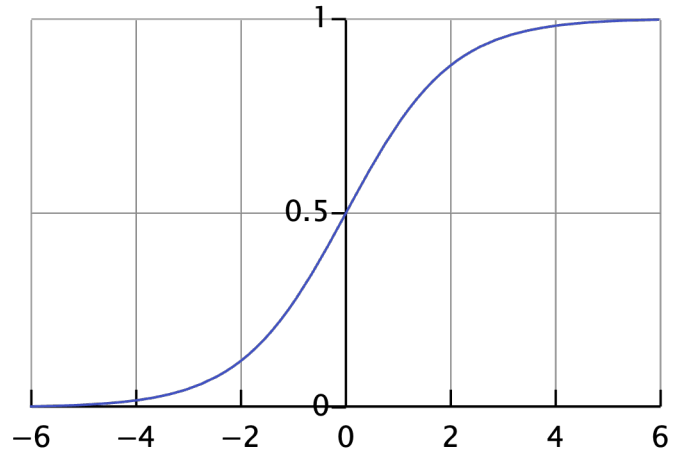
- Translation invariant
- More efficient than either fully connected linear layers or transformer architecture (though increasingly Transformers are SoTA for DV)

Node values (weights) are adjusted by activation functions.



Rectified Linear Unit (ReLU)
Keeps positive values and sets negative to 0
Better for optimization

GELU used in some models like BERT transformer
GELU can be more accurate



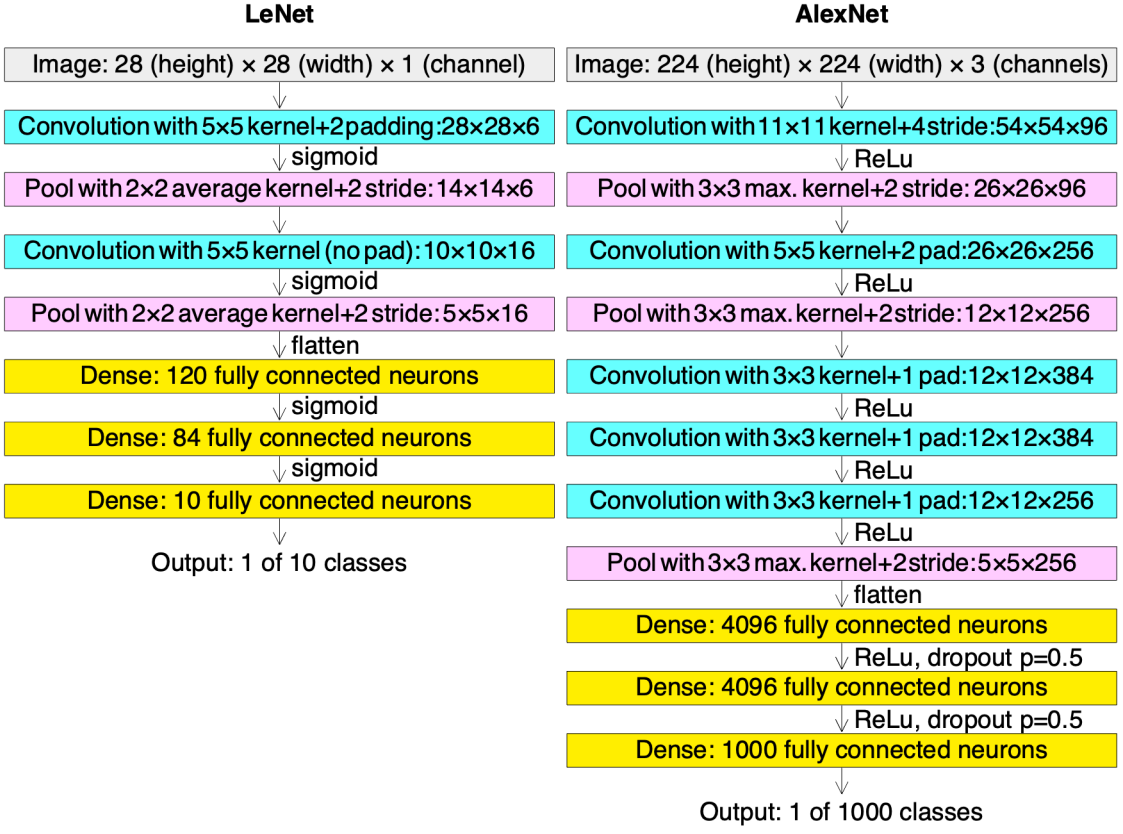
Sigmoids
Sigmoids are squashing functions that smash any value into the (0, 1) range.
Used to convert outputs to probabilities.
Especially useful for binary classification.
Still used but generally replaced by ReLU (easier to train)

By Ringdongdang - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=95947821

You have all the concepts for these architectures.
The layer size changes are due to the kernel, stride, and padding.

LeNet

- Created by LeCun in 1998
- Early convolutional network

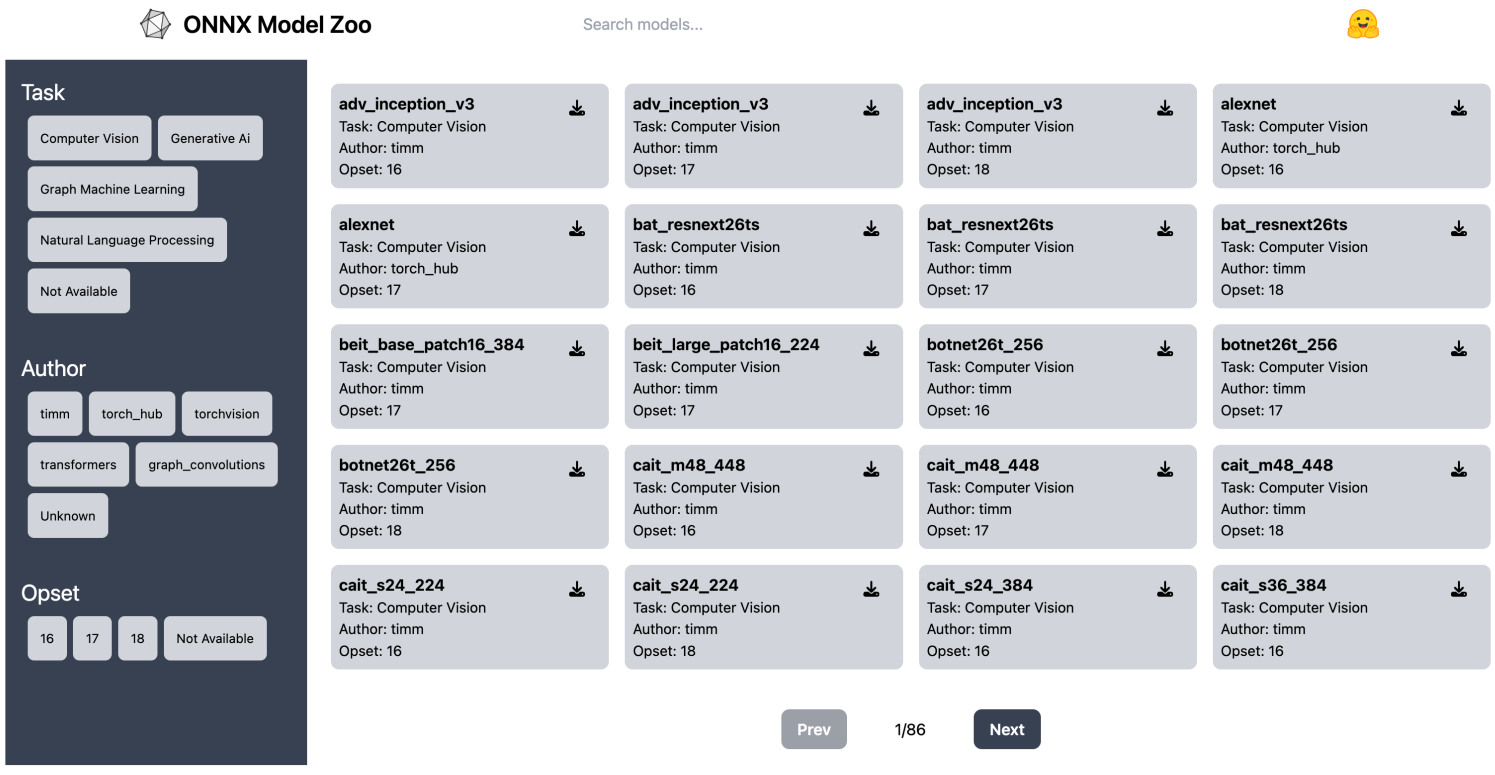


AlexNet

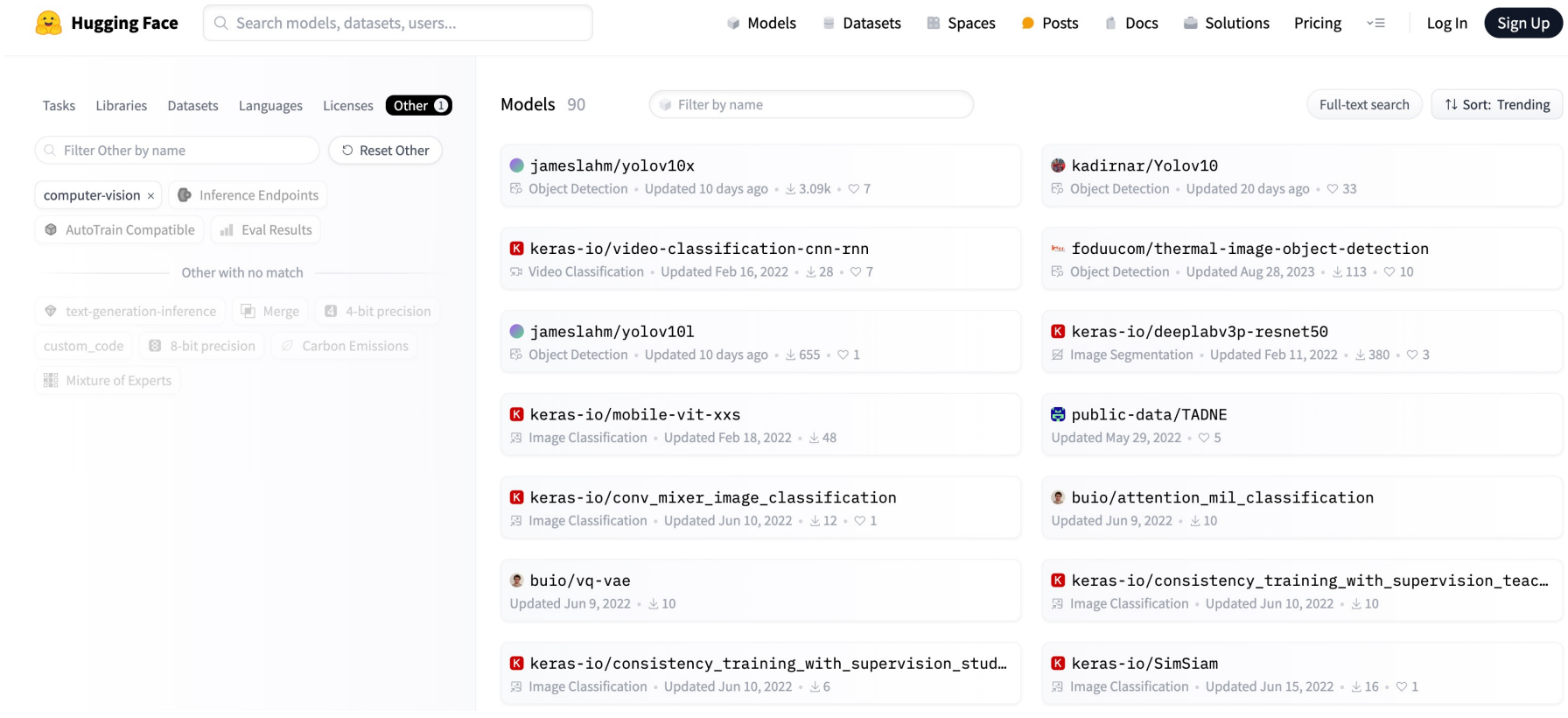
- The start of modern CV
- Classification model
- Won ImageNet Large Scale Recognition Challenge in 2012
- Top-5 error 15.3%
- 10.8% better than next model

By Cmglee - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=104937230

Training models can take a considerable amount of time. Now, there trained models to leverage.

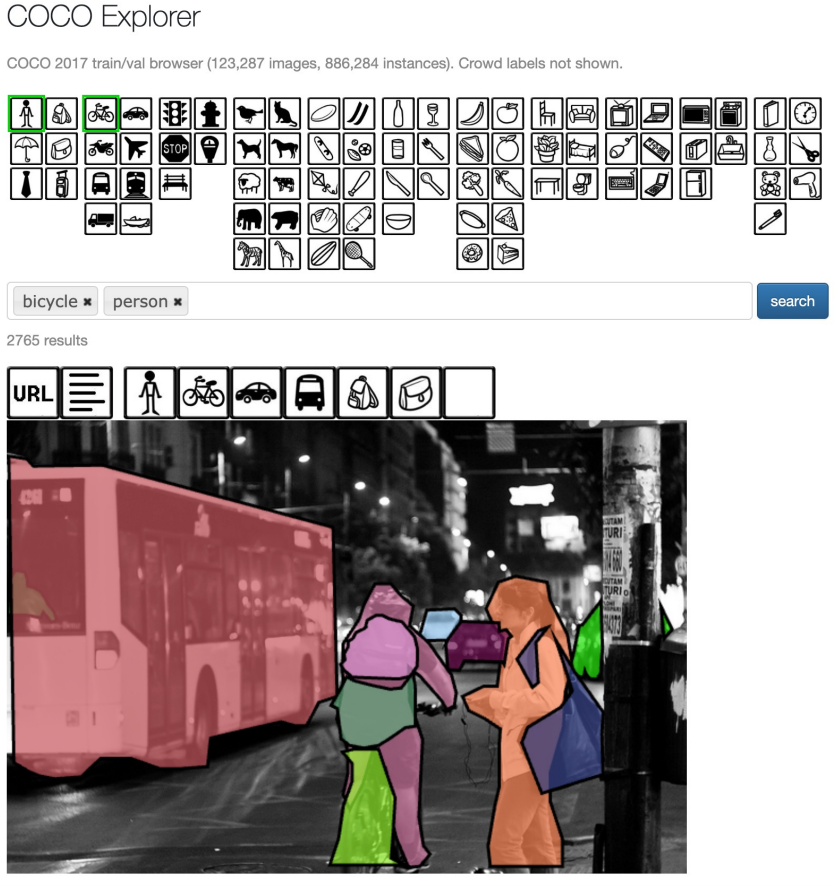


There are many places to find models.
For example, Huggingface also provides models.



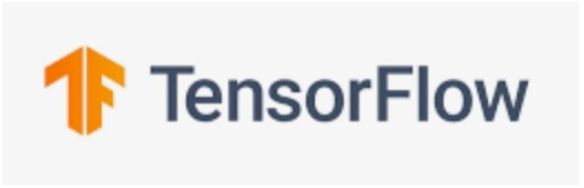
The characteristics of the dataset are important, too. Common Objects in Context will be our dataset.

- Used for segmentation and detection
- Objects are in context
- 1.5 million objects
- 80 object categories
- 5 captions per image
- https://cocodataset.org/#explore



There are multiple neural network frameworks.

- Developed by Facebook
- Used in research/academia
- Python (and C++) interface
- Autodifferentiation
- Faster prototyping



- Developed by Google
- Used in industry
- Graph-based computation
- More memory efficient

- Open Neural Network Exchange
- Common set of operators
- Converts PyTorch/TensorFlow
- Model interoperability

Exercise 4

Pretrained Neural Networks

Carnegie Mellon University
Software Engineering Institute

Links

- ONNX Model Zoo
  - https://github.com/onnx/models
- Netron Model viewer
  - https://netron.app/

Exercise 4: Pretrained Neural Networks

- Review the code imports.
- Create and configure the camera.
- Start a viewing window.
- Resize and view real-time image data.

Exercise 4: Pretrained Neural Networks

New Imports/

```
import onnx
import onnxruntime as ort
```

Exercise 4: Pretrained Neural Networks

New Imports/

```
picam2.configure(config)
picam2.start()
```

Exercise 3: Transforming Images for Reference

Letterbox Function:

```
# letterbox procedure
def letterbox(src, dest_shape):
    # get src dims
    src_width = src.shape[1]    # img.shape returns tuple (rows, cols, chan)
    src_height = src.shape[0]   # NOTE: rows => height; cols => width

    # cons dest array (filled with gray), get dest dims
    # NOTE: each 32-bit [B, G, R, A] pixel value is [128, 128, 128, 255]
    dest = np.full(dest_shape, np.uint8(128))
    dest[:, :, 3] = np.uint8(255)
    dest_width = dest.shape[1]
    dest_height = dest.shape[0]
```

Exercise 2: Image Capture

Preliminary Imports and Connect to Camera

```
# Important that this code block is only run once!
# Otherwise will need to restart kernel
from picamera2 import Picamera2
import timeit [DO WE USE TIMEIT FOR ANYTHING?]
import cv2

# instantiate camera instance
picam2 = Picamera2()
```

Debrief

- What questions or observations do you have?
- Does anything need clarification?
- Anything else you'd like to do in this notebook?

Lesson 5

Object Detection

Carnegie Mellon University
Software Engineering Institute

Lesson 5: Learning Objectives

- Distinguish and discuss object identification/detection methods
  - Different objectives, different outputs
- Explain the purpose of bounding boxes.
- Define YOLO algorithm output.
- Produce bounding boxes for images using the YOLO algorithm.

Lecture 5

Object Detection

Carnegie Mellon University
Software Engineering Institute

The Object Detection Problem

- Object Detection entails two sub-problems

1. *Localization* - determining the location on an image where certain objects are present
2. *Classification* – determining what those objects are

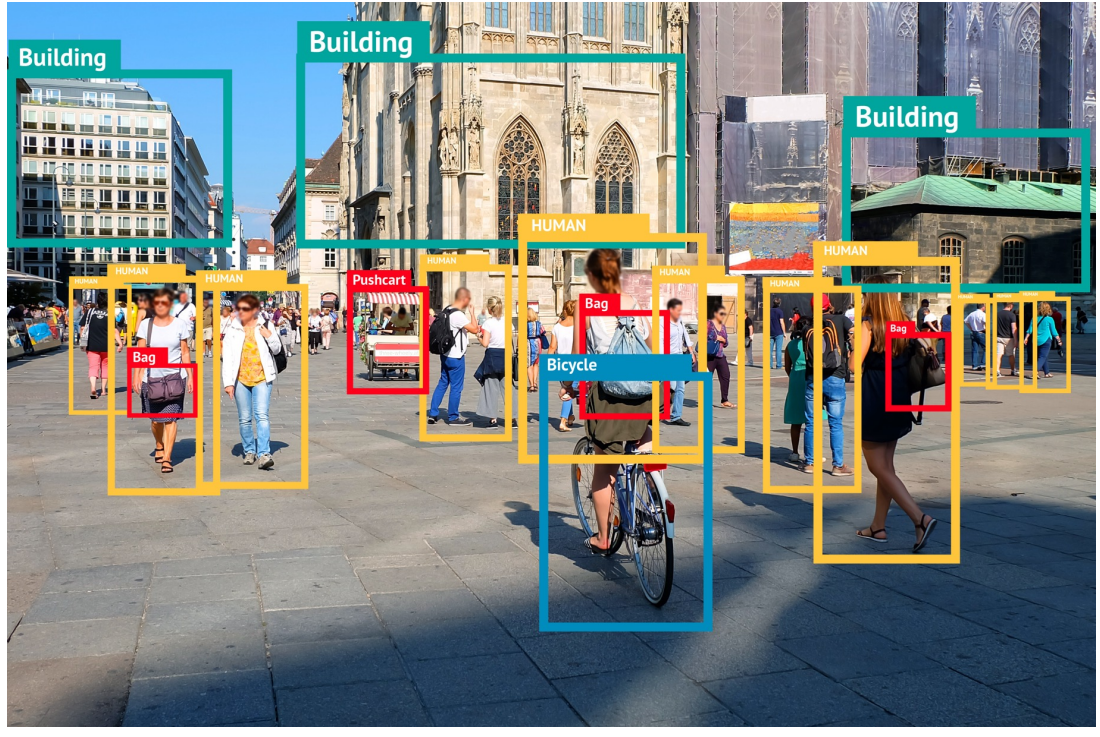
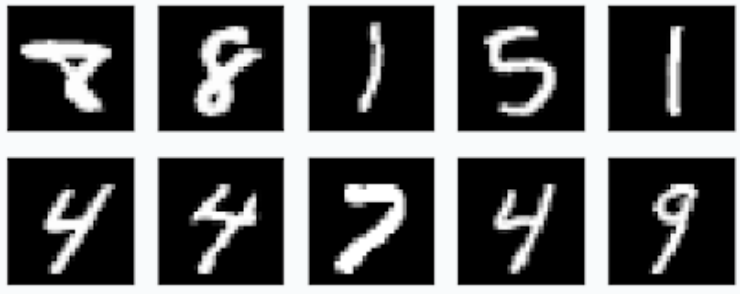
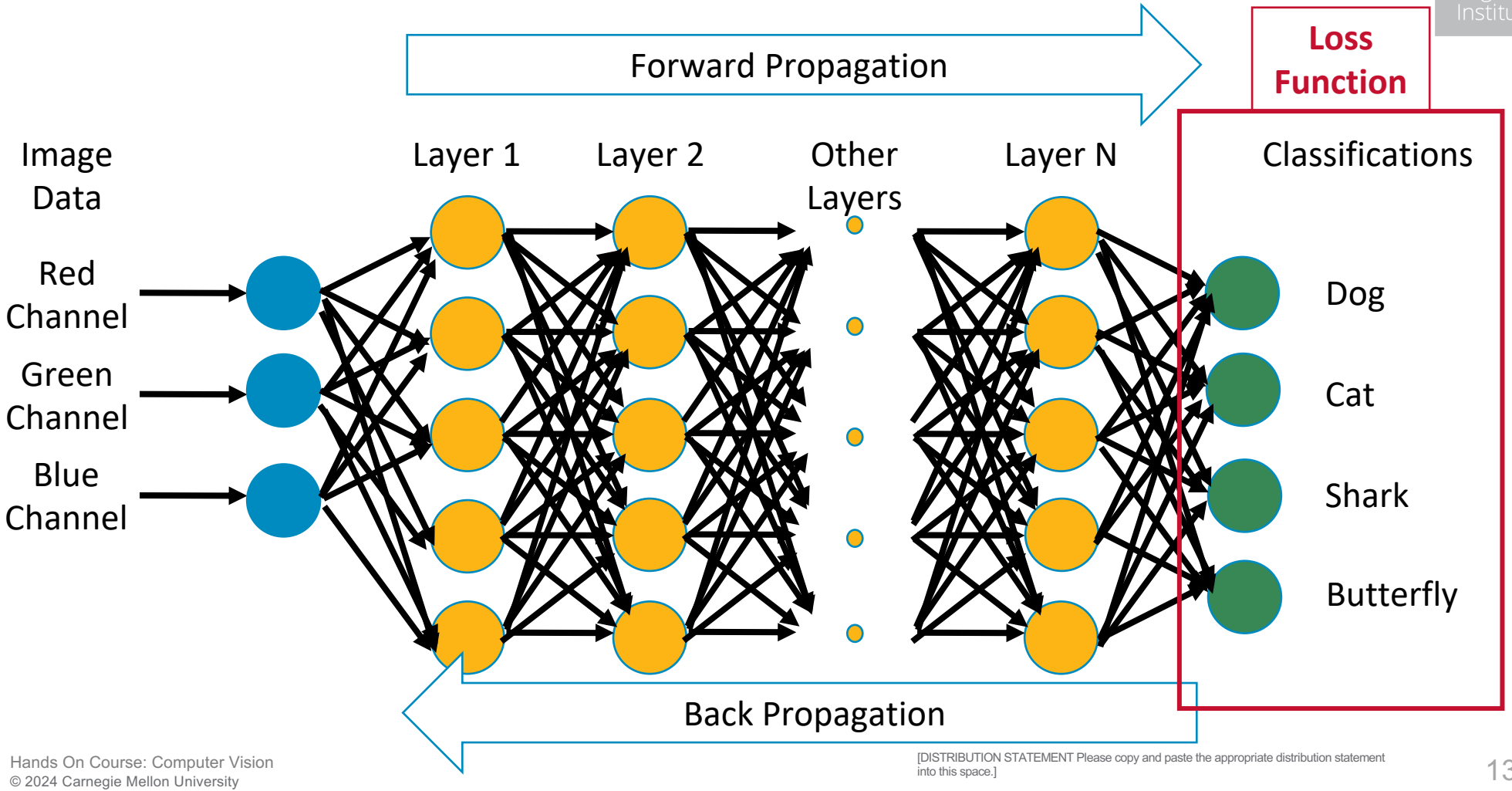


Image Classification

- A classic problem in CV is classifying digits using the MNIST dataset
- **Input x:** 28x28x1 (greyscale) image
- **Output ŷ:** 10 class probabilities corresponding to the digits 0 through 9
- **Ground truth y:** True class probabilities
- **Loss function**: Mean Squared Error: $(y_i - \hat{y}_i)^2$
  - Intuition: loss should measure how far we are from the correct answer



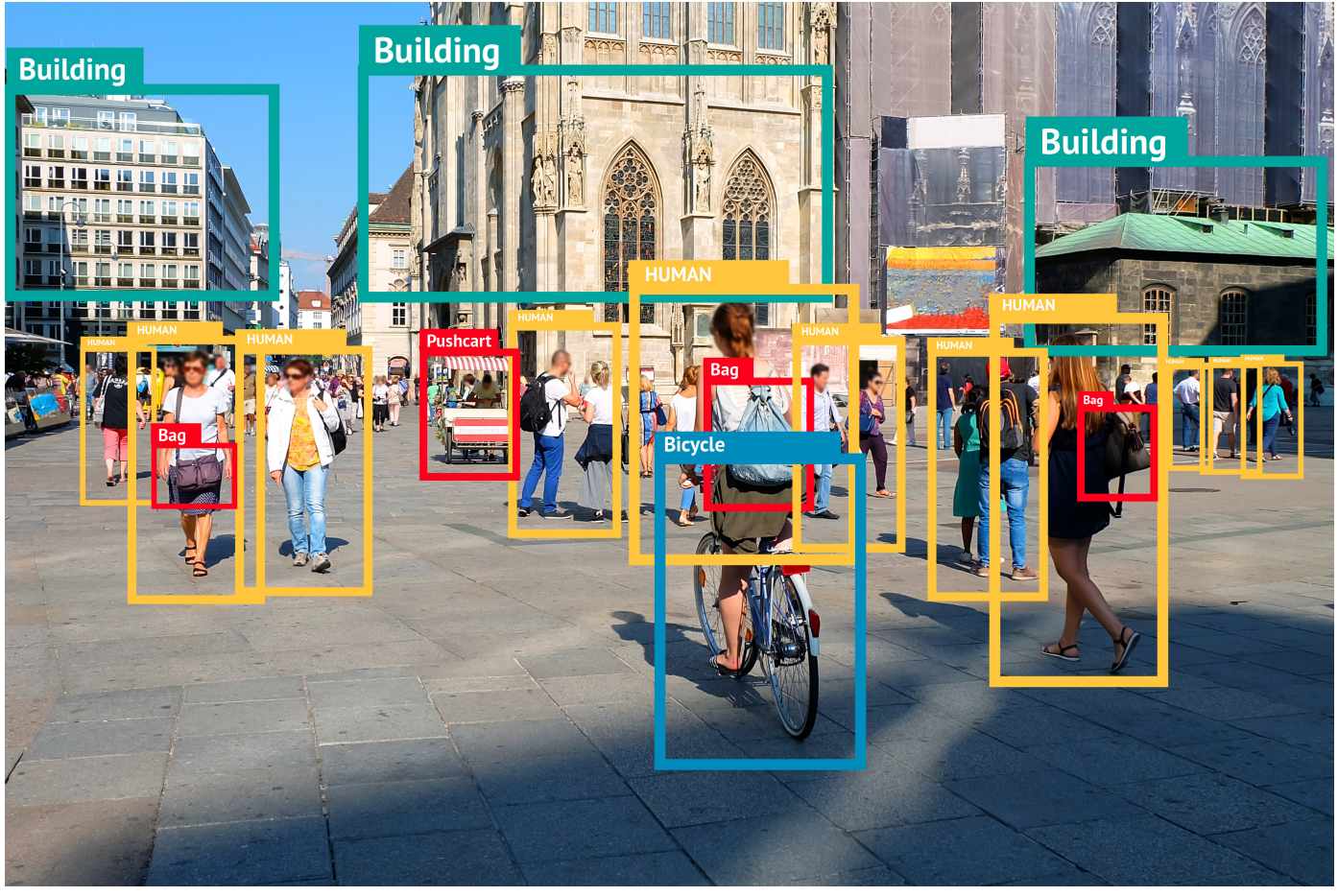
Neural networks are trained using forward and back propagation.



Object Classification for COCO

- COCO is a much larger and diverse dataset of images than MNIST
- https://cocodataset.org/#explore
- There are 80 types of labeled objects so our object classification model will have to output 80 class probabilities

Localization – How?

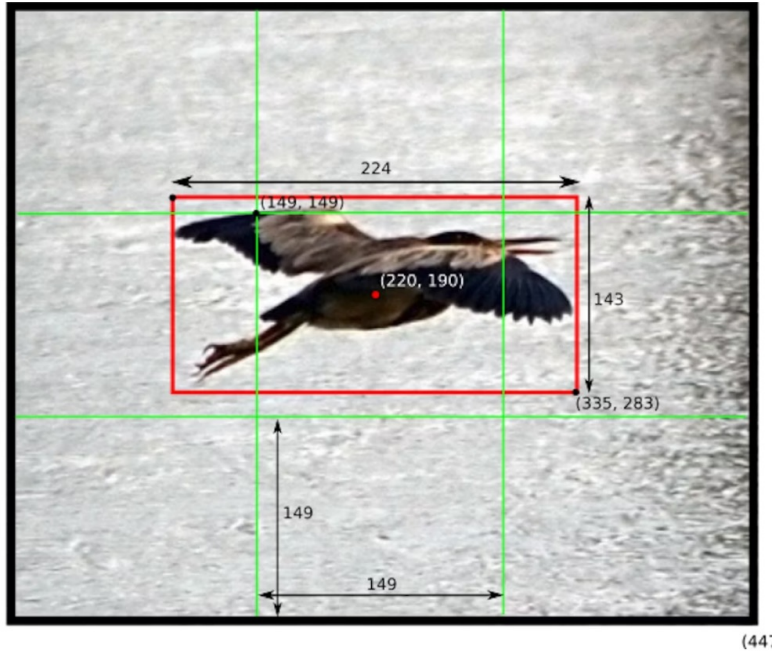


In object detection, bounding boxes identify object locations

- Bounding boxes identify the locations of objects in an image
- Bounding boxes can be specified with coordinates, typically we use:
  - **x** (x-coordinate of the center of the box)
  - **y** (y-coordinate of the center of the box)
  - **w** (width of the box)
  - **h** (height of the box)
- Object Detection models output these four coordinates

Bounding boxes and grid cells

- Images are sub-divided into an *SxS* grid of cells for object detection (here *S*=3)



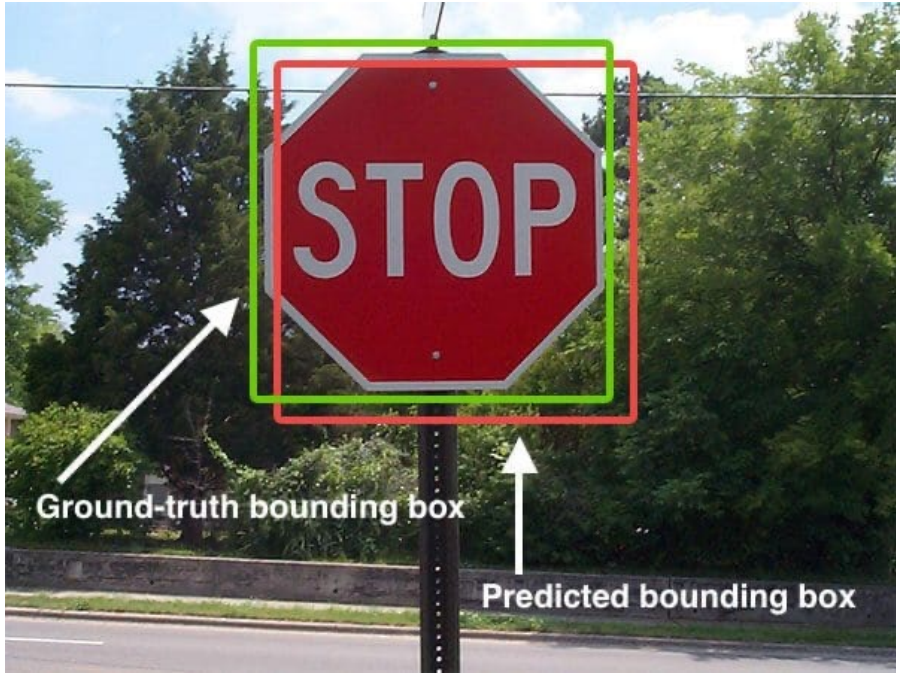
x = (220-149) / 149 = 0.48
y = (190-149) / 149 = 0.28
w = 224 / 448 = 0.50
h = 143 / 448 = 0.32

Example of how to calculate box coordinates in a 448x448 image with S=3. Note how the (x,y) coordinates are calculated relative to the center grid cell

General object detection output

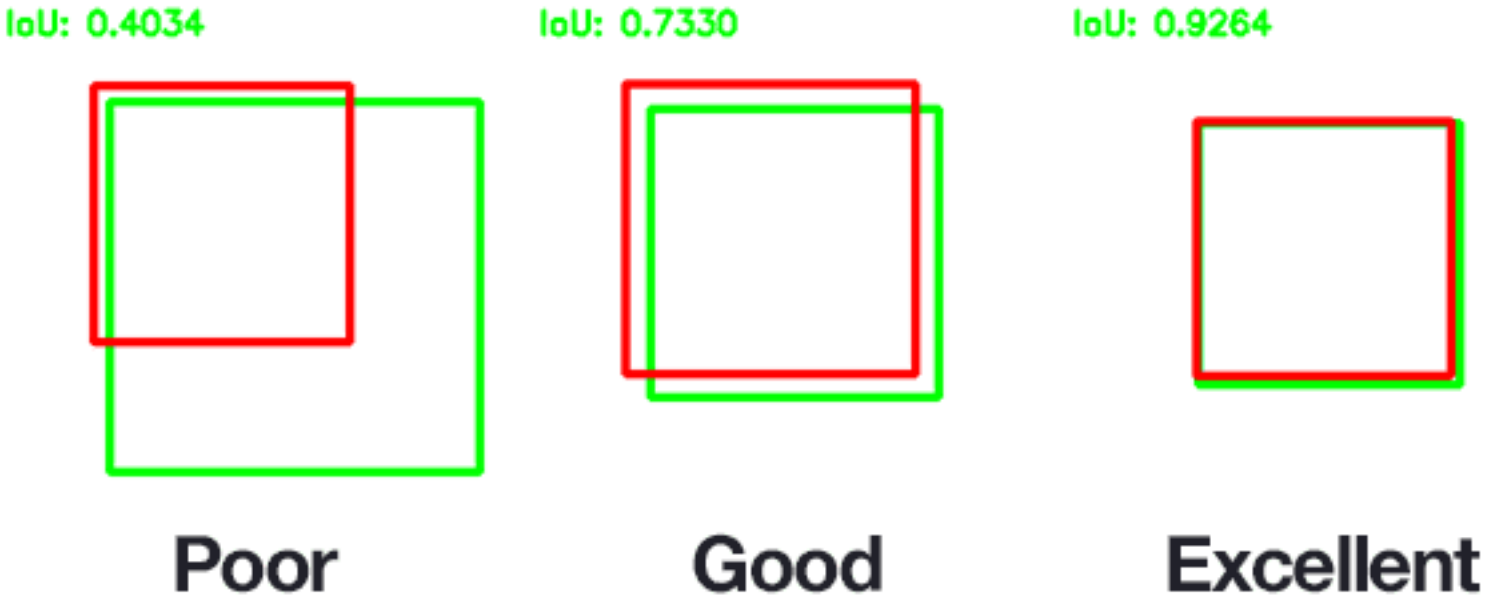
- Each of our *SxS* grid cells predicts *B* bounding boxes
- Each bounding box prediction has 5 components:
  - *x*
  - *y*
  - *w*
  - *h*
  - *confidence/objectness score* (we'll talk more about this later – for now think of it as the confidence our prediction)

During training bounding box quality is assessed with **Intersection over Union** metric



$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

Intersection over Union (cont.)



Exercise 5

Bounding Boxes

Carnegie Mellon University
Software Engineering Institute

Instructions

1. Exercise 5: Drawing BBoxes on test image.
   a. **Draw Bounding Boxes and labels on the image.**
   b. **Show too many overlapping detections???**
   c. **Failure motivates introduction of NMS**

Debrief

- What questions or observations do you have?
- Does anything need clarification?
- Anything else you'd like to do in this notebook?

Lesson 6

You Only Look Once (YOLO)

Carnegie Mellon University
Software Engineering Institute

Lesson 6: Learning Objectives

- Describe YOLO (bounding box) output.
- Explain the purpose of non-max suppression.
- Apply non-max suppression for postprocessing.
- Interpret YOLO (bounding box) output.

Lecture 6

YOLO Input and Output

Carnegie Mellon University
Software Engineering Institute

Yolov1 (2016)

You Only Look Once: Unified, Real-Time Object Detection

Joseph Redmon*, Santosh Divvala*†, Ross Girshick¶, Ali Farhadi*†
University of Washington*, Allen Institute for AI†, Facebook AI Research¶
http://pjreddie.com/yolo/

Abstract

We present YOLO, a new approach to object detection. Prior work on object detection repurposes classifiers to perform detection. Instead, we frame object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. A single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation. Since the whole detection pipeline is a single network, it can be optimized end-to-end directly on detection performance.

Our unified architecture is extremely fast. Our base YOLO model processes images in real-time at 45 frames per second. A smaller version of the network, Fast YOLO, processes an astounding 155 frames per second while still achieving double the mAP of other real-time detectors. Compared to state-of-the-art detection systems, YOLO makes more localization errors but is less likely to predict false positives on background. Finally, YOLO learns very general representations of objects. It outperforms other detection methods, including DPM and R-CNN, when generalizing from natural images to other domains like artwork.

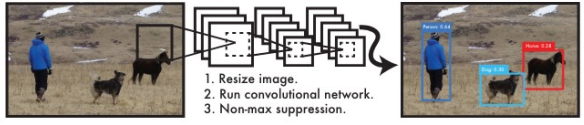


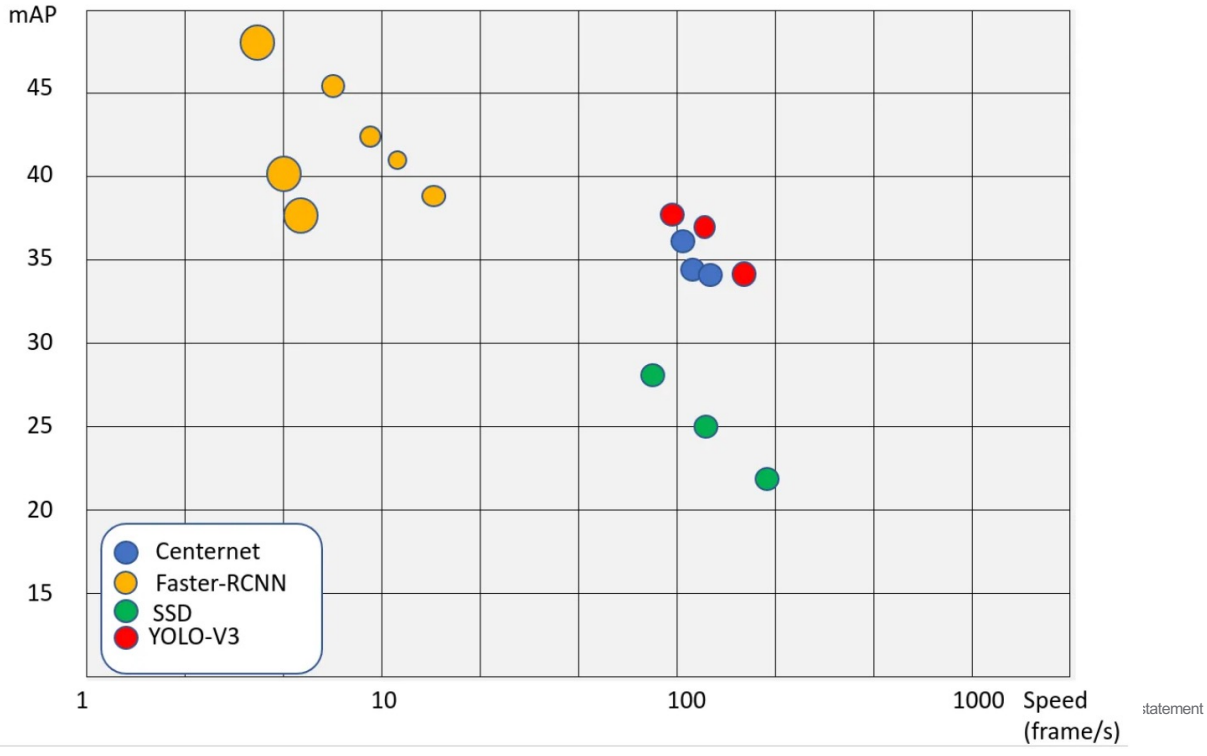
Figure 1: The YOLO Detection System. Processing images with YOLO is simple and straightforward. Our system (1) resizes the input image to 448×448, (2) runs a single convolutional network on the image, and (3) thresholds the resulting detections by the model's confidence.

methods to first generate potential bounding boxes in an image and then run a classifier on these proposed boxes. After classification, post-processing is used to refine the bounding boxes, eliminate duplicate detections, and rescore the boxes based on other objects in the scene [13]. These complex pipelines are slow and hard to optimize because each individual component must be trained separately.

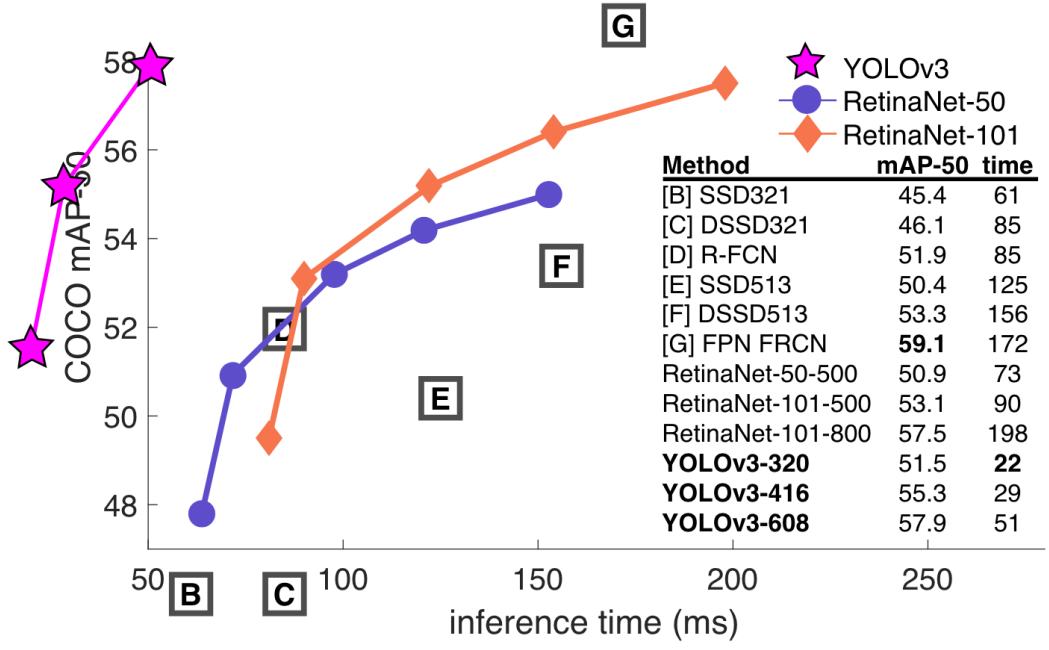
We reframe object detection as a single regression problem, straight from image pixels to bounding box coordinates and class probabilities. Using our system, you only

YOLO (You Only Look Once) is a one stage object detection algorithm used in real-time systems

- v1 published in 2016.
- New versions with better performance have been developed over the years. Most recently Yolov9
- We're using Yolov3 Tiny
- Fast and accurate



Joseph Redmon aside



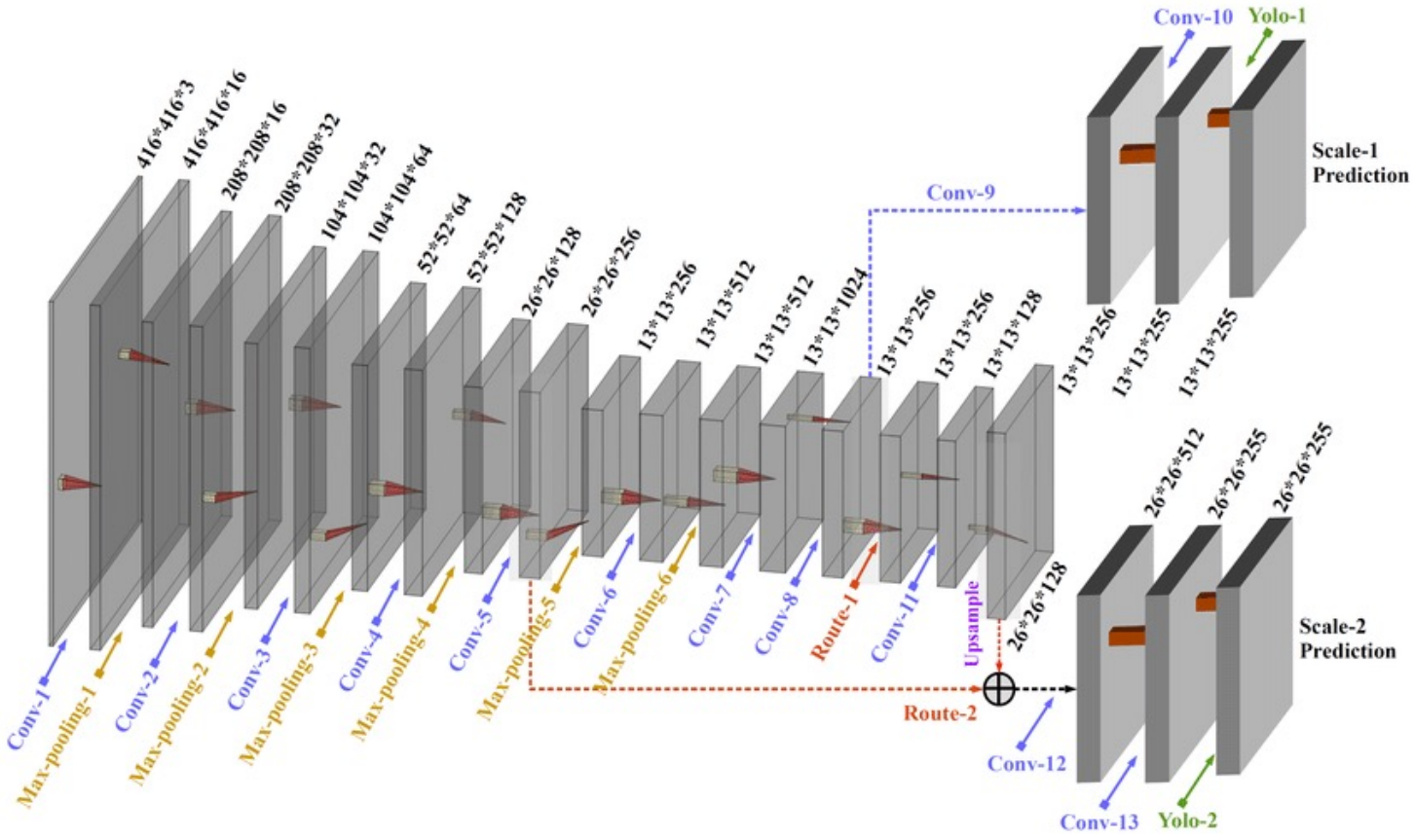
Method	mAP-50	time
[B] SSD321	45.4	61
[C] DSSD321	46.1	85
[D] R-FCN	51.9	85
[E] SSD513	50.4	125
[F] DSSD513	53.3	156
[G] FPN FRCN	**59.1**	172
RetinaNet-50-500	50.9	73
RetinaNet-101-500	53.1	90
RetinaNet-101-800	57.5	198
YOLOv3-320	51.5	**22**
YOLOv3-416	55.3	29
YOLOv3-608	57.9	51

Figure 3. Again adapted from the [9], this time displaying speed/accuracy tradeoff on the mAP at .5 IOU metric. You can tell YOLOv3 is good because it's very high and far to the left. Can you cite your own paper? Guess who's going to try, this guy → [16]. Oh, I forgot, we also fix a data loading bug in YOLOv2, that helped by like 2 mAP. Just sneaking this in here to not throw off layout.

1. Introduction

Sometimes you just kinda phone it in for a year, you know? I didn't do a whole lot of research this year. Spent a lot of time on Twitter. Played around with GANs a little. I had a little momentum left over from last year [12] [1]; I managed to make some improvements to YOLO. But, honestly, nothing like super interesting, just a bunch of small changes that make it better. I also helped out with other people's research a little.

YOLOv3 Tiny Architecture



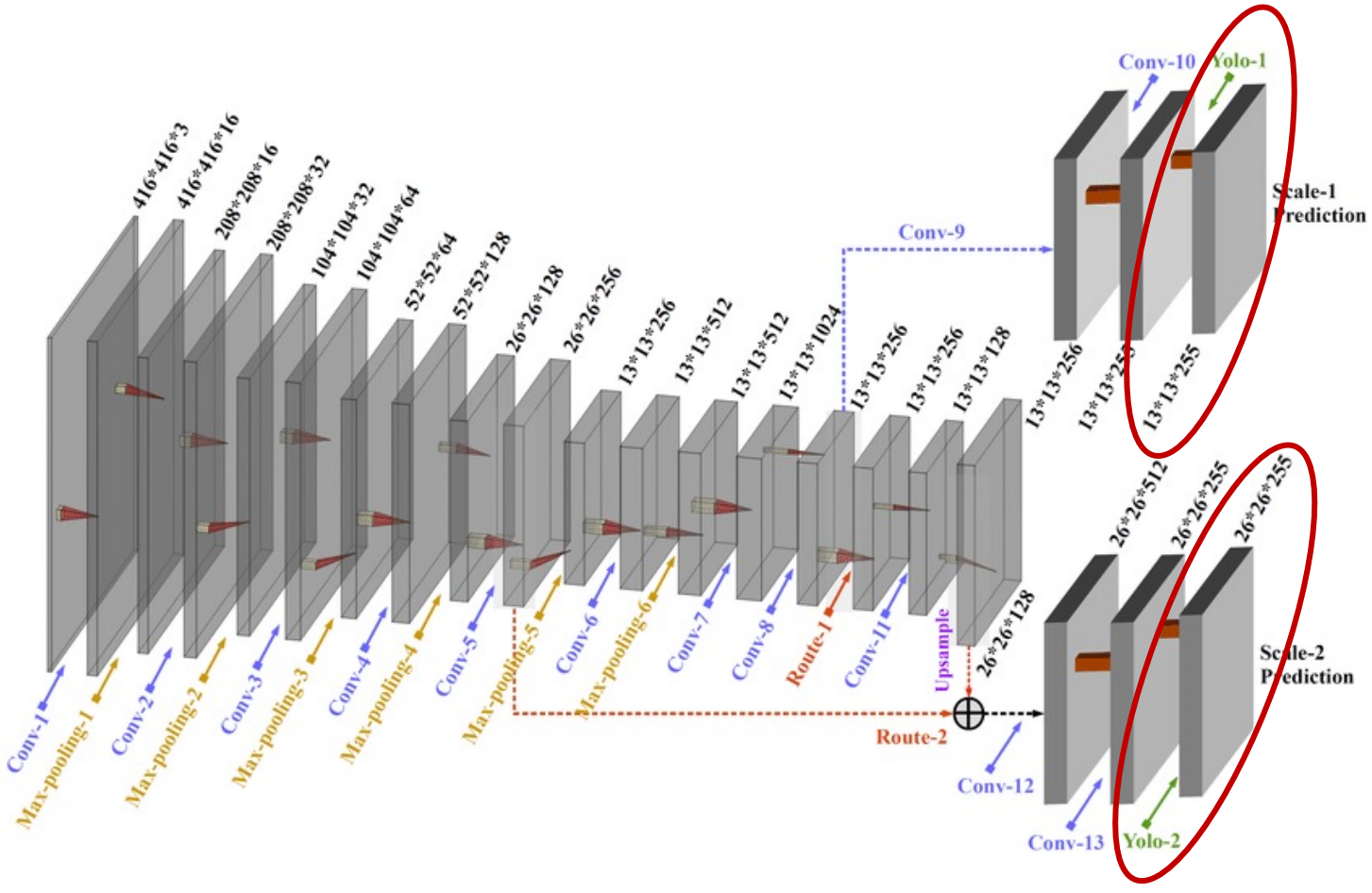
YOLO is a single convolutional neural network.

- Divides image into an *SxS* grid of cells
- Predicts bounding boxes and class probabilities for each cell in one pass of the network
- Each cell predicts *B* bounding boxes and 1 set of conditional class probabilities
  C = P(Class$_i$ | Object)
- 5 predictions for each bounding box: *x, y, w, h, confidence/objectness score*
- Confidence/Objecntess scores are interpreted as the P(object) * IOU(pred, truth)
- Class-specific confidence scores can be derived as follows:

$$\Pr(\text{Class}_i|\text{Object}) * \Pr(\text{Object}) * \text{IOU}_{\text{pred}}^{\text{truth}} = \Pr(\text{Class}_i) * \text{IOU}_{\text{pred}}^{\text{truth}} \quad (1)$$

This gives us class specific confidence scores for each box

YOLOv3 Tiny Architecture: Two Prediction Blocks



Multiple Prediction Blocks for Better Predictions

- Different prediction blocks process the image at different spatial compressions
- Allows the network to learn objects at different sizes
- Block 1 (13x13) for larger objects: broader context, poorer resolution
- Block 2 (26x26) for smaller objects: less context, greater resolution
- Prediction blocks receive both highly-processed features and partly-processed features from earlier in network

YOLOv3-Tiny Output Dimensions

- S x S x (B x (5 + C))
- Prediction Block 1
  - 13 x 13 x (3 x (5 + 80)) = 13 x 13 x 255
- Prediction Block 2
  - 26 x 26 x (3 x (5 + 80)) = 26 x 26 x 255

YOLOv1 Loss Function

$$\lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+ \lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2$$

YOLOv1 Loss Function

Coordinate Loss

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2$$

YOLOv1 Loss Function

Carnegie Mellon University Software Engineering Institute

Coordinate loss

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

where $\mathbb{1}_i^{\text{obj}}$ denotes if object appears in cell i and $\mathbb{1}_{ij}^{\text{obj}}$ denotes that the jth bounding box predictor in cell i is "responsible" for that prediction.

"responsible" box

YOLO predicts multiple bounding boxes per grid cell. At training time we only want one bounding box predictor to be responsible for each object. We assign one predictor to be "responsible" for predicting an object based on which prediction has the highest current IOU with the ground truth. This leads to specialization between the bounding box predictors. Each predictor gets better at predicting certain sizes, aspect ratios, or classes of object, improving overall recall.

YOLOv1 Loss Function

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

Objectness or Confidence Loss

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2$$

YOLOv1 Loss Function

Objectness or Confidence Loss

$$+\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{\text{obj}}\left(C_i-\hat{C}_i\right)^2$$

$$+\lambda_{\text{noobj}}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{\text{noobj}}\left(C_i-\hat{C}_i\right)^2$$

- Loss associated with the confidence score for each bounding box predictor
- C_hat is the confidence score and C is the IOU of the predicted bounding box with the ground truth (true or calibrated confidence)

YOLOv1 Loss Function

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

Classification Loss

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2$$

YOLOv1 Loss Function

Loss associated with misclassification of object

Classification Loss

$$+\sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c)\right)^2$$

YOLOv1 Loss Function – lambdas are hyperparameters

Coordinate Loss

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

Objectness or Confidence Loss

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

Classification Loss

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2$$

YOLOv1 to YOLOv2

	YOLO							YOLOv2
batch norm?		✓	✓	✓	✓	✓	✓	✓
hi-res classifier?			✓	✓	✓	✓	✓	✓
convolutional?				✓	✓	✓	✓	✓
anchor boxes?				✓	✓			
new network?					✓	✓	✓	✓
dimension priors?						✓	✓	✓
location prediction?						✓	✓	✓
passthrough?							✓	✓
multi-scale?								✓
hi-res detector?								✓
VOC2007 mAP	63.4	65.8	69.5	69.2	69.6	74.4	75.4	76.8	**78.6**

Table 2: The path from YOLO to YOLOv2. Most of the listed design decisions lead to significant increases in mAP. Two exceptions are switching to a fully convolutional network with anchor boxes and using the new network. Switching to the anchor box style approach increased recall without changing mAP while using the new network cut computation by 33%.

Anchor boxes are used to predict object locations.

- Multiple anchor boxes are proposed around a pixel
- The anchor boxes generated will have different sizes and aspect ratios
- They acts as informed priors about size and location of objects
- Introduced in Yolov2

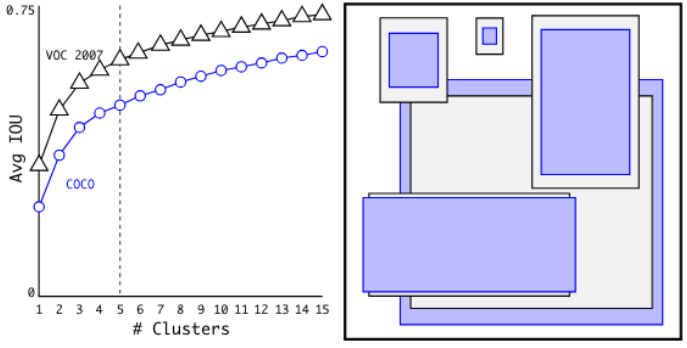


Figure 2: Clustering box dimensions on VOC and COCO. We run k-means clustering on the dimensions of bounding boxes to get good priors for our model. The left image shows the average IOU we get with various choices for k. We find that $k = 5$ gives a good tradeoff for recall vs. complexity of the model. The right image shows the relative centroids for VOC and COCO. Both sets of priors favor thinner, taller boxes while COCO has greater variation in size than VOC.

Bounding box prediction details

2.1. Bounding Box Prediction

Following YOLO9000 our system predicts bounding boxes using dimension clusters as anchor boxes [15]. The network predicts 4 coordinates for each bounding box, t_x, t_y, t_w, t_h. If the cell is offset from the top left corner of the image by (c_x, c_y) and the bounding box prior has width and height p_w, p_h, then the predictions correspond to:

$$
\begin{aligned}
b_x &= \sigma(t_x) + c_x \\
b_y &= \sigma(t_y) + c_y \\
b_w &= p_w e^{t_w} \\
b_h &= p_h e^{t_h} \\
Pr(\text{object}) * IOU(b, \text{object}) &= \sigma(t_o)
\end{aligned}
$$

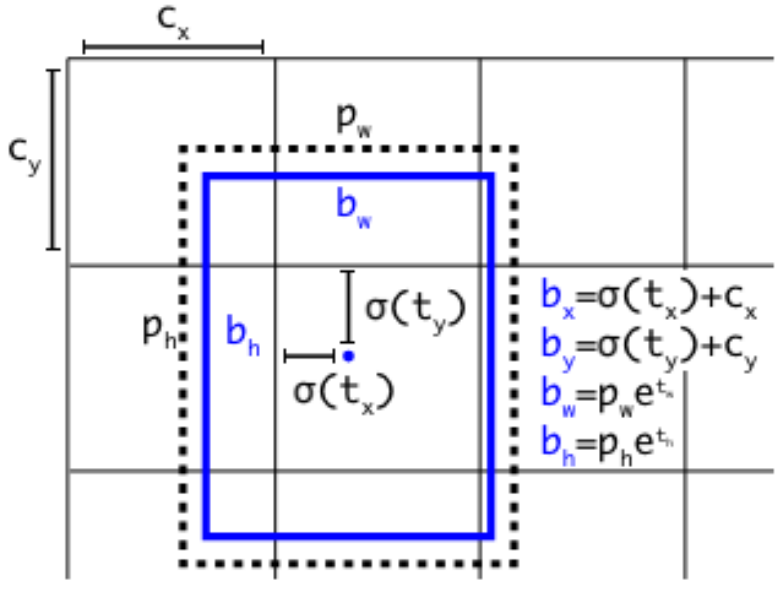


Figure 2. **Bounding boxes with dimension priors and location prediction.** We predict the width and height of the box as offsets from cluster centroids. We predict the center coordinates of the box relative to the location of filter application using a sigmoid function. This figure blatantly self-plagiarized from [15].

YOLOv3 Loss Function

Coordinate Loss

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

Objectness or Confidence Loss

$$\sum_{i=0}^{S^2} \sum_{j=0}^{B} I_{ij}^{obj} \left[\hat{C}_i \log(C_i) + \left(1 - \hat{C}_i\right) \log\left(1 - C_i\right) \right] +$$

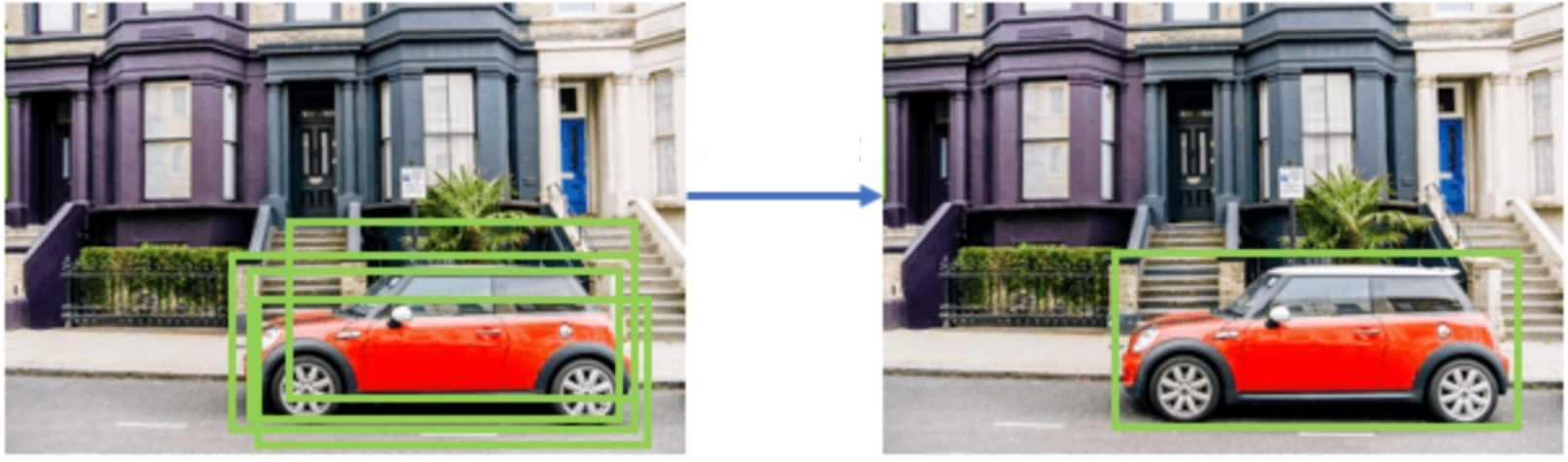
$$\lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} I_{ij}^{noobj} \left[\hat{C}_i \log(C_i) + \left(1 - \hat{C}_i\right) \log(1 - C_i) \right] +$$

Classification Loss

$$\sum_{i=0}^{S^2} I_{ij}^{obj} \sum_{c \in classes} \left[\hat{p}_i(c) \log(p_i(c)) + \left(1 - \hat{p}_i(c)\right) \log(1 - p_i(c)) \right]$$

What to do about too many detections?

Ideas?



Non-Maximum Suppression

- Algorithm for filtering the YOLO bounding box detections
- Given a list *P* of bounding box predictions each of the form (*x,y, w, h, c*), set some threshold IOU *thresh*, then:

1. Select the prediction *S* with the highest confidence score
2. Remove *S* from *P* and put it in our keep list *K*
3. Now calculate the IOU of *S* with every prediction in *P*, if the IOU is greater than *thresh* for some prediction *T*, remove *T* from *P*
4. If there are still predictions in *P* return to step 1, else return keep list *K*

- Lower *thresh* is a more aggressive filter

Speed-Performance Tradeoffs

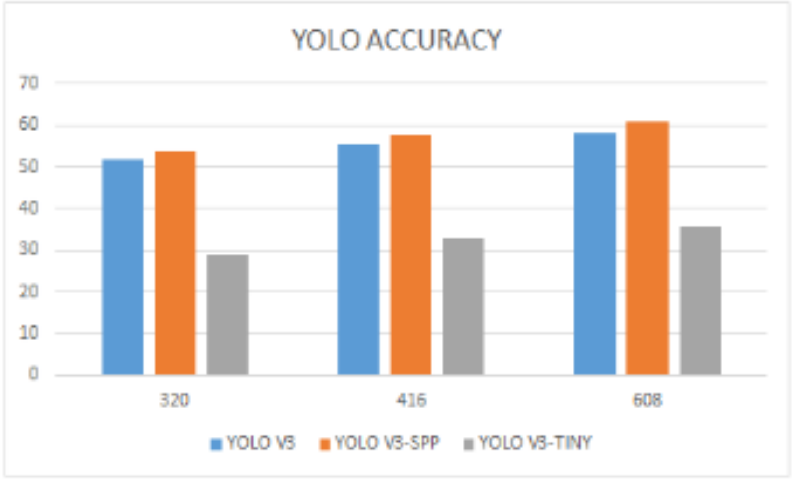


Fig.8: Accuracy comparison of YOLO algorithms

YOLO v3-Tiny is a lightweight variant of YOLO v3, which takes less running time and less accuracy when examined with YOLO v3. In Fig. 8, we compared the accuracy of different versions of YOLO algorithms for given image pixels.

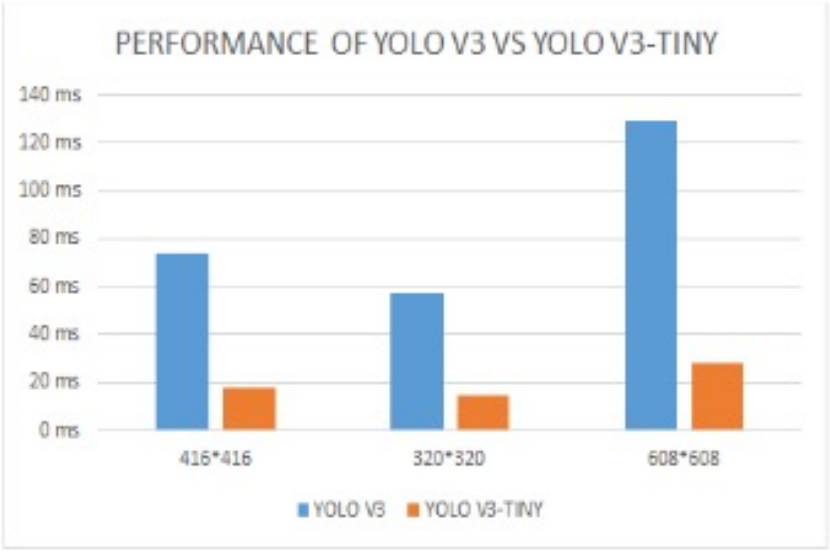


Fig.14: Speed Comparison

The above graph in Fig. 14 shows a comparison of the running time of YOLO v3 with YOLO v3-Tiny for different dimensions of images.

There are broadly two categories of object detection algorithms in current deep learning-based computer vision.

- One stage approaches like Single Shot Detector (SSD) and YOLO
- Regional-based approaches (two stage) like RCNN and later versions.
- One stage algorithms are faster but not as accurate as two stage.
- Two stage algorithms are more accurate because they identify and narrow down regions of where object(s) may be located.
- Although one stage can be less accurate, the increased speed makes them more appropriate for real-time systems. Although Faster RCNN can be used in real-time, YOLO is currently better when performance counts like in autonomous vehicles.

Exercise 6

Post-Processing and Non-Max Suppression

Carnegie Mellon University
Software Engineering Institute

Post processing the detections (non-max suppression - NMS)

After showing too many overlapping detections show how this can be reduced.

2. Output of Post-processing

•The Inference program produces a list of numpy arrays, the shape of which is displayed as the output. These arrays predict both the bounding boxes and class labels but are encoded.

•Further we will take each of the numpy arrays and decode the candidate bounding boxes and class predictions. If there are any bounding boxes that don't confidently describe an object (having class probabilities less than the threshold, 0.3) we'll ignore them.

•Here a maximum of 200 bounding boxes can be considered in an image.

•I have used the correct_yolo_boxes() function to perform translation of bounding box coordinates so that we can plot the original image and draw the bounding boxes.

•Again to remove those candidate bounding boxes that may be referring to the same object, we define the amount of overlap as non-max suppression threshold = 0.45.

•Also there is a need to rescale the coordinates of the bounding boxes to the original image along with displaying the label and scores on top of each of them.

•All these Post-processing steps need to be performed before we get the bounding boxes along with the recognized classes in our output image.

Running Object detection on NMS output

Play with 3 hyperparameters (object confidence threshold, class confidence threshold, and non-max suppression threshold)

Debrief

- What did you see?
- What did we choose for you? (maybe)

Lesson 7

Inference on Images

Carnegie Mellon University
Software Engineering Institute

Lecture 7

Pipeline Review?

Carnegie Mellon University
Software Engineering Institute

Exercise 7

Running the Computer Vision Pipeline

Carnegie Mellon University
Software Engineering Institute

Instructions

Carnegie Mellon University Software Engineering Institute

PUT IT ALL TOGETHER (CAMERA VIDEO to DETECTION STREAM)

Debrief

- What did you see?
- What did we choose for you? (maybe)

Lesson 8

Conclusion

Carnegie Mellon University
Software Engineering Institute

Lecture 8

Additional Topics

Carnegie Mellon University
Software Engineering Institute

MENTION LIMITATIONS IN FINAL LESSON. Computer vision can't do everything.

Amazon a

AI STANDS FOR "ACTUALLY, INDIANS" —

Amazon Fresh kills “Just Walk Out” shopping tech—it never really worked

"AI" checkout was actually powered by 1,000 human video reviewers in India.

RON AMADEO - 4/3/2024, 12:55 PM

from *Ars Technica*

Application developers don't program models from scratch. They use libraries like PyTorch and TensorFlow

- List reasons to use PyTorch
- List reasons to use TensorFlow
- We'll be using PyTorch

Exercise 8

TBD

Carnegie Mellon University
Software Engineering Institute

IDEAS/NEEDED

- ML Pipeline: Training data/testing data, validation
- High level description of frameworks
- Evaluation metrics
- Basic neural network definition
- Importance of ImageNet
- Some famous architectures (LeNet, AlexNet, etc.)
- Think about: What might you want to detect/measure?
- Convolutional neural networks
- Anatomy of a CNN layer (filter, padding, etc)
- Faster R-CNN (maybe)
- YOLO
- Open issues (design still experimental/heuristic, anything interesting emerging?)
- Examples of failures/current limitations
- Anything interesting emerging? Near future capabilities?

More IDEAS/NEEDED

- What else do we need to prep for Module 2?
- What about hardware considerations?
- What high level things are needed to understand processors/memory?
- What about cameras?
- How can we prep the pipeline issues/functions that the learner will see in Module 2?
- What's missing?

McMillan Notes (where does this content go)

- There is A LOT of content so far (I think it is longer than a 2 hour course)
- This content is training heavy and not enough inference considerations.
- Need to explain that the same image preprocessing that was done for training needs to be applied to the inference images (i.e. the ones coming from the camera)
- There is a lot that needs to be done to the camera images before they can be fed into the chosen model
- Do you put these manipulations/transformations in the section after explaining pixels?
- Need slides on pretrained model repositories

Final Reflections

- What are the three broad categories of computer vision applications?
- What are some applications of computer vision? How do they relate to the categories?
- How is image data structured?
- What colors are needed to create all other colors?
- What is alpha?
- How are image data values represented?
- What's the purpose of image augmentation?
- What are some common image augmentation techniques?

Module 2

Inference on the Pi

Carnegie Mellon University
Software Engineering Institute

McMillan/Steele Outline

- Unbox and set up RPi and camera module. Provide preimaged uSDCard
  - Slide on hardware contents, order of connections
  - Slide on what went into creating the uSDCard image: Raspberry OS, python+modules onnx2torch, etc
  - Exercise Slide on running camera hello world (code provided on SDCard)
- Getting a pretrained model (**TinyYoloV3 may not come with NNMS**)
  - Slide on model repositories
  - Slide on ONNX model exchange format and viewers?
  - Slide (model card??) Understanding what the model expects from input and what it provides as output, what classes is it trained on.
  - Exercise/Slide on sending a test image image through and getting data out
- Tweaking the image pipeline
  - connect the camera output to model input
  - Overlaying bounding box information (too much information)
  - Explain the NMS (non-max suppression) algorithm, provide the code, and rerun the pipeline

Loading YOLO on the Pi

Carnegie Mellon University Software Engineering Institute

Lessons about about model zoos.

Is the model preloaded

Scaling/Cropping/etc the input images

Running Object detection on raw output

Module 3

Compute Constraints

Carnegie Mellon University
Software Engineering Institute

SWaP Constraints

Model compression

Distillation or pruning?

Quanitization

Model size vs. performance tradeoffs

Computer vision falls into three broad categories.

- Object detection (bounding boxes vs. Segmentation vs localization)
- Object classification/identification
- Object tracking (in video streams for example)

- Image restoration, sharpening
- Pose estimation?
- 3D scene construction

Computer vision has many useful applications.

- Optical character recognition
- Autonomous cars and vehicles
- Medical diagnosis (imaging diagnostics)
- Face recognition
- Augmented reality (extended reality or whatever it's called now)

Lecture 4 Leftovers

Lesson 1 Leftovers

Lesson 1 Leftovers

Lesson 3 Leftovers