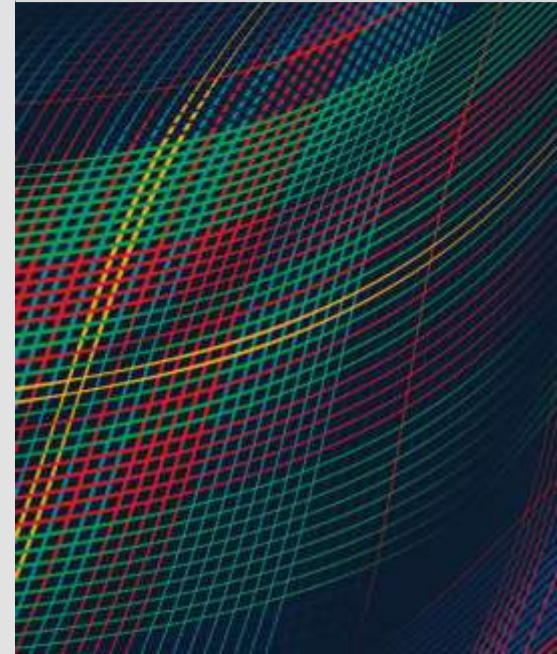


Introduction to Artificial Intelligence Computer Vision Models

GSAW 2025 Tutorial

FEBRUARY 2025

AI Division



This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

This material was prepared for the exclusive use of Ground System Architectures Workshop hosted by Aerospace Corp and may not be used for any other purpose without the written consent of permission@sei.cmu.edu.

DoD R&D Federally Funded Research and Development Center (FFRDC)



One of 10 FFRDC's sponsored by the DoD and the only one authorized to work outside DoD.

Partnerships with Academia and private sector Enable us to take innovations from concept to practice, closing the gap between research and use.

R&D in Artificial Intelligence engineering, software engineering, systems engineering, cybersecurity, and many other areas of computing,

Capable of conducting both fundamental research and classified work

Offices in Pittsburgh and DC, with employees near customer facilities in MA, TX, and CO

CMU SEI Capabilities: AI Division Overview

Mission-Focused Capabilities



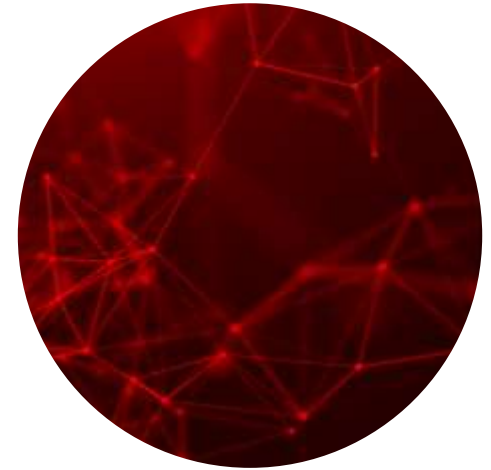
AI for Engineering



AI for Mission



Advanced Computing for AI



Secure and Counter AI

Scalable, Robust and Secure, Human-Centered

Tutorial Instructors



Jonathan Frederick

Senior Technical Project Manager



Cole Frank

Associate AI Workforce Development Engineer

Course Agenda

- Lesson 1: Introduction to Computer Vision and its applications
- Lesson 2: Computer Images
 - *Hands on 1: Representing and processing digital images*
- Lesson 3: Neural Networks for Computer Vision
 - *Hands on 2: Convolution and Pre-Trained Neural Nets (PTNNs)*
- Lesson 4: Object Detection
 - *Hands on 3: YOLO Output*
- Lesson 5: You Only Look Once (YOLO) Model Deep Dive
 - *Hands on 4: Post-processing YOLO output with Non Max Suppression (NMS) algorithm*

Course Learning Objectives

BLUF: Give you a low-level and hands-on overview of how computer vision works

- 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
- Use a computer vision model for object detection
- Postprocess computer vision algorithm (YOLO) results

Please ask questions along the way!

Lesson 1

Introduction to Computer Vision

Lesson 1: Learning Objectives

- Discuss applications of computer vision.
- Define what machine learning is.
- Describe computer vision's place within machine learning.

Lesson 1

Computer Vision Applications

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?

Computer vision has many useful applications.



Before we dive in, some helpful terminology

- Parameters, weights, coefficients
- Predictions, outputs, guesses
- Arrays, matrices, tensors

Arrays are **structured/ordered** collections of numbers

A 1D array with 3 integers: [25 120 75]

Matrices are two dimensional arrays:
$$\begin{bmatrix} 25 & 120 & 75 \\ 120 & 25 & 75 \\ 75 & 120 & 25 \end{bmatrix}$$

Tensors are multidimensional arrays.

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*.
- Effectively, it's now a subfield of machine learning

Situating ourselves (AI, ML, NLP, DL, CNNs)

Artificial Intelligence (AI)

Systems capable of performing tasks that typically require human intelligence

Machine Learning

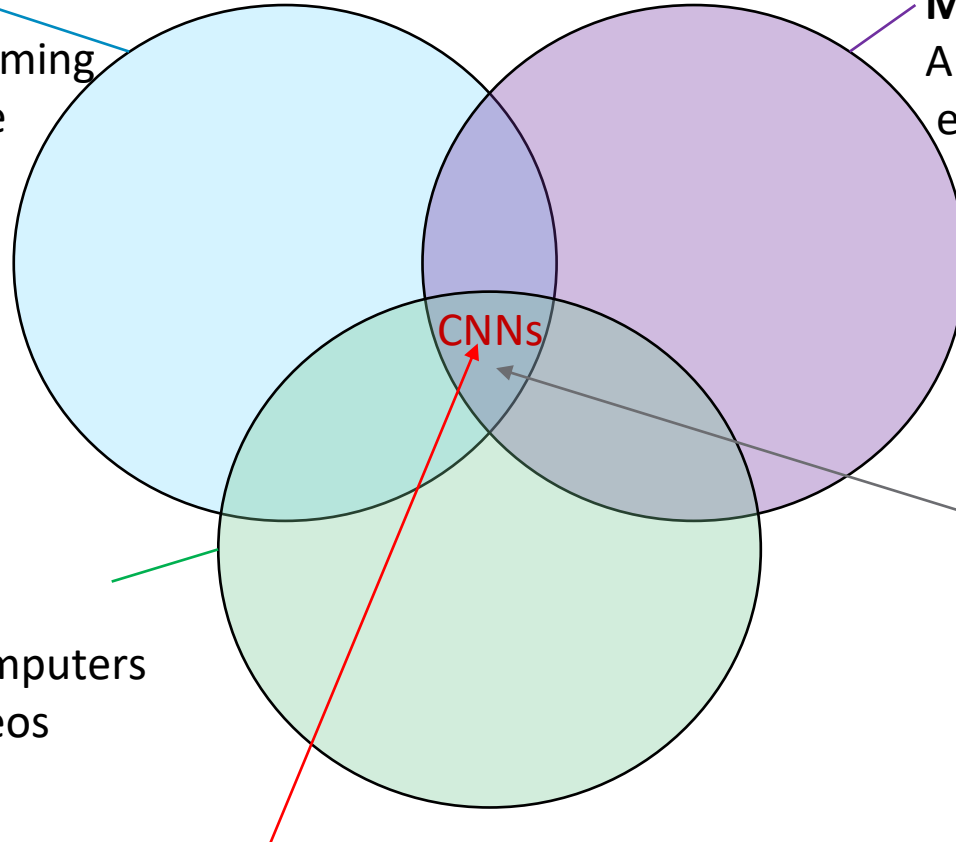
Algorithms/techniques that enable computers to “learn” from data (e.g. neural networks, support vector machines, decision trees, random forests)

Computer Vision (CV)

Algorithms that enable computers to process images and videos

Deep Learning

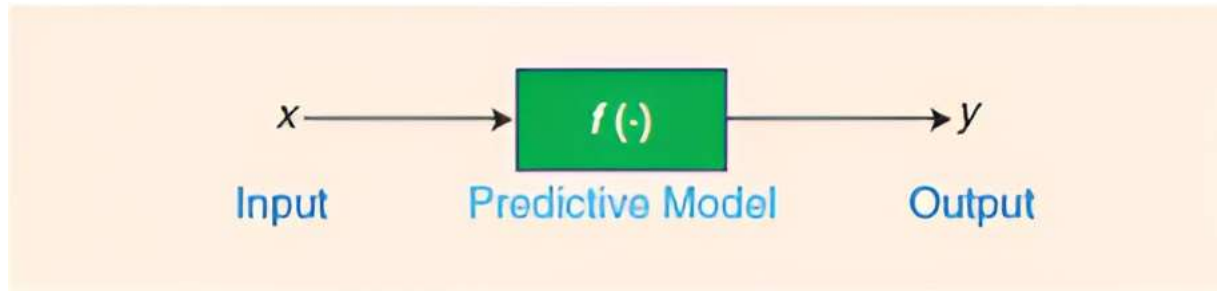
A specific subset of machine learning methods where much of the most notable progress in AI over the last decade has occurred



Convolutional Neural Nets (CNNs)

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

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



At their core ML models are simply mathematical functions!*

$$y = f(x) = 3x + 2$$

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

- A function is a rule that assigns **inputs** to **outputs** over some domain
- The domain refers to the types of objects the function is defined over
- Put differently, a function specifies a relationship between inputs and outputs
- The two equations above are simple functions over the domain of integers

* 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

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

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

$$y = mx + b$$

- 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

* 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

Linear Regression is simple ML

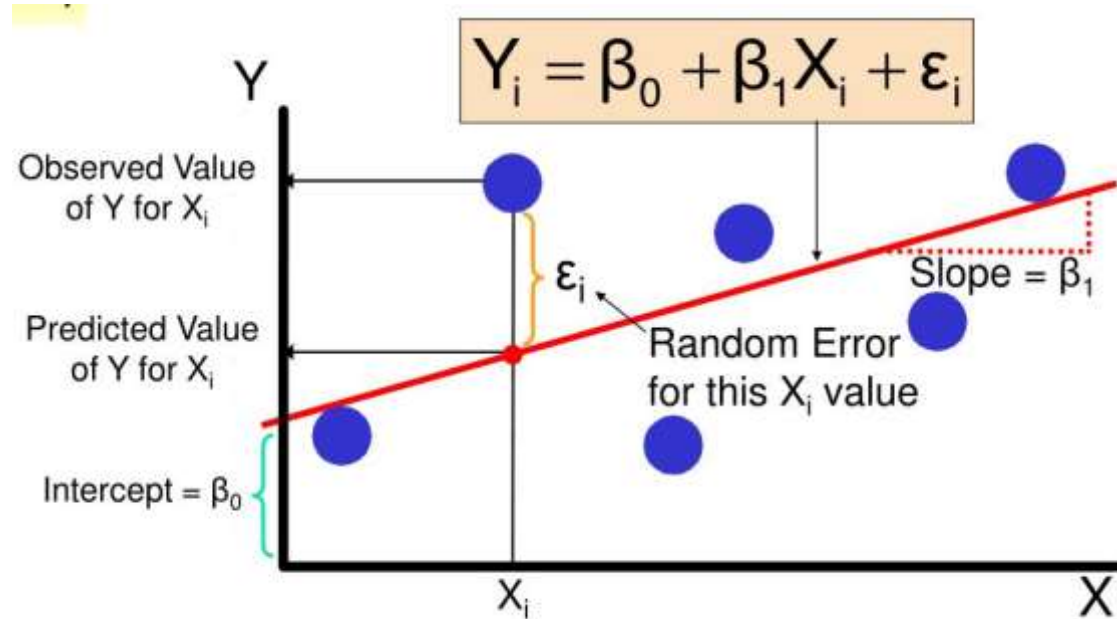
$$y = mx + b$$

Linear Regression: Single Variable

$$\hat{y} = \beta_0 + \beta_1 x$$

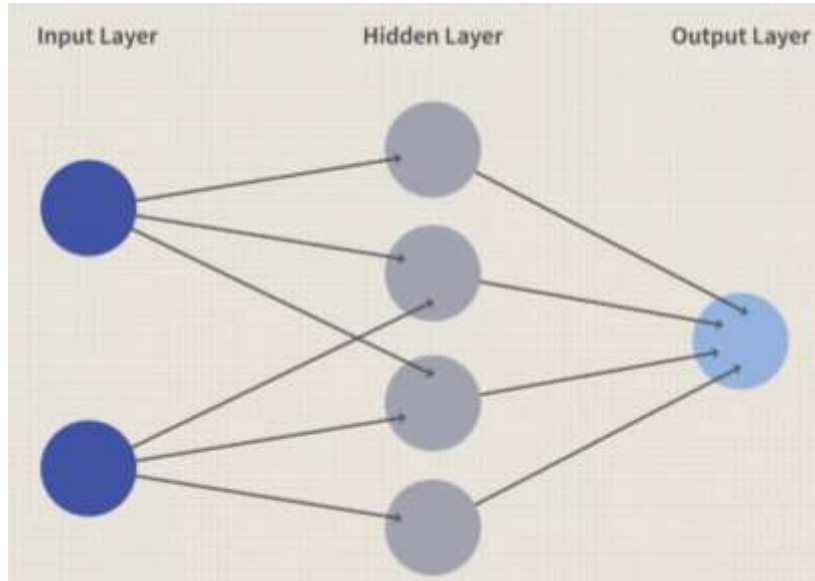
Predicted output Coefficients Input

- We learn the “right” parameters during **training**
- During **inference** we freeze those parameters and use them to make predictions



Neural Networks

- Linear regression is a very simple neural network
- Neural networks are also functions!
- Complicated and nested functions, but still functions
- Uniquely expressive and flexible functions



$$\mathbf{h} = \sigma(\mathbf{W}_{\text{hidden}} \cdot \mathbf{x} + \mathbf{b}_{\text{hidden}})$$

$$y = \sigma(\mathbf{W}_{\text{output}} \cdot \mathbf{h} + b_{\text{output}})$$

Linear Algebra is the Language of Deep Learning

- A deep learning model itself is just lots and lots of transformations of its inputs
- Linear Algebra is the field of mathematics that deals with transformations of vectors and matrices

$$\mathbf{Q} = \mathbf{XW}^{\mathbf{Q}}; \quad \mathbf{K} = \mathbf{XW}^{\mathbf{K}}; \quad \mathbf{V} = \mathbf{XW}^{\mathbf{V}}$$

$$q_i = x_i \mathbf{W}^{\mathbf{Q}}; \quad k_i = x_i \mathbf{W}^{\mathbf{K}}; \quad v_i = x_i \mathbf{W}^{\mathbf{V}}$$

$$\mathbf{A} = \text{SelfAttention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax} \left(\frac{\mathbf{QK}^{\mathbf{T}}}{\sqrt{d_k}} \right) \mathbf{V}$$

Machine Learning



Machine Learning Functions tend to be Complex

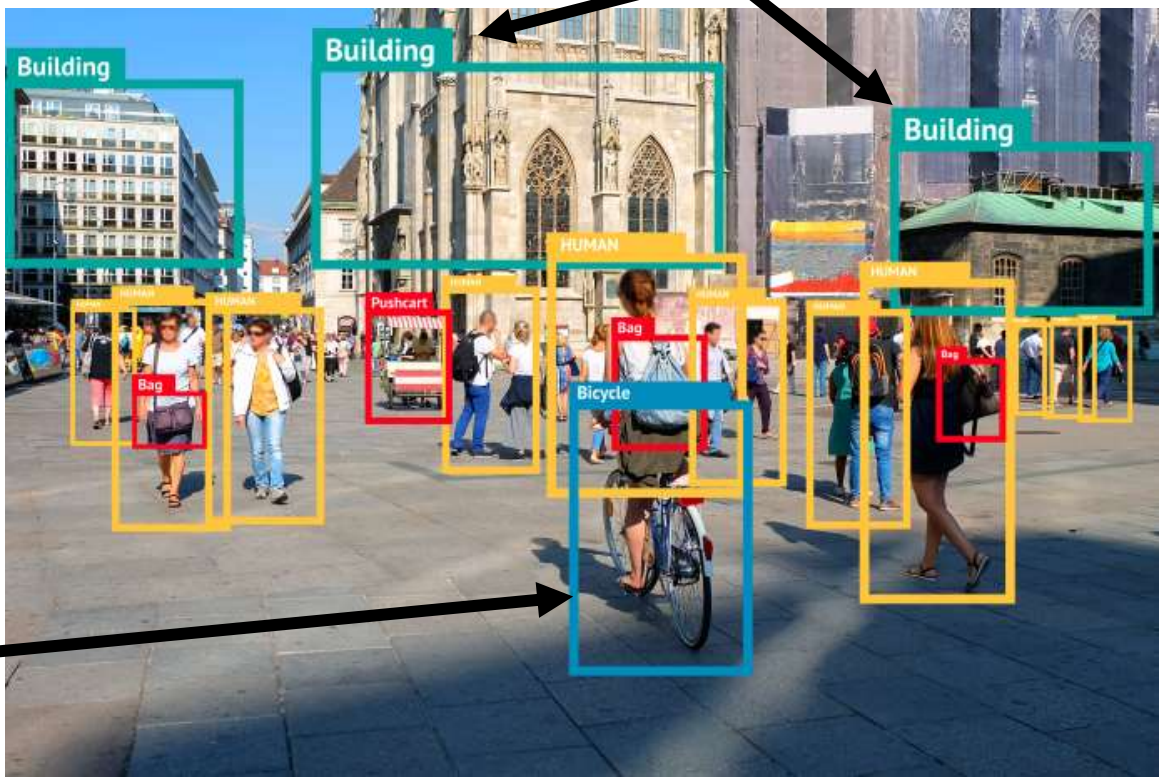
- These are very complicated functions
- Inputs and outputs are high-dimensional vectors (lists) of numbers rather than scalar values
- Many more parameters (millions, billions, even trillions)
- Complicated and non-linear transformations

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 is also numerical
- The form of the output depends on the application (for example, object detection, object classification, identification, object tracking, etc.) but it's always numerical
- **Examples:**
 - **Tank classification model**
 - **Binary vs Multi-label classification**
 - **Object detection model**

Computer vision falls into three broad categories.

2. Object Classification/Identification



1. Object Detection

3. Object Segmentation



Lesson 2

Computer Images

Lesson 2: Learning Objectives

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

Lecture 2

The Anatomy of an Image

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.



For a computer images, are arrays of numbers



0	2	15	0	0	11	10	0	0	0	0	9	9	0	0	0
0	0	0	4	60	157	236	255	255	177	95	61	32	0	0	29
0	10	16	119	238	255	244	245	243	250	249	255	222	103	10	0
0	14	170	255	255	244	254	255	253	245	255	249	253	251	124	1
2	98	255	228	255	251	254	211	141	116	122	215	251	238	255	49
13	217	243	255	155	33	226	52	2	0	10	13	232	255	255	36
16	229	252	254	49	12	0	0	7	7	0	70	237	252	235	62
6	141	245	255	212	25	11	9	3	0	115	236	243	255	137	0
0	87	252	250	248	215	60	0	1	121	252	255	248	144	6	0
0	13	113	255	255	245	255	182	181	248	252	242	208	36	0	19
1	0	5	117	251	255	241	255	247	255	241	162	17	0	7	0
0	0	0	4	58	251	255	246	254	253	255	120	11	0	1	0
0	0	4	97	255	255	255	248	252	255	244	255	182	10	0	4
0	22	206	252	246	251	241	100	24	113	255	245	255	194	9	0
0	111	255	242	255	158	24	0	0	6	39	255	232	230	56	0
0	218	251	250	137	7	11	0	0	0	2	62	255	250	125	3
0	173	255	255	101	9	20	0	13	3	13	182	251	245	61	0
0	107	251	241	255	230	98	55	19	118	217	248	253	255	52	4
0	18	148	250	255	247	255	255	255	249	255	240	255	129	0	5
0	0	23	113	215	255	250	248	255	255	248	248	118	14	12	0
0	0	6	1	0	52	153	233	255	252	147	37	0	0	4	1
0	0	5	5	0	0	0	0	0	14	1	0	6	6	0	0

0	2	15	0	0	11	10	0	0	0	0	9	9	0	0	0
0	0	0	4	60	157	236	255	255	177	95	61	32	0	0	29
0	10	16	119	238	255	244	245	243	250	249	255	222	103	10	0
0	14	170	255	255	244	254	255	253	245	255	249	253	251	124	1
2	98	255	228	255	251	254	211	141	116	122	215	251	238	255	49
13	217	243	255	155	33	226	52	2	0	10	13	232	255	255	36
16	229	252	254	49	12	0	0	7	7	0	70	237	252	235	62
6	141	245	255	212	25	11	9	3	0	115	236	243	255	137	0
0	87	252	250	248	215	60	0	1	121	252	255	248	144	6	0
0	13	113	255	255	245	255	182	181	248	252	242	208	36	0	19
1	0	5	117	251	255	241	255	247	255	241	162	17	0	7	0
0	0	0	4	58	251	255	246	254	253	255	120	11	0	1	0
0	0	4	97	255	255	255	248	252	255	244	255	182	10	0	4
0	22	206	252	246	251	241	100	24	113	255	245	255	194	9	0
0	111	255	242	255	158	24	0	0	6	39	255	232	230	56	0
0	218	251	250	137	7	11	0	0	0	2	62	255	250	125	3
0	173	255	255	101	9	20	0	13	3	13	182	251	245	61	0
0	107	251	241	255	230	98	55	19	118	217	248	253	255	52	4
0	18	148	250	255	247	255	255	255	249	255	240	255	129	0	5
0	0	23	113	215	255	250	248	255	255	248	248	118	14	12	0
0	0	6	1	0	52	153	233	255	252	147	37	0	0	4	1
0	0	5	5	0	0	0	0	0	14	1	0	6	6	0	0

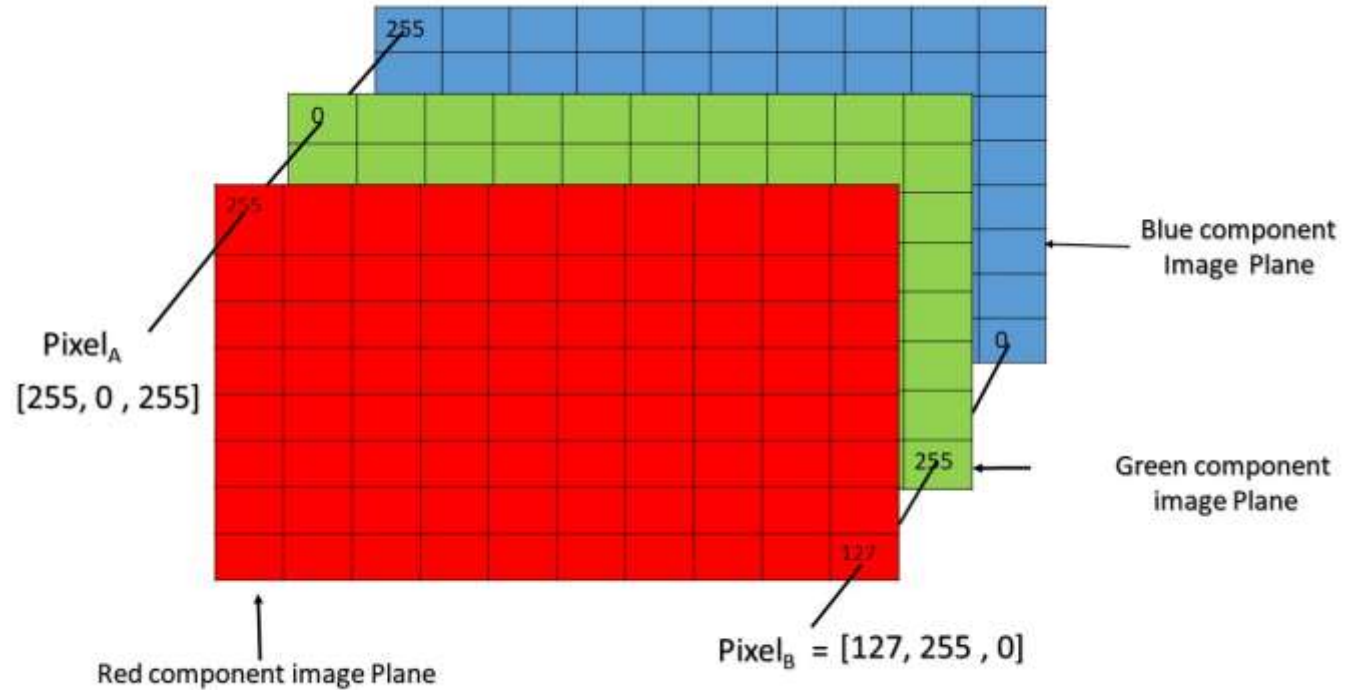
For color, we need another dimension

- Three dimensional array:

- Height
- Width
- Color channel

- (H x W x C)

- Red Green Blue (RGB)
color space



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

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]] ← Pixel (Each Row of 3 Numbers)
 ← Columns (Image Width)
 ← Row (Image Height)

[[[ 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]]

```



Colors are represented with combinations of red, green, and blue (RGB).

[0 0 0] is



[50 58 57] is



[255 255 255] is



[47 71 63] is



[255 0 0] is



[37 74 124] is



[0 255 0] is



[179 168 160] is



[0 0 255] is



[127 127 127] is



Note: RGB over integers can represent a finite number of colors, which also means it can represent a finite number of images of a given dimension

Images usually must be transformed for model input.

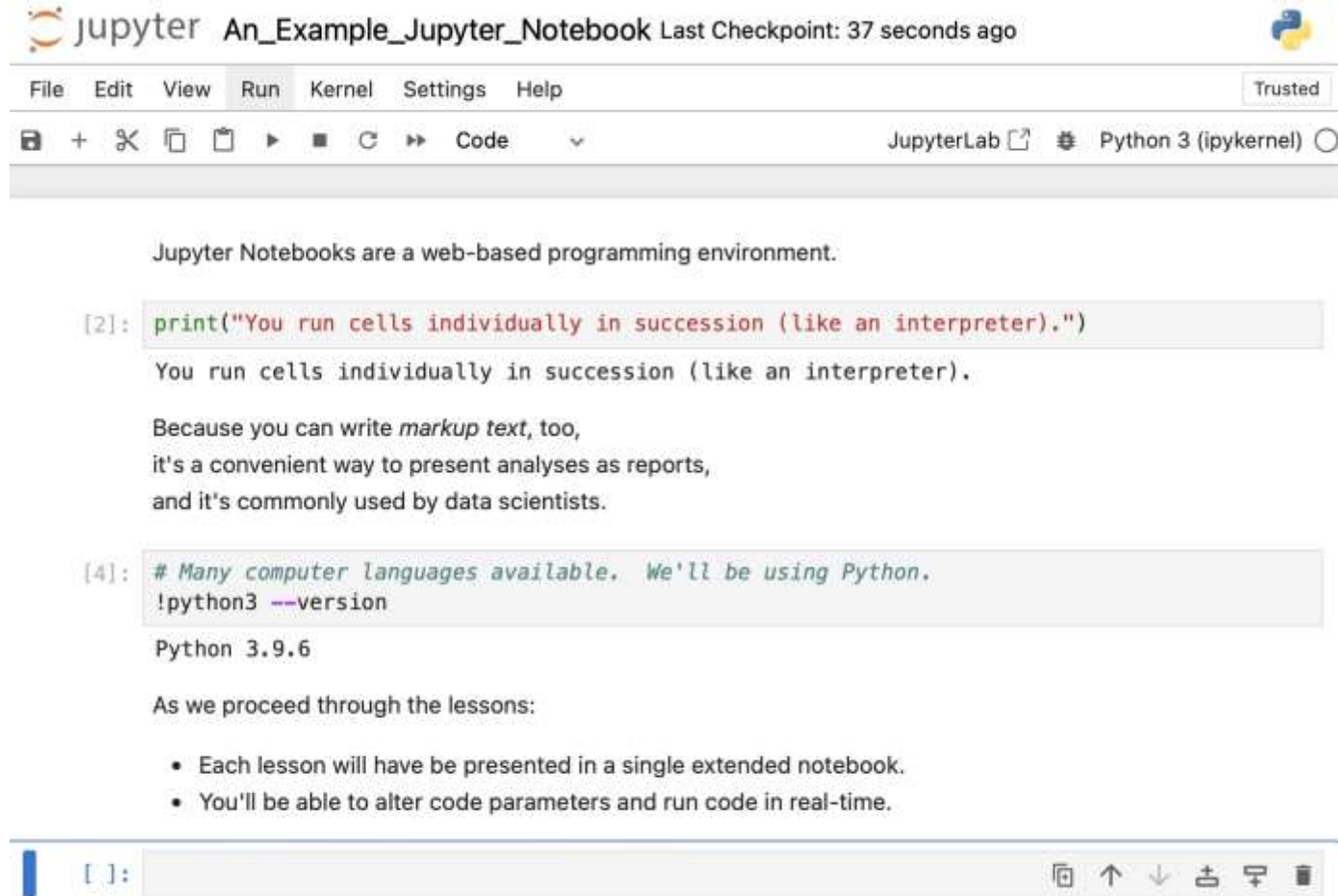
- Images may not be captured in the format necessary for input into the computer vision model
- If this happens, it's necessary to transform the image:
 - Scaling: changing the size of the image
 - Cropping: cutting out part of the image
 - Letterboxing: reduce image but keep aspect ratio
 - Flipping: 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 1

Computer Images and Processing

Lab Set Up

We'll use Jupyter Notebooks to run the software.



The screenshot shows a Jupyter Notebook interface. At the top, the title bar reads "jupyter An_Example_Jupyter_Notebook Last Checkpoint: 37 seconds ago". Below the title bar is a menu bar with "File", "Edit", "View", "Run", "Kernel", "Settings", and "Help". A "Trusted" badge is visible on the right side of the menu bar. Below the menu bar is a toolbar with various icons for file operations and execution. The main content area shows two code cells. The first cell contains the code `print("You run cells individually in succession (like an interpreter).")` and its output is the same string. The second cell contains the code `# Many computer languages available. We'll be using Python.
!python3 --version` and its output is `Python 3.9.6`. Below the second cell, there is a list of bullet points: "Each lesson will have be presented in a single extended notebook." and "You'll be able to alter code parameters and run code in real-time." At the bottom of the notebook, there is a prompt `[]:` and a toolbar with icons for copy, paste, and other actions.

jupyter An_Example_Jupyter_Notebook Last Checkpoint: 37 seconds ago

File Edit View Run Kernel Settings Help Trusted

Code Python 3 (ipykernel)

Jupyter Notebooks are a web-based programming environment.

```
[2]: print("You run cells individually in succession (like an interpreter).")
```

You run cells individually in succession (like an interpreter).

Because you can write *markup text*, too,
it's a convenient way to present analyses as reports,
and it's commonly used by data scientists.

```
[4]: # Many computer languages available. We'll be using Python.  
!python3 --version
```

Python 3.9.6

As we proceed through the lessons:

- Each lesson will have be presented in a single extended notebook.
- You'll be able to alter code parameters and run code in real-time.

[]:

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.



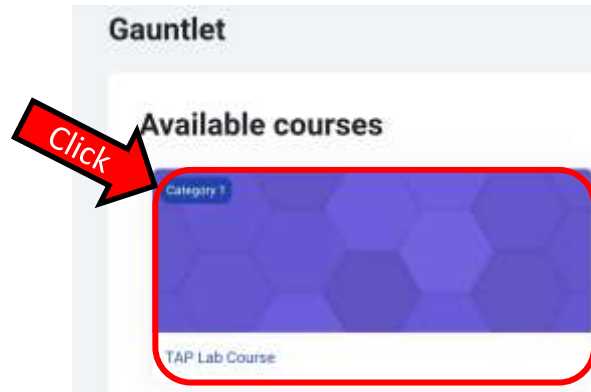
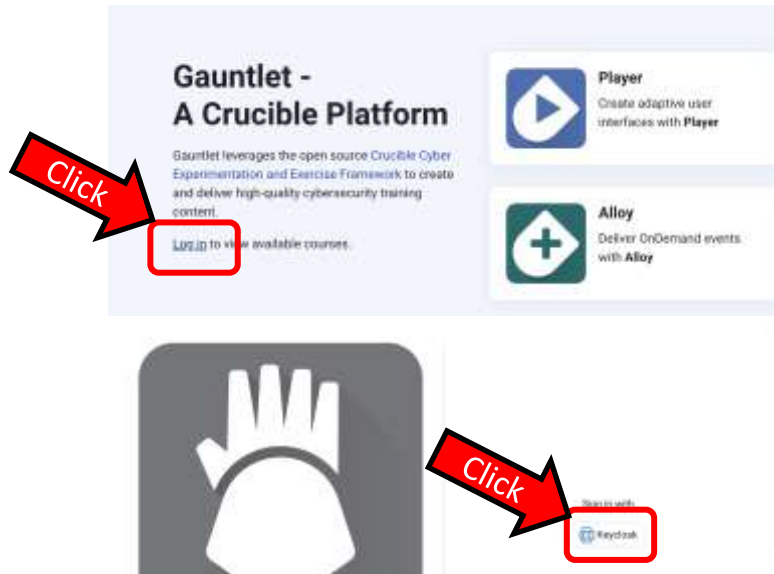
Open Neural Network Exchange supports the creation of deep learning models and supports converting models between PyTorch and TensorFlow, popular neural network packages.



Open Computer Vision supports image processing.

Setup:

- Go to: <https://lms.gauntlet.sei.cmu.edu/> and sign in with a student account
- Open "TAP Lab Course"
- Select "Computer Vision" then open the lab



Gauntlet/VM Setup

- Gauntlet
- CVIntro Repo (github)
- Virtual environment venv, requirements.txt
- **First duplicate the Jupyter Exercise Notebook in case something goes wrong**

Debrief

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

Exercise 1: Transforming Images for Reference

- Images as arrays of integers
- Image manipulation as transformations of these arrays
- Resizing, scaling, letterboxing images for model input specifications

Debrief

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

Lesson 3

Neural Networks for Computer Vision

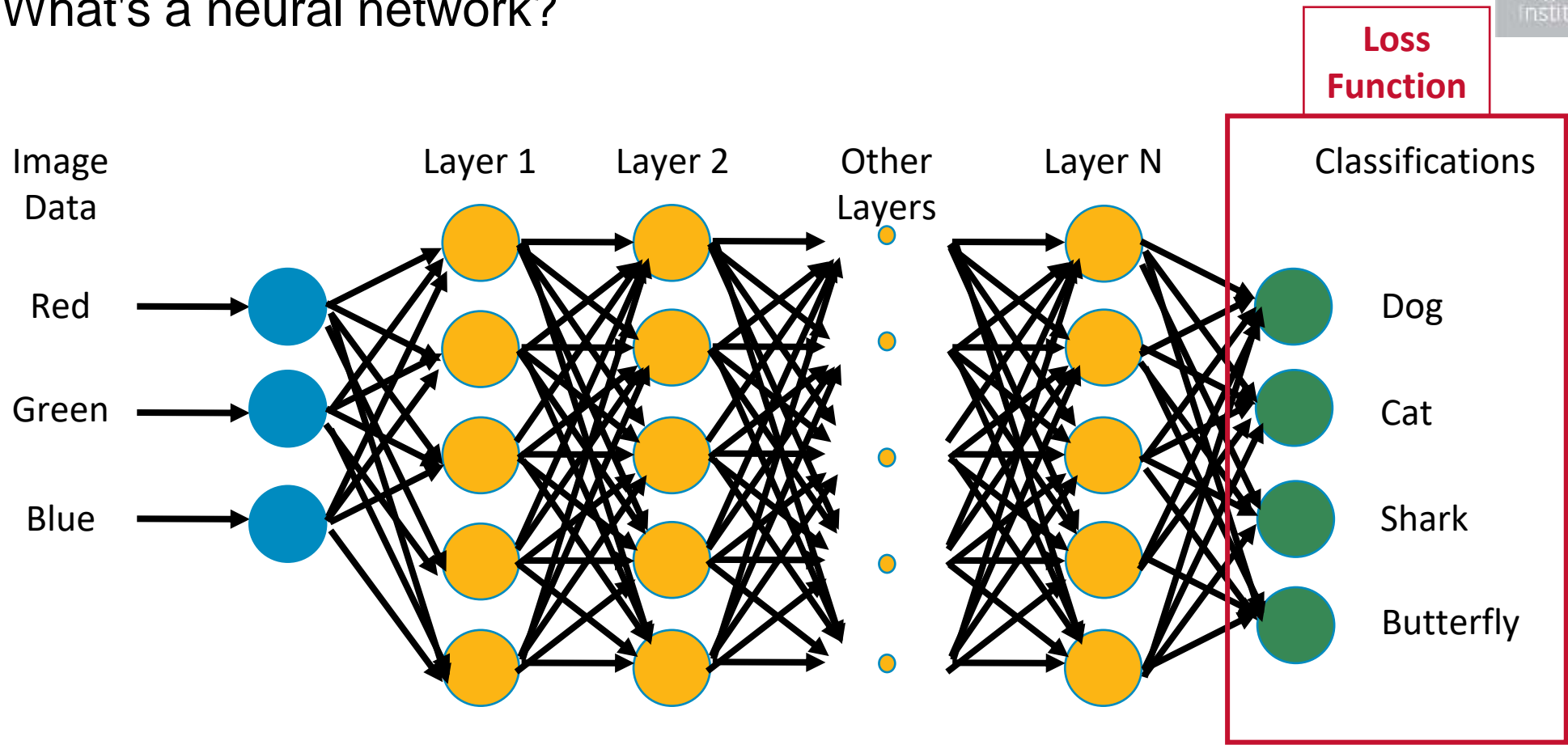
Lesson 3: Learning Objectives

- 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.

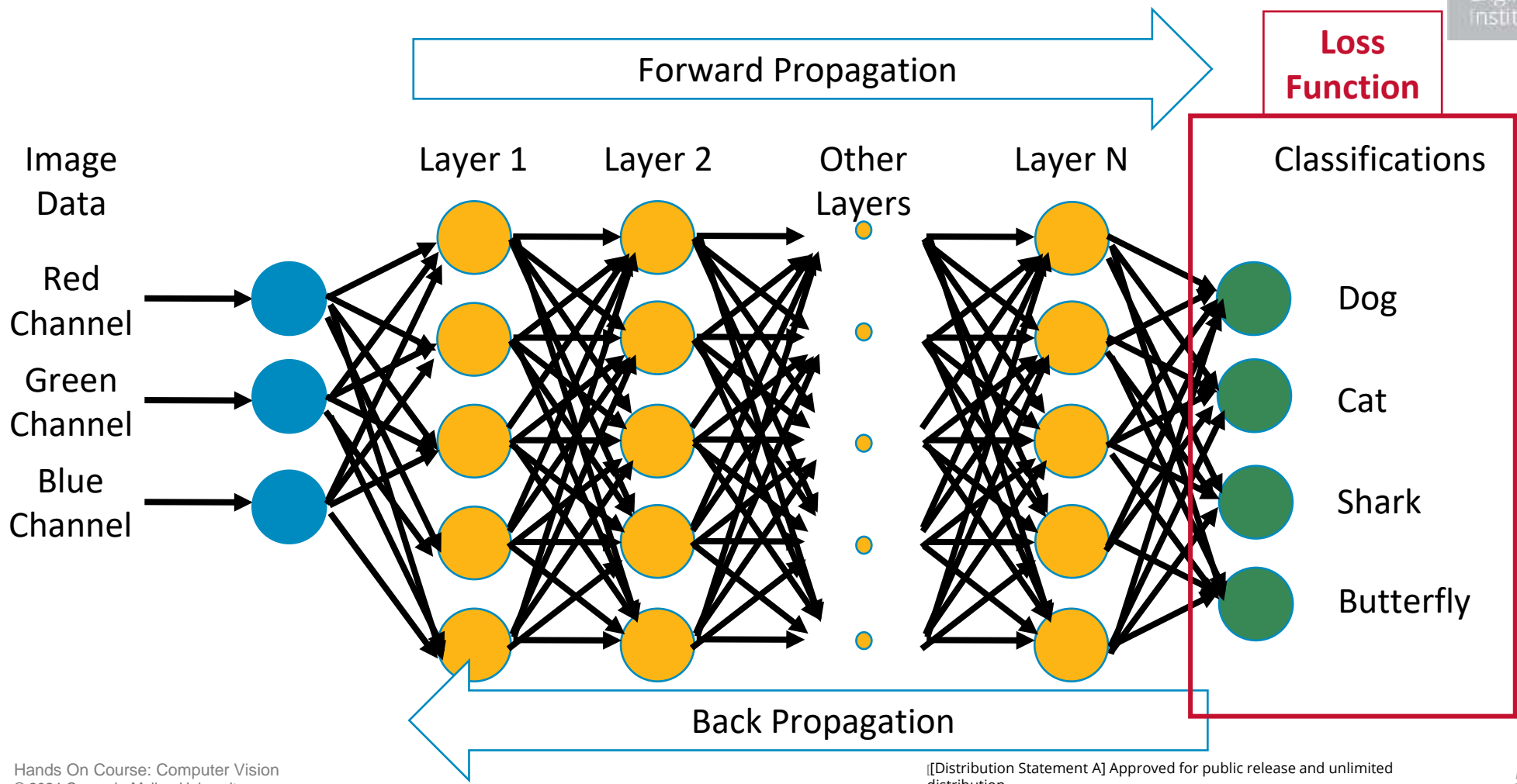
Lecture 3

Neural Networks for Computer Vision

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

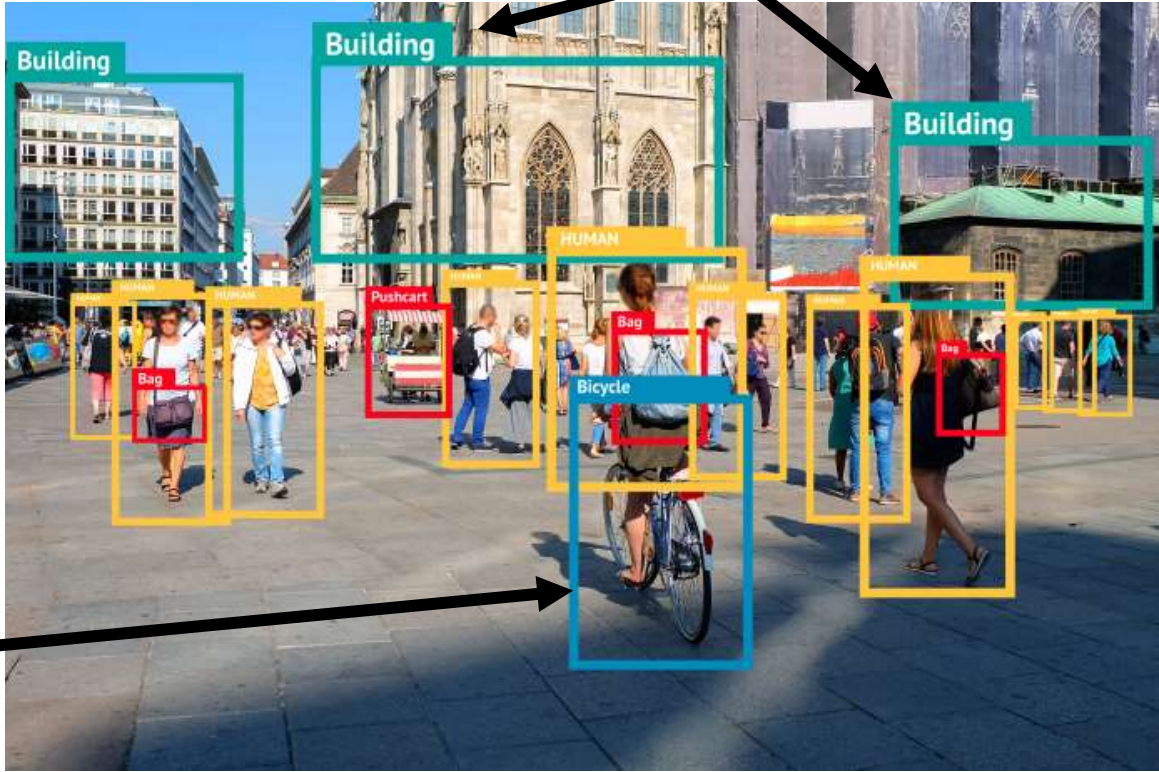


Neural networks are trained using forward and back propagation.



Neural networks can do different computer vision tasks

2. Object Classification/Identification



3. Object Segmentation



1. Object Detection

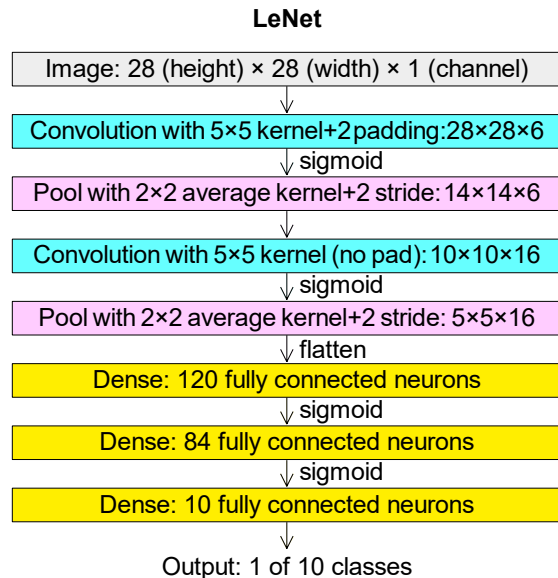
Deep Learning Revolutionized Machine Learning



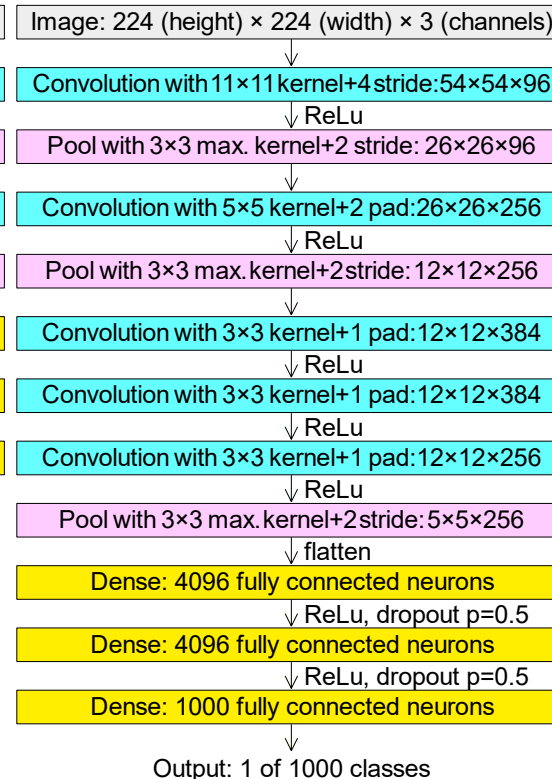
Deep learning revolutionized computer vision. Convolutional neural networks are especially important.

LeNet

- Created by LeCun in 1998
- Early convolutional network



AlexNet



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>

The layers have different functions.

- Convolutional Layer: Selects/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

Convolution basics

- A convolution is a way to transform an image array
- Convolutions for image processing pre-date their use for computer vision/machine learning
- Lots of common image effects (blurring, sharpening, etc.) are achieved using a convolution filter
- A convolution of an image array is specified by three things
 - The kernel
 - The stride length
 - The padding

During convolution a kernel slides across the image to make the feature data.

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			Kernel		Output	
0	1	2	0	1	19	25
3	4	5	2	3	37	43
6	7	8				

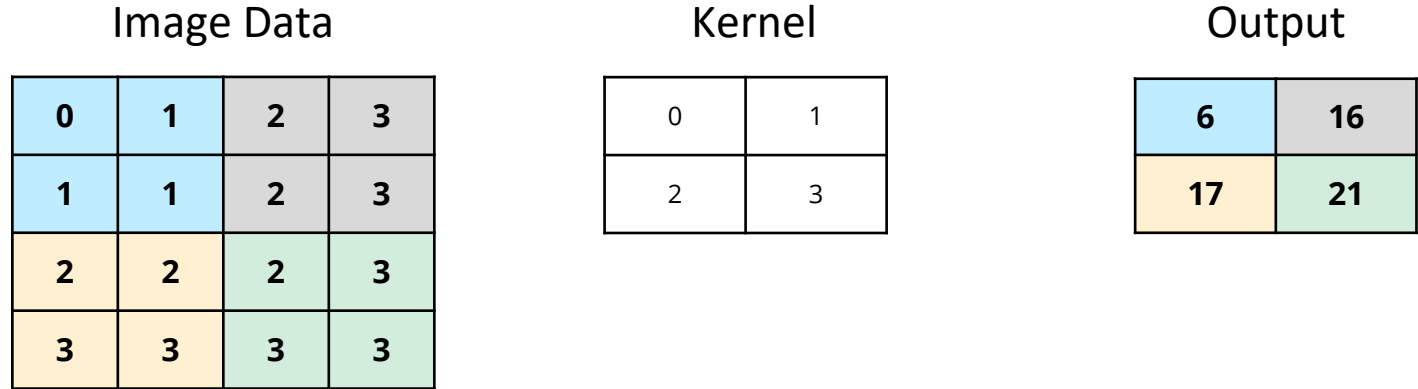
For example: $1 * 0 + 2 * 1 + 4 * 2 + 5 * 3 = 0 + 2 + 8 + 15 = 25$ (the top right output)

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.



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.

Different kernels highlight/capture different information in the underlying image

Identity Kernel

0	0	0
0	1	0
0	0	0



Simple Blur

1/9

1	1	1
1	1	1
1	1	1



Image Sharpening

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

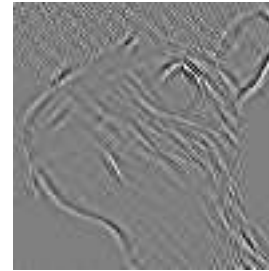


Kernels (continued)

- Edge detection kernel



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



These are examples of handcrafted kernels for achieving specific desired effects
In contrast, Convolutional Neural Nets are learning their own kernels which may or may not be interpretable

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

Image Data

0	1	2
3	4	5
6	7	8

Max Pooling
2 x 2 Matrix

Output

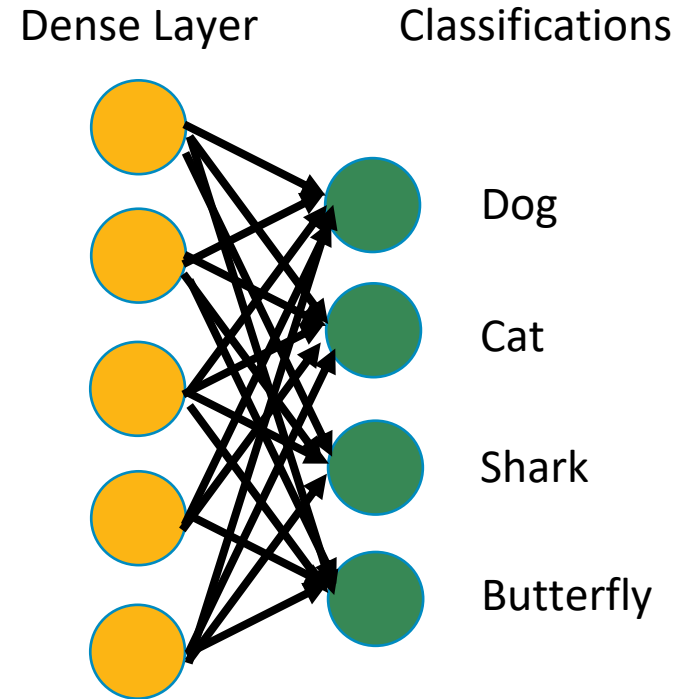
4	5
7	8

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.

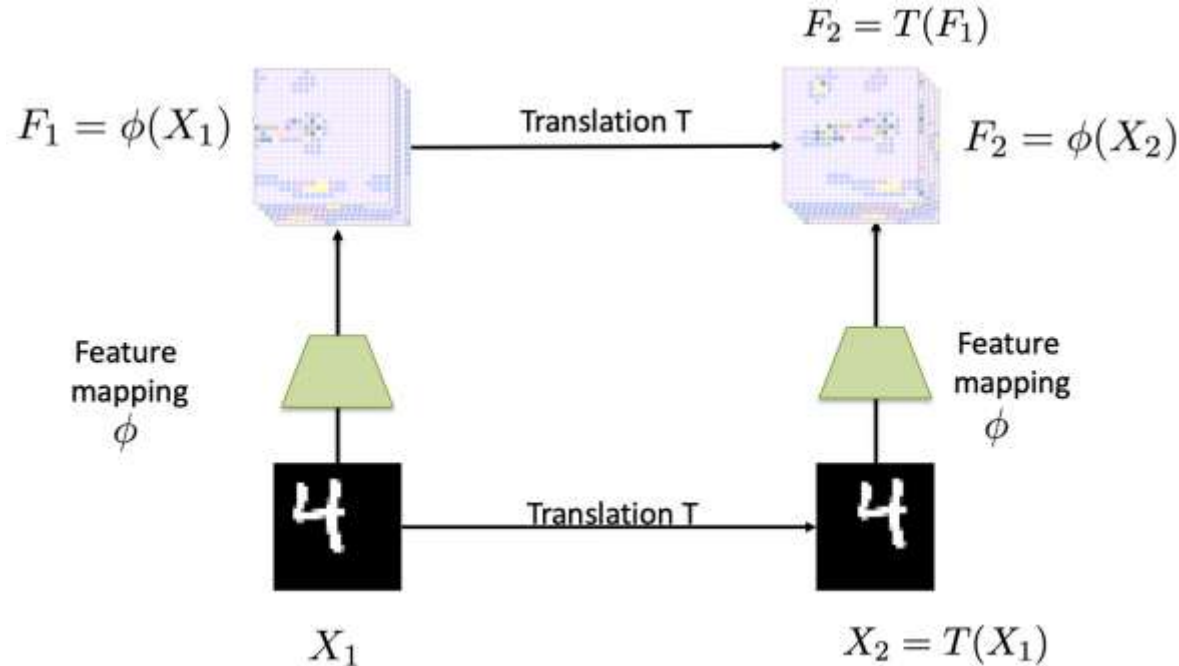
Dense (fully connected) layers have all input and output nodes connected

- 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.

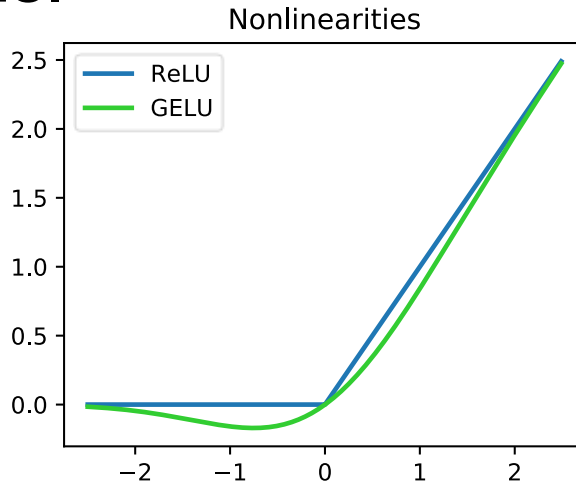


Why convolutions?

- Translation equivariant
- More efficient than fully connected or transformers architecture



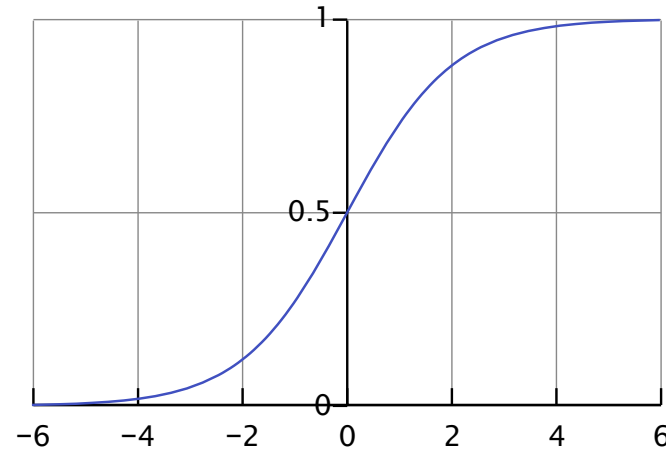
Node values (weights) are adjusted by activation functions.



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.

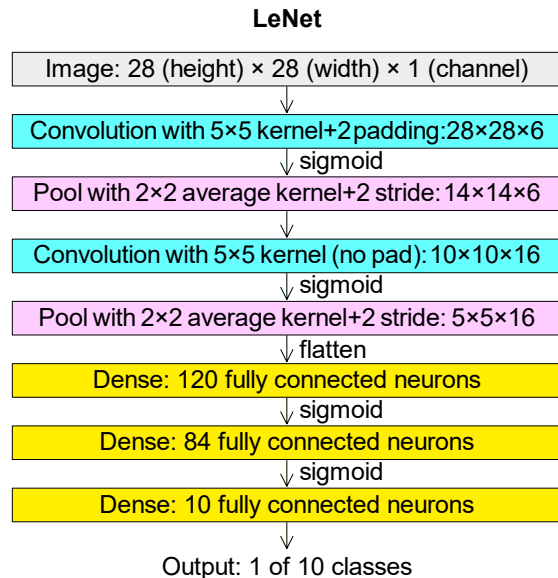
Still used but generally replaced by ReLU (easier to train)

You have all the concepts for these architectures.

The layer size changes are due to the kernel, stride, and padding.

LeNet

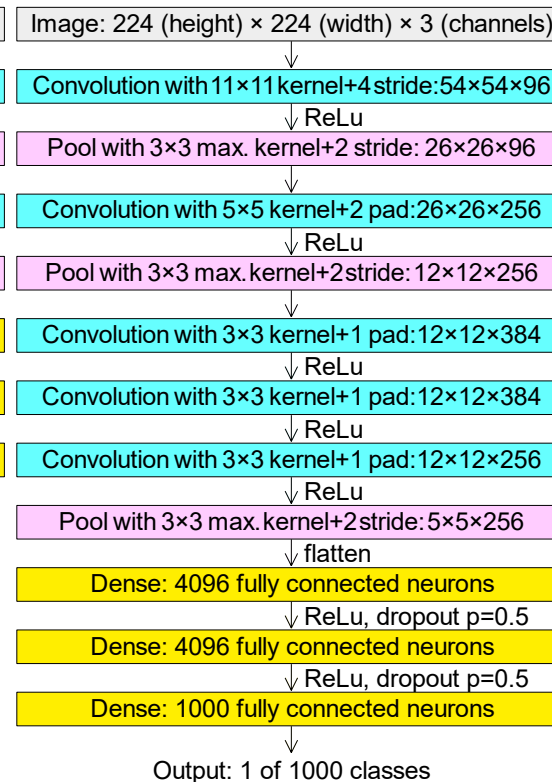
- Created by LeCun in 1998
- Early convolutional network



AlexNet


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.

ONNX Model Zoo Search models... 

Task

Computer Vision Generative AI

Graph Machine Learning

Natural Language Processing

Not Available

Author

timm torch_hub torchvision

transformers graph_convolution

Unknown

Opset

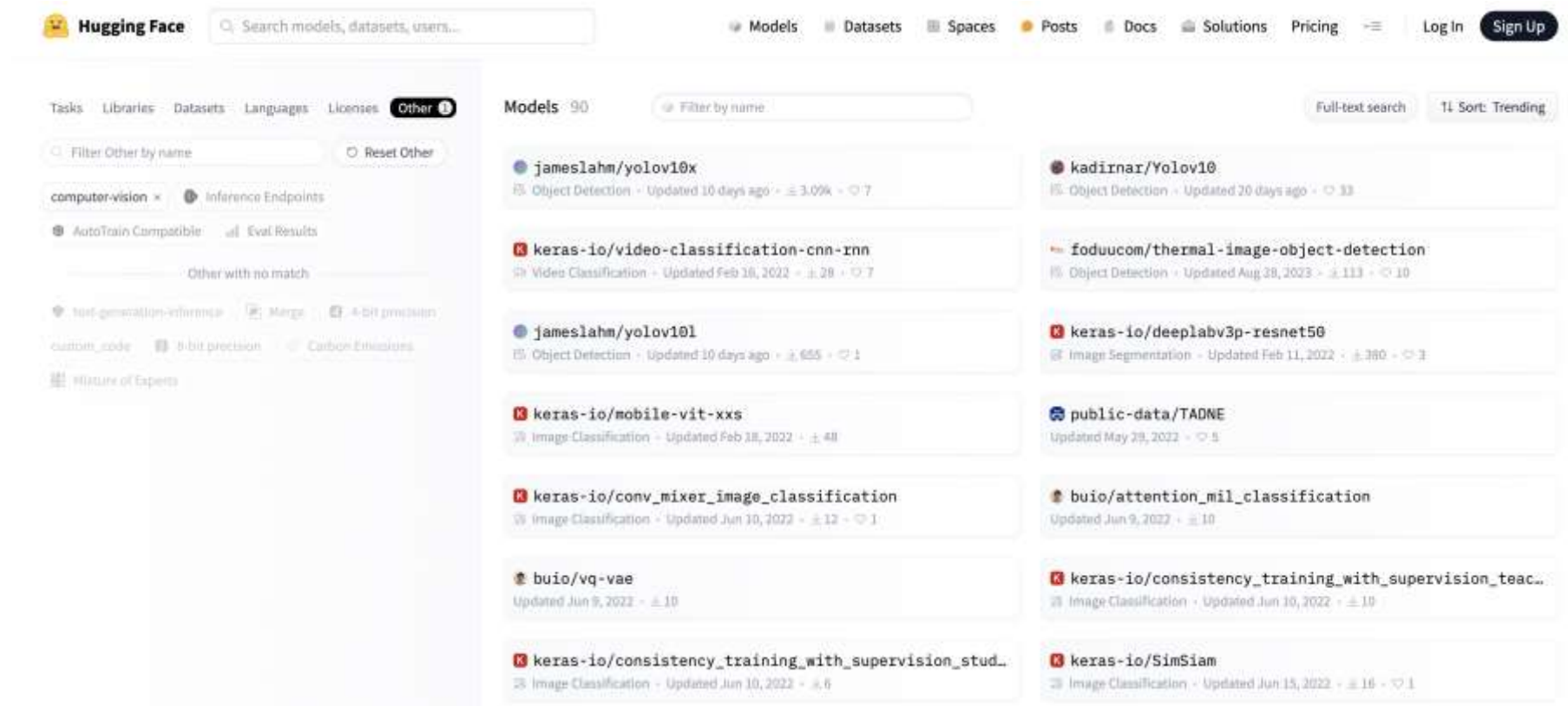
16 17 18 Not Available

adv_inception_v3 Task: Computer Vision Author: timm Opset: 16	adv_inception_v3 Task: Computer Vision Author: timm Opset: 17	adv_inception_v3 Task: Computer Vision Author: timm Opset: 18	alexnet Task: Computer Vision Author: torch_hub Opset: 16
alexnet Task: Computer Vision Author: torch_hub Opset: 17	bat_resnext26ts Task: Computer Vision Author: timm Opset: 16	bat_resnext26ts Task: Computer Vision Author: timm Opset: 17	bat_resnext26ts Task: Computer Vision Author: timm Opset: 18
beit_base_patch16_384 Task: Computer Vision Author: timm Opset: 17	beit_large_patch16_224 Task: Computer Vision Author: timm Opset: 17	botnet26t_256 Task: Computer Vision Author: timm Opset: 16	botnet26t_256 Task: Computer Vision Author: timm Opset: 17
botnet26t_256 Task: Computer Vision Author: timm Opset: 18	cait_m48_448 Task: Computer Vision Author: timm Opset: 16	cait_m48_448 Task: Computer Vision Author: timm Opset: 17	cait_m48_448 Task: Computer Vision Author: timm Opset: 18
cait_s24_224 Task: Computer Vision Author: timm Opset: 16	cait_s24_224 Task: Computer Vision Author: timm Opset: 18	cait_s24_384 Task: Computer Vision Author: timm Opset: 16	cait_s36_384 Task: Computer Vision Author: timm Opset: 18

Prev 1/86 Next

There are many places to find models.

For example, Huggingface also provides models.

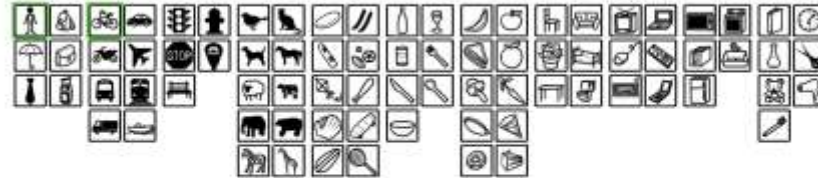


The characteristics of the dataset are important, too. Common Objects in Context (COCO) will be our dataset.

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

COCO Explorer

COCO 2017 Train/val browser (123,287 images, 886,284 instances). Crowd labels not shown.



bicycle person

search

2765 results



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 3

Convolution and Pretrained Neural Networks

Links

- ONNX Model Zoo

- <https://github.com/onnx/models>

- Netron Model viewer

- <https://netron.app/>

Exercise 2: Convolution and Pretrained Neural Networks

- Illustration of convolution with different types of kernels
- Run inference on an image with object detection model, show raw output of YOLO
- Motivates discussion of converting outputs to visual annotations

Debrief

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

Lesson 4

Object Detection

Lesson 4: 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.

Lesson 4

Object Detection

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

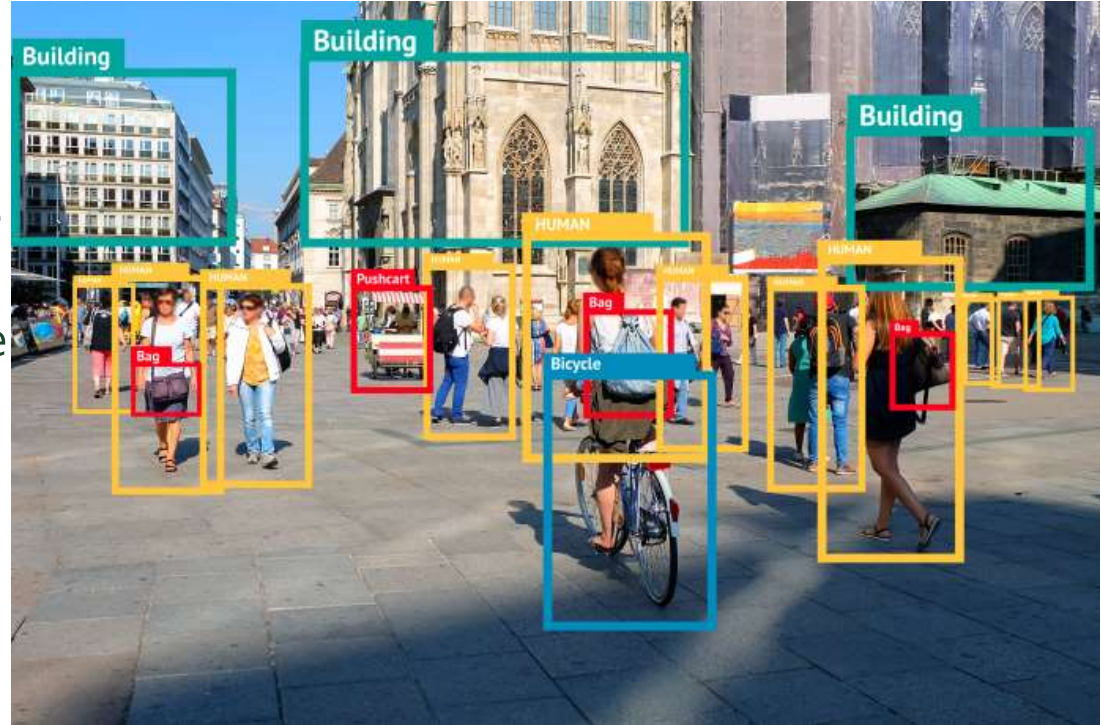
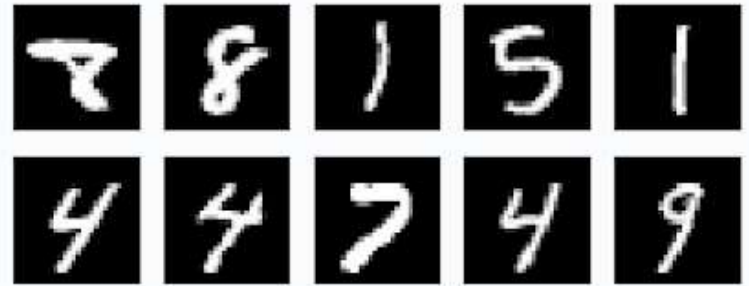
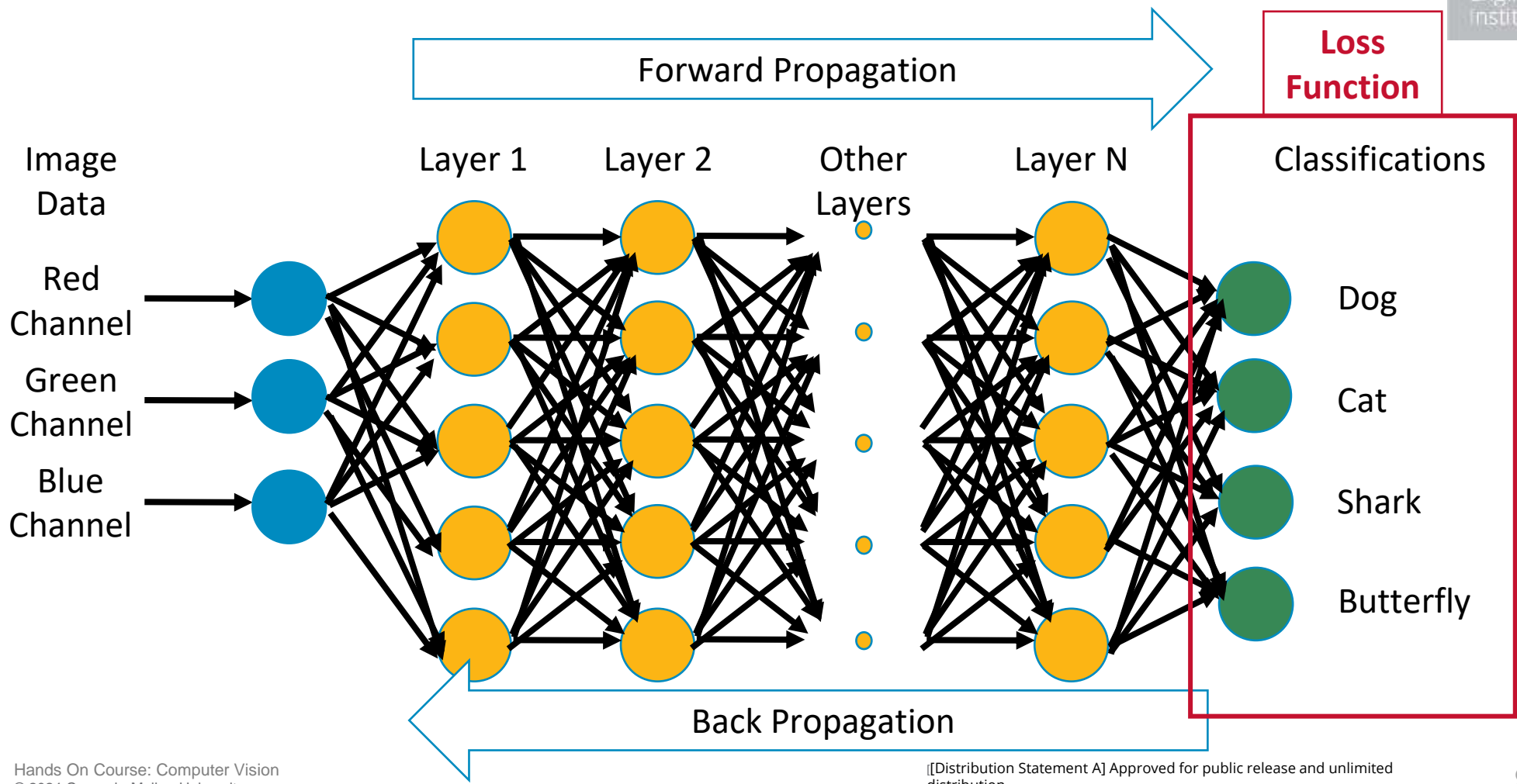


Image Classification

- A classic problem in CV is classifying digits using the MNIST dataset
- **Input x :** 28x28x1 (greyscale) image
- **Output \hat{y} :** a 1D vector of 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

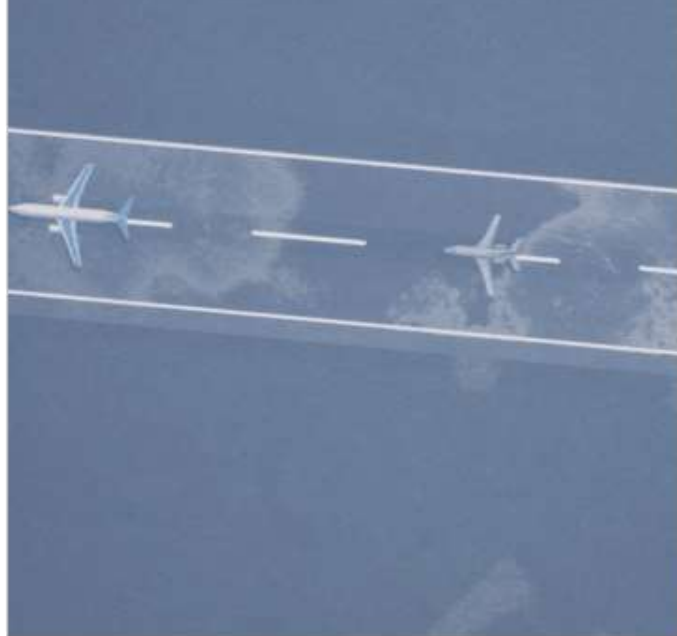
- COCO is a much larger and diverse dataset of images than MNIST
- <https://cocodataset.org/#explore>
- There are 80 types of labeled objects so our object classification model will have to output 80 class probabilities



RarePlanes

- “largest openly-available very high resolution dataset built to test the value of synthetic data from an overhead perspective”
- Real data
 - 253 Maxar WorldView-3 satellite scenes spanning 112 locations and 2,142 km² with 14,700 hand-annotated aircraft
- Synthetic data
 - 50,000 synthetic satellite images with ~630,000 aircraft annotations.

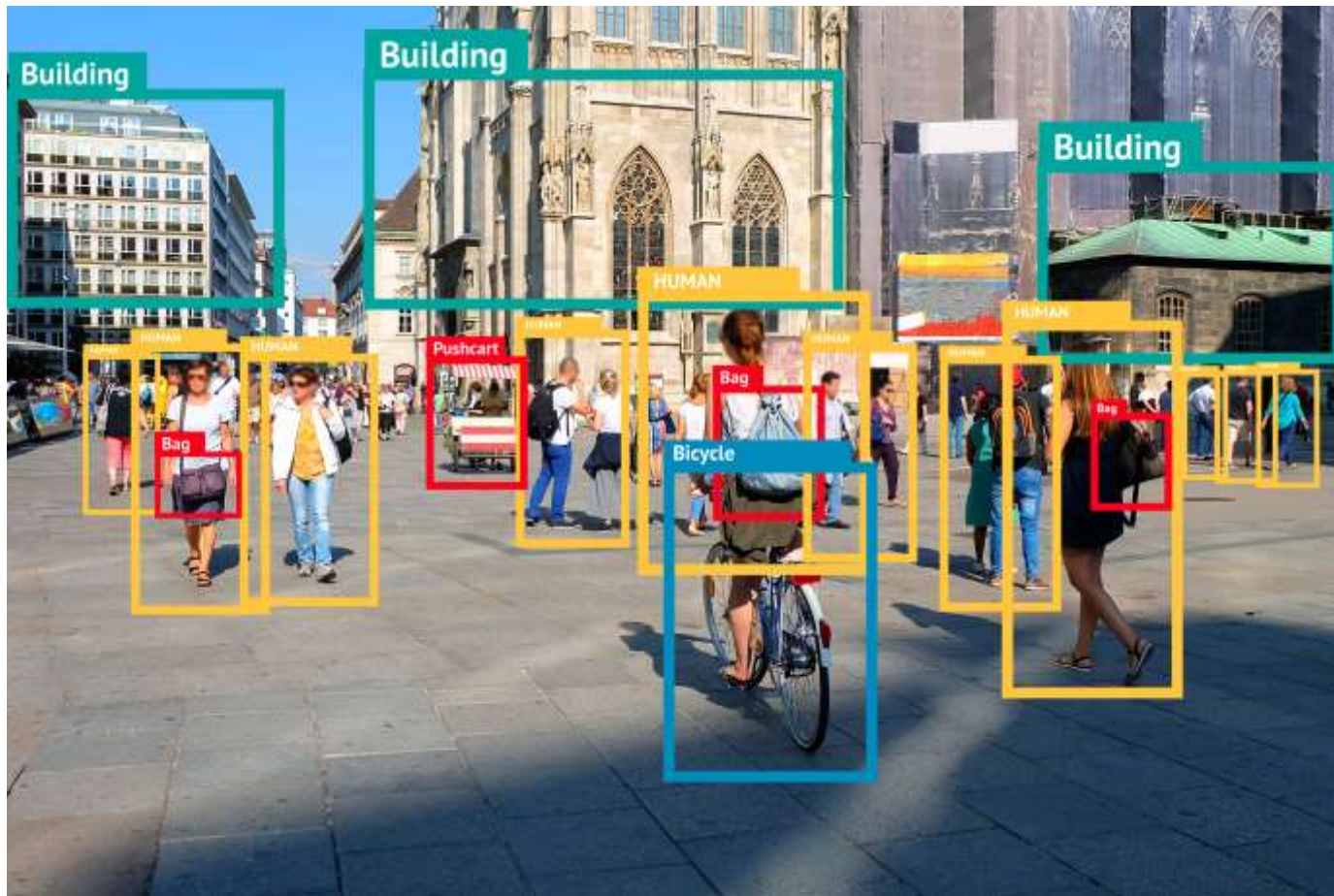




“airplane” is a category in COCO dataset



Localization – How?

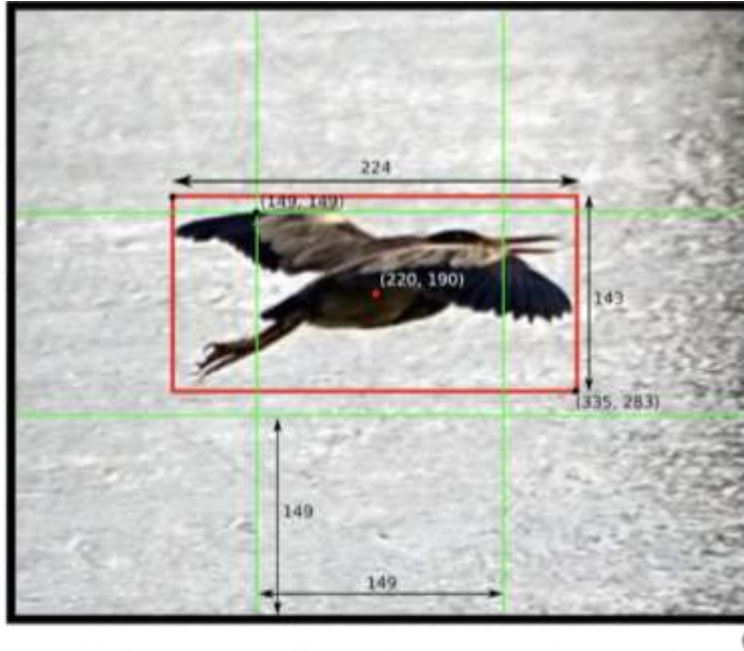


In object detection, bounding boxes identify object locations

- 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 $S \times S$ grid of cells for object detection (here $S=3$)



$$x = (220 - 149) / 149 = 0.48$$

$$y = (190 - 149) / 149 = 0.28$$

$$w = 224 / 448 = 0.50$$

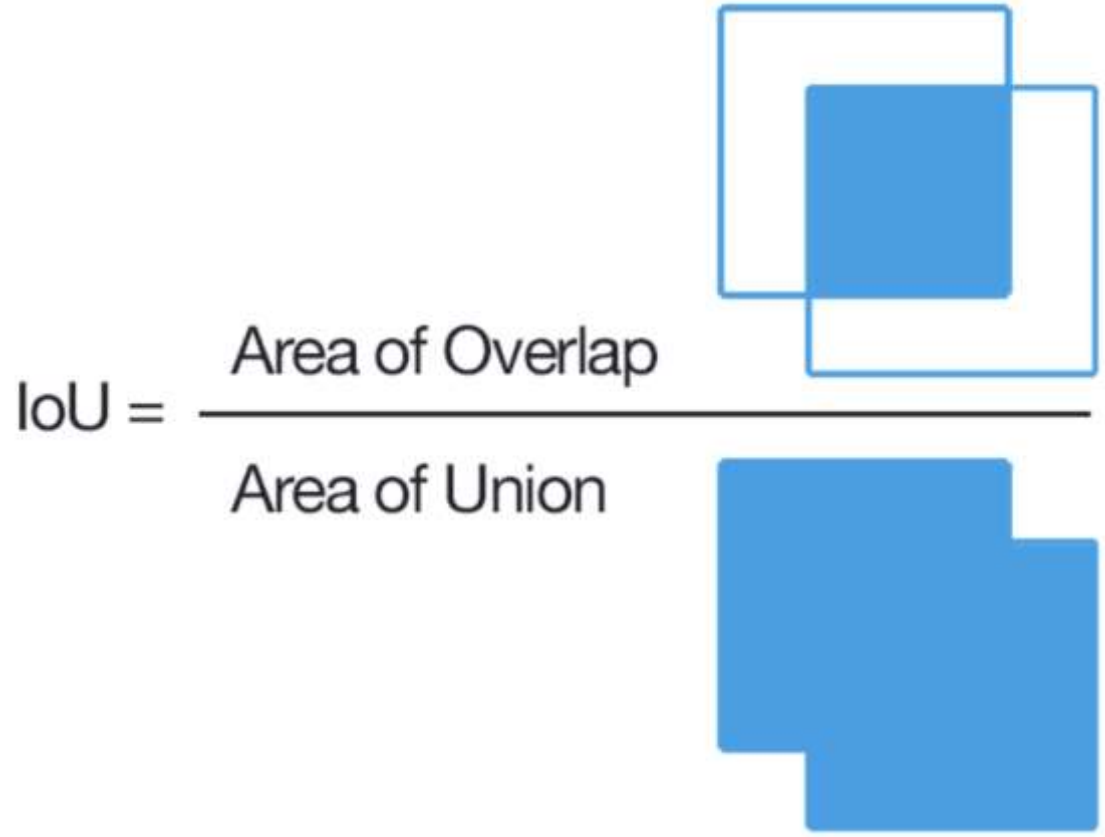
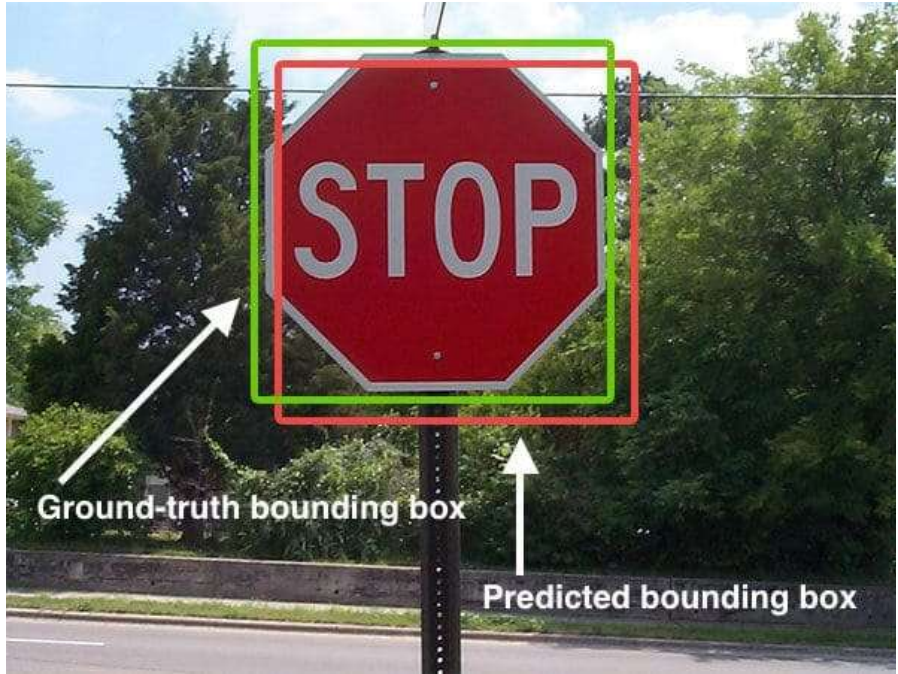
$$h = 143 / 448 = 0.32$$

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

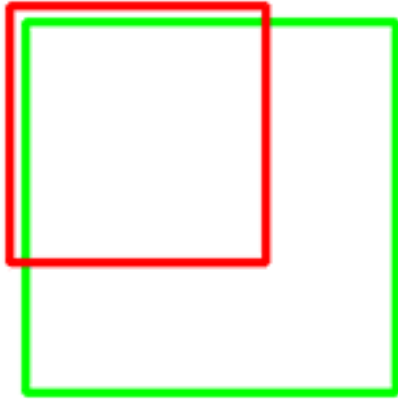
- Each of our $S \times S$ grid cells predicts B bounding boxes
- Each bounding box prediction has 5 components:
 - x
 - y
 - w
 - h
 - *confidence/objectness score* (we'll talk more about this later – for now think of it as the confidence our prediction)

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



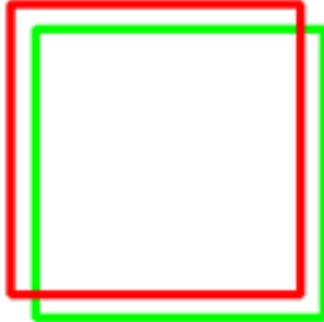
Intersection over Union (cont.)

IoU: 0.4034



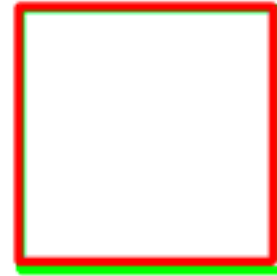
Poor

IoU: 0.7330



Good

IoU: 0.9264



Excellent

Exercise 3

Bounding Boxes

Hands-on Exercise 3: Bounding boxes

- Introduce initial code for handling Yolo output and drawing annotations on image
- Motivate need for NMS and objectness threshold

Debrief

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

Lesson 5

You Only Look Once (YOLO) Details

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 5

YOLO Input and Output

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.

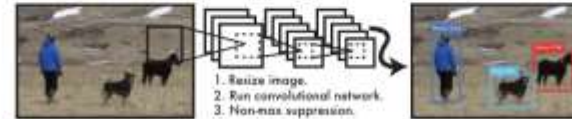


Figure 1: The YOLO Detection System. Processing images with YOLO is simple and straightforward. Our system (1) resizes the input image to 448×448 , (2) runs a single convolutional network on the image, and (3) thresholds the resulting detections by the model's confidence.

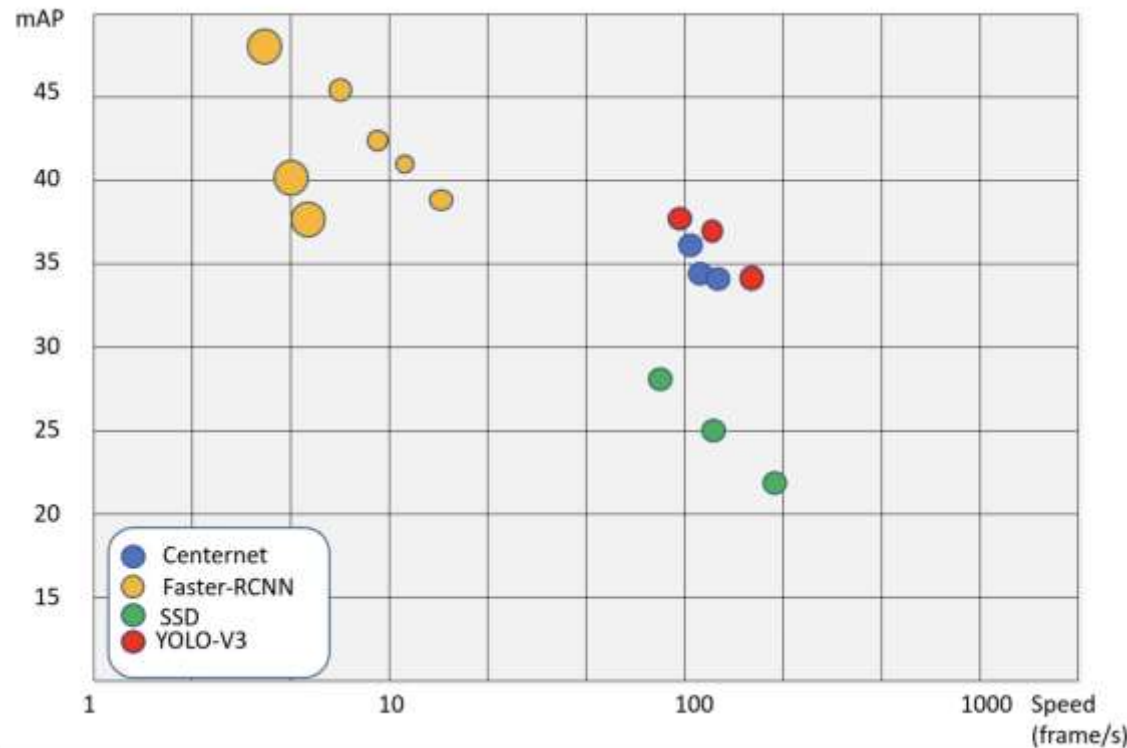
methods to first generate potential bounding boxes in an image and then run a classifier on these proposed boxes. After classification, post-processing is used to refine the bounding boxes, eliminate duplicate detections, and rescore the boxes based on other objects in the scene [13]. These complex pipelines 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

.02640v5 [cs.CV] 9 May 2016

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



my LITTLE RESUME

JOSEPH REDMON
University of Washington - CSE
Box 352350
Seattle WA 98195-2350
jredmie@uw.edu

This little pony went to Middlebury College, in Middlebury Vermont. He graduated with highest honors and even received a departmental award for academic excellence! While at Middlebury he cultivated an ongoings love for...

COMPUTER SCIENCE & MATHEMATICS!

For two years he worked as a tutor for the Computer Science department. He loves passing on his knowledge and setting everypony excited about computer science!

One summer he worked for the National Institute of Standards and Technology. He developed an online tool for analyzing thermal neutron triple-axis spectrometry data. He even had his very own nuclear reactor to play with!

The next year he had an Extreme Blue Internship with IBM. He worked with a team of interns at the Almaden Research Center to develop technology related to online shopping. But he's not allowed to talk about it too much! (He signed a non-disclosure)

After graduation he galloped off to Unalaska, Alaska to work as a radio DJ, and freelance web developer. He produces stunning, high quality websites for all the nice ponies hanging out in the Alaskan Bush.

For a few months he dabbled in domestication, working for a startup in San Francisco called ZeroCater. He spearheaded major projects, infrastructure upgrades and code cleaning frenzies while ensuring that thousands of hardworking ponies got quality, catered lunches of grains, oats, and grasses every day! The daily plough just wasn't for him though, so he threw off the bit and bridle and galloped back to Alaska. He still does contract work for ZeroCater on other companies on occasion, and in the fall he'll be heading off to a computer science Ph.D. program at the University of Washington!

PONY STATS

Education
School: Middlebury College '12
Major: Computer Science
Minor: Mathematics
GPA: 3.74
Major GPA: 3.98

Favorites
Languages: C, Python
Editor: Vim
Subjects: Machine Learning, Computer Vision, Compilers

Awards/Achievements
Timothy T. Ruess Award for Academic Achievement
3rd Place at ACM-ICPC BOSPRE 2 years in a row, 5th at NE North American Regionals in 2010
Top 10% in 4 separate Kaggle competitions, top 3% of active users

The adventure is far from over... What will this little pony do next, who knows??

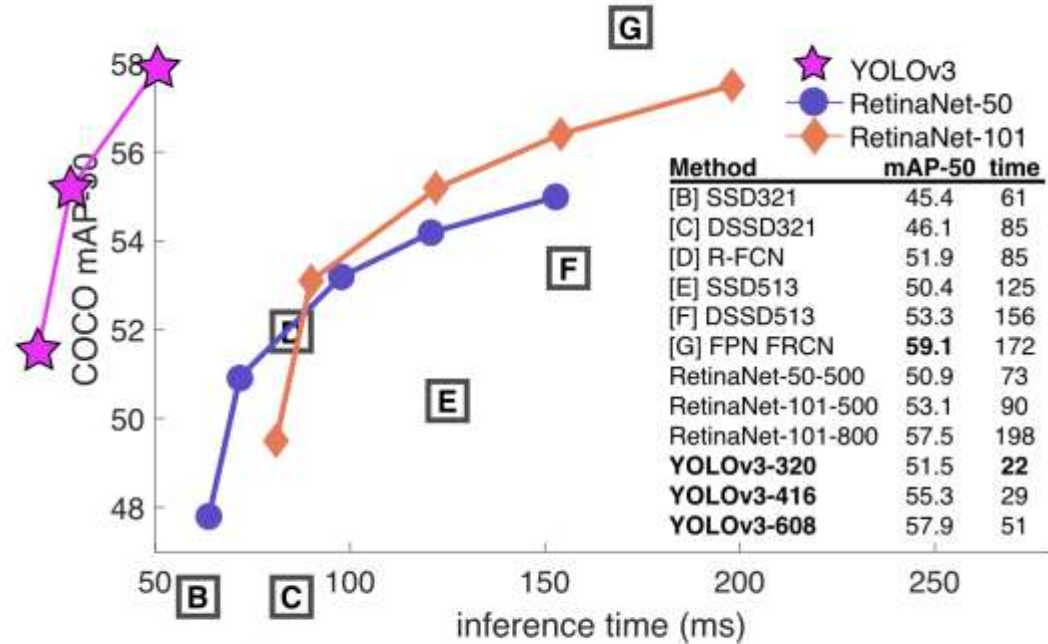


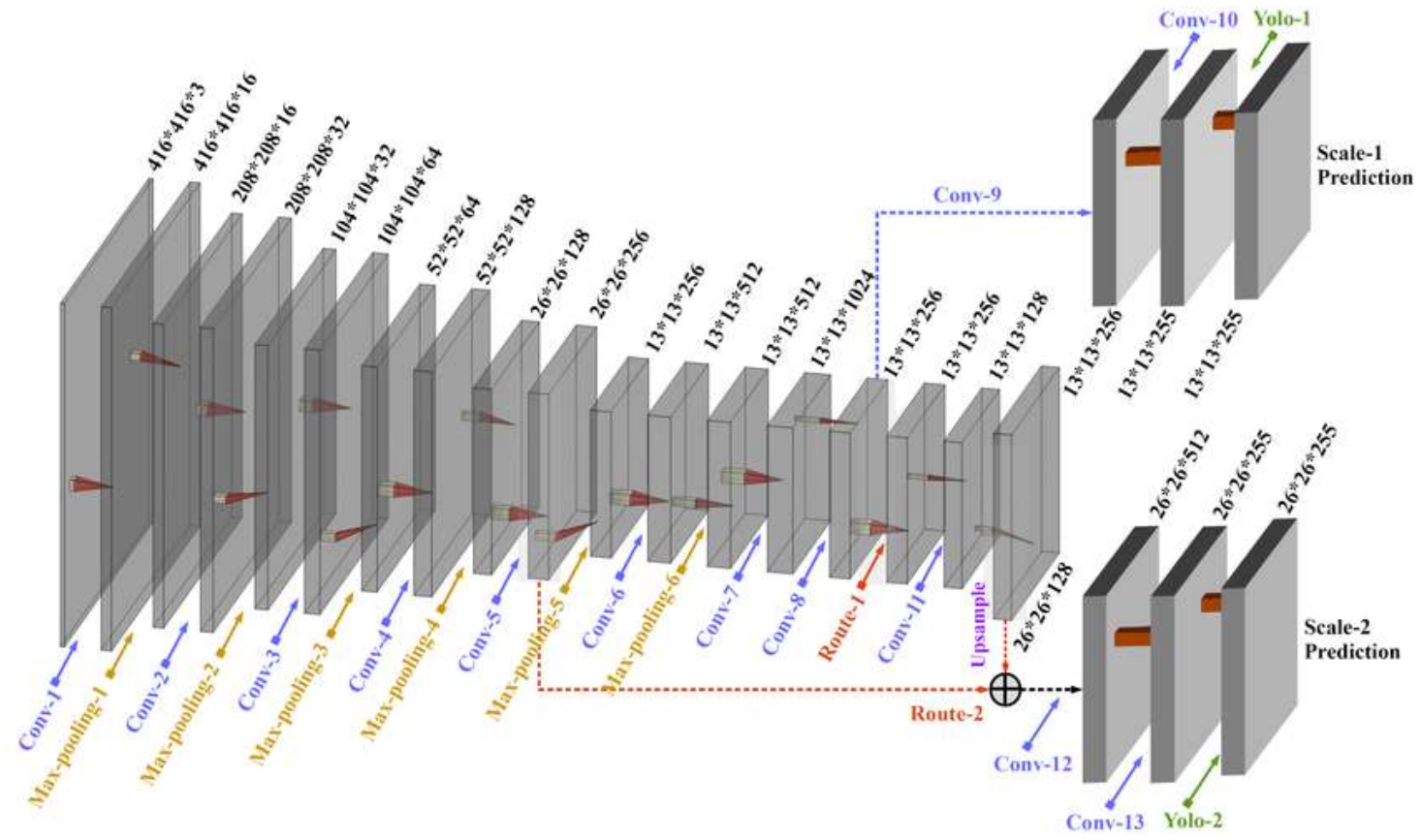
Figure 3. Again adapted from the [1], this time displaying speed/accuracy tradeoff on the mAP at .5 IOU metric. You can tell YOLOv3 is good because it's very high and far to the left. Can you cite your own paper? Guess who's going to try, this guy → [16]. Oh, I forgot, we also fix a data loading bug 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.

ited

YOLOv3 Tiny Architecture



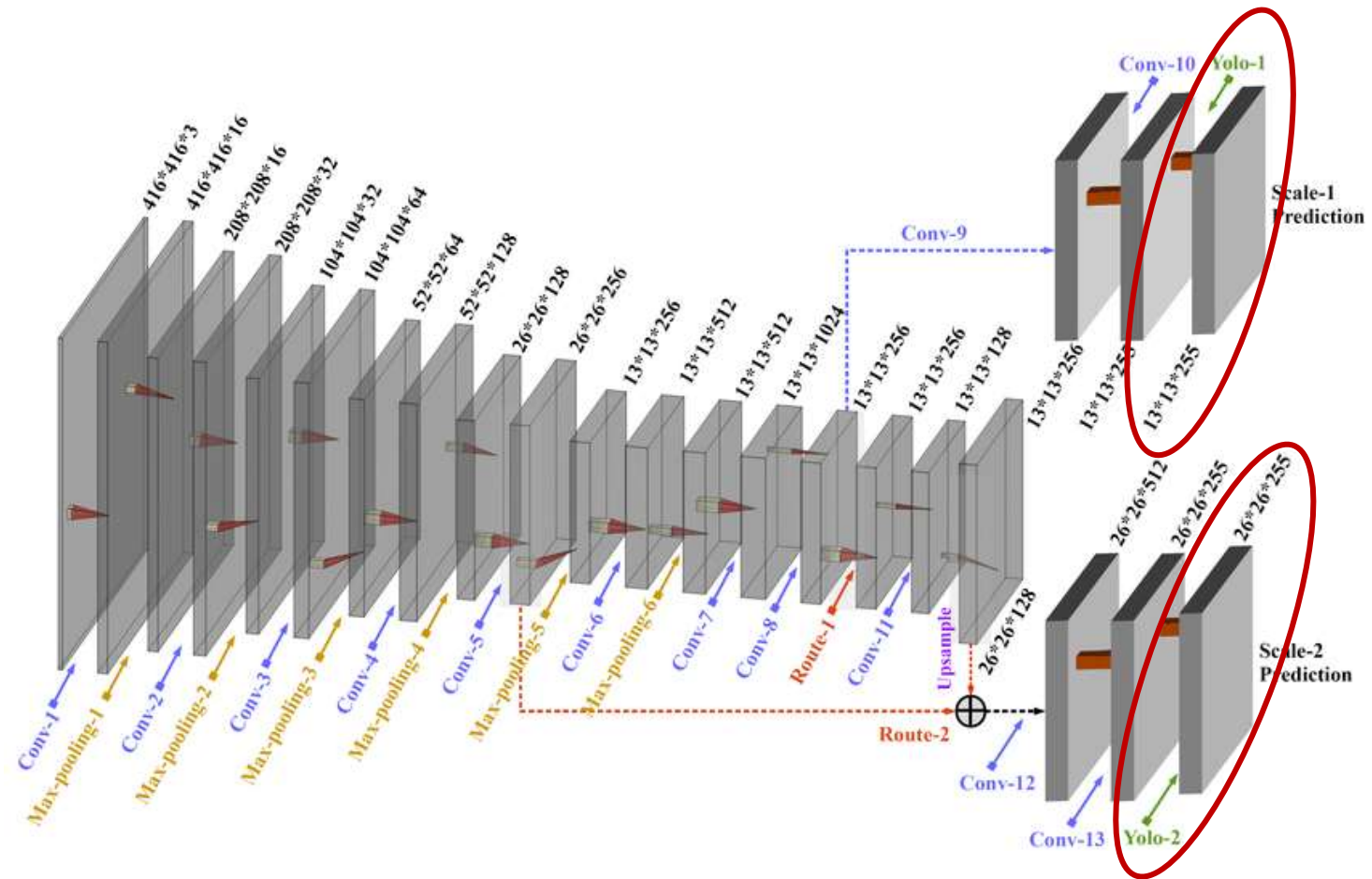
YOLO is a single convolutional neural network.

- Divides image into an $S \times S$ 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 $C = P(\text{Class}_i | \text{Object})$
- 5 predictions for each bounding box: $x, y, w, h, \text{confidence/objectness score}$
- Confidence/Objectness scores are interpreted as the $P(\text{object}) * \text{IOU}(\text{pred}, \text{truth})$
- Class-specific confidence scores can be derived as follows:

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

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

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

YOLOv1 Loss Function

$$\begin{aligned} & \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\ & + \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\ & + \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} (C_i - \hat{C}_i)^2 \\ & + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{noobj}} (C_i - \hat{C}_i)^2 \\ & + \sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \end{aligned}$$

YOLOv1 Loss Function

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

Coordinate Loss

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

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

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

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

YOLOv1 Loss Function

Coordinate loss

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

where $\mathbb{1}_i^{\text{obj}}$ denotes if object appears in cell i and $\mathbb{1}_{ij}^{\text{obj}}$ denotes that the j th 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

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

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

YOLOv1 Loss Function

Objectness or Confidence Loss

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

- Loss associated with the confidence score for each bounding box predictor
- \hat{C} 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

$$\begin{aligned} & \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\ & + \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\ & + \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} (C_i - \hat{C}_i)^2 \\ & + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{noobj}} (C_i - \hat{C}_i)^2 \end{aligned}$$

Classification Loss

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

YOLOv1 Loss Function

Loss associated with misclassification of object

Classification Loss

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

YOLOv1 Loss Function – **lambdas** are hyperparameters

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

Coordinate Loss

Objectness or
Confidence Loss

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

Classification Loss

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

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

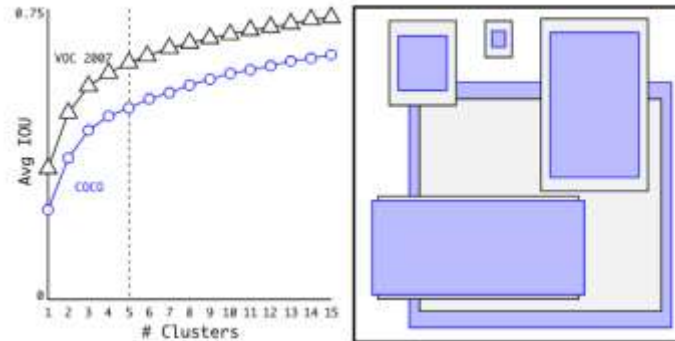


Figure 2: Clustering box dimensions on VOC and COCO. We run k-means clustering on the dimensions of bounding boxes to get good priors for our model. The left image shows the average IOU we get with various choices for k . We find that $k = 5$ gives a good tradeoff for recall vs. complexity of the model. 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:

$$b_x = \sigma(t_x) + c_x$$

$$b_y = \sigma(t_y) + c_y$$

$$b_w = p_w e^{t_w}$$

$$b_h = p_h e^{t_h}$$

$$Pr(\text{object}) * IOU(b, \text{object}) = \sigma(t_o)$$

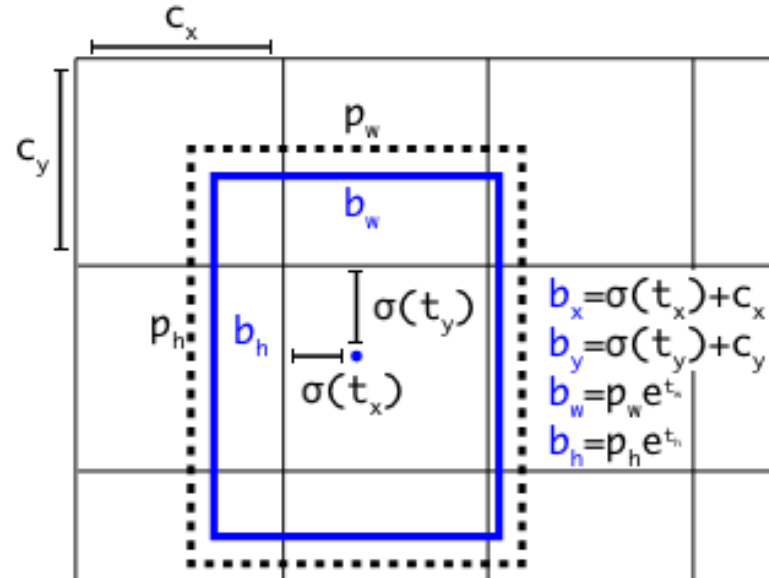


Figure 2. **Bounding boxes with dimension priors and location prediction.** We predict the width and height of the box as offsets from cluster centroids. We predict the center coordinates of the box relative to the location of filter application using a sigmoid function. This figure blatantly self-plagiarized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

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

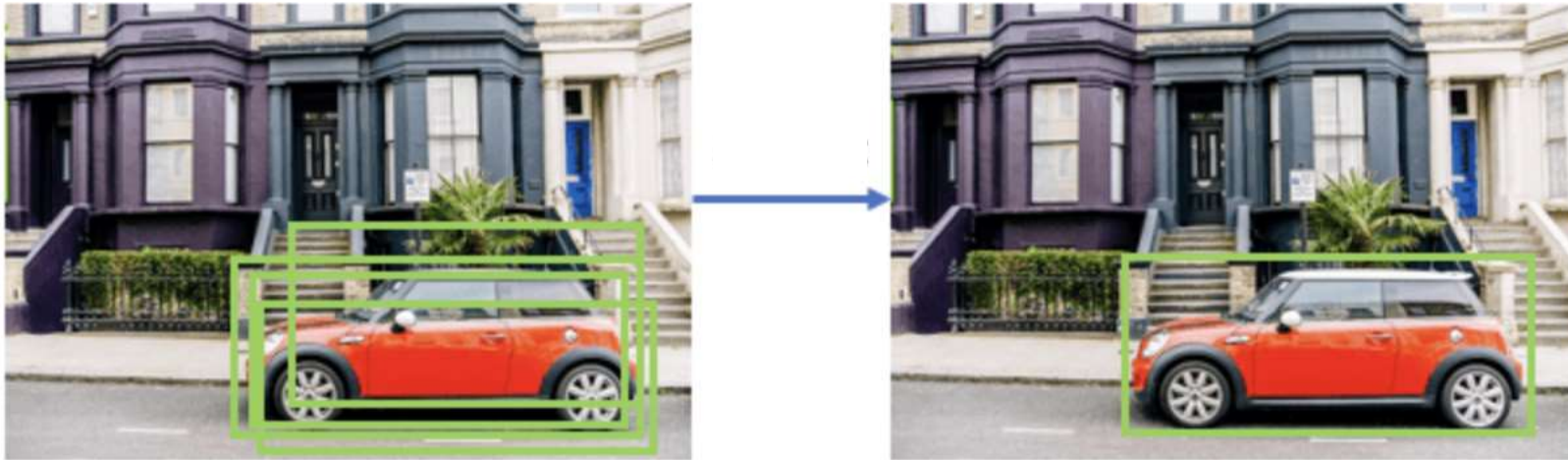
$$\lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^B I_{ij}^{\text{noobj}} \left[\hat{C}_i \log(C_i) + (1 - \hat{C}_i) \log(1 - C_i) \right] +$$

Classification Loss

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

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

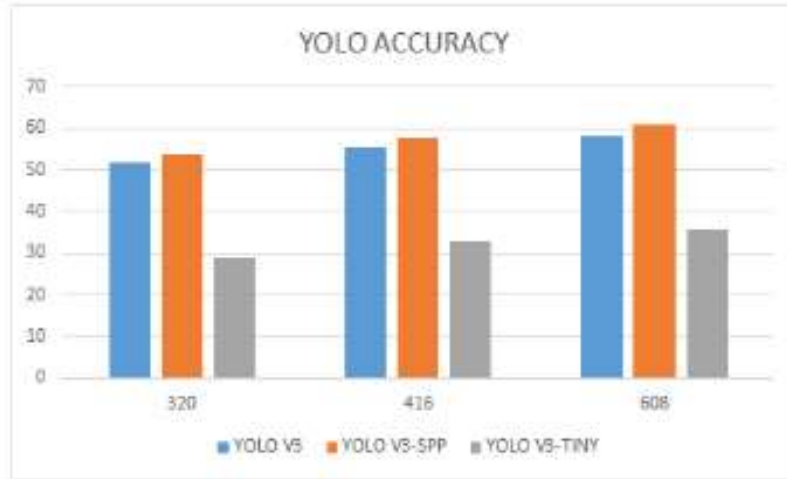


Fig.8: Accuracy comparison of YOLO algorithms

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.

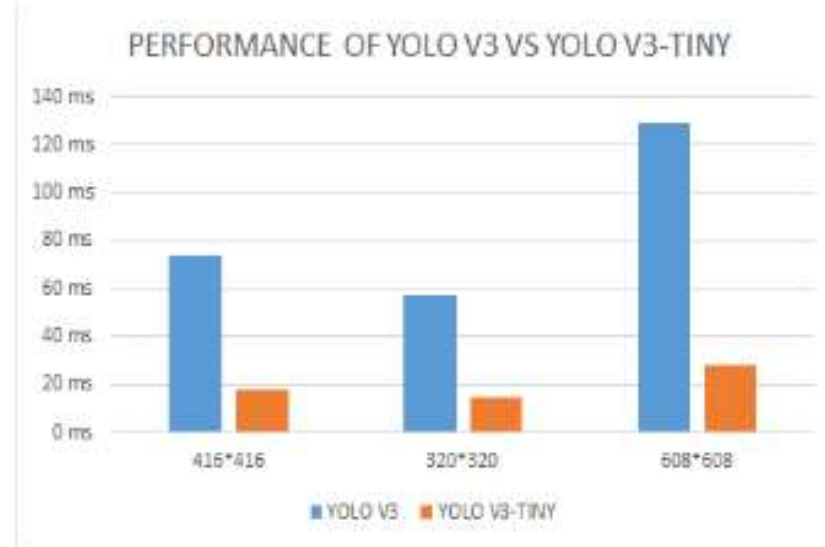


Fig.14: Speed Comparison

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 4

Post-Processing and Non-Max Suppression

Running Object detection on NMS output

Play with 3 hyperparameters

1. Object confidence threshold
2. class confidence threshold, and
3. non-max suppression threshold

Debrief

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

