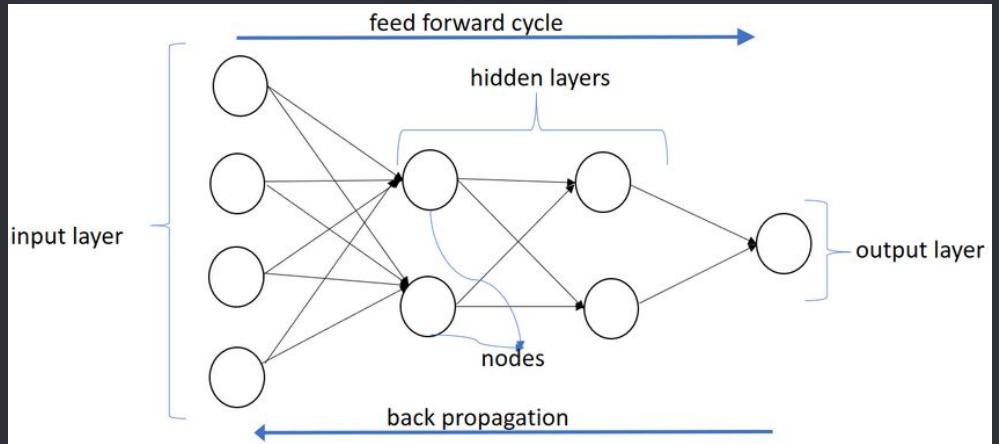
Real-Time Animated Characters Detection and Recognition with YOLOv5

Neural Network Basic

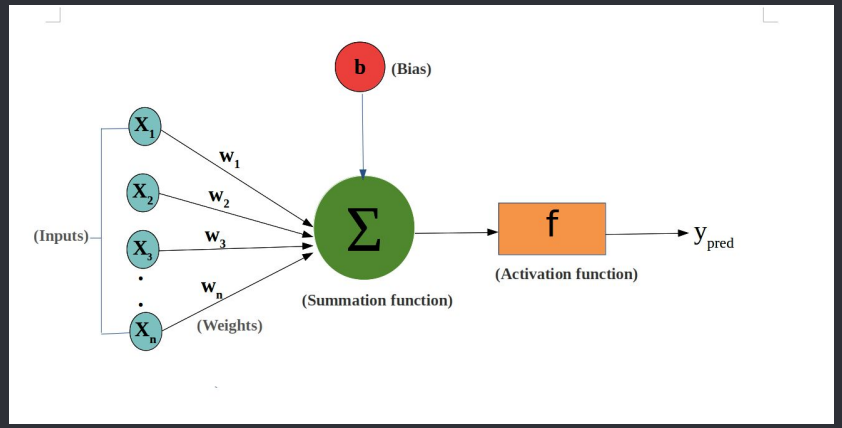
What is a neural network?

- A neural network is an universal approximator.
- Given significantly sized dataset and significantly complexed network, a neural network can estimate any function.
- It makes up of layers where each layer contain an arbitrary number of nodes



How does a neural network approximate?

- Connections between nodes have **weights** and **biases** attached to them.
- A network takes into account these variables and perform **forward propagation** to produce predicted outputs.



- However, since all these variables are initialized randomly, the initial prediction is inaccurate.
- Thus, a network needs to learn from the error and makes adjustment to **weights** and **biases** to reduce such an error. This process is called **backpropagation**.

Approximate bitwise xor operation.

- Goal: Train a neural network to estimate outputs of a bitwise xor operation given two single-digit of 0 or 1.

Inputs		**Output**
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

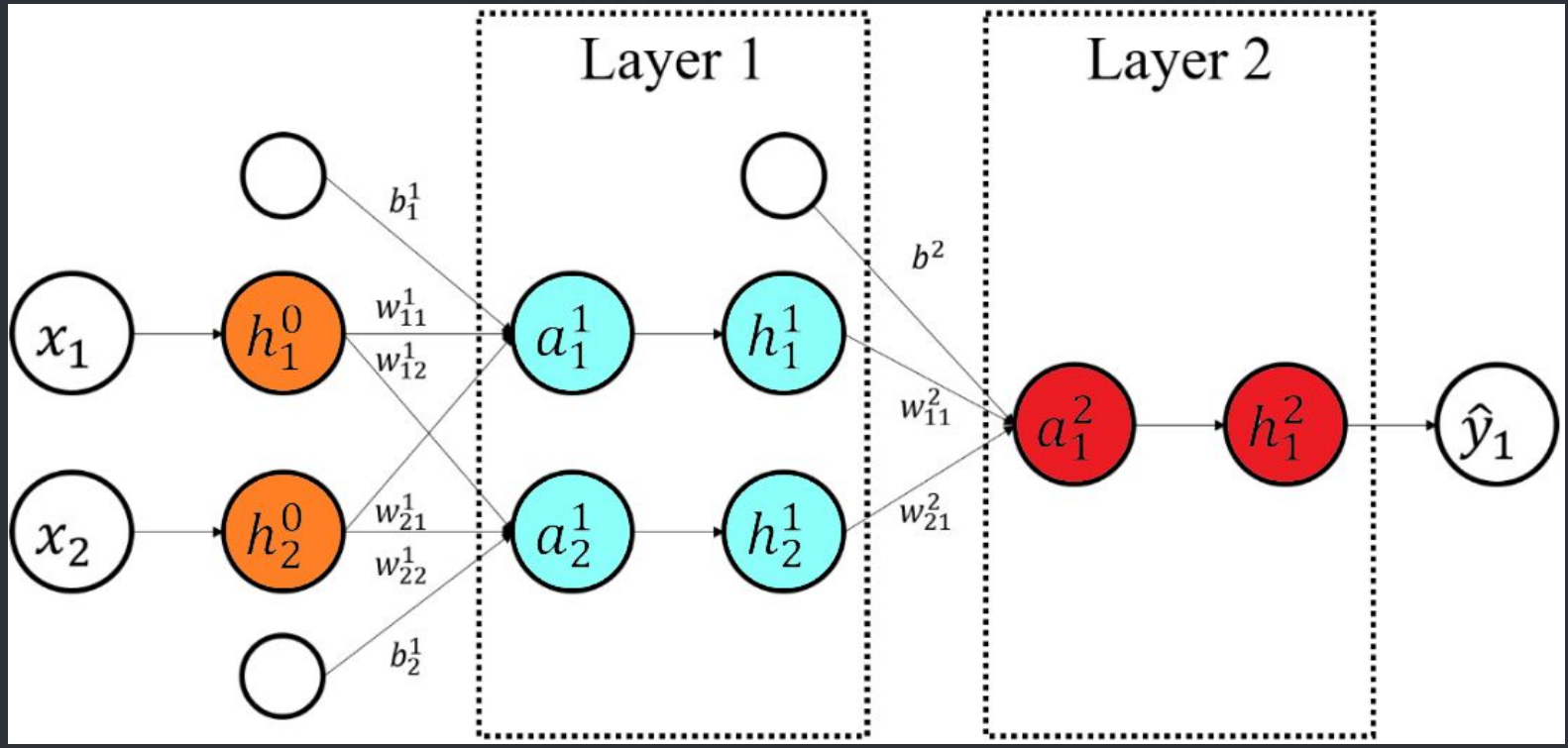
Considerations

- What are inputs and outputs?
  - More good data is better
- How many layers?
  - More layers results in a deeper network.
  - Increase trainable parameters
  - Need more time and data to optimize.
- How many nodes per each layer?
  - More nodes results in a wider network.
  - Increase trainable parameters
  - Need more time and data to optimize.
- What is the **activation function** for each node?
  - Activation function is a nonlinear function that increases the expressivity of a network by enabling it to better approximate nonlinear function.
- What is the **loss function**?
  - A loss is a single value represents the error between predicted outputs and truth outputs.

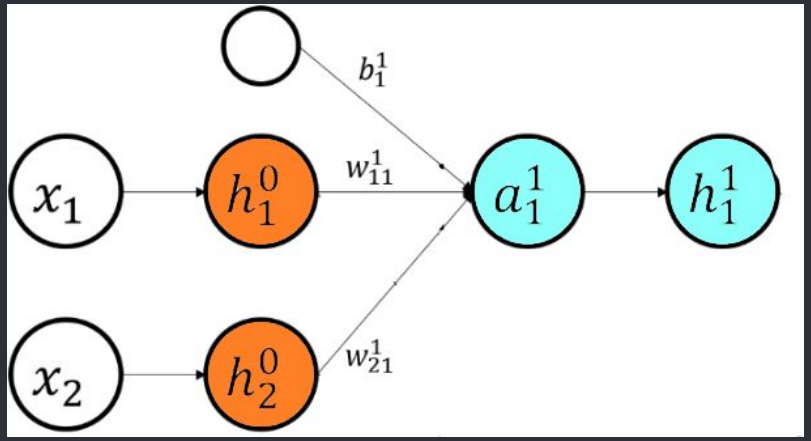
Model Design

- Type: Sequential
- Structure:
  - Input Layer
    - Node: 2
  - Hidden Layer 1
    - Type: Fully Connected
    - Node: 2
    - Activation function: Hyperbolic Tangent
  - Hidden Layer 2
    - Type: Fully Connected
    - Node: 1
    - Activation function: Sigmoid
  - Output Layer
    - Node: 1
- Loss function: Binary cross-entropy

Network Architecture Visualization



Forward Propagation - Normal Operation



$$Given : x_1,\ x_2$$

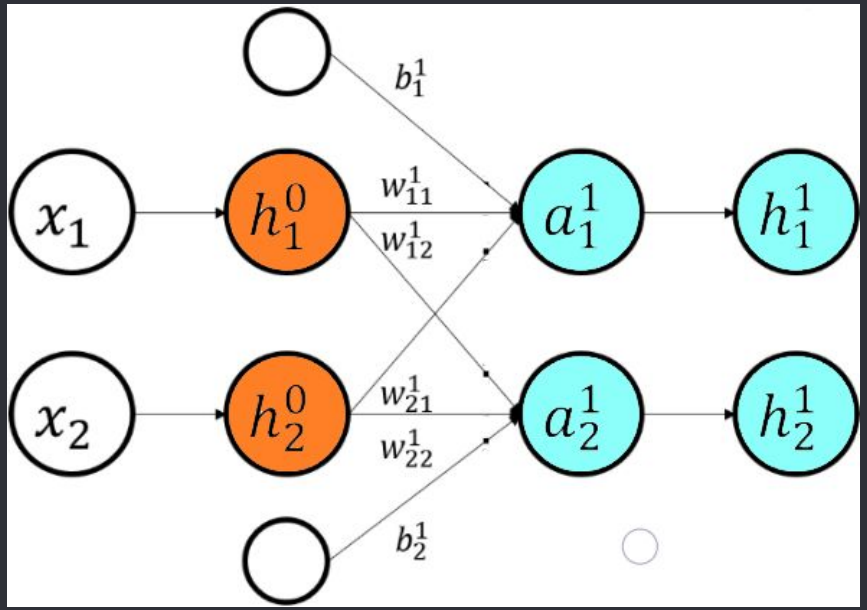
$$Result : h_1^0 = x_1$$

$$h_2^0 = x_2$$

$$a_1^1 = h_1^0 \cdot w_{11}^1 + h_2^0 \cdot w_{21}^1 + b_1^1$$

$$h_1^1 = \tanh(a_1^1)$$

Forward Propagation - Matrix Operation



$$Given : x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$h^0 = \begin{bmatrix} h_1^0 \\ h_2^0 \end{bmatrix}$$

$$w^1 = \begin{bmatrix} w_{11}^1 & w_{12}^1 \\ w_{21}^1 & w_{22}^1 \end{bmatrix}$$

$$b^1 = \begin{bmatrix} b_1^1 \\ b_2^1 \end{bmatrix}$$

$$a^1 = \begin{bmatrix} a_1^1 \\ a_2^1 \end{bmatrix}$$

$$h^1 = \begin{bmatrix} h_1^1 \\ h_2^1 \end{bmatrix}$$

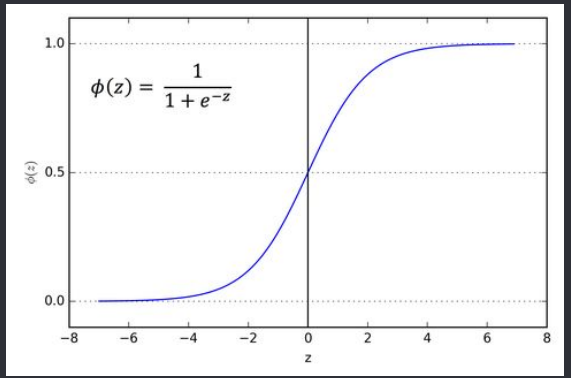
$$Result : h^0 = x$$

$$a^1 = (w^1)^T \cdot h^0 + b^1$$

$$h^1 = \tanh(a^1)$$

Predicted Outputs

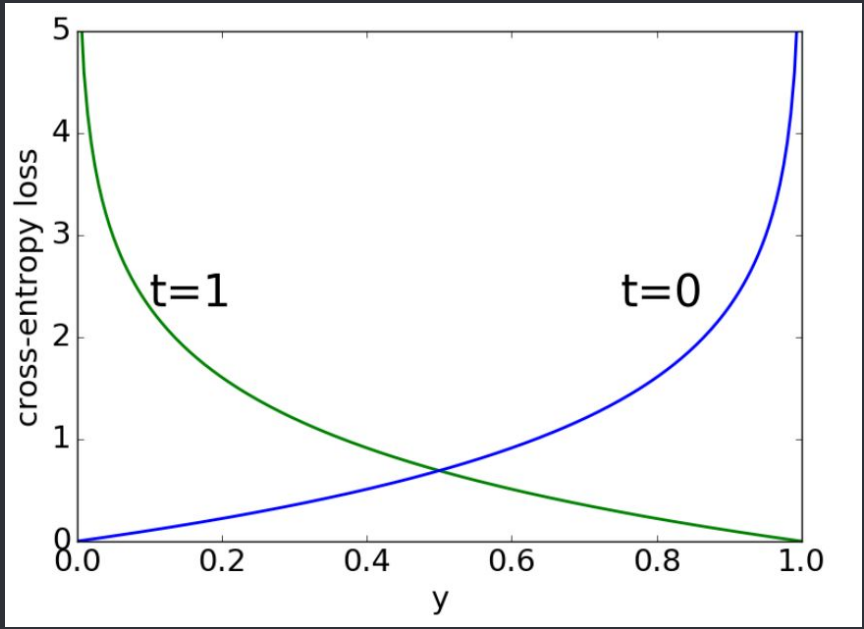
- Since binary cross-entropy is the loss function of choice, the activation function of the last hidden layer will be Sigmoid.
  - Sigmoid will bound any input between 0 and 1.
- Table below shows predicted outputs produced from randomize weights and biases.



x_1	x_2	y	ŷ
0	0	0	0.5618
0	1	1	0.2649
1	0	1	0.7879
1	1	1	0.1479

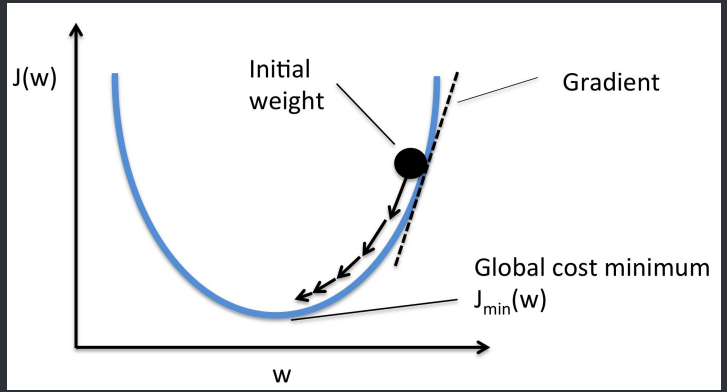
Binary Cross-Entropy

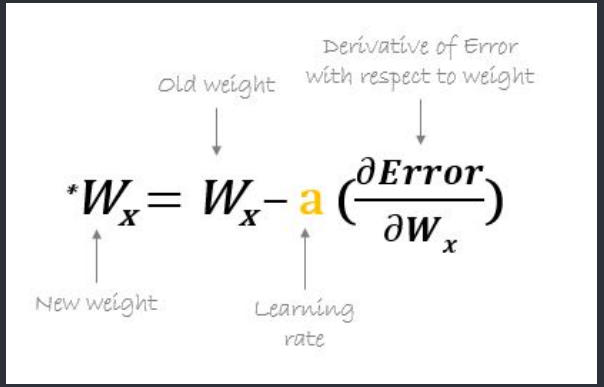
- **Binary Cross-Entropy** or **log-loss** function is used to calculate the loss value when there are exactly two truth outputs.
- When the truth output is 0, the loss value produced by the binary cross-entropy will converge to 0 as predicted output goes to 0.
- When the truth output is 1, the loss value produced by the binary cross-entropy will converge to 0 as predicted output goes to 1.
- $$L(\hat{y}, y) = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}))$$



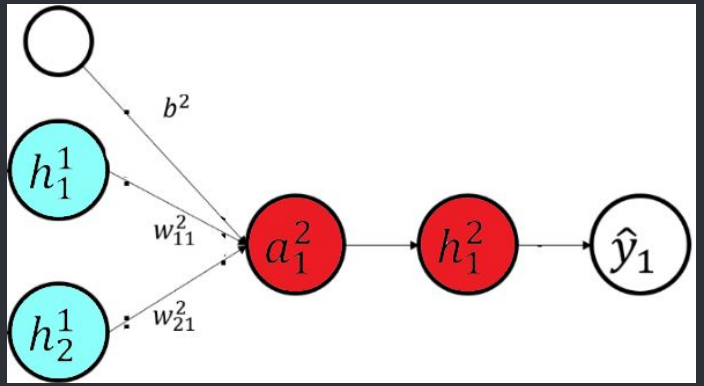
Backpropagation - Stochastic Gradient Descent

- Stochastic Gradient Descent is an iterative method for adjusting weights and biases such that the loss value is minimized .
- It relies on the differential of the loss value with respect to each trainable parameter.
- Simply put, the gradient (△) represents the direction and magnitude of the change that a given trainable variable contributed to the loss value.
- Thus, the loss value can be minimized by subtracting a gradient from an optimizable variable.





Backpropagation - Gradient Calculation



$$\frac{\partial L}{\partial \vec{w}^2} = \begin{bmatrix} \frac{\partial L}{\partial w_{11}^2} \\ \frac{\partial L}{\partial w_{21}^2} \end{bmatrix}$$

$$\frac{\partial L}{\partial w_{11}^2} = \frac{\partial L}{\partial \hat{y}_1} \cdot \frac{\partial \hat{y}_1}{\partial h_1^2} \cdot \frac{\partial h_1^2}{\partial a_1^2} \cdot \frac{\partial a_1^2}{\partial w_{11}^2}$$

$$\frac{\partial L}{\partial \hat{y}_1} = \left(\frac{1-y}{(1-\hat{y}_1)} - \frac{y}{\hat{y}_1}\right)$$

$$\frac{\partial \hat{y}_1}{\partial h_1^2} = 1$$

$$\frac{\partial h_1^2}{\partial a_1^2} = \frac{\partial}{\partial a_1^2} \cdot \sigma(a_1^2) = \sigma(a_1^2) \cdot (1 - \sigma(a_1^2))$$

$$\frac{\partial a_1^2}{\partial w_{11}^2} = \frac{\partial}{\partial w_{11}^2} \cdot (h_1^1 \cdot w_{11}^2 + h_2^1 \cdot w_{21}^2 + b^2) = h_1^1$$

$$\frac{\partial L}{\partial w_{11}^2} = \left(\frac{1-y}{(1-\hat{y}_1)} - \frac{y}{\hat{y}_1}\right) \cdot [\sigma(a_1^2) \cdot (1 - \sigma(a_1^2))] \cdot h_1^1)$$

$$\frac{\partial L}{\partial w_{21}^2} = \frac{\partial L}{\partial \hat{y}_1} \cdot \frac{\partial \hat{y}_1}{\partial h_1^2} \cdot \frac{\partial h_1^2}{\partial a_1^2} \cdot \frac{\partial a_1^2}{\partial w_{21}^2} = \left(\frac{1-y}{(1-\hat{y}_1)} - \frac{y}{\hat{y}_1}\right) \cdot [\sigma(a_1^2) \cdot (1 - \sigma(a_1^2))] \cdot h_2^1)$$

$$\frac{\partial L}{\partial b^2} = \frac{\partial L}{\partial \hat{y}_1} \cdot \frac{\partial \hat{y}_1}{\partial h_1^2} \cdot \frac{\partial h_1^2}{\partial a_1^2} \cdot \frac{\partial a_1^2}{\partial b^2} = \left(\frac{1-y}{(1-\hat{y}_1)} - \frac{y}{\hat{y}_1}\right) \cdot [\sigma(a_1^2) \cdot (1 - \sigma(a_1^2))] \cdot 1)$$

Predicted Outputs

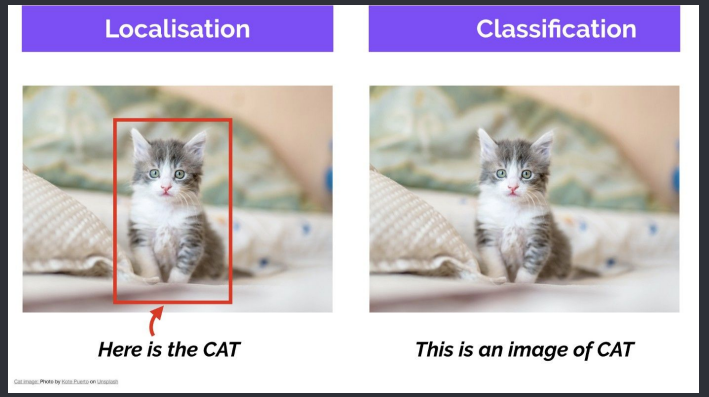
x_1	x_2	y	Pre-train ŷ	Post-train ŷ
0	0	0	0.5618	0.01430112
0	1	1	0.2649	0.99191275
1	0	1	0.7879	0.9918608
1	1	1	0.1479	0.01547358

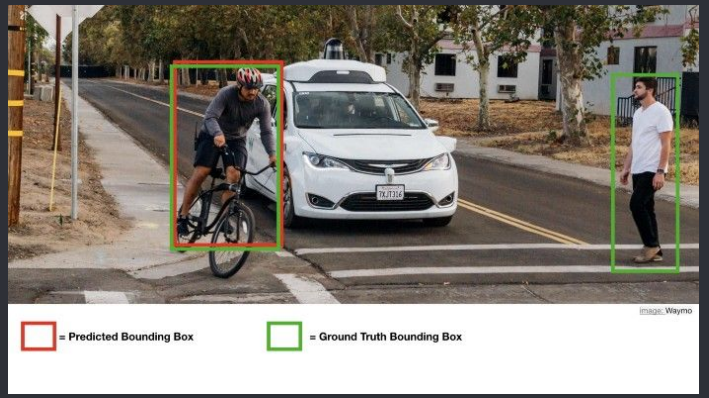
- Complete solution available at https://github.com/sotheanith/CECS-551-Collection/tree/master/Assignment%204

History

When precision is not good enough...

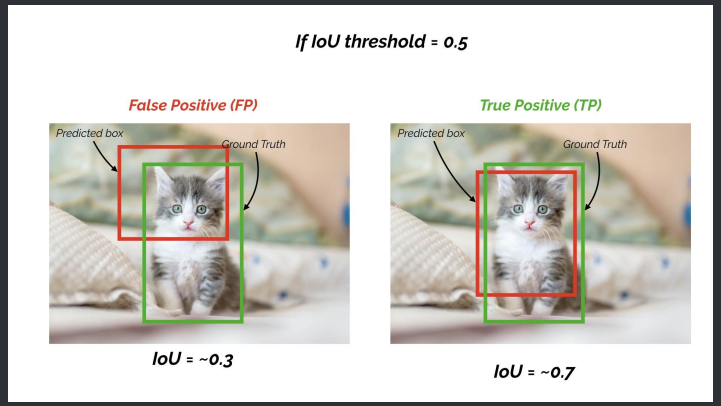
- Precision is not a good metric for measuring performances of object detection and recognition algorithms because it only account for classification; not localization.
- Consider the following example:
  - True Positive = 1
  - False Positive = 0
  - Precision = TP/(TP+FP) = 1





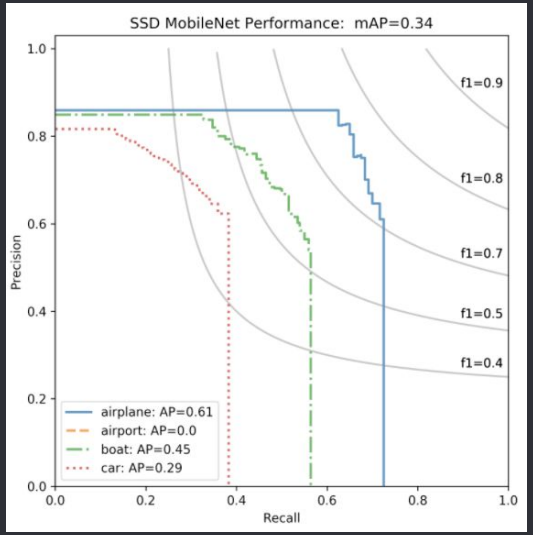
Mean Average Precision

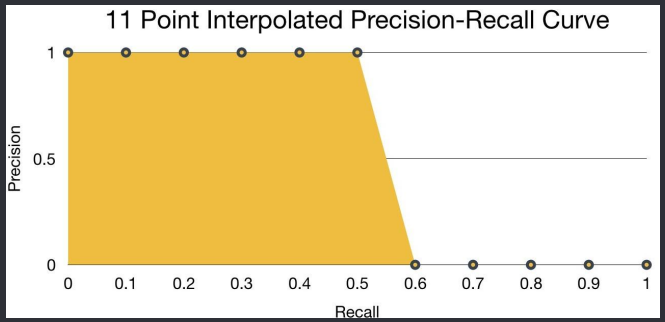
- Mean Average Precision or mAP is the go to metric for measuring performances of object detection and recognition models.
- It factors in, both, a model's object localization ability and a model's object classification ability.
- To start with, an arbitrary IOU threshold (t) and the predicted class (ŷ) are used to generate the confusion matrix for a class y.
  - TP: IOU > t and ŷ == y.
  - FP: IOU > t and ŷ != y.
  - FN: IOU < t and y exists
  - TN: Every possible boundary boxes that is not a ground truth.



Mean Average Precision

- From the confusion matrix, precision and recall of a class are calculated and plot on graph.
- The AP is the area under the precision-recall curve and it can be approximate using various methods such as
  - The 11-points Interpolated Precision Recall Curve
    - Max precision at a given recall and up
  - Riemann Sum
- Finally, mAP@t is the average of AP of all classes.
- Furthermore, mAP across various IOUs are also used as a secondary metric.





$$AP = \frac{1}{11} \sum_{Recall_i} Precision(Recall_i) = 1$$

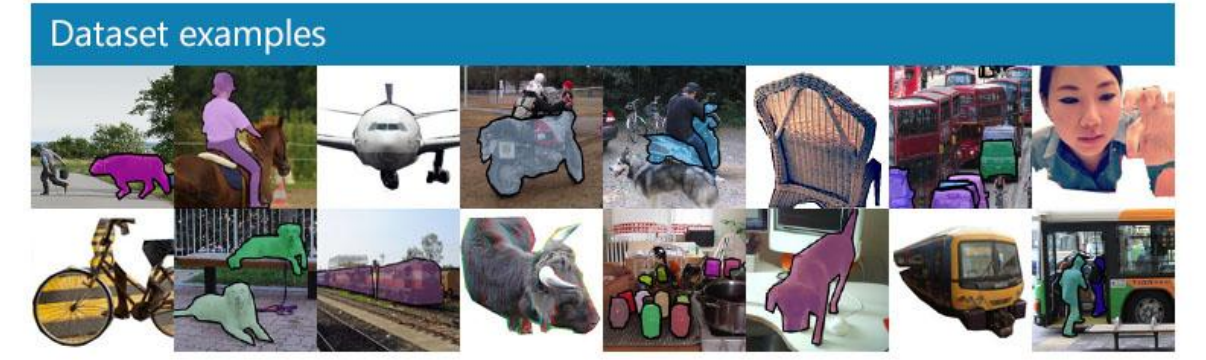
$$AP = \frac{1}{11} \sum_{0,0.1,0.2,0.3,..0.9,1} (1 * 6) + (0 * 5) = 0.545$$

PASCAL VOC Dataset

- PASCAL VOC or PASCAL Visual Object Classes Challenge was a popular dataset used to benchmark object detection and recognition.
- It combines 2007 and 2012 datasets for training, validating, and testing.
- Features:
  - ~10k images (50% train and 50% val/test)
  - 20 object categories (dogs, chairs, horses, etc…)
  - ~20k object instances

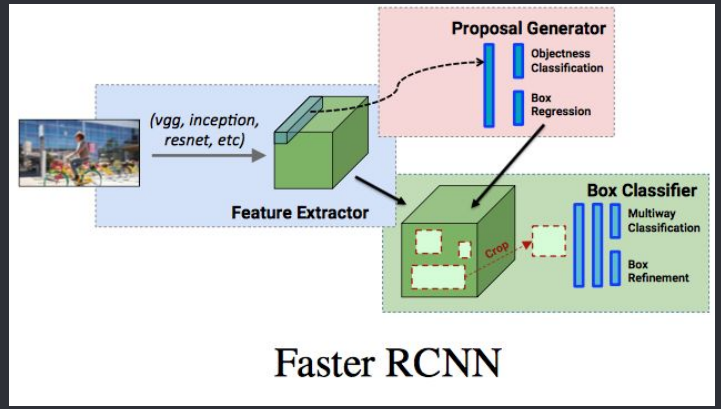
COCO Dataset

- COCO or Common Objects in Context is one of the most popular dataset used to benchmark performances of various deep learning models and algorithms.
- Features:
  - 300k images with more than 200k images labels
  - 1.5 million object instances
  - 80 object categories (Cat, Dog, Car, etc...)
  - 91 stuff categories (Sky, road, etc...)
  - and more...



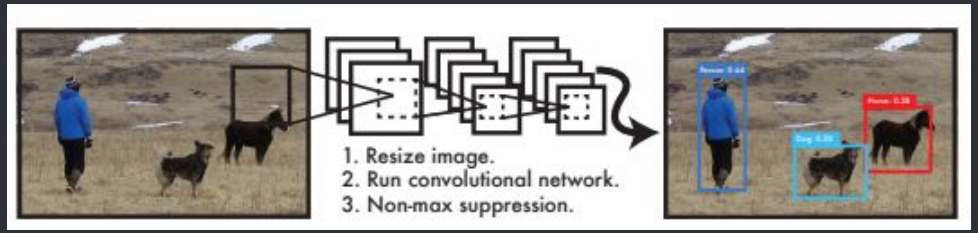
R-CNN, Fast R-CNN, Faster R-CNN

- Prior to the introduction of YOLO algorithm, object detection and object recognition are considered independent tasks.
- As such, many popular algorithms have two distinct models where one model is responsible for find regions that contain objects (object detection) and another model is responsible for classifying objects (object recognition).
- Example: R-CNN, Fast R-CNN, Faster R-CNN, etc...
- Advantages:
  - Relatively high mAP@.5
    - 73.2 (Pascal VOC 2007+2012)
  - Modularity
- Disadvantages:
  - Hyperparameters sensitivity
  - Low inference time
    - 7 fps (Pascal VOC 2007)



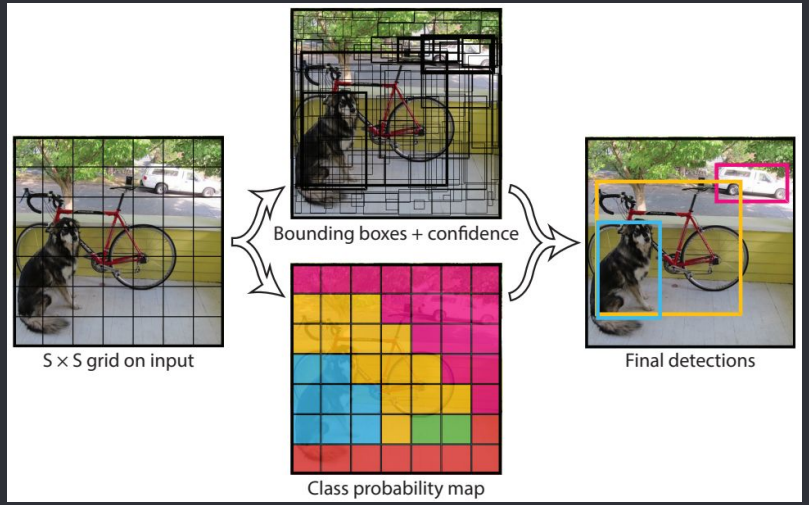
YOLOv1

- In 2015, Joseph Redmon et al introduces YOLO or You Only Look Once algorithm which aims at unifying object detection and object recognition into a single problem and achieving real-time image processing.
- YOLO algorithm works by restructuring the problem of finding bounding boxes and classify bounding boxes' contents into a regression problem where a model has to predict both simultaneously.
- Advantages:
  - Respectable mAP
    - 63.4 (Pascal VOC 2007+2012)
  - Real-time processing
    - 45 fps (Pascal VOC 2007)
- Disadvantages:
  - Can only detect up to 49 (7x7) objects per image
  - Terrible at detecting overlapping objects
  - Can only approximate 2 shapes per object



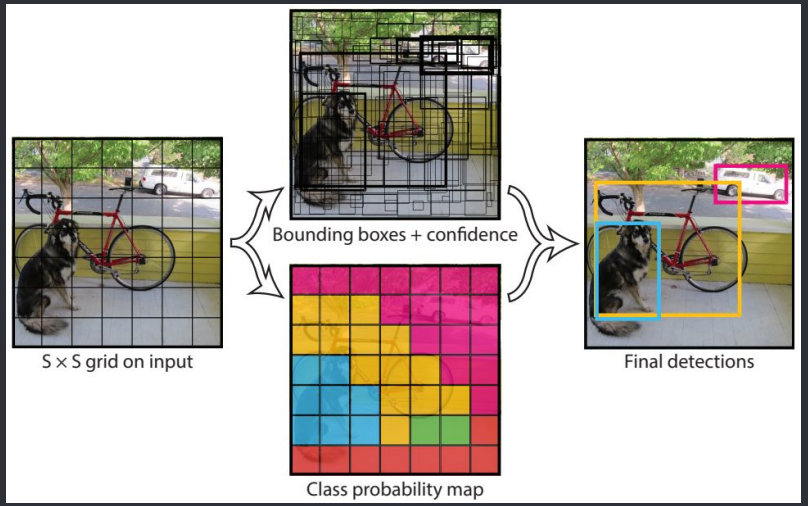
YOLOv1 - How does it work?

- At its fundamental level, YOLOv1 algorithm process information in two main steps: predicting and filtering.
- To start with, during the predicting phase, a model is used to produce information related bounding boxes and class probability.
- Then, the Non-maximum Suppression algorithm is used to filter irrelevant information and produce the final prediction.



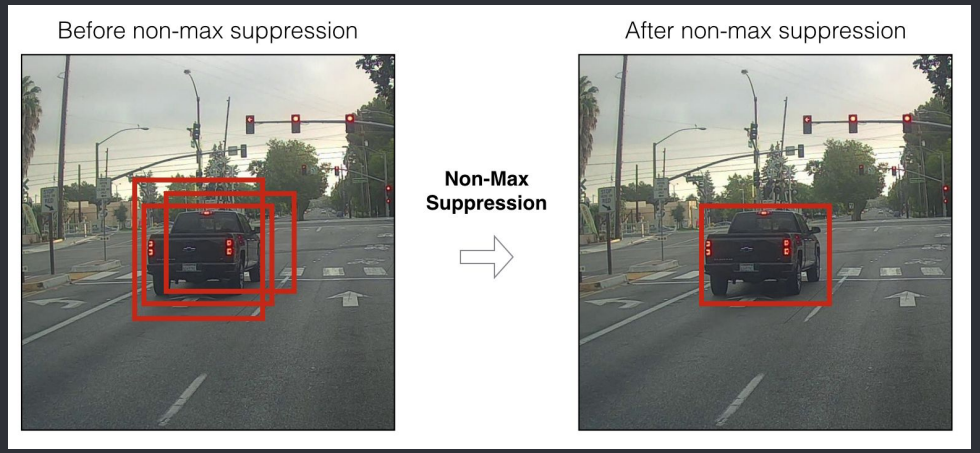
YOLOv1 - The Predicting Phase

- YOLOv1 accepts a fixed size image of 448x448 as an input and splits such image into 7 x 7 (S x S) cells.
- Each cell is responsible for predicting 2 bounding boxes and a class probability with following information:
  - Bounding boxes (B):
    - X
    - Y
    - Width
    - Height
    - Confidence score
  - Class (C)
    - Probability of every possible classes
- Thus, YOLOv1 output shape is S x S x (5 * B + C).
- In the case of Pascal VOC dataset, the YOLOv1 output is 7x7x30 since S = 7, B = 2, and C = 20



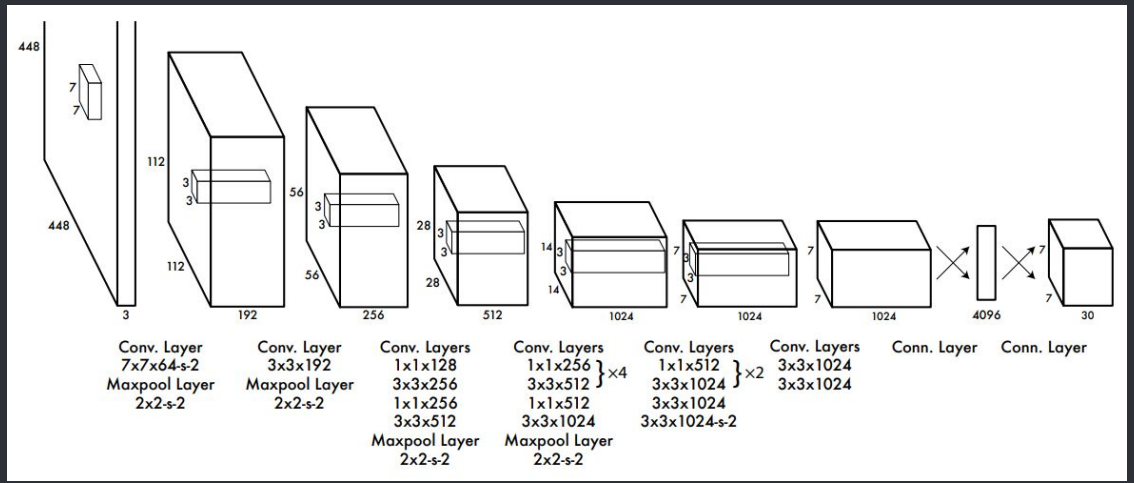
YOLOv1 - The Filtering Phase

- After predicting bounding boxes and class probability, Non-maximum Suppression or NMS algorithm is used to filter out overlapping proposed bounding boxes.
- NMS starts with picking a bounding box with the highest confidence score from the list of proposed boxes and add it to the list of picked boxes.
- Then, the algorithm finds all boxes that predicted the same class as the picked box and calculate their IOUs.
- If a proposed box and the picked box have an IOU over a certain threshold, such proposed box is removed from the list of proposed boxes.
- This process is repeated until there is no more boxes in the list of proposed boxes.



YOLOv1 - Network Architecture

- YOLOv1 network architecture is inspired by the GoogLeNet model.
- It composed of 24 convolution layers followed by 2 fully-connected layers for outputs.



YOLOv1 - Loss Function

- YOLOv1 uses a custom loss function that penalizes each component of the predicted output independently.
- It consists of 5 terms.
- The first term is the loss for predicted x and y.
- The second term is the loss for predicted width and height.
  - Square root is used to slow its grow.
- The third term is the loss for the predicted confidence score when the model predict that there is a box.
- The fourth term is the loss for the predicted confidence score when the model predict that there is not a box.
- The fifth term is the loss for the class probability.

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

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

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

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

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

- The three λs are used to scale the importance of each term.
- 1^{obj}_{ij} : 1 where there is an object in cell i and bounding box j has the highest confidence score. Else, 0.
- 1^{obj}_{i} : 1 if there is a object in cell i. Else, 0.
- 1^{noob}_{ij}: 1 if there is no object in cell i. Else, 0.

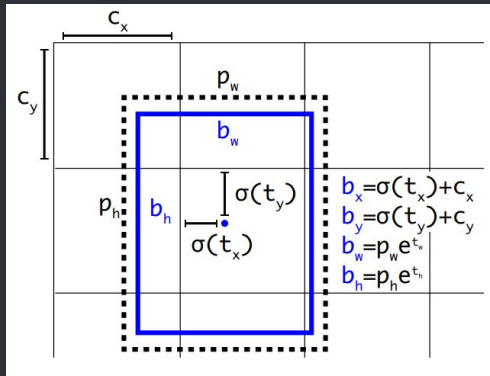
YOLOv2

- YOLOv2 improves over YOLOv1 by incorporating new techniques and best-practices which translate to overall improvement in mAP and inference time.
- It, also, offers a way for prioritizing mAP vs inference time through model selection.

Model	mAP@.5 (Pascal VOC 2007+2012)	FPS (Pascal VOC 2007)
YOLOv2 288 × 288:	69.0	91
YOLOv2 352 × 352	73.7	81
YOLOv2 416 × 416	76.8	67
YOLOv2 480 × 480	77.8	59
YOLOv2 544 × 544	78.6	40

YOLOv2 - Improvements

- Train/Detection Process
  - **High-Resolution Classifier** and **Detector**
    - It is being train with 448 x 448 images
    - It can detect 448 x 448 images and up.
  - **Multi-Scale Training**
    - Size of train images are dynamically changed at certain epochs.
- New Architecture (Darknet 19)
  - **Batch Normalization**
  - **Pure Convolution Layers**
    - Without fully-connected layers, model can accept any image size.
  - **Passthrough**
- Bounding Boxes Prediction
  - **Anchor Boxes**
    - Initial shape of bounding boxes are determined by applying KNN to train images.
  - **Location Prediction**
    - A bounding box is predicted relative to its parent cell.
  - **Dimension Priors**
    - A bounding box is predicted with respect to some functions.



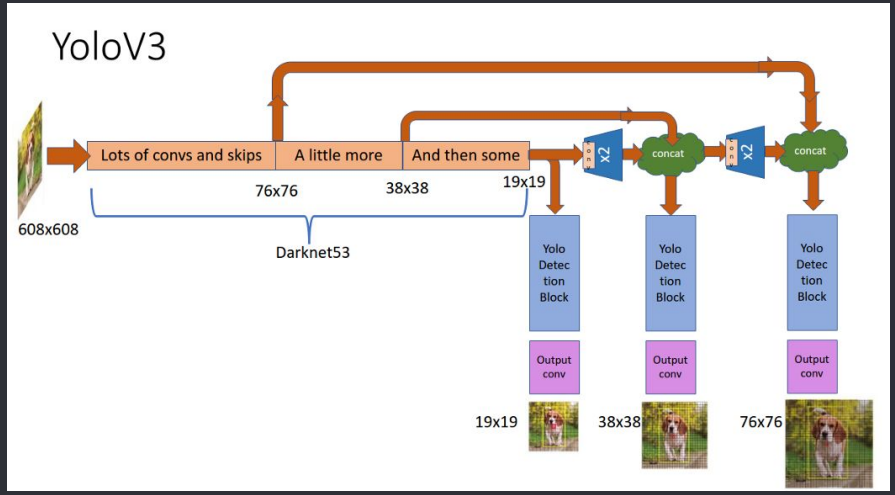
YOLOv3

- YOLOv3 represents a major improvement over YOLOv2 as it has a new architecture that improvement its ability to detect small objects and a new outputs that enable it to detect overlapping objects.
- Performances (COCO):

Model	mAP@.5	FPS	mAP_{small}	mAP_{medium}	mAP_{large}
Faster R-CNN+++	55.7	N/A	15.6	38.7	50.9
RetinaNet	61.1	5.05	24.1	44.2	51.2
YOLOv2	44.0	N/A	5.0	22.4	35.5
YOLOv3 608 × 608	57.9	19.6	18.3	35.4	41.9

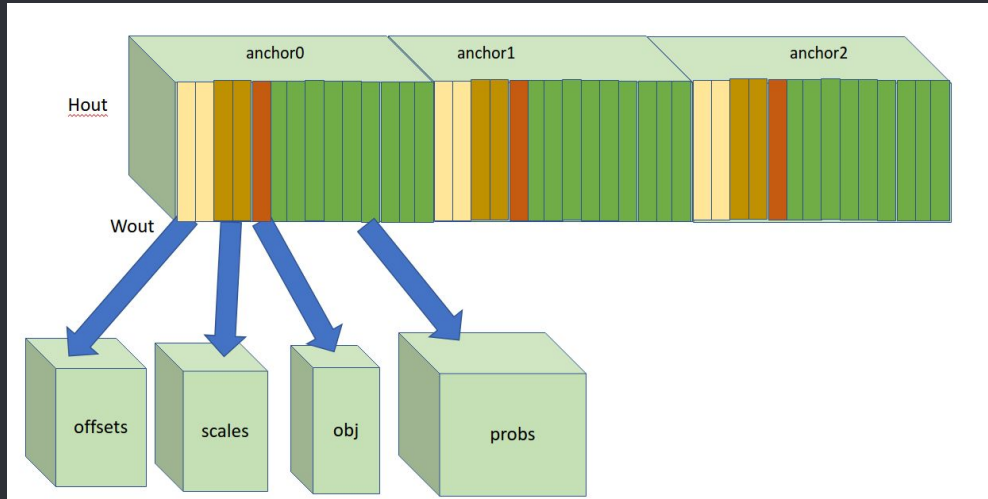
YOLOv3 - Network Architecture

- YOLOv3 architecture or Darknet-53 is inspired of ResNet and FPN (Feature-Pyramid Network).
- It has 52 convolution layers for feature extracts and 3 prediction heads.
- Each prediction head is tasked with outputting different size prediction.
- Thus, a network has to learn to detect and recognize objects at various sizes.



YOLOv3 - New Outputs

- In order to detect overlapping objects, YOLOv3 modifies the outputs structure such that bounding boxes, not cells, are responsible for predicting class probability.
- Given:
  - S: number of cells
  - B: number of bounding boxes
  - C: number of classes
- Output: S x S x B *(5 + C)



YOLOv5

By Ultralytics

What is YOLOv5?

- YOLOv5 is the latest iteration of YOLO algorithm and a spiritual successor to YOLOv3.
- Released in June 2020 by Ultralytics, it represents their continue research in the field of computer vision and it incorporates “lessons learned and best practices evolved over thousands of hours of research and development.”
- Unlike previous iteration of YOLO, YOLOv5 is implemented purely in Python with PyTorch framework as the backbone and all of its models are pre-trained on COCO dataset.
- Github: https://github.com/ultralytics/yolov5

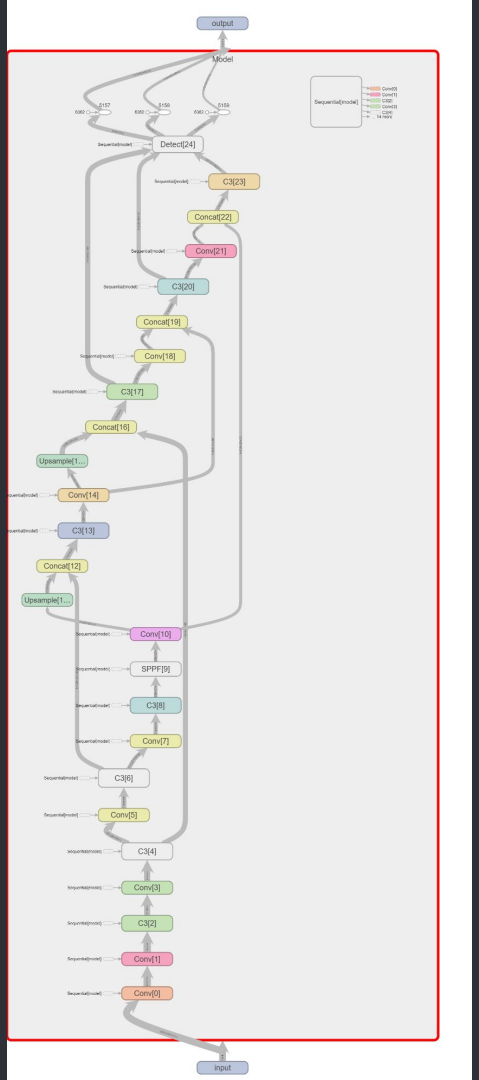
YOLOv5 - Models

- Currently, YOLOv5 offers 10 pre-trained models of various sizes where 5 models are designed for 640x640 images and another 5 models are designed for 1280 x 1280 images.

Model	size (pixels)	mAP^{val} 0.5:0.95	mAP^{val} 0.5	Speed CPU b1 (ms)	Speed V100 b1 (ms)	Speed V100 b32 (ms)	params (M)	FLOPs @640 (B)
YOLOv5n	640	28.4	46.0	**45**	**6.3**	**0.6**	**1.9**	**4.5**
YOLOv5s	640	37.2	56.0	98	6.4	0.9	7.2	16.5
YOLOv5m	640	45.2	63.9	224	8.2	1.7	21.2	49.0
YOLOv5l	640	48.8	67.2	430	10.1	2.7	46.5	109.1
YOLOv5x	640	50.7	68.9	766	12.1	4.8	86.7	205.7
YOLOv5n6	1280	34.0	50.7	153	8.1	2.1	3.2	4.6
YOLOv5s6	1280	44.5	63.0	385	8.2	3.6	16.8	12.6
YOLOv5m6	1280	51.0	69.0	887	11.1	6.8	35.7	50.0
YOLOv5l6	1280	53.6	71.6	1784	15.8	10.5	76.8	111.4
YOLOv5x6 + TTA	1280 1536	54.7 **55.4**	**72.4** 72.3	3136 -	26.2 -	19.4 -	140.7 -	209.8 -

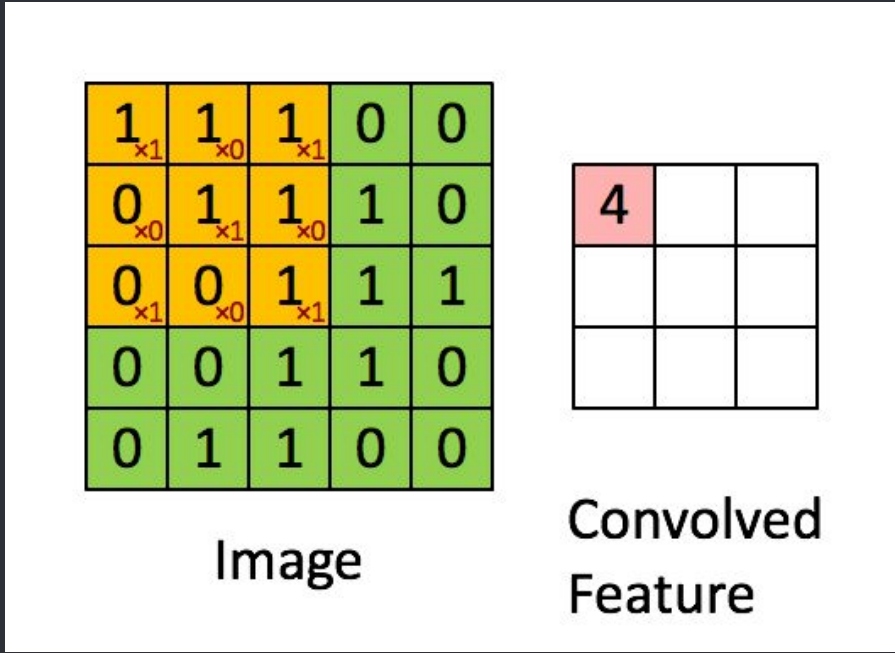
YOLOv5 - Architecture

- YOLOv5 network is divided into two parts:
  - **Body**: responsibles for feature extraction
  - **Heads**: responsibles for prediction
- Both parts are composed of three special layers:
  - **Convolution**
  - **SPPF**
  - **C3**



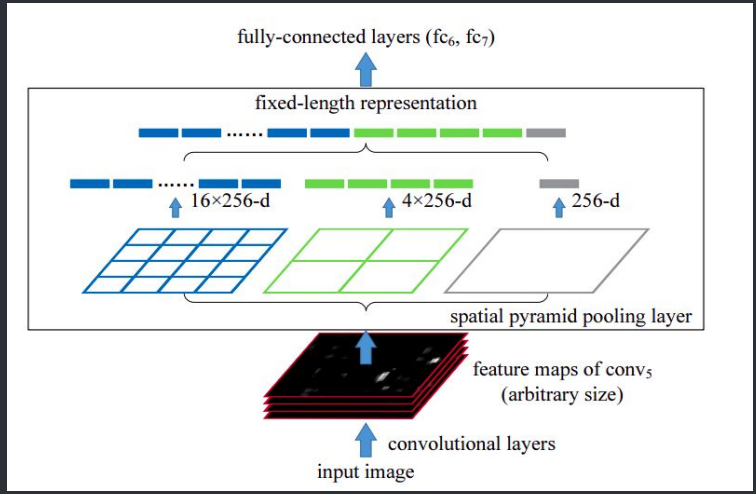
YOLOv5 - Convolution Layers

- A convolution layer is a major building block of a neural network and it has shown to performance better at extracting features from an image than a fully-connected layer.
- It works under the same principle as a convolution operation where kernels are slide over and convolve an image.
- In case of a convolution layer, kernels represent weights and biases that should be optimized.



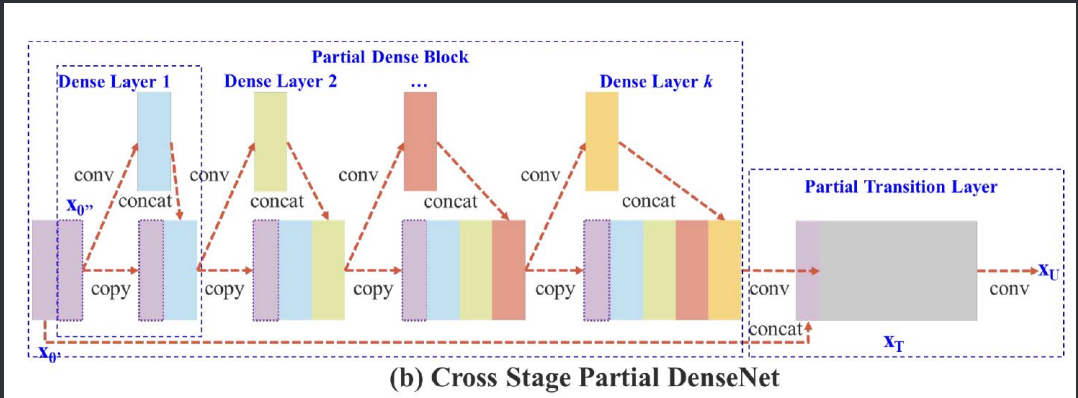
YOLOv5 - SPPF

- A SPPF layer is a special implementation of SPP or Spiral Pyramid Pooling layer that achieves identical result with less computing cost.
- The goal of any pooling layer is to reduce the size of an input such that a subsequent convolution layer can sample larger area.
- However, unlike other pooling layers, a SPPF layer is designed such that it will always produce an output of fixed size for any giving input.
- It accomplishes this by dynamically change the size of kernels and strides based on given input size and a desire output size.



YOLOv5 - C3

- A C3 layer is a modify implementation of the CSP or Cross Stage Partial bottleneck layer from DenseNet with one of the four main convolution layers removed.
- The idea behind a bottleneck layer is that by introducing a smaller layer between two large layers, the second large layer is forced to learn from an encoded information produced by the smaller layer.
- Basically, a C3 layer is a bottleneck layer with multiple passthrough that enable convolution layers to learn from an encoded and a raw informations while avoiding the problem of vanishing gradient.

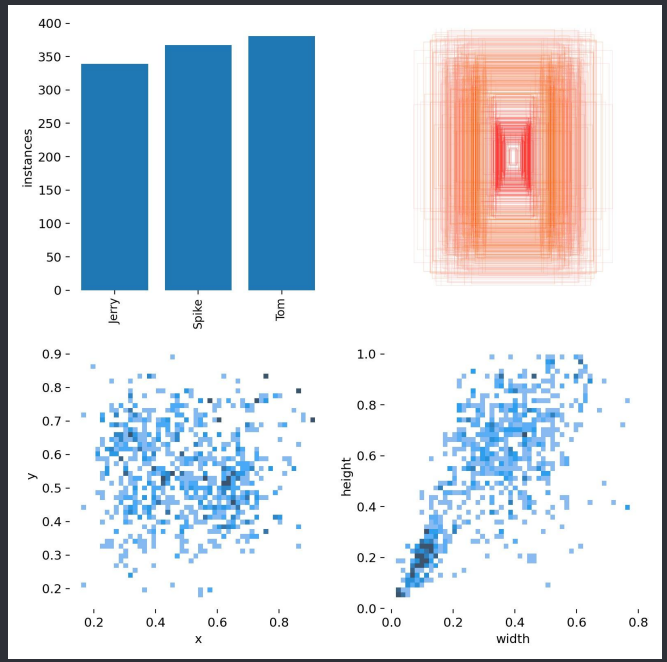


(b) Cross Stage Partial DenseNet

Experiment

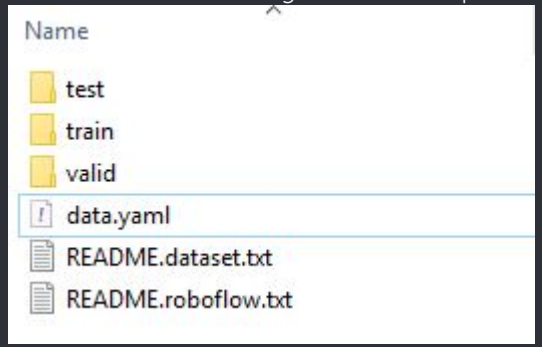
Dataset

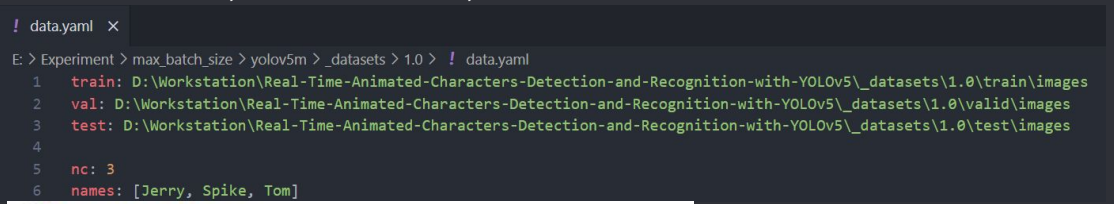
- Source: Tom & Jerry | Triple Trouble | Classic Cartoon Compilation | WB Kids
- Number of Images: 1000
  - Contains objects: 931
  - Null examples: 69
- Object Instances:
  - Tom: 562
  - Spike: 538
  - Jerry: 490
- Preprocessing:
  - Auto-Orient
  - Stretch to 640x640
- Split Ratio:
  - Train: 70%
  - Validation: 21%
  - Test: 9%

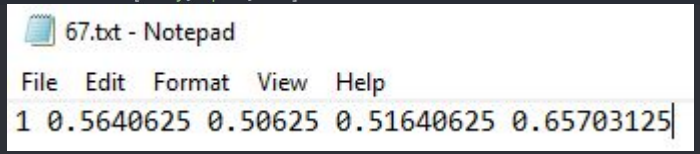


Dataset

- All images are annotated using Roboflow and split manually after download.
- YOLOv5 dataset is structured as three folders and data.yaml
  - Each of the three folders represents train, validation, and test dataset respectively and they contains an images folder and a labels folder.
    - The labels folder contains txt files correspond to images in the images folder
    - Each txt file contains following informations
      - Class
      - x
      - y
      - width
      - height
  - data.yaml contains paths to the three folders, number of classes, and names of classes.







Source Code

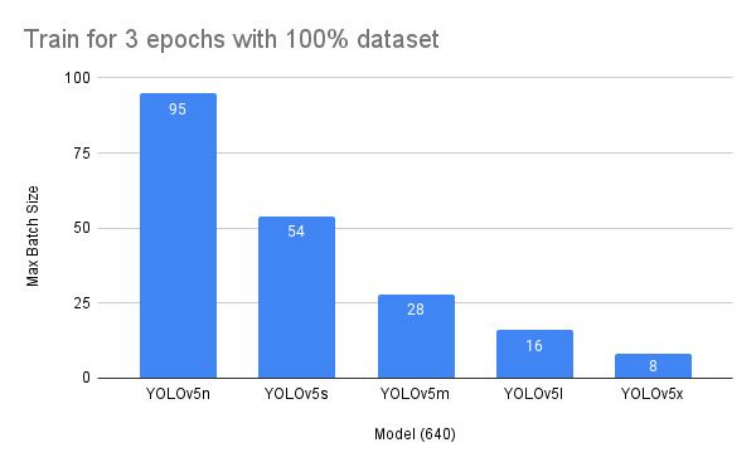
- videos.py: download videos and save them with specific names.
  - train.mp4: Tom & Jerry | Triple Trouble | Classic Cartoon Compilation | WB Kids
  - test.mp4: Tom & Jerry | Best Buddies 🐭🐱🐶 | Classic Cartoon Compilation | WB Kids
- extract.py: extract frames from train.mp4 without duplication and use frame index as the name.
- datasets.py: download dataset.zip and split them into multiple datasets of various sizes.
- yolov5.py: download YOLOv5 source code.
- train.py: train models using existing datasets.
- detect.py: use trained models to detect characters in test.mp4
- settings.json: contain parameters used by scripts.
- requirements.txt: contain all dependencies required to run the program.

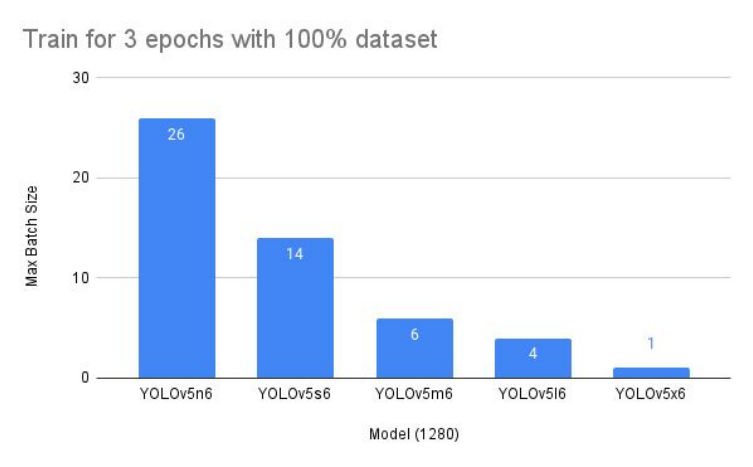
Hardware

- CPU: Ryzen 9 5900x
- Memory: 32GB @ 3600Mhz
- GPU: RTX 3060 12GB
- Storage: WD Blue Hard Drive

Experiment 1 - Hardware Limitation

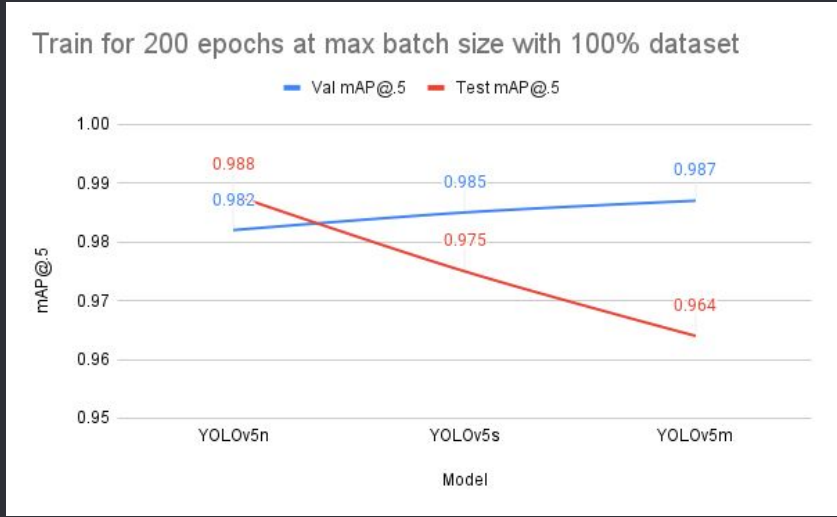
- Determine the maximum possible batch size of each model by training them for 3 epochs with a 100% dataset.
- Observation:
  - This hardware configuration can train nano, small, medium and large models of the 640p variant and a nano model of the 1280p variant.
  - However, since we plan to examine the performance difference between models being trained at max batch size vs those being trained at normalized batch size, we select nano, small, and medium models of the 640 variant for further testing.





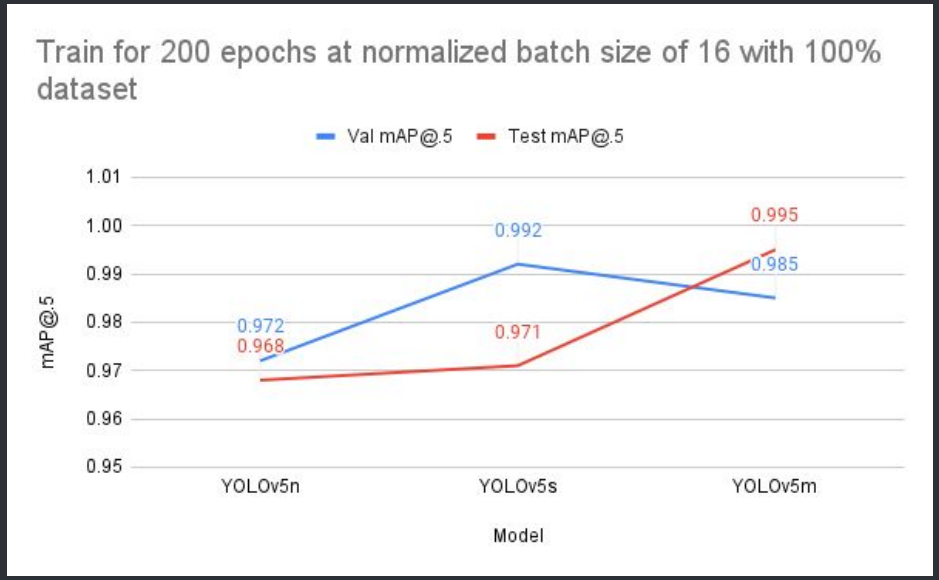
Experiment 2 - Train Selected Models at Max Batch Size

- Determine the performance of trained models when they are being trained for 200 epochs at max batch size with a 100% dataset.
- Observation:
  - Trained at max batch size causes instability in trained models where they failed to generalize.
  - Ideally, increasing complexity of a well-trained model should also increase its performance on the test dataset.



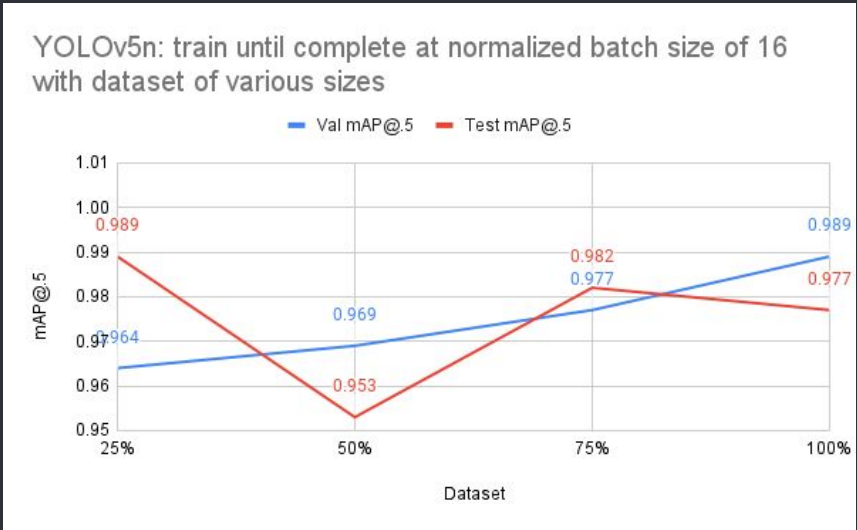
Experiment 3 - Train Selected Models at Normalized Batch Size

- Determine the performance of trained models when they are being trained for 200 epochs at a normalized batch size of 16 with a 100% dataset.
- Observation:
  - Unlike the previous case, there is a positive correlation between model complexity and model performance on the test dataset.
  - Thus, we decide on a normalized batch size for further testing.



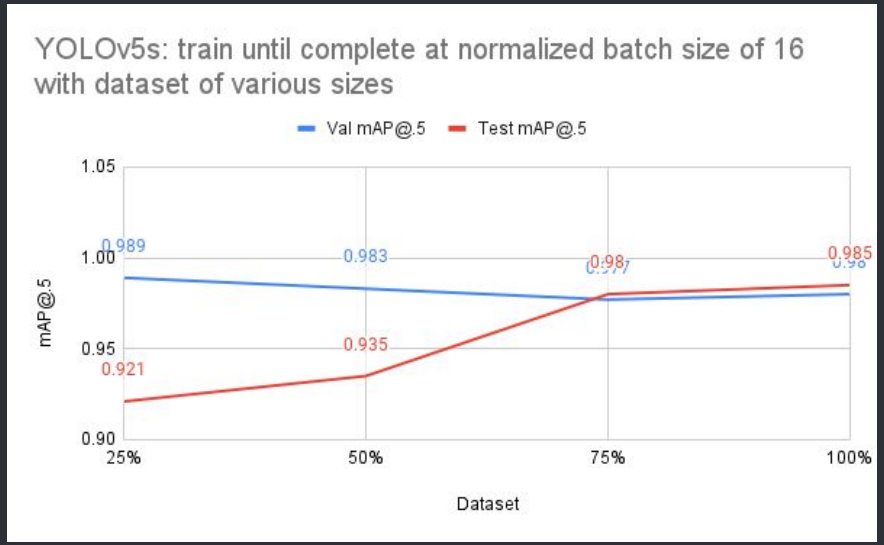
Experiment 4 - Nano Model

- Determine the performance of the nano model when it is being trained until completion at normalized batch size of 16 with various dataset sizes.
- Observation:
  - When we increase the size of dataset, the performance of the nano model on the test dataset fluctuate greatly.
  - This is a good indication that this model is too simple for our dataset.



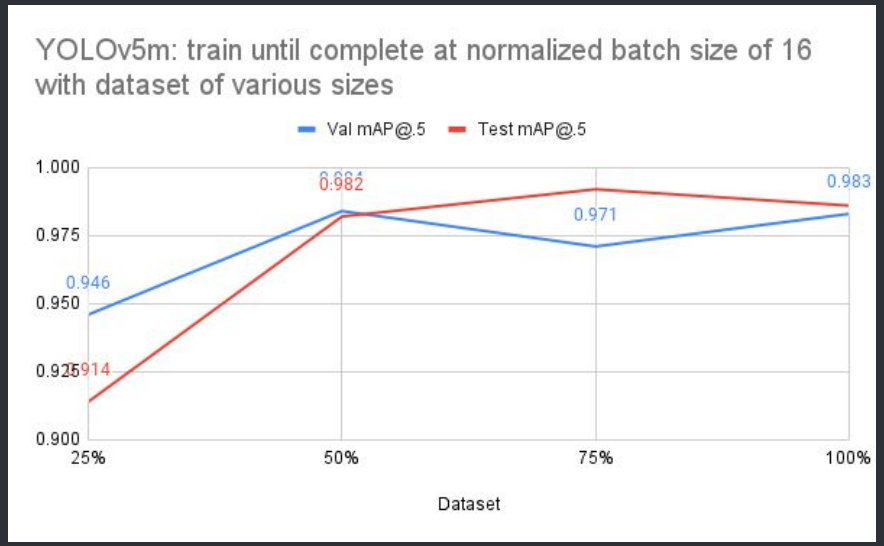
Experiment 4 - Small Model

- Determine the performance of the small model when it is being trained until completion at normalized batch size of 16 with various dataset sizes.
- Observation:
  - The small model performance better as we increase the size of the dataset which is a good indication that such model is suited for our dataset.



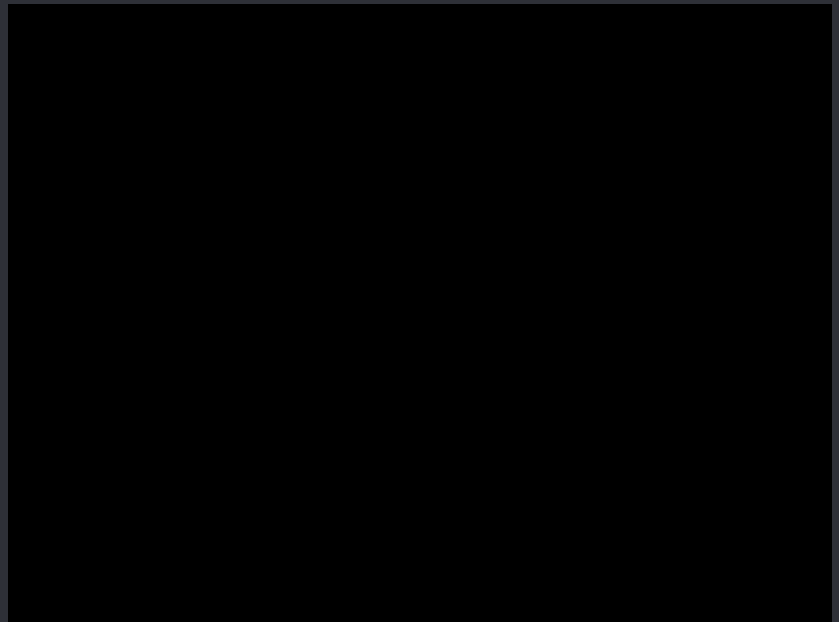
Experiment 4 - Medium Model

- Determine the performance of the medium model when it is being trained until completion at normalized batch size of 16 with various dataset sizes.
- Observation:
  - The medium model performance fluctuate on both validation and test datasets as we increase the size of the dataset.
  - This could indicate that our dataset is too small and thus, this model is unable to generalize well when it observe such a small dataset.



Detection Demonstrations

- Source: test.mp4
  - Tom & Jerry | Best Buddies 🐭🐱🐶 | Classic Cartoon Compilation | WB Kids
- Playlist: Detection Demo Videos



Links

- Source Code: https://github.com/sotheanith/Real-Time-Animated-Characters-Detection-and-Recognition-with-YOLOv5
- YOLOv5: https://github.com/ultralytics/yolov5
- Detection Demo Videos: https://drive.google.com/drive/folders/1IrzEbeN1YUsLuAcWPxh-CivSbFcn73Ns?usp=sharing
- Train Video: https://youtu.be/rilFfbm7j8k
- Test Video: https://youtu.be/cqyziA30whE
- Annotation Tools: https://roboflow.com/
- 640p Dataset: https://mega.nz/file/z3YCWBYC#n6Klmpr3XB6ula_WOSriem5W0gnNgEZk3tZBVm5wDQ8
- 1280p Dataset: https://mega.nz/file/uyAwFZaK#9lZAk6_Pn0W9yB40KlfZx7e5WjYgTjdzIVogt6qv1jA

References

- F. Lau, “Speed/accuracy trade-offs for modern convolutional object detectors,” *Medium*, 09-Mar-2017. [Online]. Available: https://medium.com/@phelixlau/speed-accuracy-trade-offs-for-modern-convolutional-object-detectors-bbad4e4e0718. [Accessed: 21-Nov-2021].
- G. Jocher, “ultralytics/yolov5: v6.0 - YOLOv5n 'Nano' models, Roboflow integration, TensorFlow export, OpenCV DNN support”. Zenodo, Oct. 12, 2021. doi: 10.5281/zenodo.5563715.
- H. M. K., “Backpropagation step by step,” *HMKCode*, 03-Nov-2019. [Online]. Available: https://hmkcode.com/ai/backpropagation-step-by-step/. [Accessed: 21-Nov-2021].
- J. Redmon and A. Farhadi, “Yolo9000: Better, faster, stronger,” *arXiv.org*, 25-Dec-2016. [Online]. Available: https://arxiv.org/abs/1612.08242. [Accessed: 21-Nov-2021].
- J. Redmon and A. Farhadi, “Yolov3: An incremental improvement,” *arXiv.org*, 08-Apr-2018. [Online]. Available: https://arxiv.org/abs/1804.02767. [Accessed: 21-Nov-2021].
- J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” *arXiv.org*, 09-May-2016. [Online]. Available: https://arxiv.org/abs/1506.02640. [Accessed: 21-Nov-2021].
- R. Grosse, “CSC321 lecture 4: Learning a ... - university of toronto,” *Computer Science University of Toronto*. [Online]. Available: http://www.cs.toronto.edu/~rgrosse/courses/csc321_2017/slides/lec4.pdf. [Accessed: 21-Nov-2021].
- S. An, “Convolutional Neural Networks,” *Sungtae's awesome homepage*, 09-Oct-2017. [Online]. Available: https://www.cc.gatech.edu/~san37/post/dlhc-cnn/. [Accessed: 21-Nov-2021].
- S. Ganesh, “What's the role of weights and bias in a neural network?,” *Medium*, 30-Jul-2020. [Online]. Available: https://towardsdatascience.com/whats-the-role-of-weights-and-bias-in-a-neural-network-4cf7e9888a0f?gi=e0eb90752e82. [Accessed: 21-Nov-2021].
- S. K, “Non-maximum suppression (NMS),” *Medium*, 01-Oct-2019. [Online]. Available: https://towardsdatascience.com/non-maximum-suppression-nms-93ce178e177c. [Accessed: 21-Nov-2021].
- S. R. Das, K. Mokashi, and R. Culkin, “Are markets truly efficient? experiments using deep learning algorithms for market movement prediction,” *MDPI*, 13-Sep-2018. [Online]. Available: https://www.mdpi.com/1999-4893/11/9/138. [Accessed: 21-Nov-2021].
- S. Raschka, “Gradient descent and stochastic gradient descent,” *Gradient Descent and Stochastic Gradient Descent - mlxtend*. [Online]. Available: http://rasbt.github.io/mlxtend/user_guide/general_concepts/gradient-optimization/. [Accessed: 21-Nov-2021].
- S. Yohanandan, “Map (mean average precision) might confuse you!,” *Medium*, 09-Jun-2020. [Online]. Available: https://towardsdatascience.com/map-mean-average-precision-might-confuse-you-5956f1bfa9e2. [Accessed: 21-Nov-2021].
- U. Almog, “Yolo V3 explained,” *Medium*, 09-Oct-2020. [Online]. Available: https://towardsdatascience.com/yolo-v3-explained-ff5b850390f. [Accessed: 21-Nov-2021].