Advanced Deep Learning in Computer Vision

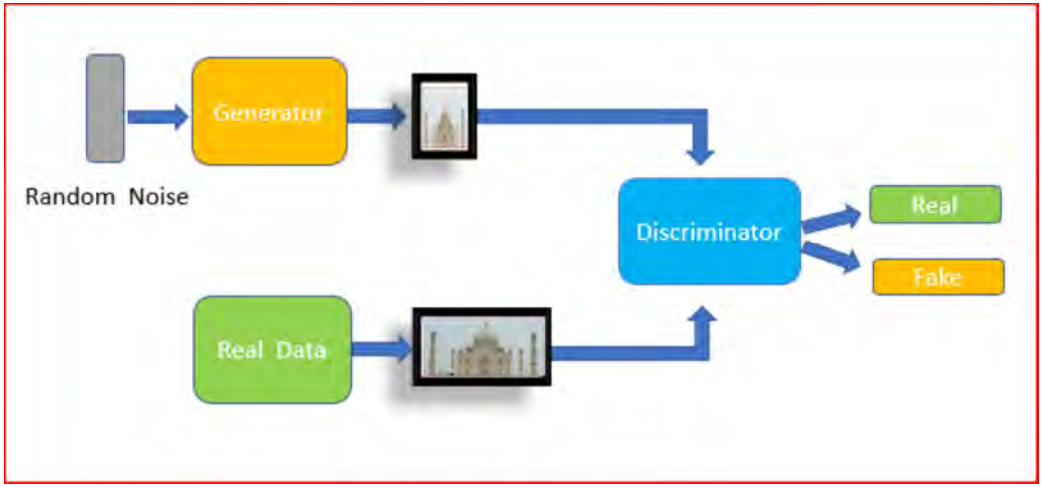
Day 3

GAN and specialized images

Materials:
http://bit.ly/ADLCV_Jan21

Programme

Day 1	Transfer Learning Activity – Transfer Learning Fine Tuning Object Detection and Localization Activity – Localization using Haar Cascades	More Object Detection and Localization Activity – Using YOLOv3 and SSD Annotation Activity – Annotation Hands-on
Day 2	Image Segmentation Activity – - OpenCV Mask RCNN - Keras Mask RCNN - Training Customized Mask RCNN	Activity: Using Customized Mask RCNN Face Detection and Recognition Activity: - Create Face Database - Face Recognition
Day 3	Advanced Generative Adversarial Network Activity - DCGAN for small color photographs - Conditional GAN	Customised Dataset with Yolo Activity - Thermal Images - Aerial Images



GAN

ADVANCED GENERATIVE ADVERSARIAL NETWORKS

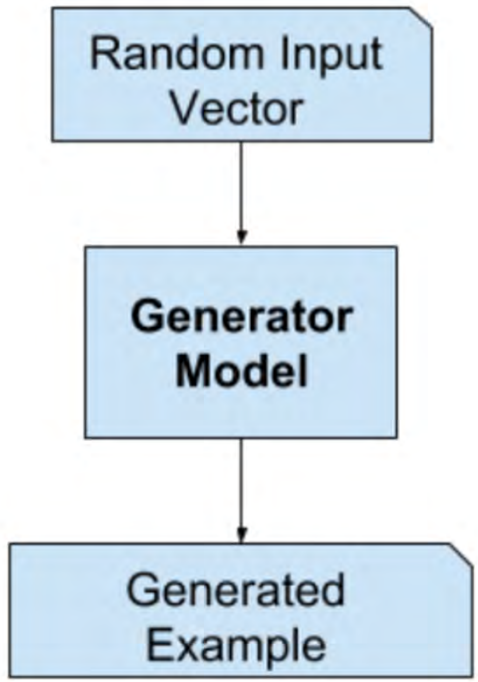
Image ref: https://medium.com/datadriveninvestor/generative-adversarial-network-gan-using-keras-ce1c05cfdfd3

Recap: What are GANs?

- Generative modelling is an unsupervised learning task in machine learning that involves automatically discovering and learning the regularities or patterns in input data in such a way that the model can be used to generate or output new examples that plausibly could have been drawn from the original dataset
- A model architecture for training a generative model, and it is most common to use deep learning models in this architecture
- GANs are a clever way of training a generative model by framing the problem as a supervised learning problem with two sub-models:
  - the **generator model** that we train to generate new examples, and
  - the **discriminator model** that tries to classify examples as either real (from the domain) or fake (generated)

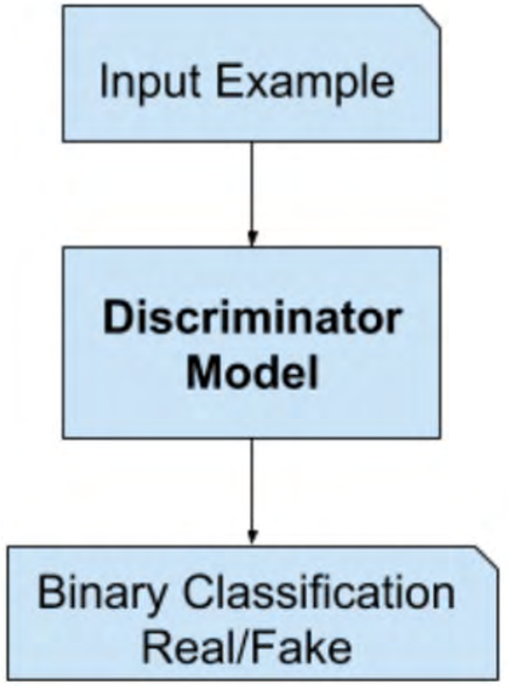
Recap: The Generator Model

- The generator model takes a fixed-length random vector as input and generates a sample in the domain, such as an image.
- This **vector space is referred to as a latent space**, or a vector space comprised of latent variables.
- In the case of GANs, the generator model applies meaning to points in a chosen latent space, such that new points drawn from the latent space can be provided to the generator model as input and used to generate new and different output examples.
- After training, the generator model is kept and used to generate new samples.



Recap: The Discriminator Model

- The discriminator model takes an example from the problem domain as input (real or generated) and predicts a binary class label of real or fake (generated).
- The real example comes from the training dataset.
- The generated examples are output by the generator model.
- The discriminator is a normal classification model.
- After the training process, the discriminator model is discarded as we are interested in the generator.

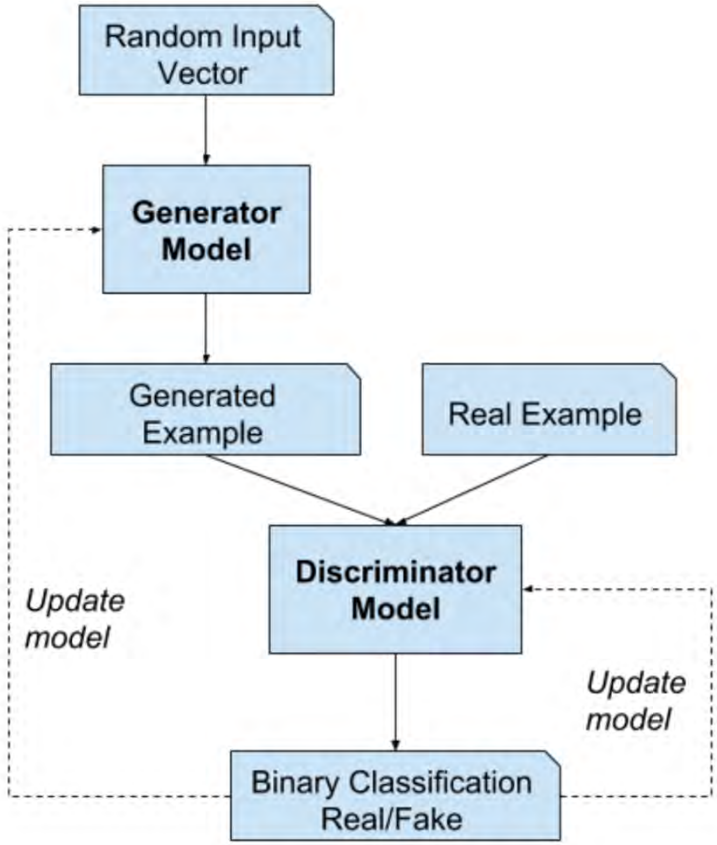


Recap: GANs as a Two Player Game

- Generative modelling is an unsupervised learning problem, although a clever property of the GAN architecture is that the training of the generative model is framed as a **supervised learning** problem.
- The two models, the generator and discriminator, are trained together.
- The generator generates a batch of samples, and these, along with real examples from the domain, are provided to the discriminator and classified as real or fake.
- ***The discriminator is then updated to get better at discriminating real and fake samples in the next round, and importantly, the generator is updated based on how well, or not, the generated samples fooled the discriminator.***
- In this way, the two models are competing against each other. They are adversarial in the game theory sense and are playing a zero-sum game.

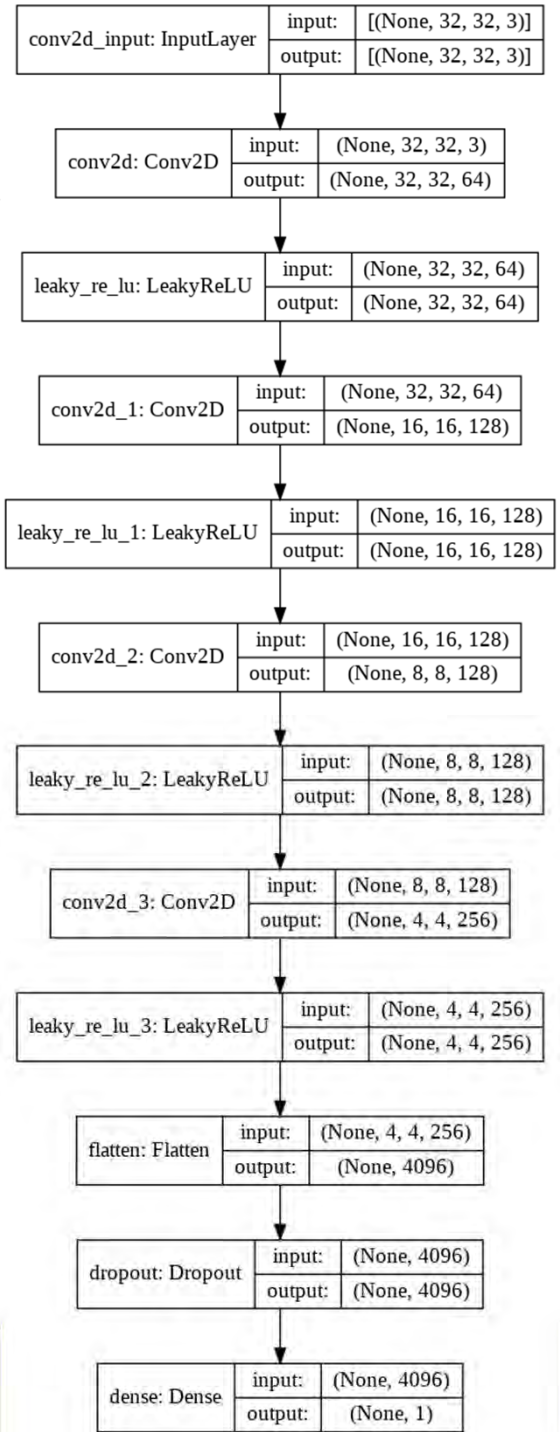
Recap: GANs as a Two Player Game

- At a limit, the generator generates perfect replicas from the input domain every time, and the discriminator cannot tell the difference and predicts unsure (e.g. 50% for real and fake) in every case. This is just an example of an idealized case; we do not need to get to this point to arrive at a useful generator model.



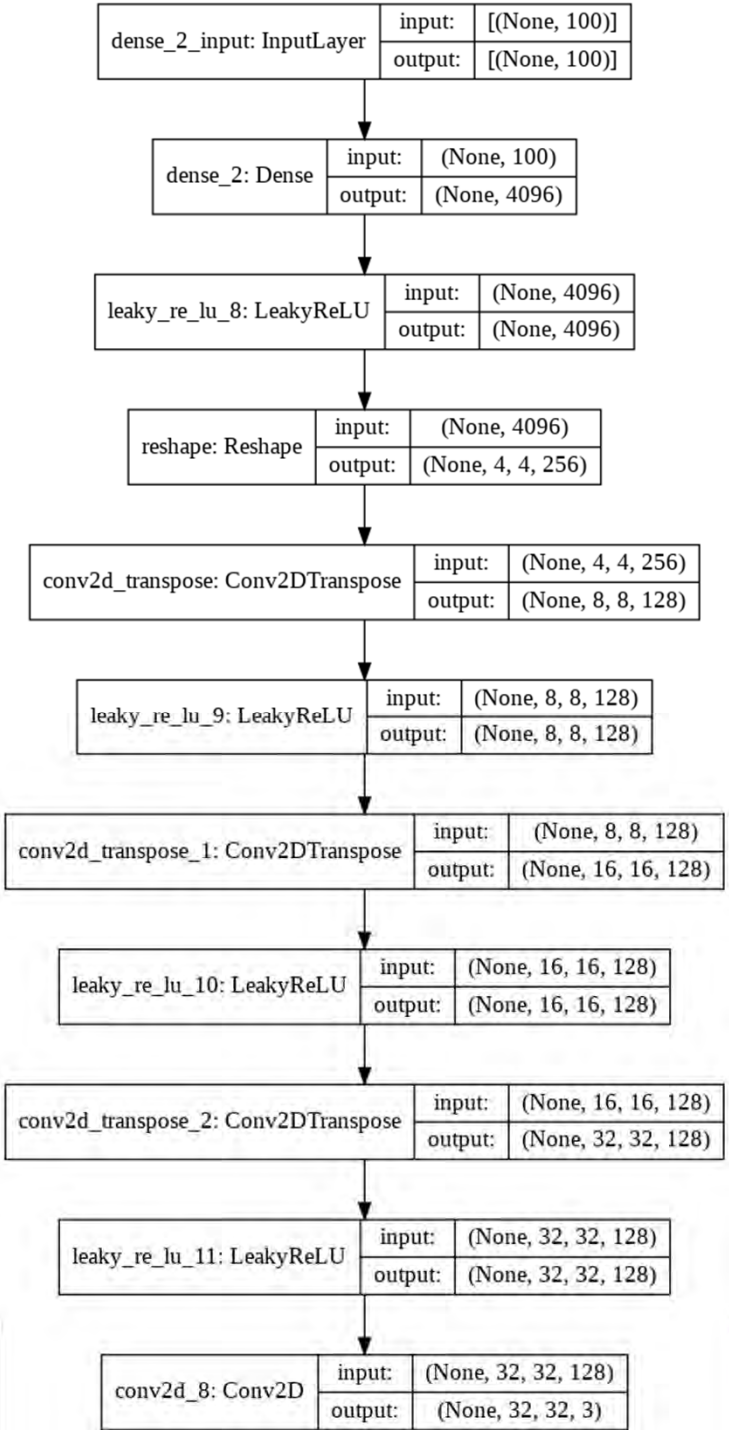
Discriminator

```
# define the standalone discriminator model
def define_discriminator(in_shape=(32,32,3)):
  model = Sequential()
  # normal
  model.add(Conv2D(64, (3,3), padding="same",
                   input_shape=in_shape))
  model.add(LeakyReLU(alpha=0.2))
  # downsample
  model.add(Conv2D(128, (3,3), strides=(2,2),
                   padding="same"))
  model.add(LeakyReLU(alpha=0.2))
  # downsample
  model.add(Conv2D(128, (3,3), strides=(2,2),
                   padding="same"))
  model.add(LeakyReLU(alpha=0.2))
  # downsample
  model.add(Conv2D(256, (3,3), strides=(2,2),
                   padding="same"))
  model.add(LeakyReLU(alpha=0.2))
  # classifier
  model.add(Flatten())
  model.add(Dropout(0.4))
  model.add(Dense(1, activation="sigmoid"))
  # compile model
  opt = Adam(lr=0.0002, beta_1=0.5)
  model.compile(loss="binary_crossentropy",
                optimizer=opt,
                metrics=["accuracy"])
  return model
```



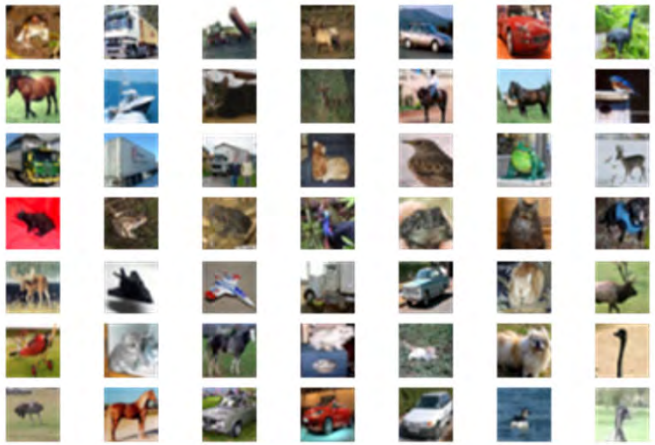
Generator Model

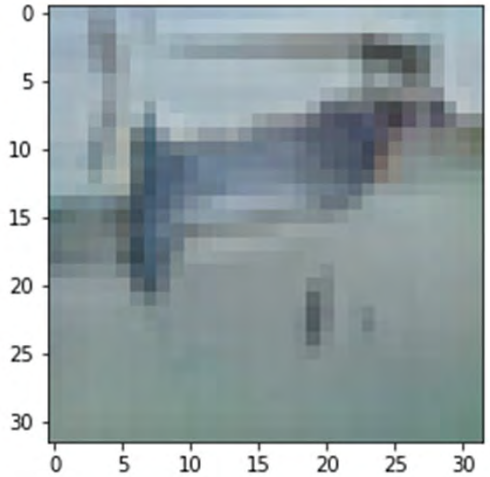
```python
# define the standalone generator model
def define_generator(latent_dim):
  model = Sequential()
  # foundation for 4x4 image
  n_nodes = 256 * 4 * 4
  model.add(Dense(n_nodes, input_dim=latent_dim))
  model.add(LeakyReLU(alpha=0.2))
  model.add(Reshape((4, 4, 256)))
  # upsample to 8x8
  model.add(Conv2DTranspose(128, (4,4),
                            strides=(2,2),
                            padding="same"))
  model.add(LeakyReLU(alpha=0.2))
  # upsample to 16x16
  model.add(Conv2DTranspose(128, (4,4),
                            strides=(2,2),
                            padding="same"))
  model.add(LeakyReLU(alpha=0.2))
  # upsample to 32x32
  model.add(Conv2DTranspose(128, (4,4),
                            strides=(2,2),
                            padding="same"))
  model.add(LeakyReLU(alpha=0.2))
  # output layer
  model.add(Conv2D(3, (3,3), activation="tanh",
                   padding="same"))
  return model
```



Activity 1 – DCGAN for small color photographs

CIFAR-10 : Small Object Photograph Dataset





Exercises:

Change Latent Space. Update the example to use a larger or smaller latent space and compare the quality of the results and speed of training.

Model Configuration. Update the model configuration to use deeper or more shallow discriminator and/or generator models, perhaps experiment with the UpSampling2D layers in the generator.

Share two photographs you created at http://bit.ly/activity_21

Step 1:
Watch and listen to the instructor's demonstration

15 mins

Step 2:
Work through the activities

45 mins

Individual Activity

30 Mins Break

Conditional GAN

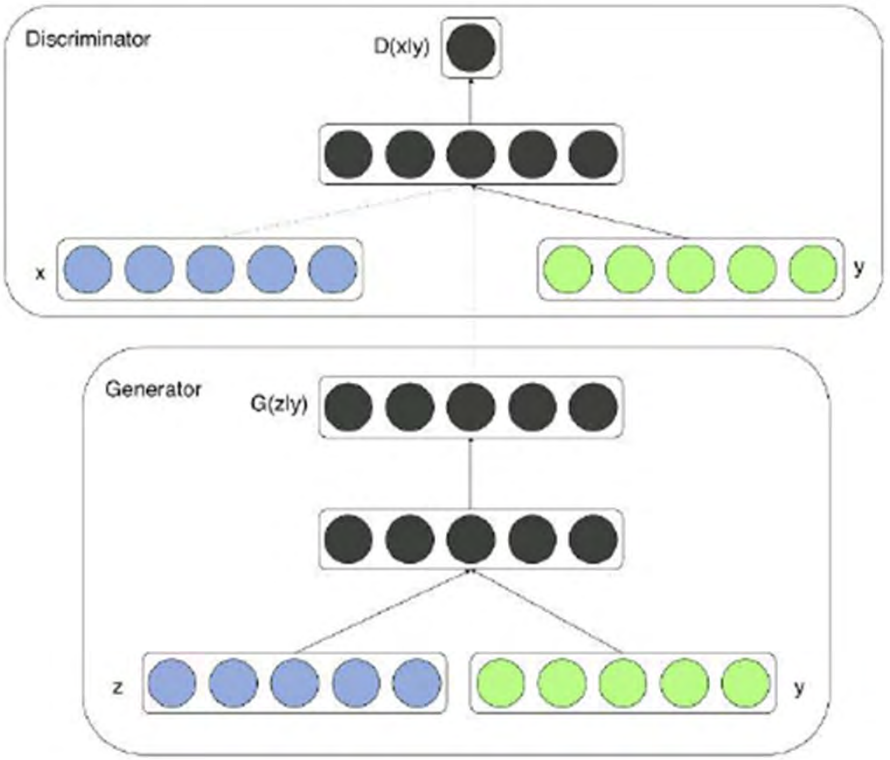
- Some datasets have additional information, such as a class label, and it is desirable to make use of this information
- There are two motivations for making use of the class label information in a GAN model.
  - Improve the GAN.
  - Targeted Image Generation.
- Additional information that is correlated with the input images, such as class labels, can be used to improve the GAN.
  - more stable training, faster training, and/or generated images that have better quality.
  - Class labels can also be used for the deliberate or targeted generation of images of a given type
- Generative adversarial nets can be extended to a conditional model if both the generator and discriminator are conditioned on some extra information y. [...] We can perform the conditioning by feeding y into the both the discriminator and generator as additional input layer. — Conditional Generative Adversarial Nets, 2014.

MNIST

- For example, in the case of MNIST, specific handwritten digits can be generated, such as the number 9; in the case of CIFAR-10, specific object photographs can be generated such as frogs; and in the case of the Fashion-MNIST dataset, specific items of clothing can be generated, such as dress.
- This type of model is called a Conditional Generative Adversarial Network, CGAN or cGAN for short. The cGAN was first described by Mehdi Mirza and Simon Osindero in their 2014 paper titled Conditional Generative Adversarial Nets. In the paper, the authors motivate the approach based on the desire to direct the image generation process of the generator model.

cGAN

- Their approach is demonstrated in the **MNIST handwritten digit dataset where the class labels are one hot encoded and concatenated with the input to both the generator and discriminator models**. The image below provides a summary of the model architecture.



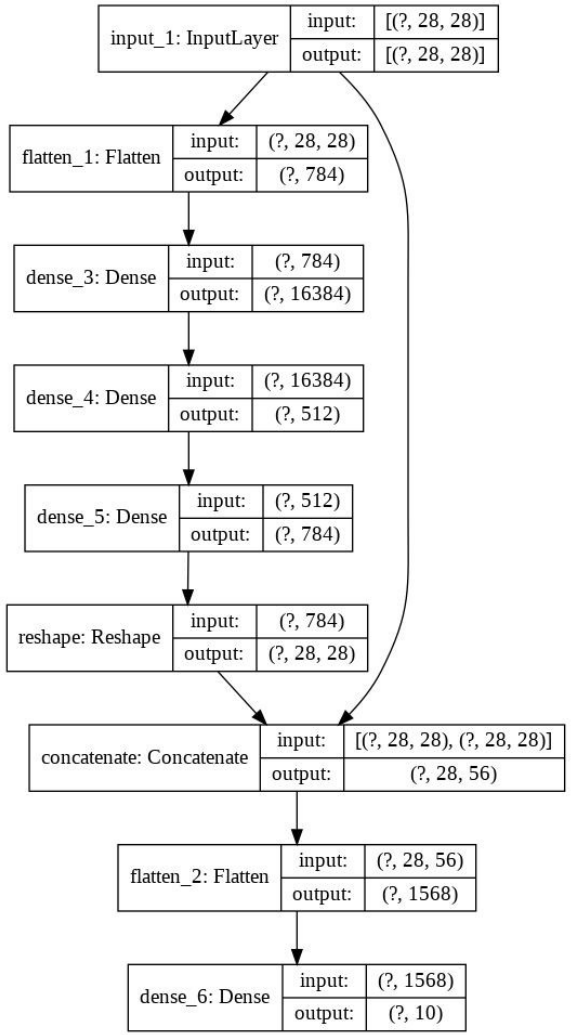
Keras Functional API

- Sequential and Functional are two ways to build Keras models.
- Sequential model is simplest type of model, a linear stock of layers. Easy to use , but it is most limited.
- In Sequential API we can not create a model that:
  - share layers
  - have branches
  - have multiple inputs
  - have multiple output
- Keras functional API allows us to build arbitrary graphs of layers

Example

```
1   input_ = keras.layers.Input(shape=[28, 28])
2   flatten = keras.layers.Flatten(input_shape=[28, 28])(input_)
3   hidden1 = keras.layers.Dense(2**14, activation="relu")(flatten)
4   hidden2 = keras.layers.Dense(512, activation='relu')(hidden1)
5   hidden3 = keras.layers.Dense(28*28, activation='relu')(hidden2)
6   reshap = keras.layers.Reshape((28, 28))(hidden3)
7   concat_ = keras.layers.Concatenate()([input_, reshap])
8   flatten2 = keras.layers.Flatten(input_shape=[28, 28])(concat_)
9   output = keras.layers.Dense(10, activation='softmax')(flatten2)
10  model = keras.Model(inputs=[input_], outputs=[output] )
```

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 28, 28)]     0
__________________________________________________________________________________________________
flatten_1 (Flatten)             (None, 784)          0           input_1[0][0]
__________________________________________________________________________________________________
dense_3 (Dense)                 (None, 16384)        12861440    flatten_1[0][0]
__________________________________________________________________________________________________
dense_4 (Dense)                 (None, 512)          8389120     dense_3[0][0]
__________________________________________________________________________________________________
dense_5 (Dense)                 (None, 784)          402192      dense_4[0][0]
__________________________________________________________________________________________________
reshape (Reshape)               (None, 28, 28)       0           dense_5[0][0]
__________________________________________________________________________________________________
concatenate (Concatenate)       (None, 28, 56)       0           input_1[0][0]
                                                                 reshape[0][0]
__________________________________________________________________________________________________
flatten_2 (Flatten)             (None, 1568)         0           concatenate[0][0]
__________________________________________________________________________________________________
dense_6 (Dense)                 (None, 10)           15690       flatten_2[0][0]
==================================================================================================
Total params: 21,668,442
Trainable params: 21,668,442
Non-trainable params: 0
__________________________________________________________________________________________________
```



https://hanifi.medium.com/sequential-api-vs-functional-api-model-in-keras-266823d7cd5e

Discriminator Model

```python
# define the standalone discriminator model
def define_discriminator(in_shape=(28,28,1)):
  model = Sequential()
  # downsample
  model.add(Conv2D(128, (3,3), strides=(2,2), padding="same", input_shape=in_shape))
  model.add(LeakyReLU(alpha=0.2))
  # downsample
  model.add(Conv2D(128, (3,3), strides=(2,2), padding="same"))
  model.add(LeakyReLU(alpha=0.2))
  # classifier
  model.add(Flatten())
  model.add(Dropout(0.4))
  model.add(Dense(1, activation="sigmoid"))
  # compile model
  opt = Adam(lr=0.0002, beta_1=0.5)
  model.compile(loss="binary_crossentropy", optimizer=opt, metrics=["accuracy"])
  return model
```

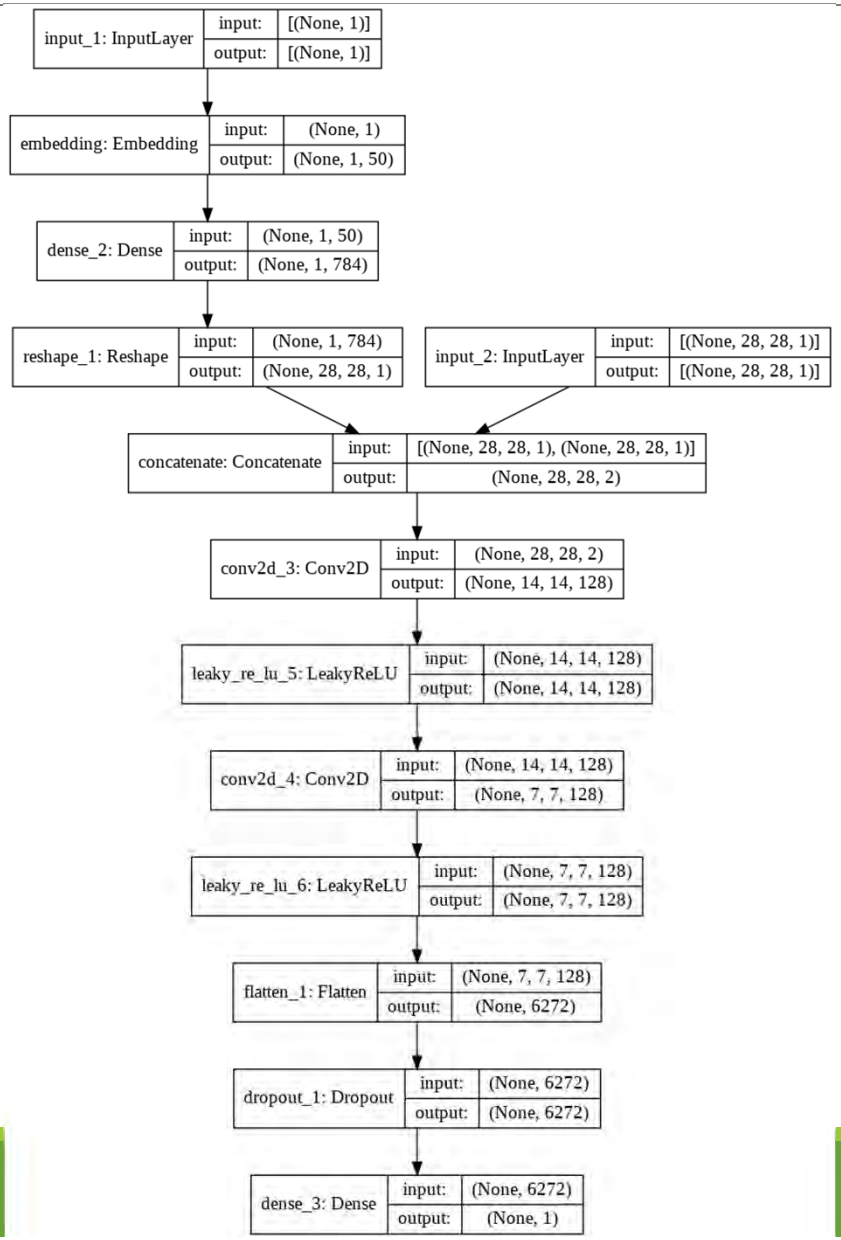
Unconditioned discriminator

Embedding layer - Turns positive integers (indexes) into dense vectors of fixed size.

```python
# define the standalone discriminator model
def define_discriminator(in_shape=(28,28,1), n_classes=10):
  # label input
  in_label = Input(shape=(1,))
  # embedding for categorical input
  li = Embedding(n_classes, 50)(in_label)
  # scale up to image dimensions with linear activation
  n_nodes = in_shape[0] * in_shape[1]
  li = Dense(n_nodes)(li)
  # reshape to additional channel
  li = Reshape((in_shape[0], in_shape[1], 1))(li)
  # image input
  in_image = Input(shape=in_shape)
  # concat label as a channel
  merge = Concatenate()([in_image, li])
  # downsample
  fe = Conv2D(128, (3,3), strides=(2,2), padding="same")(merge)
  fe = LeakyReLU(alpha=0.2)(fe)
  # downsample
  fe = Conv2D(128, (3,3), strides=(2,2), padding="same")(fe)
  fe = LeakyReLU(alpha=0.2)(fe)
  # flatten feature maps
  fe = Flatten()(fe)
  # dropout
  fe = Dropout(0.4)(fe)
  # output
  out_layer = Dense(1, activation="sigmoid")(fe)
  # define model
  model = Model([in_image, in_label], out_layer)
  # compile model
  opt = Adam(lr=0.0002, beta_1=0.5)
  model.compile(loss="binary_crossentropy", optimizer=opt, metrics=["accuracy"])
  return model
```

Conditioned discriminator

Discriminator Model



Generator Model

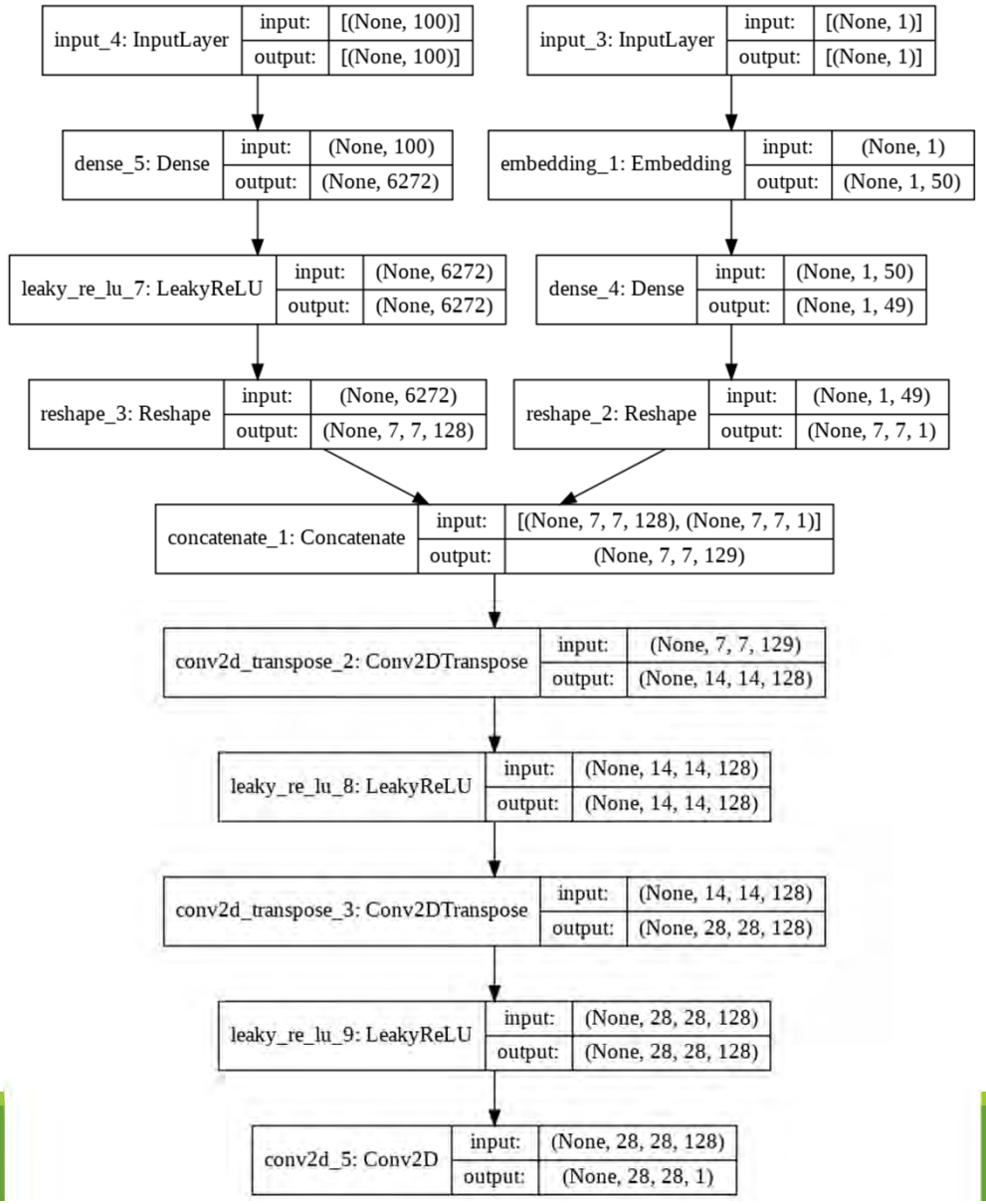
```python
# define the standalone generator model
def define_generator(latent_dim):
  model = Sequential()
  # foundation for 7x7 image
  n_nodes = 128 * 7 * 7
  model.add(Dense(n_nodes, input_dim=latent_dim))
  model.add(LeakyReLU(alpha=0.2))
  model.add(Reshape((7, 7, 128)))
  # upsample to 14x14
  model.add(Conv2DTranspose(128, (4,4), strides=(2,2), padding="same"))
  model.add(LeakyReLU(alpha=0.2))
  # upsample to 28x28
  model.add(Conv2DTranspose(128, (4,4), strides=(2,2), padding="same"))
  model.add(LeakyReLU(alpha=0.2))
  # generate
  model.add(Conv2D(1, (7,7), activation="tanh", padding="same"))
  return model
```

Unconditioned Generator

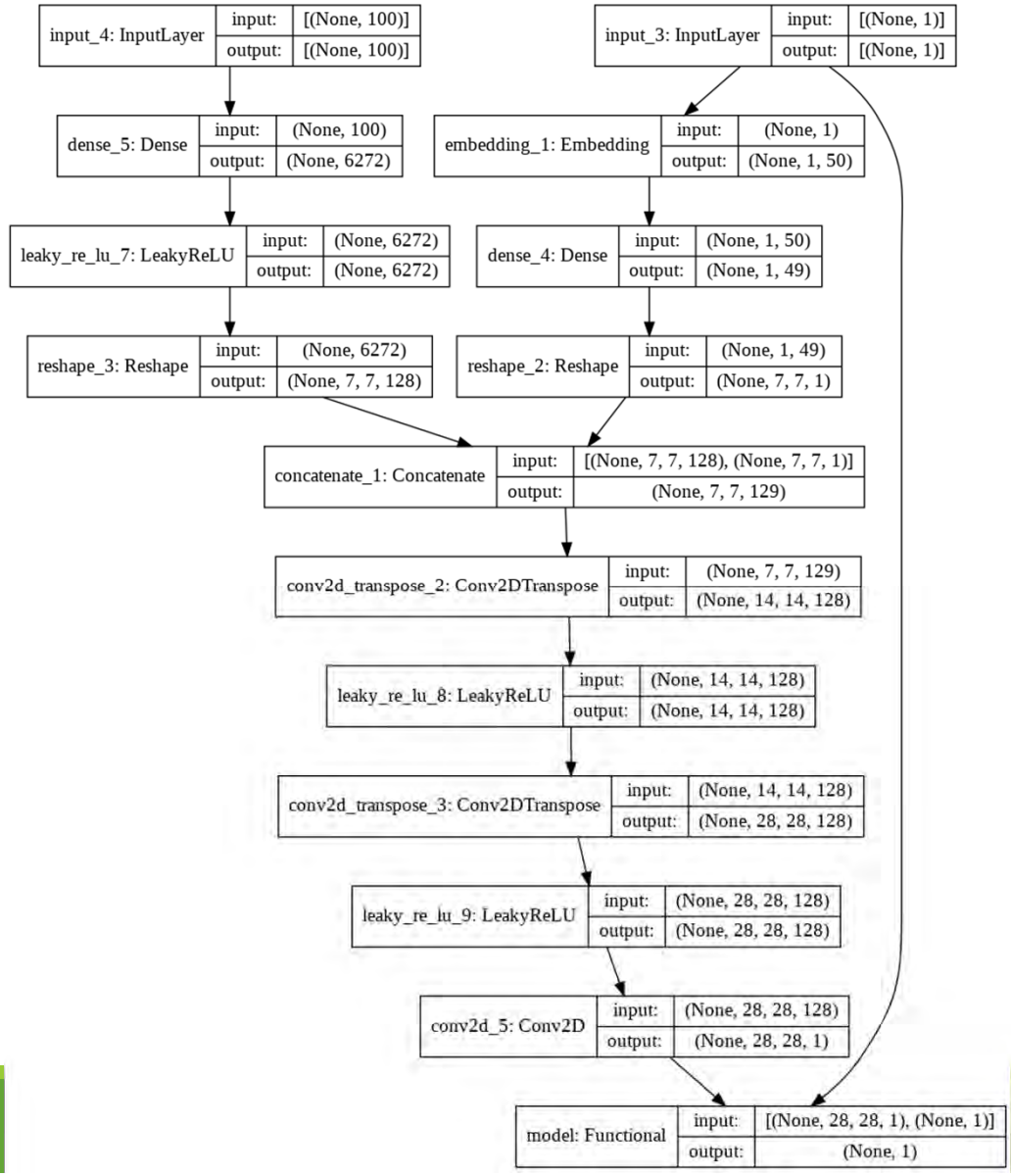
```python
# define the standalone generator model
def define_generator(latent_dim, n_classes=10):
  # label input
  in_label = Input(shape=(1,))
  # embedding for categorical input
  li = Embedding(n_classes, 50)(in_label)
  # linear multiplication
  n_nodes = 7 * 7
  li = Dense(n_nodes)(li)
  # reshape to additional channel
  li = Reshape((7, 7, 1))(li)
  # image generator input
  in_lat = Input(shape=(latent_dim,))
  # foundation for 7x7 image
  n_nodes = 128 * 7 * 7
  gen = Dense(n_nodes)(in_lat)
  gen = LeakyReLU(alpha=0.2)(gen)
  gen = Reshape((7, 7, 128))(gen)
  # merge image gen and label input
  merge = Concatenate()([gen, li])
  # upsample to 14x14
  gen = Conv2DTranspose(128, (4,4), strides=(2,2), padding="same")(merge)
  gen = LeakyReLU(alpha=0.2)(gen)
  # upsample to 28x28
  gen = Conv2DTranspose(128, (4,4), strides=(2,2), padding="same")(gen)
  gen = LeakyReLU(alpha=0.2)(gen)
  # output
  out_layer = Conv2D(1, (7,7), activation="tanh", padding="same")(gen)
  # define model
  model = Model([in_lat, in_label], out_layer)
  return model
```

Conditioned Generator

Generator Model



cGAN


input_4: InputLayer
input: [(None, 100)]
output: [(None, 100)]
input_3: InputLayer
input: [(None, 1)]
output: [(None, 1)]
dense_5: Dense
input: (None, 100)
output: (None, 6272)
embedding_1: Embedding
input: (None, 1)
output: (None, 1, 50)
leaky_re_lu_7: LeakyReLU
input: (None, 6272)
output: (None, 6272)
dense_4: Dense
input: (None, 1, 50)
output: (None, 1, 49)
reshape_3: Reshape
input: (None, 6272)
output: (None, 7, 7, 128)
reshape_2: Reshape
input: (None, 1, 49)
output: (None, 7, 7, 1)
concatenate_1: Concatenate
input: [(None, 7, 7, 128), (None, 7, 7, 1)]
output: (None, 7, 7, 129)
conv2d_transpose_2: Conv2DTranspose
input: (None, 7, 7, 129)
output: (None, 14, 14, 128)
leaky_re_lu_8: LeakyReLU
input: (None, 14, 14, 128)
output: (None, 14, 14, 128)
conv2d_transpose_3: Conv2DTranspose
input: (None, 14, 14, 128)
output: (None, 28, 28, 128)
leaky_re_lu_9: LeakyReLU
input: (None, 28, 28, 128)
output: (None, 28, 28, 128)
conv2d_5: Conv2D
input: (None, 28, 28, 128)
output: (None, 28, 28, 1)
model: Functional
input: [(None, 28, 28, 1), (None, 1)]
output: (None, 1)

Generating images

```
# load model
model = load_model(output_dir+"cgan_generator.h5")
# generate images
latent_points, labels = generate_latent_points(100, 100)
# specify labels
labels = asarray([x for _ in range(10) for x in range(10)])
# generate images
X = model.predict([latent_points, labels])
# scale from [-1,1] to [0,1]
X = (X + 1) / 2.0
# plot the result
save_plot(X, 10)
```

Activity 2 – Conditional GAN

Step 1:
Watch and listen to the instructor's demonstration

15 mins

Step 2:
Work through the activities

45 mins

Exercises:

Latent Space Size. Experiment by varying the size of the latent space and review the impact on the quality of generated images.

Embedding Size. Experiment by varying the size of the class label embedding, making it smaller or larger, and review the impact on the quality of generated images.

Alternate Architecture. Update the model architecture to concatenate the class label elsewhere in the generator and/or discriminator model, perhaps with different dimensionality, and review the impact on the quality of generated images.

Share two sets of fashion you created at http://bit.ly/activity_21

Individual Activity

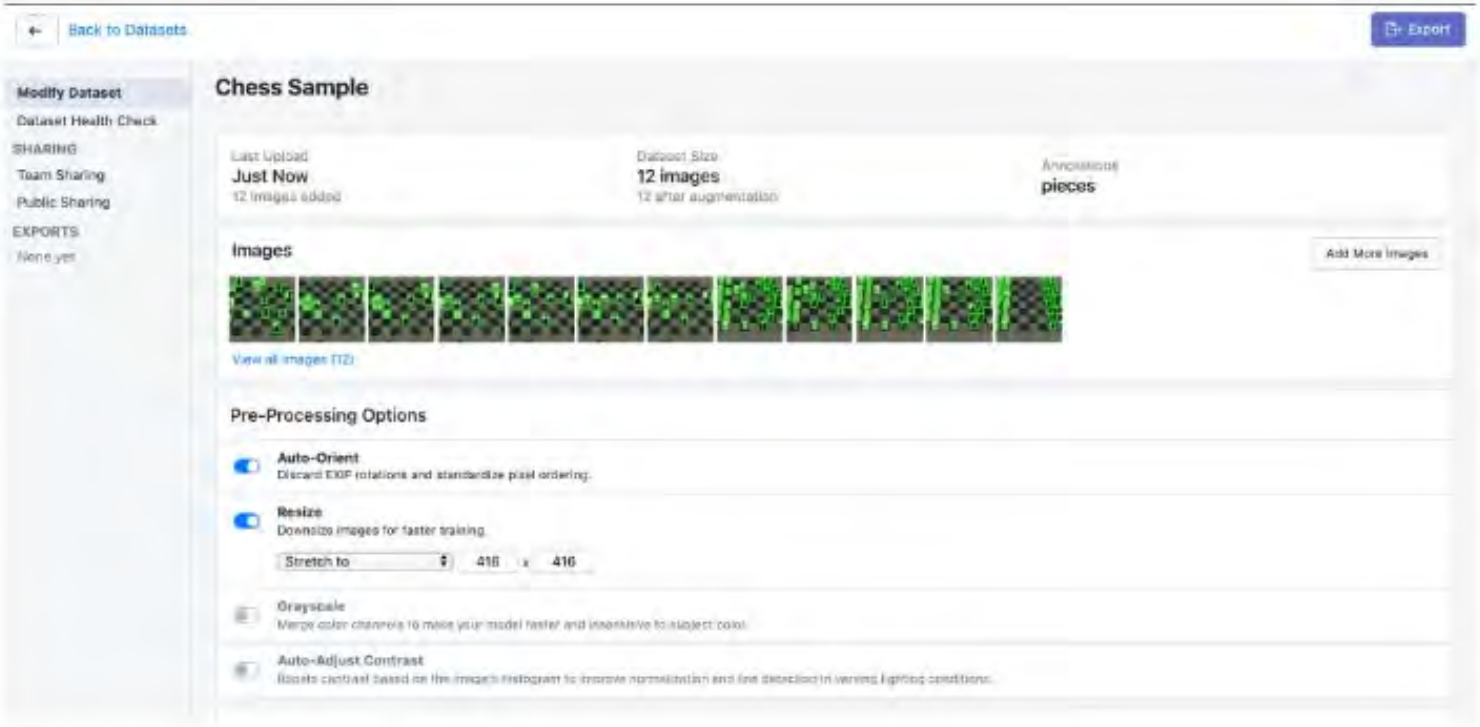
60 mins Lunch Break

Lunch break 12:20 - 13:30

Specialised Images

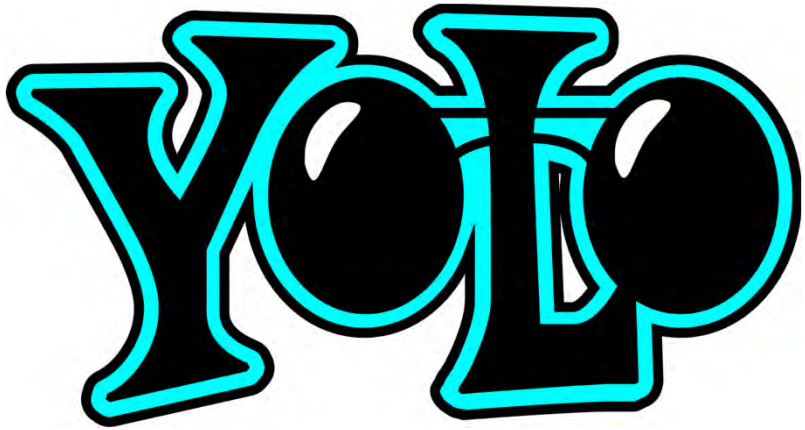
Roboflow introduction

- a web application designed to streamline image datasets and annotations for developers
- Supports and converts multiple annotation formats.



Yolo – You Only Look Once

- One of the most popular and most favorite algorithms for AI engineers. It always has been the first preference for real-time object detection.
- **Biggest advantages:**
  - Speed (45 frames per second — better than realtime)
  - Network understands generalized object representation (This allowed them to train the network on real world images and predictions on artwork was still fairly accurate).
  - faster version (with smaller architecture) — 155 frames per sec but is less accurate.
  - open source

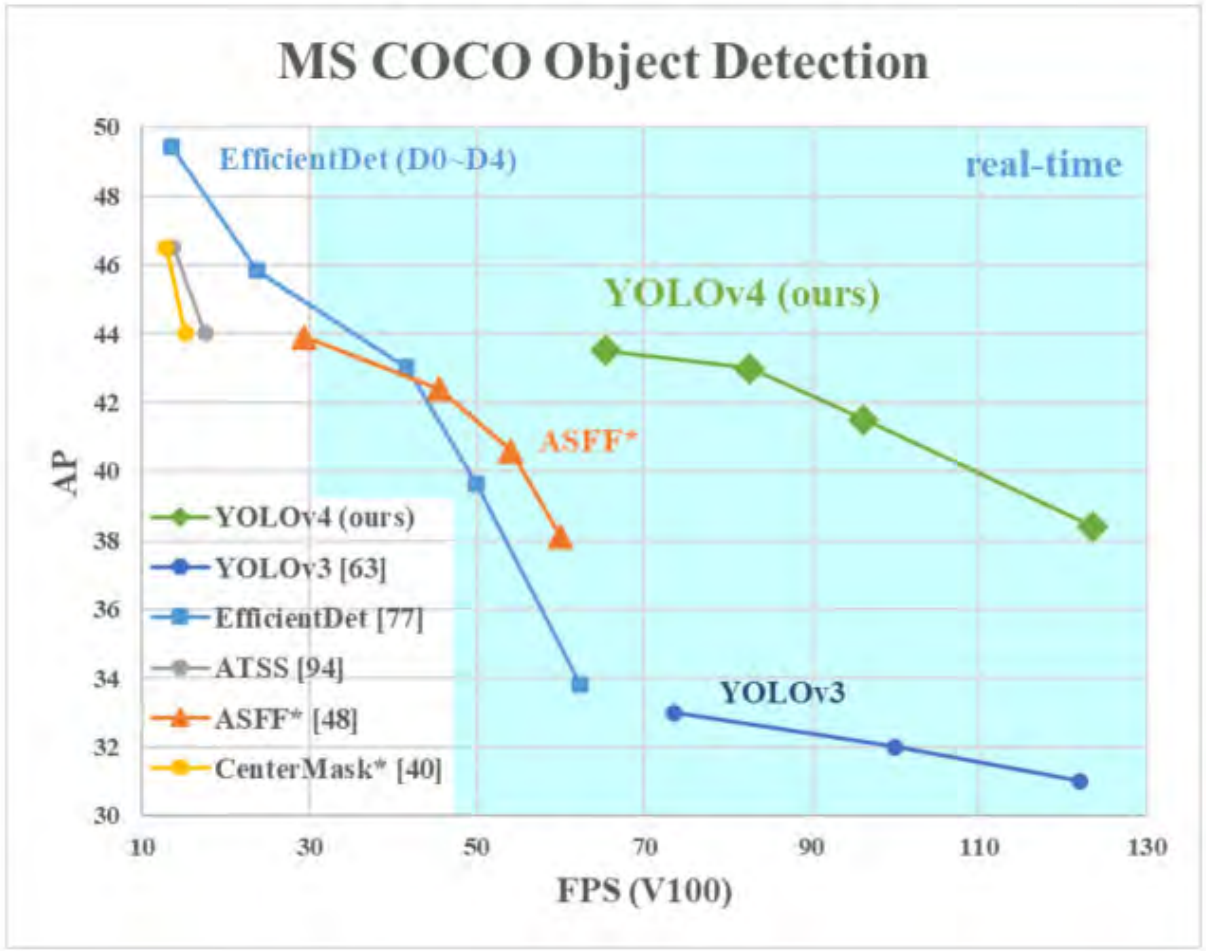


Yolo timeline

- YOLO v1 was introduced in May 2016 by Joseph Redmon with paper "You Only Look Once: Unified, Real-Time Object Detection." This was one of the biggest evolution in real-time object detection.
- In December 2017, Joseph introduced another version of YOLO with paper "YOLO9000: Better, Faster, Stronger." it was also known as YOLO 9000.
- YOLOv2 (2017) made a number of iterative improvements on top of YOLO including **BatchNorm, higher resolution, and anchor boxes**.
- After a year in April 2018, the most popular and stable version of YOLO was introduced. Joseph had a partner this time and they released YOLOv3 with paper "YOLOv3: An Incremental Improvement". Adds an **objectness score to bounding box prediction**, added connections to the backbone network layers, and made predictions at three separate levels of granularity to improve performance on smaller objects
- In April 2020, Alexey Bochkovskiy introduced YOLOv4 with paper "YOLOv4: Optimal Speed and Accuracy of Object Detection" Alexey is not the official author of previous versions of YOLO but Joseph and Ali took a step back from YOLO someone has to handle the era. YOLOv4 makes **realtime detection a priority** and conducts training on a single GPU
- In June 2020, just four days back another unofficial author Glenn Jocher released YOLOv5. There are lots of controversies about the selection of the name "YOLOv5" and other stuff. Glenn introduced PyTorch based version of YOLOv5 with exceptional improvements. Hence he has not released any official paper yet.

https://towardsdatascience.com/yolo-v5-is-here-b668ce2a4908

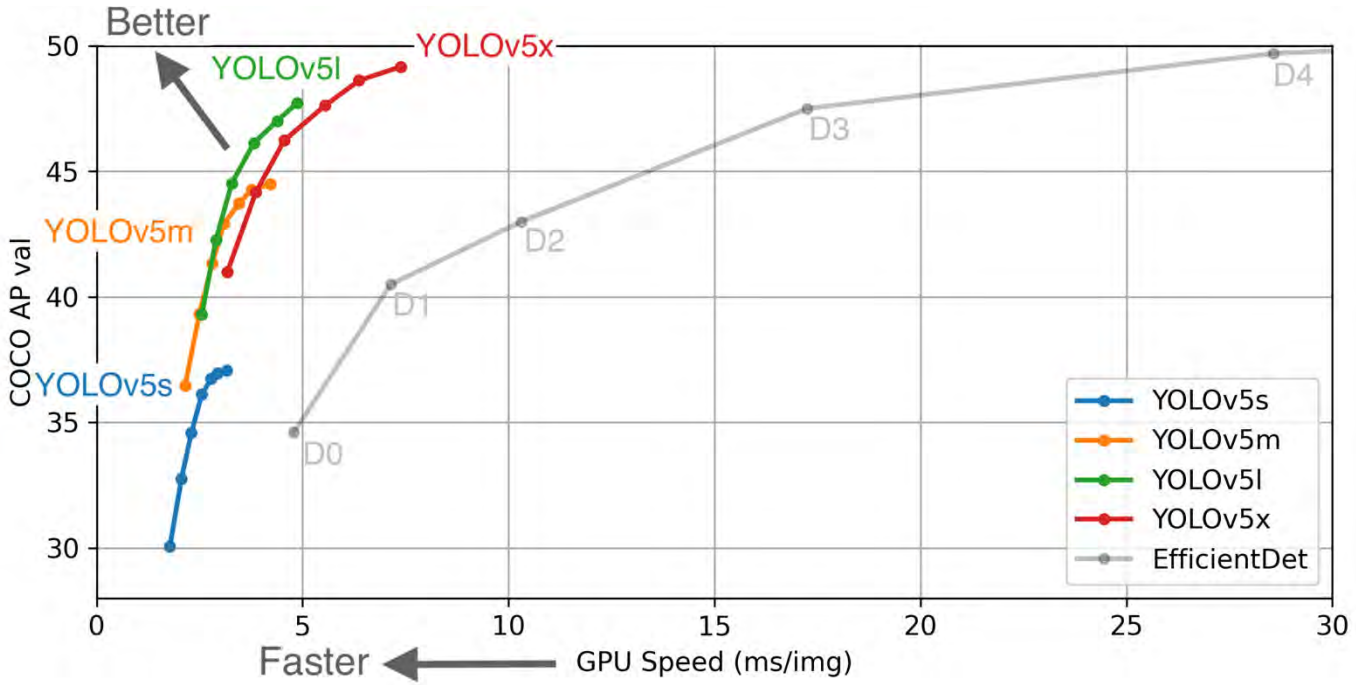
Yolo vs others



Source: YOLOv4 paper.

YOLOv5

- Ultralytics's open-source research into future object detection methods
- incorporates lessons learned and best practices evolved over training thousands of models on custom client datasets with their previous YOLO repository https://github.com/ultralytics/yolov3



Big idea

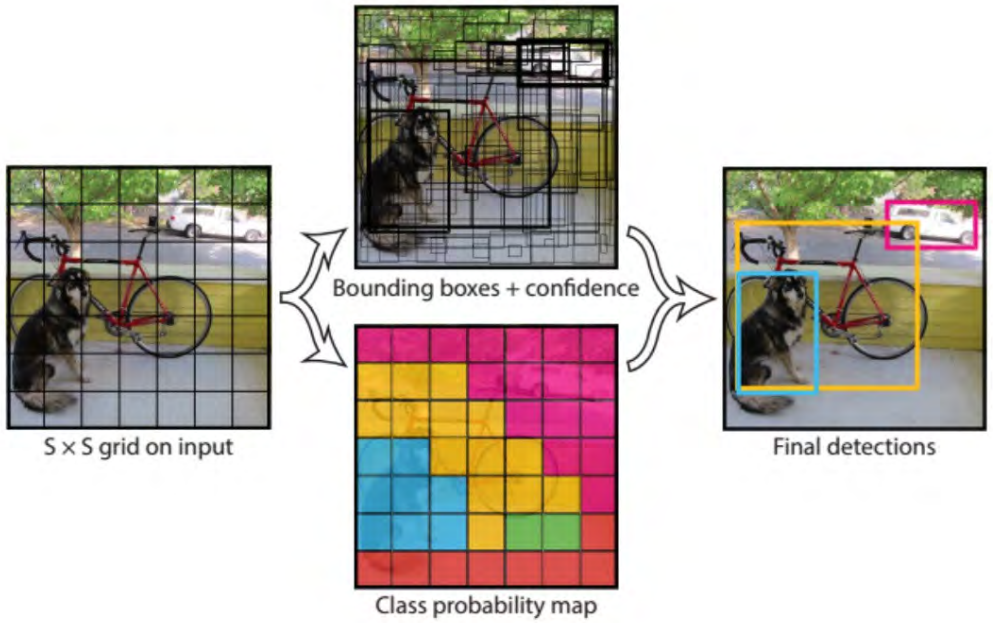
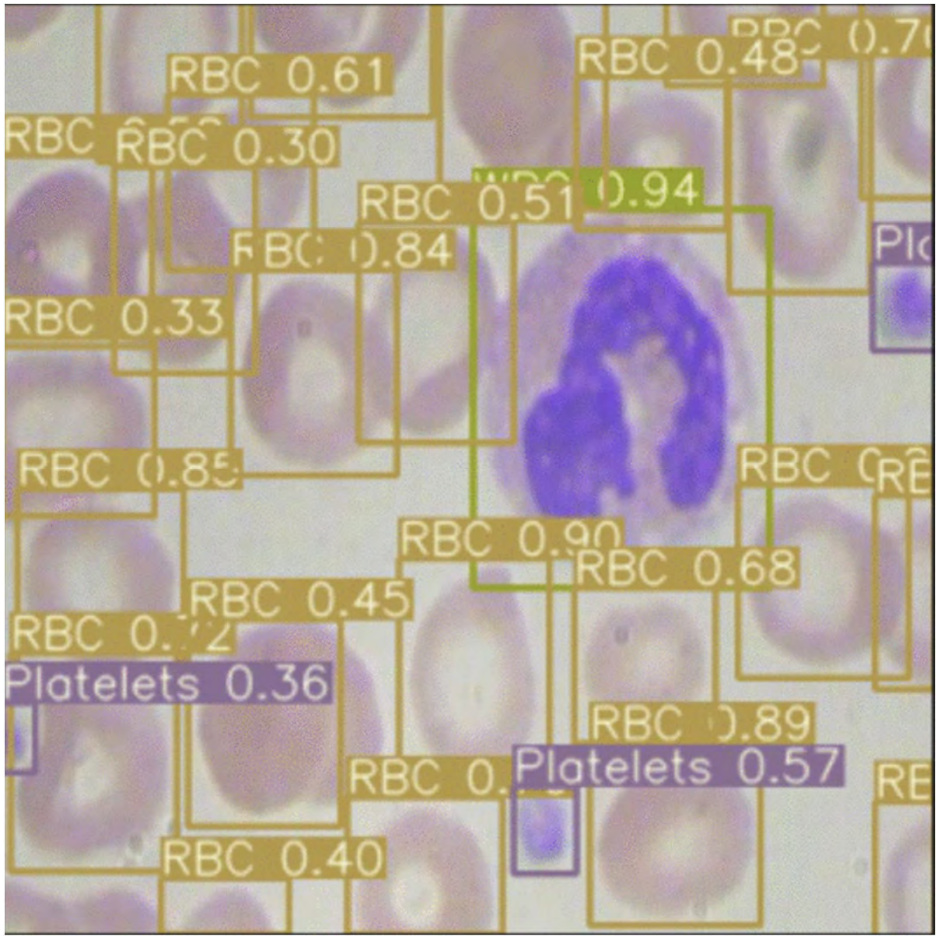


Figure 2: The Model. Our system models detection as a regression problem. It divides the image into an $S \times S$ grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an $S \times S \times (B * 5 + C)$ tensor.

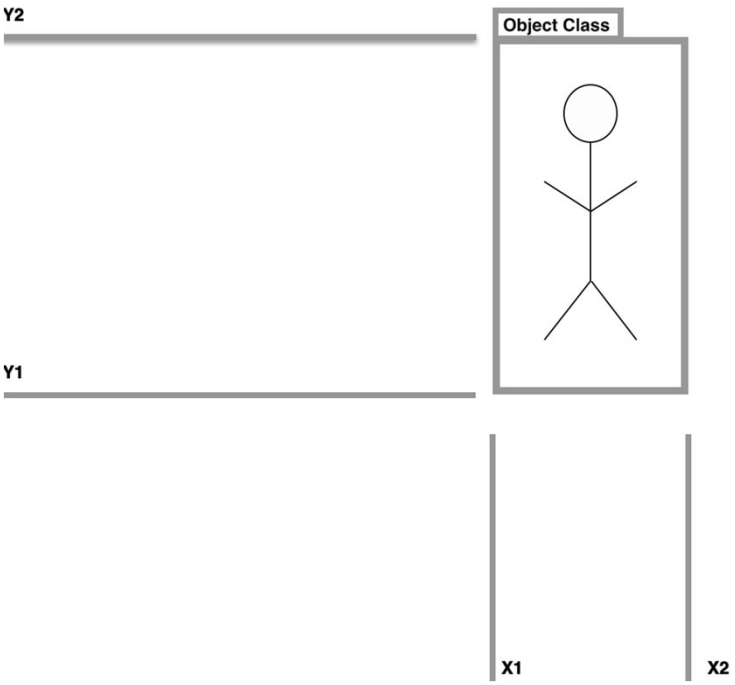
Evaluation

Which is better? Green or Yellow based on your intuition?



mAP

Object detection systems make predictions in terms of a bounding box and a class label.



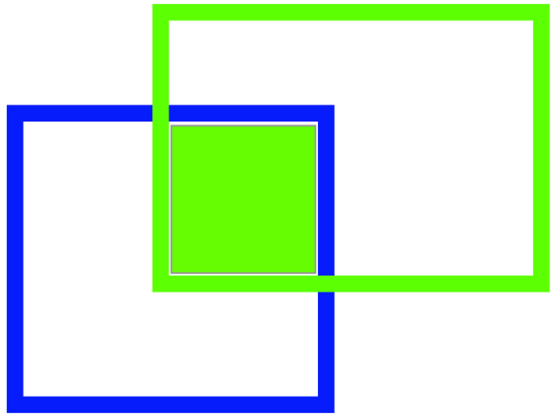
In practice, the bounding boxes predicted in the X1, X2, Y1, Y2 coordinates are sure to be off (even if slightly) from the ground truth label.

We know that we should count a bounding box prediction as incorrect if it is the wrong class, but where should we draw the line on bounding box overlap?

https://blog.roboflow.com/mean-average-precision/

mAP

The **Intersection over Union (IoU)** provides a metric to set this boundary at, measured as the amount of predicted bounding box that overlaps with the ground truth bounding box divided by the total area of both bounding boxes.



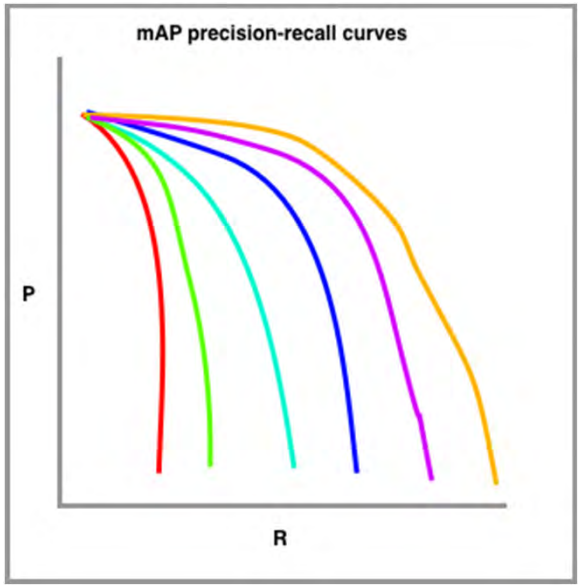
Picking the right single threshold for the IoU metric seems arbitrary. One researcher might justify a 60 percent overlap, and another is convinced that 75 percent seems more reasonable.

So why not have all of the thresholds considered in a single metric? Enter mAP.

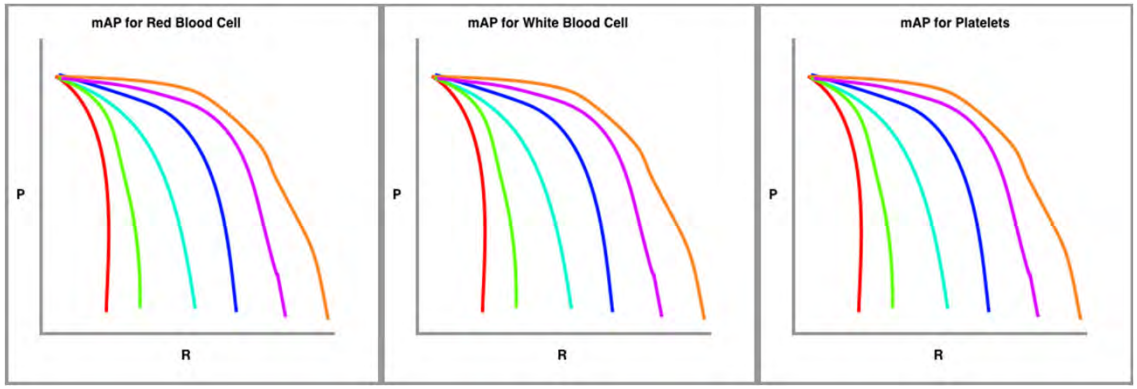
https://blog.roboflow.com/mean-average-precision/

mAP

- In order to calculate mAP, we draw a series of precision recall curves with the IoU threshold set at varying levels of difficulty.



In my sketch, red is drawn with the highest requirement for IoU (perhaps 90 percent) and the orange line is drawn with the most lenient requirement for IoU (perhaps 10 percent). The number of lines to draw is typically set by challenge.

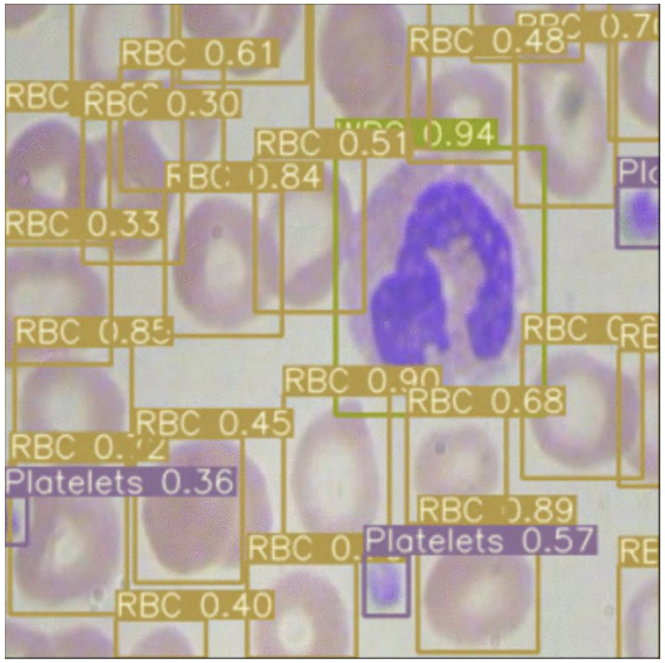


Then we draw these precision-recall curves for the dataset split out by class type.

The metric calculates the average precision (AP) for each class individually across all of the IoU thresholds. Then the metric averages the mAP for all classes to arrive at the final estimate.

Evaluation

Which is better? Green or Yellow?



<u>Evaluation of EfficientDet (green) on cell object detection:</u>

78.59% = Platelets AP

77.87% = RBC AP

96.47% = WBC AP

mAP = 84.31%

<u>Evaluation of YOLOv3 (yellow) on cell object detection:</u>

72.15% = Platelets AP

74.41% = RBC AP

95.54% = WBC AP

mAP = 80.70%

Is your intuition correct?

Steps:

- Install YOLOv5 dependencies
- Download Custom YOLOv5 Object Detection Data
- Define YOLOv5 Model Configuration and Architecture
- Train a custom YOLOv5 Detector
- Evaluate YOLOv5 performance
- Visualize YOLOv5 training data
- Run YOLOv5 Inference on test images
- Export Saved YOLOv5 Weights for Future Inference

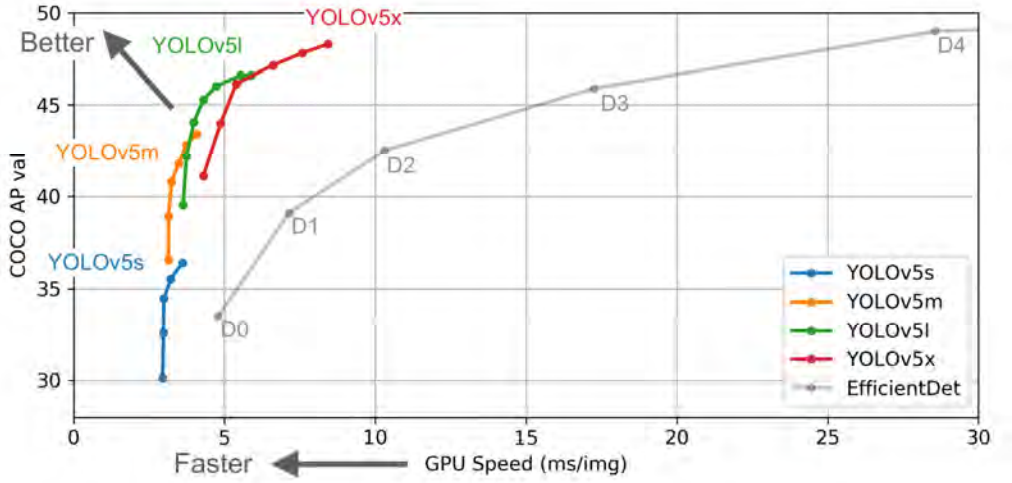
Data yaml

- Before training you need to modify the **YAML** file which specifies the **location** or path of the training folder and validation folder,
- and also information on the **names** and number of **classes**.

```
train: ../train/images
val: ../valid/images

nc: 2
names: ['dog', 'person']
```

Model yaml

- option to pick any of the four YOLOv5 models which includes:
  - yolov5-s
  - yolov5-m
  - yolov5-l
  - yolov5-x




```
# parameters
nc: 1 # number of classes
depth_multiple: 0.33  # model depth multiple
width_multiple: 0.50  # layer channel multiple


# anchors
anchors:
  - [10,13, 16,30, 33,23]  # P3/8
  - [30,61, 62,45, 59,119]  # P4/16
  - [116,90, 156,198, 373,326]  # P5/32
```

YOLOv5 model description

Small	Medium	Large	XLarge
YOLOv5s	**YOLOv5m**	**YOLOv5l**	**YOLOv5x**
14 MB_{FP16} 2.2 ms_{V100} 36.8 mAP_{COCO}	41 MB_{FP16} 2.9 ms_{V100} 44.5 mAP_{COCO}	90 MB_{FP16} 3.8 ms_{V100} 48.1 mAP_{COCO}	168 MB_{FP16} 6.0 ms_{V100} 50.1 mAP_{COCO}

Model	size	AP^{val}	AP^{test}	AP_{50}	$Speed_{V100}$	FPS_{V100}	params	GFLOPS
YOLOv5s	640	36.8	36.8	55.6	**2.2ms**	**455**	7.3M	17.0
YOLOv5m	640	44.5	44.5	63.1	2.9ms	345	21.4M	51.3
YOLOv5l	640	48.1	48.1	66.4	3.8ms	264	47.0M	115.4
YOLOv5x	640	**50.1**	**50.1**	**68.7**	6.0ms	167	87.7M	218.8
YOLOv5x + TTA	832	**51.9**	**51.9**	**69.6**	24.9ms	40	87.7M	1005.3

https://pytorch.org/hub/ultralytics_yolov5/

Changing model

Step 1 – show the content of the original template

```
#this is the model configuration we will use for our tutorial
%cat /content/yolov5/models/yolov5x.yaml
```

```
# parameters
nc: 80  # number of classes
depth_multiple: 1.33  # model depth multiple
width_multiple: 1.25  # layer channel multiple

# anchors
anchors:
  - [10,13, 16,30, 33,23]  # P3/8
  - [30,61, 62,45, 59,119]  # P4/16
  - [116,90, 156,198, 373,326]  # P5/32

# YOLOv5 backbone
backbone:
  # [from, number, module, args]
  [[-1, 1, Focus, [64, 3]],  # 0-P1/2
   [-1, 1, Conv, [128, 3, 2]],  # 1-P2/4
   [-1, 3, C3, [128]],
   [-1, 1, Conv, [256, 3, 2]],  # 3-P3/8
   [-1, 9, C3, [256]],
   [-1, 1, Conv, [512, 3, 2]],  # 5-P4/16
   [-1, 9, C3, [512]],
   [-1, 1, Conv, [1024, 3, 2]],  # 7-P5/32
   [-1, 1, SPP, [1024, [5, 9, 13]]],
   [-1, 3, C3, [1024, False]],  # 9
  ]

# YOLOv5 head
head:
  [[-1, 1, Conv, [512, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 6], 1, Concat, [1]],  # cat backbone P4
   [-1, 3, C3, [512, False]],  # 13

   [-1, 1, Conv, [256, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 4], 1, Concat, [1]],  # cat backbone P3
   [-1, 3, C3, [256, False]],  # 17 (P3/8-small)

   [-1, 1, Conv, [256, 3, 2]],
   [[-1, 14], 1, Concat, [1]],  # cat head P4
   [-1, 3, C3, [512, False]],  # 20 (P4/16-medium)

   [-1, 1, Conv, [512, 3, 2]],
   [[-1, 10], 1, Concat, [1]],  # cat head P5
   [-1, 3, C3, [1024, False]],  # 23 (P5/32-large)

   [[17, 20, 23], 1, Detect, [nc, anchors]],  # Detect(P3, P4, P5)
  ]
```

Step 2 – Copy the content to the writetemplate cell

```
%%writetemplate /content/yolov5/models/custom_yolov5x.yaml

# parameters
nc: {num_classes}  # number of classes
depth_multiple: 1.33  # model depth multiple
width_multiple: 1.25  # layer channel multiple

# anchors
anchors:
  - [10,13, 16,30, 33,23]  # P3/8
  - [30,61, 62,45, 59,119]  # P4/16
  - [116,90, 156,198, 373,326]  # P5/32

# YOLOv5 backbone
backbone:
  # [from, number, module, args]
  [[-1, 1, Focus, [64, 3]],  # 0-P1/2
   [-1, 1, Conv, [128, 3, 2]],  # 1-P2/4
   [-1, 3, C3, [128]],
   [-1, 1, Conv, [256, 3, 2]],  # 3-P3/8
   [-1, 9, C3, [256]],
   [-1, 1, Conv, [512, 3, 2]],  # 5-P4/16
   [-1, 9, C3, [512]],
   [-1, 1, Conv, [1024, 3, 2]],  # 7-P5/32
   [-1, 1, SPP, [1024, [5, 9, 13]]],
   [-1, 3, C3, [1024, False]],  # 9
  ]

# YOLOv5 head
head:
  [[-1, 1, Conv, [512, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 6], 1, Concat, [1]],  # cat backbone P4
   [-1, 3, C3, [512, False]],  # 13

   [-1, 1, Conv, [256, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 4], 1, Concat, [1]],  # cat backbone P3
   [-1, 3, C3, [256, False]],  # 17 (P3/8-small)
```

Training

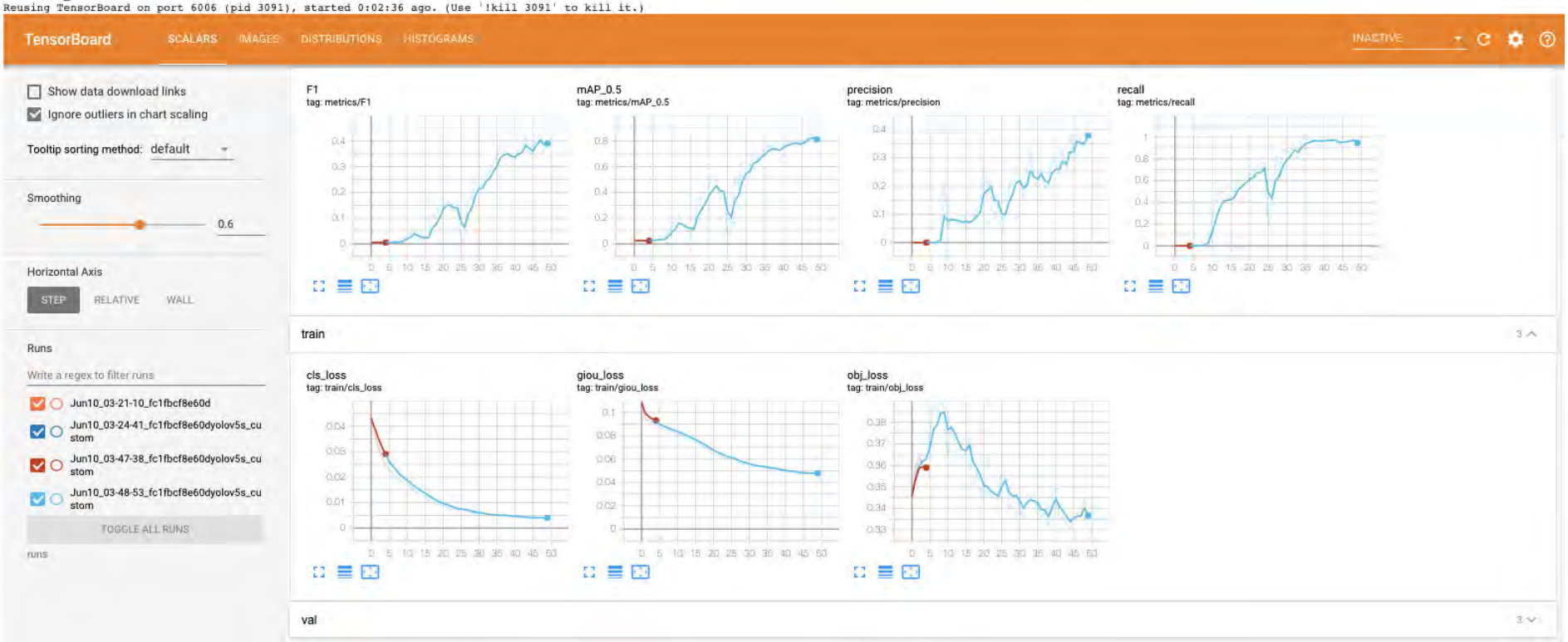
- Use training script provided
- E.g.

```
python train.py --img 416 --batch 16 --epochs 1000 --data '../data.yaml' --cfg ./models/custom_yolov5s.yaml --weights '' --name yolov5s_results  --cache

python train.py --img 640 --batch 1 --epochs 30 --data ./data/data.yaml --cfg ./models/custom_yolov5s.yaml --weights yolov5s.pt --device cpu
```

Remember to use the correct model definition yaml file

Evaluation

```
[ ] # Start tensorboard
    # Launch after you have run training
    # logs save in the folder "runs"
    %load_ext tensorboard
    %tensorboard --logdir runs

    The tensorboard extension is already loaded. To reload it, use:
      %reload_ext tensorboard
    Reusing TensorBoard on port 6006 (pid 3091), started 0:02:36 ago. (Use '!kill 3091' to kill it.)
```



You want to take the trained model weights at the point where the validation mAP reaches its highest.

Inference

- we take our trained model and make inference on test images.
- For inference we invoke those weights along with a **conf** specifying model confidence (higher confidence required makes less predictions), and a inference **source**.
- **source** can accept a directory of images, individual images, video files, and also a device's webcam port. For source, I have moved mytest1/*jpg to test_infer/.

E.g.

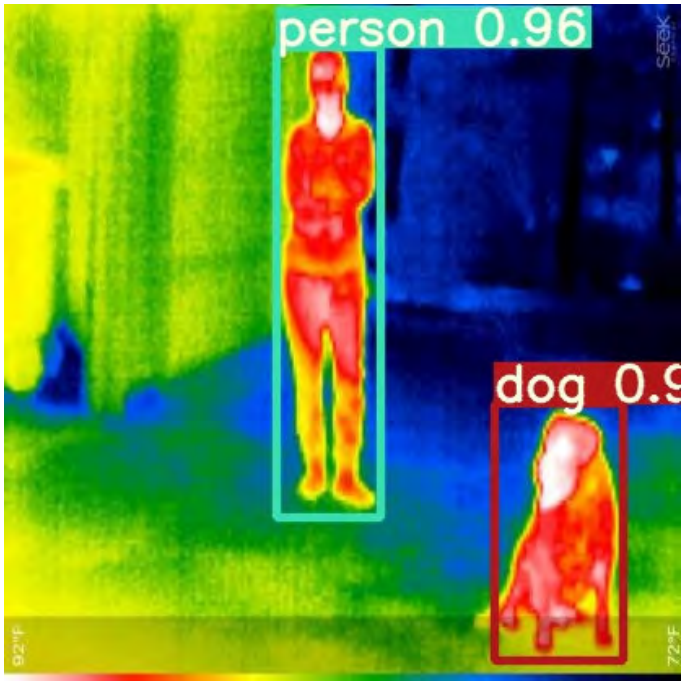
python detect.py --weights runs/train/yolov5s_results/weights/best.pt --img 416 --conf 0.4 --source ../test/images

* Use –save-txt to save the coordinates of the detected objects

Steps:

- Install YOLOv5 dependencies
- Download Custom YOLOv5 Object Detection Data
- Define YOLOv5 Model Configuration and Architecture
- Train a custom YOLOv5 Detector
- Evaluate YOLOv5 performance
- Visualize YOLOv5 training data
- Run YOLOv5 Inference on test images
- Export Saved YOLOv5 Weights for Future Inference

Activity 3a – Dogs and People Detection (Thermal) (Yolov5 inference)



Step 1:
Watch and listen to the instructor's demonstration

10 mins

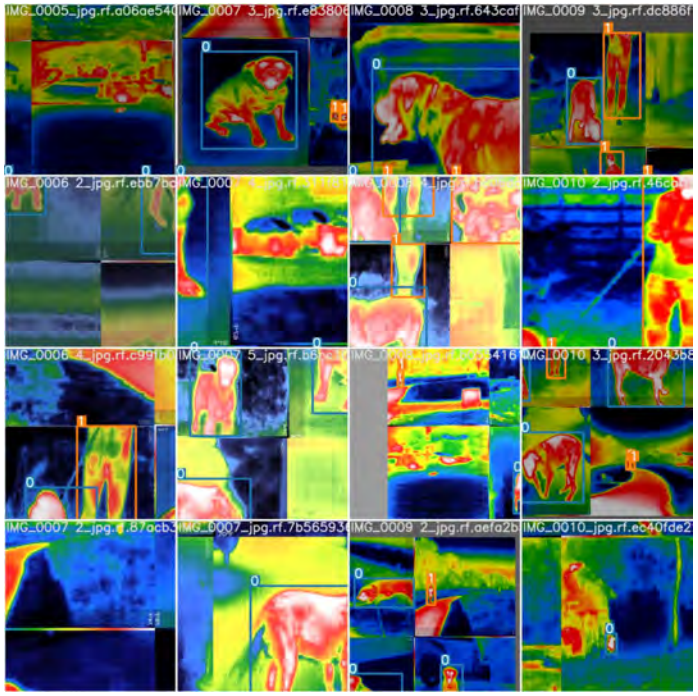
Step 2:
Work through the activities

Individual Activity

10 mins

47

Activity 3b – Dogs and People Detection (Thermal) (Yolov5 training)



Exercises:
- Use a different yolo model, e.g. m or l
- Compare the training time and performance improvement

Step 1:
Watch and listen to the instructor's demonstration

10 mins

Step 2:
Work through the activities

Individual Activity

30 mins

48

30 Mins Break

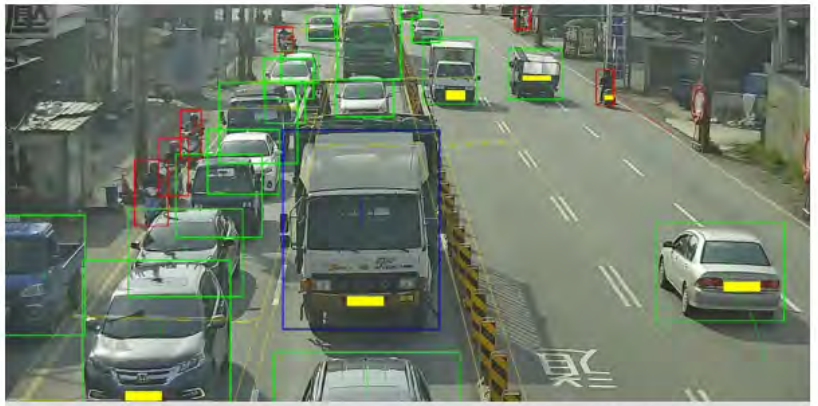
Use of YOLOv4

- Taiwanese government: Traffic control https://www.taiwannews.com.tw/en/news/3957400 and https://youtu.be/IiU6wFmfVnk
- Amazon: Anti-Covid19 Distance-assistant https://github.com/amzn/distance-assistant and Amazon Neurochip / Amazon EC2 Inf1 instances: https://aws.amazon.com/ru/blogs/machine-learning/improving-performance-for-deep-learning-based-object-detection-with-an-aws-neuron-compiled-yolov4-model-on-aws-inferentia/
- BMW Innovation Lab: https://github.com/BMW-InnovationLab

Taiwan and Russia develop real-time object detection system

YOLOv4 helps solve traffic problems in Taoyuan and Hsinchu

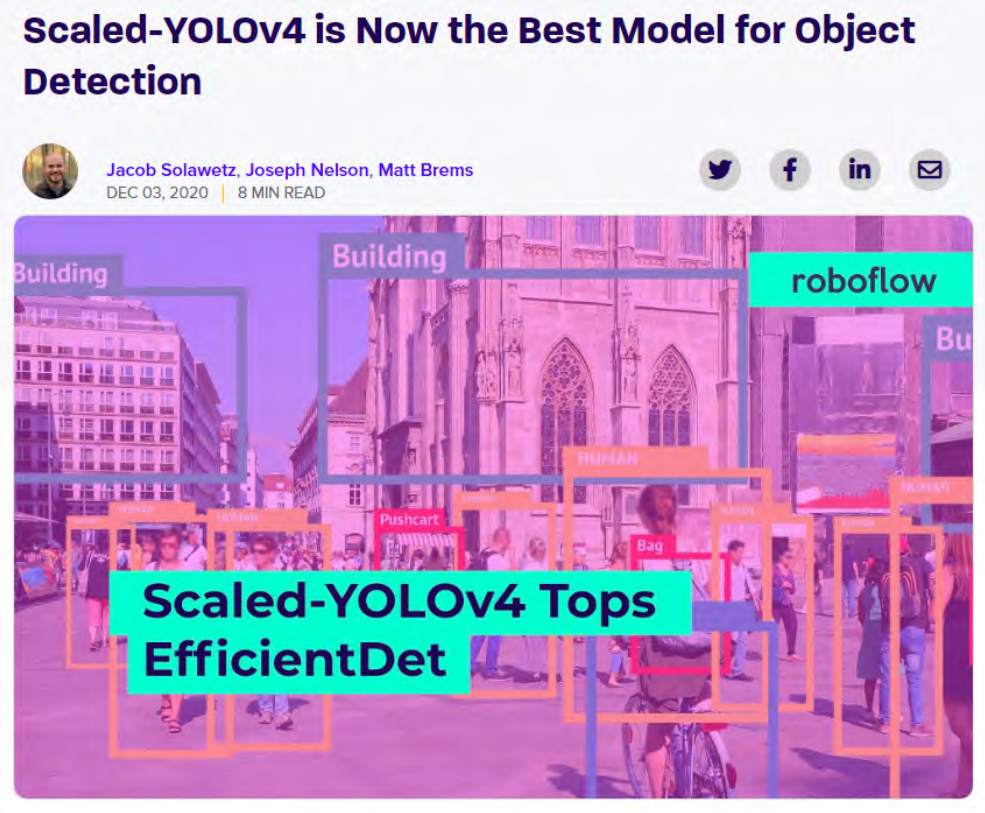
By Matthew Strong, Taiwan News, Staff Writer
2020/07/02 17:02

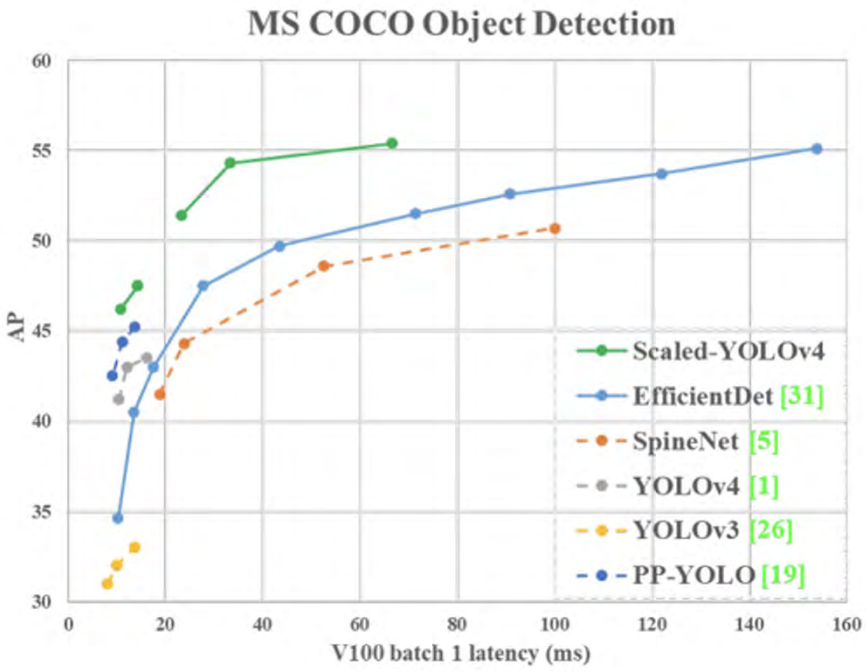


The YOLOv4 system can be use to solve traffic problems (CNA photo courtesy of Academia Sinica)

Scaled-YOLOv4

- Cross Stage Partial Network (https://arxiv.org/pdf/1911.11929.pdf)





EfficientDet Models - Model Scaling

- The EfficientDet family of models has been the preferred object detection models since it was published by the Google Research/Brain team in recent months. The most novel contribution in the EfficientDet paper was to think strategically about how to **scale object detection models up and down**.

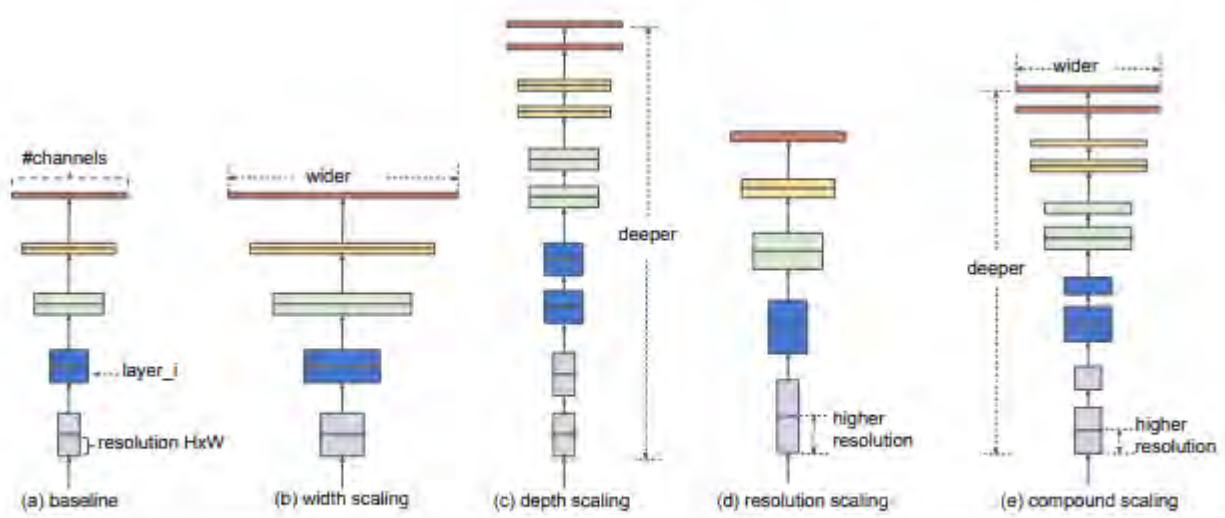


Figure 2. **Model Scaling.** (a) is a baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is our proposed compound scaling method that uniformly scales all three dimensions with a fixed ratio.

https://arxiv.org/pdf/1911.09070.pdf

Scaled-YOLOv4

- a series of neural networks built on top of the improved and scaled YOLOv4 network.
- Scaled YOLOv4 utilizes massively parallel devices such as GPUs much more efficiently than EfficientDet
- Improvements in Scaled YOLOv4 over YOLOv4:
  - Scaled YOLOv4 used optimal network scaling techniques to get YOLOv4-CSP -> P5 -> P6 -> P7 networks
  - Improved network architecture: Backbone is optimized and Neck (PAN) uses Cross-stage-partial (CSP) connections and Mish activation
  - Exponential Moving Average (EMA) is used during training — this is a special case of SWA: https://pytorch.org/blog/pytorch-1.6-now-includes-stochastic-weight-averaging/
  - For each resolution of the network, a separate neural network is trained (in YOLOv4, only one neural network was trained for all resolutions)
  - Improved objectness normalizers
  - Changed activations for Width and Height, which allows faster network training

YOLO Models

- **YOLOv4 -** Early this spring, **AlexyAB** formalized his fork of the Darknet repo, publishing the state-of-the-art **YOLOv4 model**. YOLOv4 was considered one of the best models for speed and accuracy performance, but did not top EfficientDet's largest model for overall accuracy on the COCO dataset.
- **YOLOv5 -** Shortly after the release of YOLOv4, Glenn Jocher (Github moniker **glenn-jocher**) published his version of the YOLO model in PyTorch as **YOLOv5**. Comparing how YOLOv4 and YOLOv5 models stacked up against each other was nuanced –a bunch on the **YOLOv4 vs**. **YOLOv5** debate here.
- The YOLOv5 PyTorch training and architecture conversion was the most notable contribution, making YOLO easier than ever to train, ***speeding up training time 10x relative to Darknet*.**
- One of the main reasons Scaled-YOLOv4 is implemented in the YOLOv5 PyTorch framework is, no doubt, the training routines. Glenn Jocher notably gets a shoutout in the Acknowledgments of Scaled-YOLOv4.
- **PP-YOLO -** This summer, researchers at **Baidu** released their version of the YOLO architecture, **PP-YOLO, surpassing YOLOv4 and YOLOv5**. The network, written in **the Paddle-Paddle framework**, performed well but has yet to gain much traction among practitioners.

Activity 4 – Aerial Maritime Drone Dataset (Scaled-YOLOv4)

Step 1:
Watch and listen to the instructor's demonstration

10 mins

Step 2:
Work through the activities

Individual Activity

20 mins

Activity 5 – Aerial Maritime Drone Dataset (Yolov5) (optional)

Step 1:
Watch and listen to the instructor's demonstration

10 mins

Step 2:
Work through the activities

Individual Activity

20 mins

56

Quiz

https://bit.ly/kw_poll

SCAN ME

Thank you