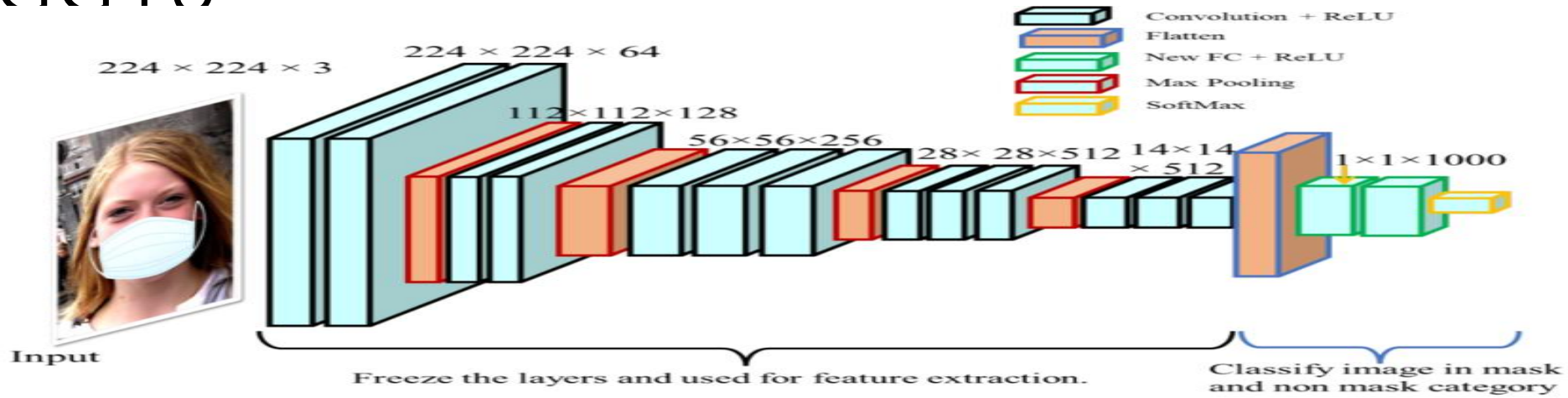
Pre-Trained Models

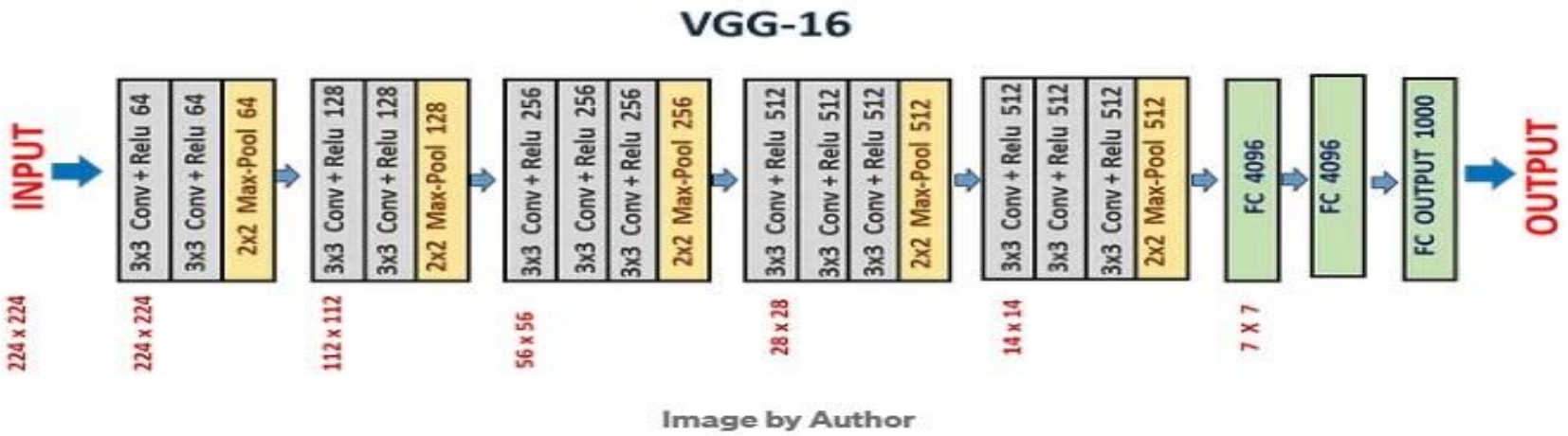
Dr. K. Adi Narayana Reddy

VGG16

- Researchers from the Oxford Visual Geometry Group, or VGG for short, participate in the ILSVRC challenge
- In 2014, convolutional neural network models (CNN) developed by the VGG won the image classification tasks.
- VGG released two different CNN models, specifically a 16-layer model and a 19-layer model.

VGG16

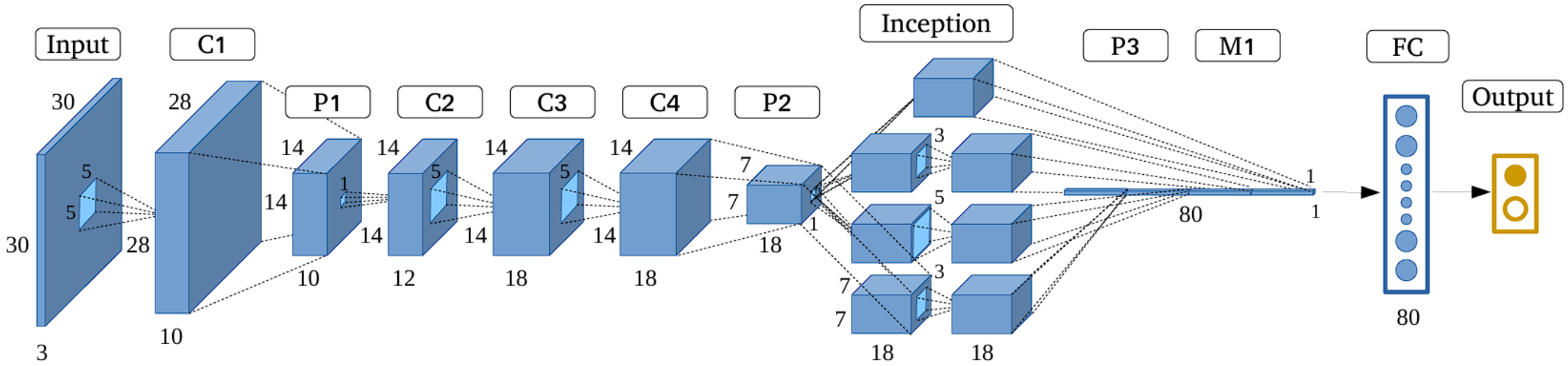

Convolution + ReLU
Flatten
New FC + ReLU
Max Pooling
SoftMax
224 × 224 × 3
224 × 224 × 64
112×112×128
56×56×256
28× 28×512
14×14 × 512
1×1×1000
Input
Freeze the layers and used for feature extraction.
Classify image in mask and non mask category


VGG-16
INPUT
3x3 Conv + Relu 64
3x3 Conv + Relu 64
2x2 Max-Pool 64
3x3 Conv + Relu 128
3x3 Conv + Relu 128
2x2 Max-Pool 128
3x3 Conv + Relu 256
3x3 Conv + Relu 256
3x3 Conv + Relu 256
2x2 Max-Pool 256
3x3 Conv + Relu 512
3x3 Conv + Relu 512
3x3 Conv + Relu 512
2x2 Max-Pool 512
3x3 Conv + Relu 512
3x3 Conv + Relu 512
3x3 Conv + Relu 512
2x2 Max-Pool 512
FC 4096
FC 4096
FC OUTPUT 1000
OUTPUT
224 x 224
224 x 224
112 x 112
56 x 56
28 x 28
14 x 14
7 X 7
Image by Author

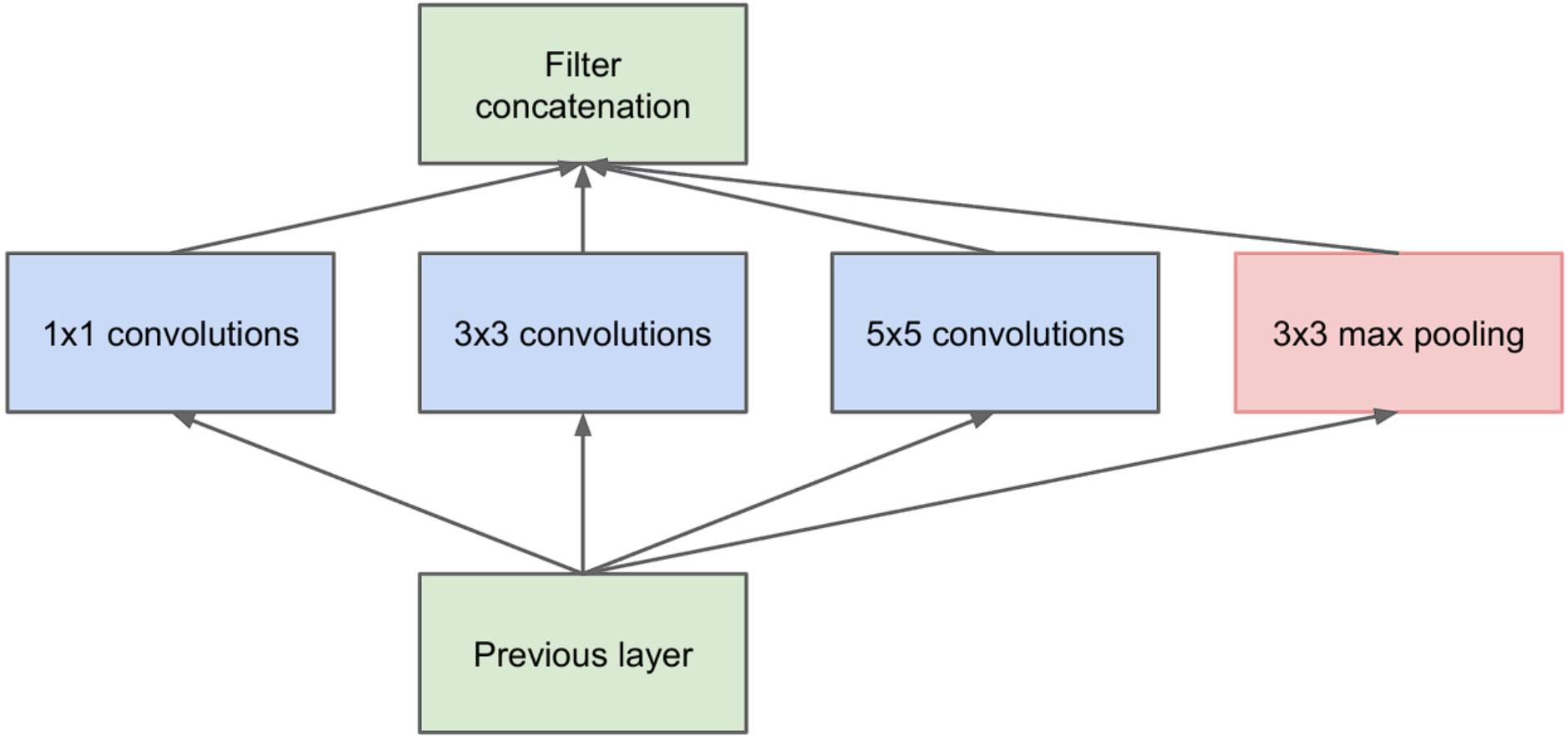
Inception V3

- When designing a deep learning model, one needs to decide what convolution filter size to use (whether it should be 3×3, 5×5, or 1×3) as it affects the model's learning and performance, and when to max pool the layers.
- Instead of deciding what filter size to use and when to perform a max pooling operation, they combined multiple convolution filters.
- Stacking multiple convolution filters together instead of just one increases the parameter count many times.

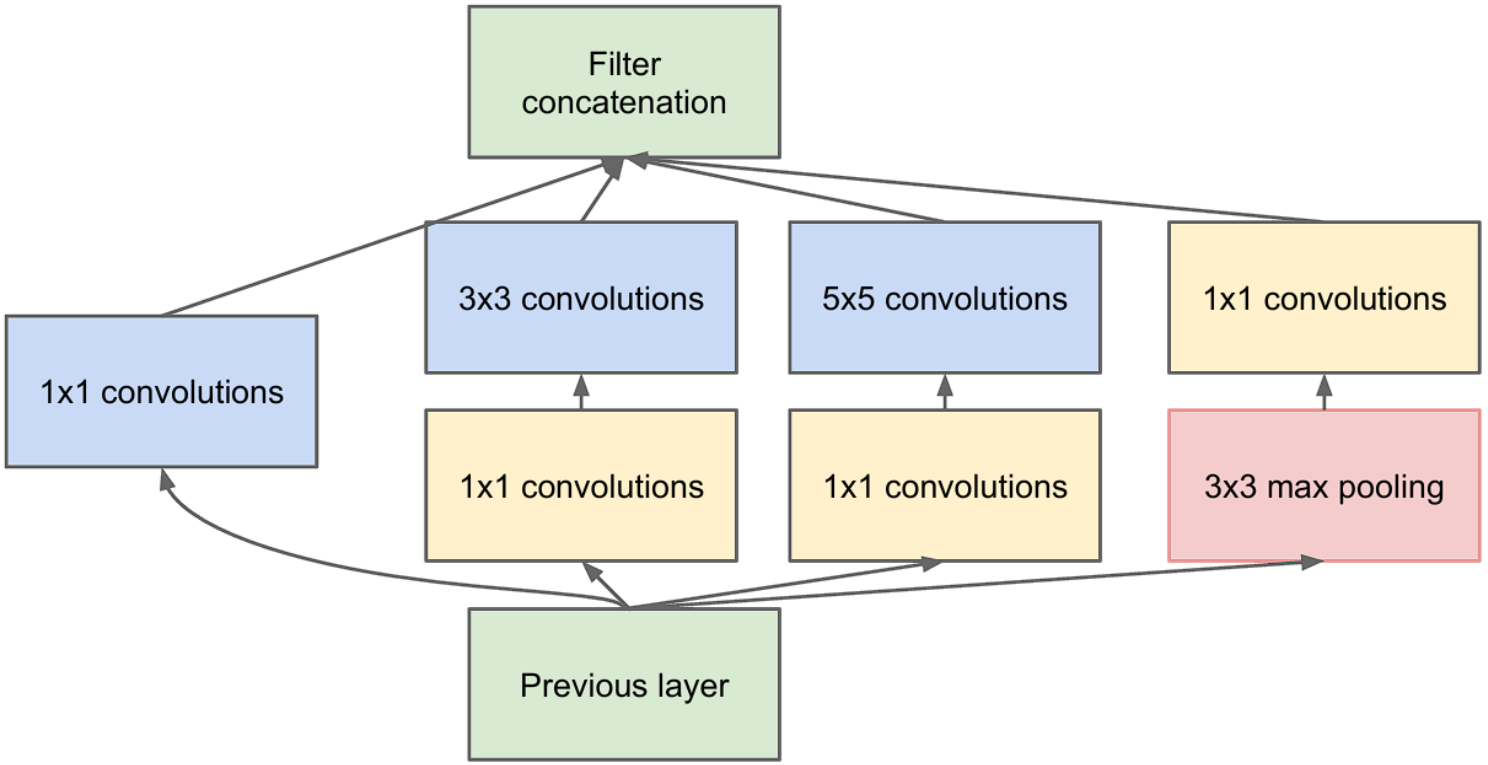
Inception V3



Inception Module Without Dimensionality Reduction



Inception Module With Dimensionality Reduction



Benefits

- Parameter Efficiency: By using 1×1 convolutions, the module reduces dimensionality before applying the more expensive 3×3 and 5×5 convolutions and pooling operations.
- Increased Representation: By incorporating filters of varying sizes and more layers, the network captures a wide range of features in the input data. This results in better representation.
- Enhancing Feature Combination: The 1×1 convolution is also called network in the network. This means that each layer is a micro-neural network that learns to abstract the data before the main convolution filters are applied.
-

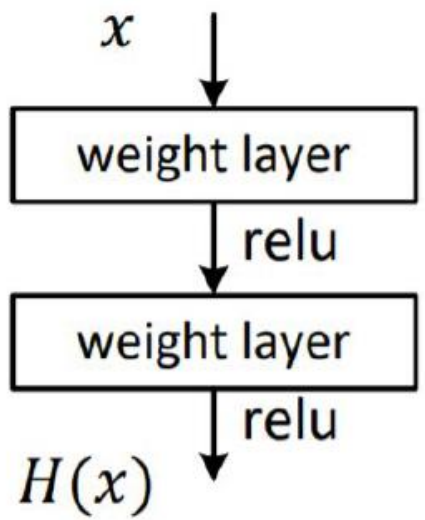
ResNet

- ResNet-50 is CNN architecture that belongs to the ResNet (Residual Networks) family, a series of models designed to address the challenges associated with training deep neural networks.
- Developed by researchers at Microsoft Research Asia, ResNet-50 is renowned for its depth and efficiency in image classification tasks. ResNet architectures come in various depths, such as ResNet-18, ResNet-32, and so forth, with ResNet-50 being a mid-sized variant.

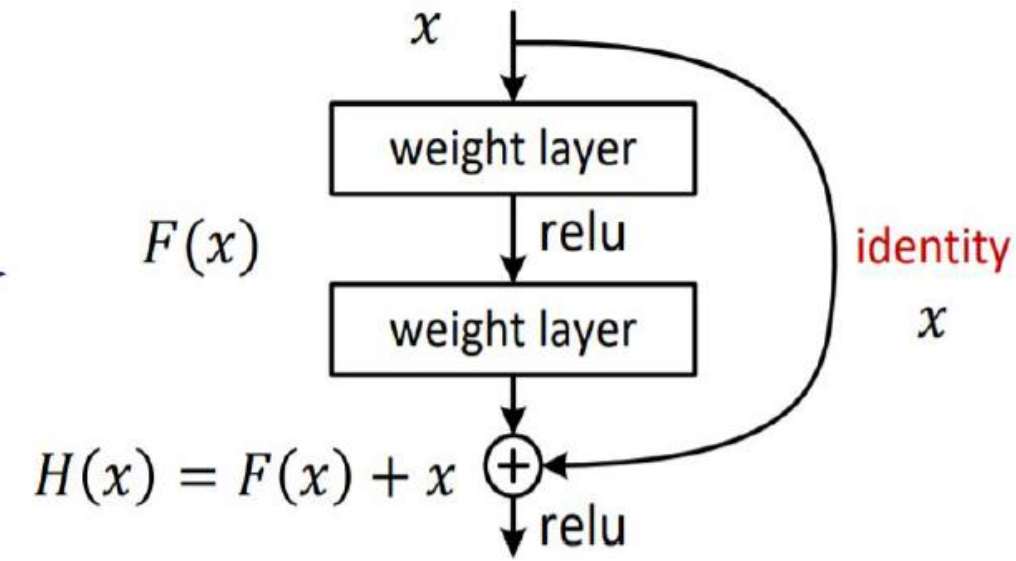
Deep Residual Learning

- Plain net

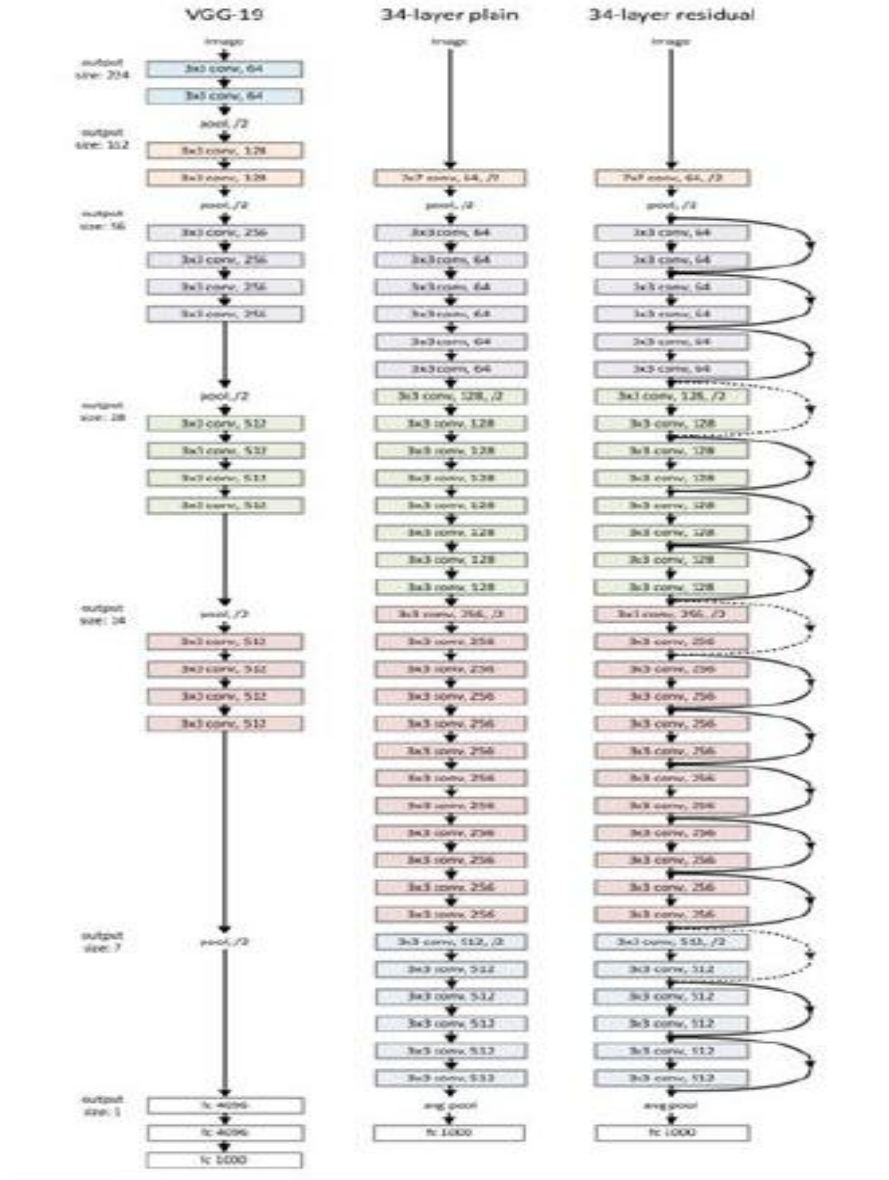
any small subnet



- Residual net



Learn the residual mapping F(x) rather than unreferenced H(x)


VGG-19
34-layer plain
34-layer residual
image
3x3 conv, 64
pool, /2
3x3 conv, 128
3x3 conv, 256
3x3 conv, 512
avg pool
fc 1000

ResNet	Inception
The core innovation in ResNet is the **residual block**, which introduces "skip connections" or "identity mappings" that allow the input to bypass one or more layers and be added directly to the output. This helps with training very deep networks by mitigating the vanishing gradient problem. ResNet models are typically composed of a stack of these residual blocks.	InceptionNet, also known as GoogLeNet, uses the **Inception module**, which applies multiple convolutional filters of different sizes (e.g., 1x1, 3x3, 5x5) in parallel within a single layer. It concatenates the results of these filters into a single output, allowing the network to capture different types of features at various scales.It also uses **1x1 convolutions** to reduce dimensionality before applying the larger convolutions, making it more efficient.
ResNet is designed for **very deep networks**, with versions like ResNet-50, ResNet-101, and ResNet-152, where the depth is a critical factor. It can scale to very large architectures because of its residual connections.	Inception networks are typically shallower compared to ResNet but have a more **complex architecture** due to the multiple filter sizes and operations within each module.
While ResNet is efficient in terms of training deep networks, it does not focus heavily on reducing the computational cost per layer.	InceptionNet is designed with **efficiency** in mind. The use of **1x1 convolutions** reduces the computational burden, especially in deeper layers, and the multi-branch design allows the network to capture a variety of features while keeping the number of parameters relatively low.

Vision Transformer

- A **Vision Transformer (ViT)** is a type of neural network architecture designed for image recognition tasks and inspired by the Transformer model, which has been highly successful in natural language processing. The Vision Transformer was introduced by Google Research in the paper "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale" in 2020.

ViT

- **Image to Patches**:
  - Instead of using convolutional layers like in traditional CNNs (Convolutional Neural Networks),
  - the image is divided into fixed-size patches. Each patch is flattened into a vector.
- **Embedding**:
  - These vectors are linearly embedded, similar to word embeddings in NLP.
  - The embeddings of the patches are combined with positional encodings to retain spatial information.
- **Transformer Encoder**:
  - The embedded patches are passed through a standard Transformer encoder,
  - which consists of multi-head self-attention and feed-forward layers.
  - This helps the model learn the global relationships between different parts of the image.
- **Classification Head**:
  - The output of the Transformer is fed into a classification head to produce the final predictions.

Transformer Encoder

- The **Transformer Encoder** is a core component of the Transformer model architecture, which was originally introduced in the paper "Attention is All You Need" by Vaswani et al. in 2017.
- Multi-Head Self-Attention Mechanism
  - The self-attention mechanism is what allows the Transformer Encoder to consider the relationships between different positions in the input sequence simultaneously.
  - The goal is to calculate attention scores for each position in the sequence relative to every other position. This enables the model to understand how each element (e.g., word in NLP or patch in vision tasks) relates to every other element in the input.

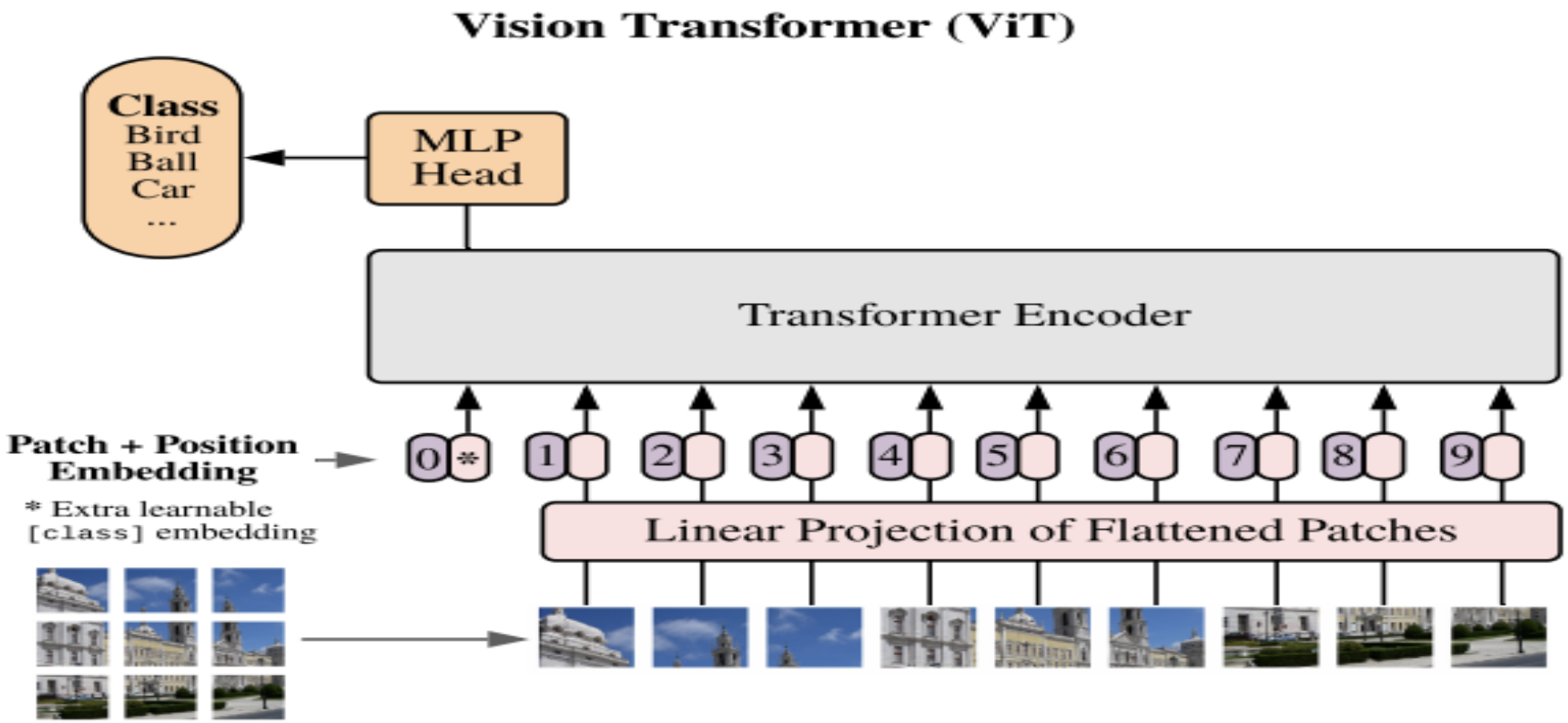
Transformer Encoder

- Query (Q), Key (K), and Value (V) Vectors
  - Each input element is transformed into three vectors: Query, Key, and Value. These vectors are learned representations that are used to compute attention.
  - The attention score for a pair of elements is computed as the dot product of the Query of one element and the Key of the other. The result determines how much "attention" should be paid from one element to the other.
  - The attention score is scaled by the square root of the dimensionality of the Key vectors to prevent very large values. This score is then passed through a softmax function to get normalized attention weights, which sum to 1.
  - The Value vectors are weighted by the attention scores, and the results are summed to produce the output for each element.

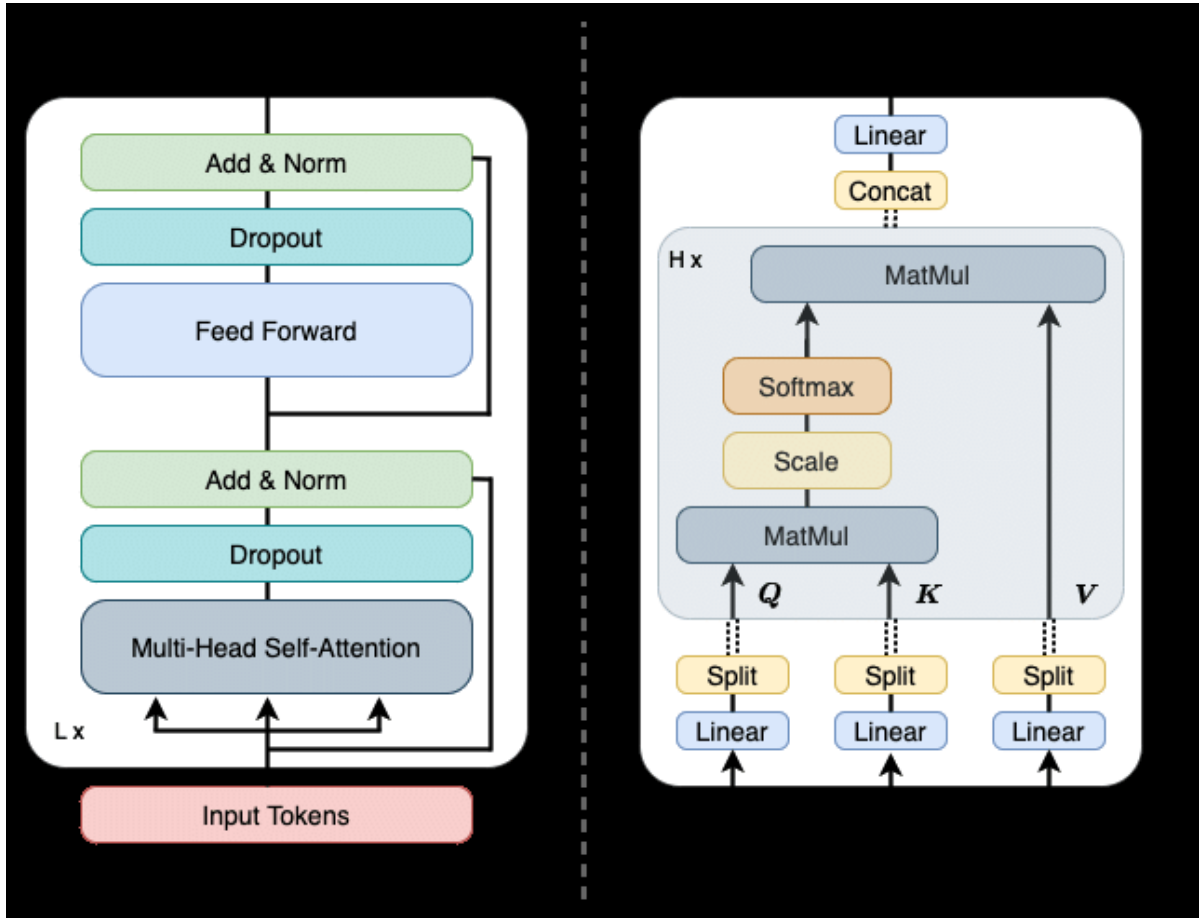
Multi-Head Attention

- Instead of performing a single self-attention calculation, the Transformer uses multiple attention "heads." Each head learns different types of relationships between elements in the sequence, capturing more nuanced patterns. The outputs of all heads are concatenated and linearly transformed to produce the final output of the self-attention block.

Vision Transformer



Multi-Head Attention



Limitations of Traditional CNN Architectures in Segmentation Tasks

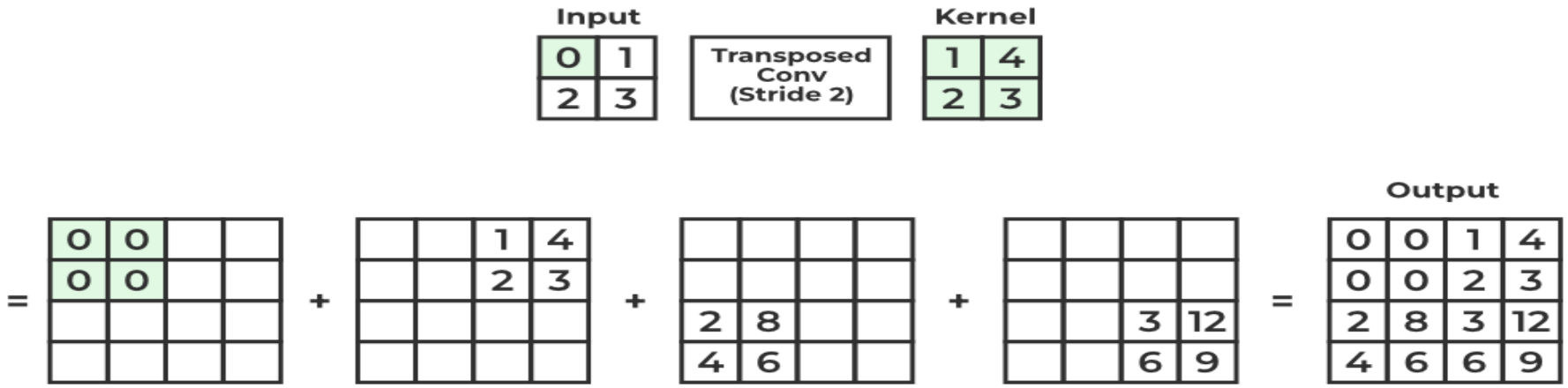
- **Loss of Spatial Information**
  - Traditional CNNs use pooling layers to downsample feature maps, which helps capture high-level features but leads to a loss of spatial information, making precise object detection and segmentation challenging.
- **Fixed Input Size**
  - CNNs often require fixed input sizes, making it difficult to handle variable-sized images in segmentation tasks
- **Limited Localisation Accuracy**
  - fully connected layers limits localization accuracy, as they discard spatial information needed for precise object or region detection.

Fully Convolutional Networks (FCNs) as a Solution for Semantic Segmentation

- Fully Convolutional Networks (FCNs) address the limitations of traditional CNNs in segmentation tasks by using only convolutional layers and retaining spatial information. They make pixel-level predictions, assigning labels to each pixel and creating dense segmentation maps. FCNs replace fully connected layers with transposed convolutions, which upsample feature maps to match the input image size, enabling precise segmentation.

Transposed Convolution

- The operation of a transposed convolutional layer is similar to that of a normal convolutional layer, except that it performs the convolution operation in the opposite direction. **Instead of sliding the kernel over the input and performing element-wise multiplication and summation, a transposed convolutional layer slides the input over the kernel and performs element-wise multiplication and summation**



Transposed Convolution

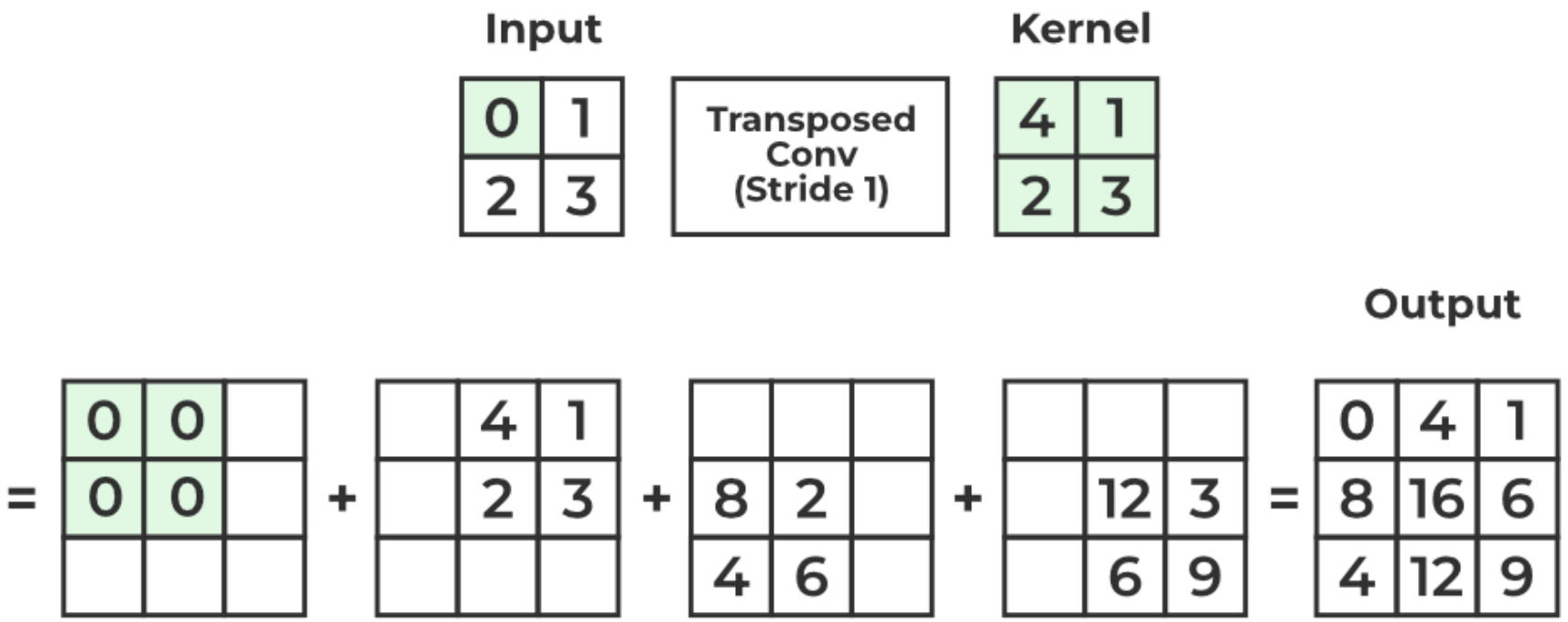


Image Segmentation

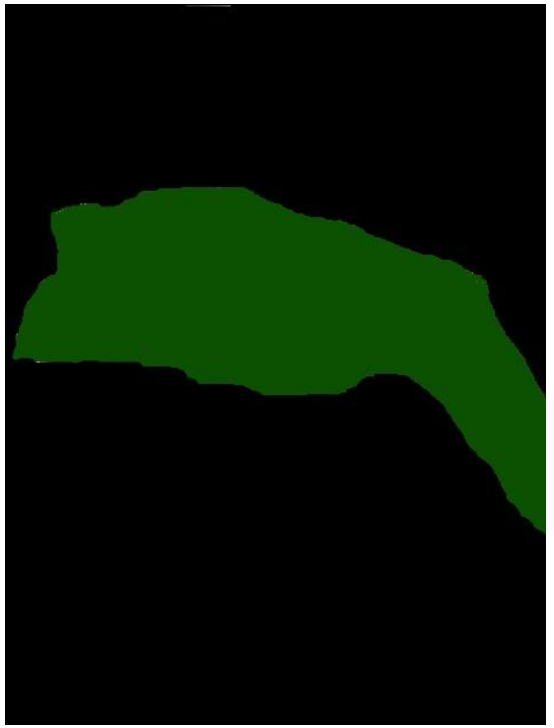
- Image segmentation is a fundamental process in **computer vision** in which an image is divided into many meaningful and separate parts or segments.
- Image classification, which provides a single label to a complete image, segmentation adds labels to each pixel or group of pixels, essentially splitting the image into semantically significant parts

What does a U-Net do?

Input Image

Learns Segmentation

U-Net

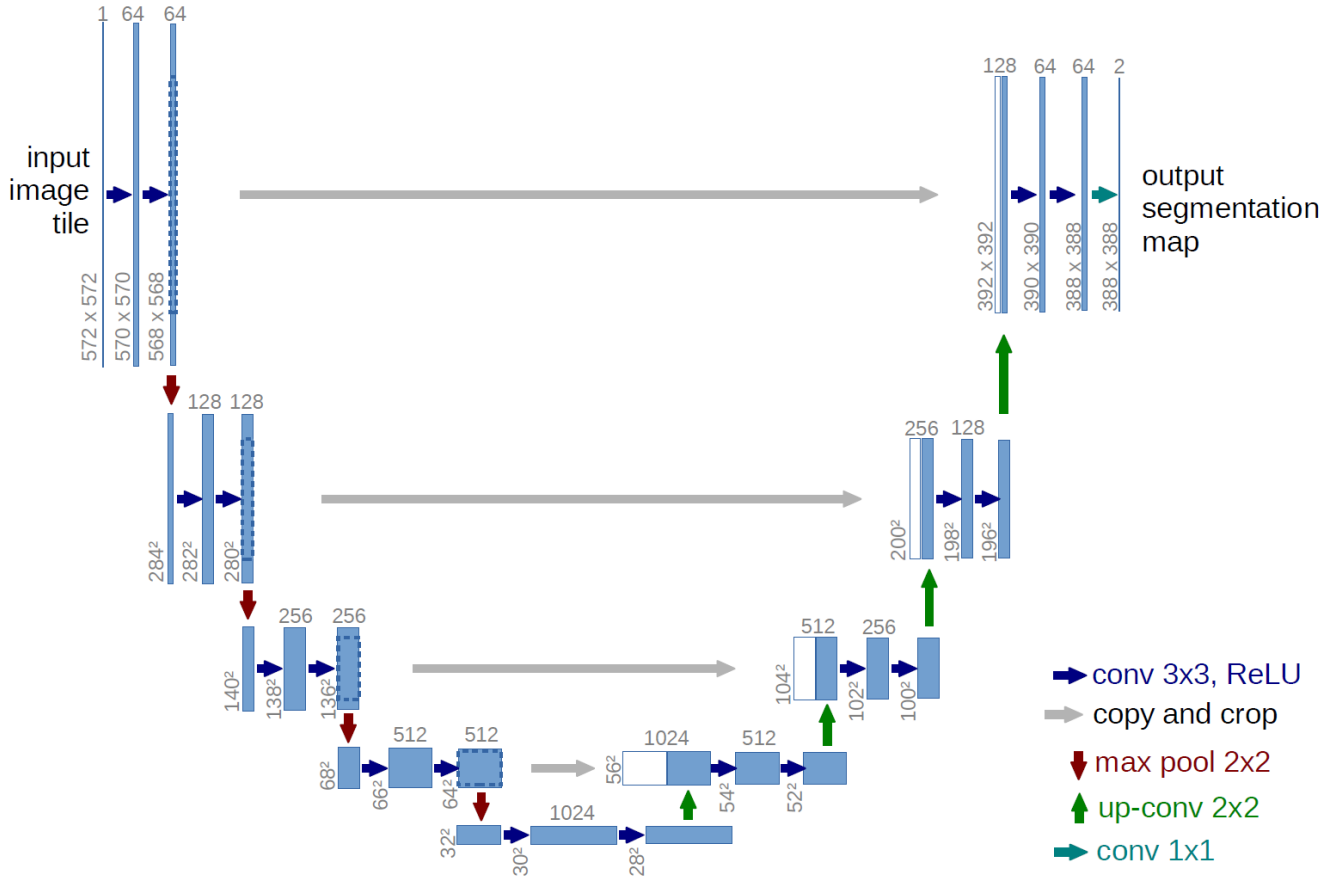


Output Segmentation Map

UNET

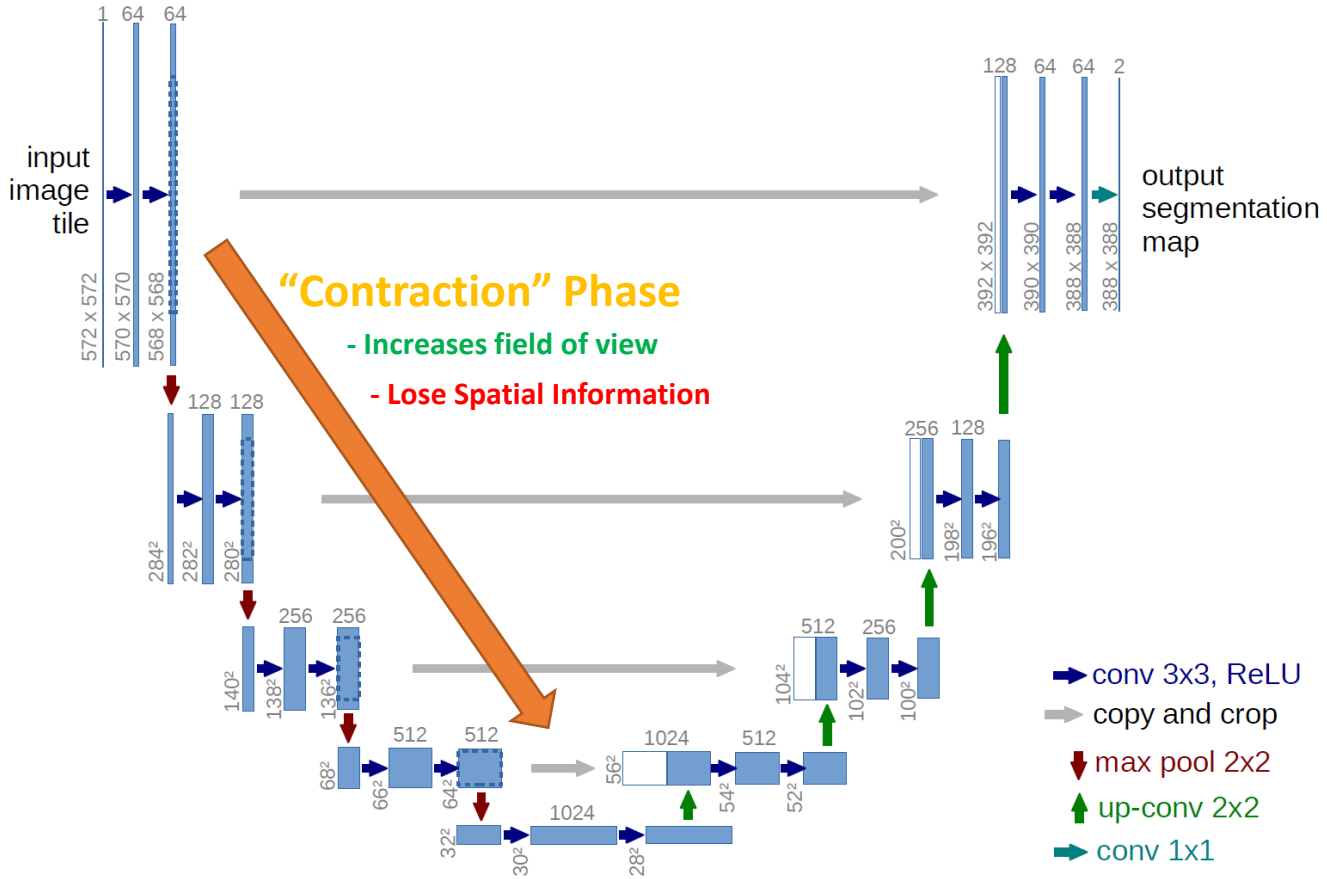
- UNET is a fully convolutional neural network (FCN) architecture built for image segmentation applications. It was first proposed in 2015 by Olaf Ronneberger, Philipp Fischer, and Thomas Brox.
- UNET combines an encoding path, also called the contracting path, with a decoding path called the expanding path
- The architecture is named after its U-shaped look when depicted in a diagram.

U-Net Architecture



Ronneberger et al. (2015) U-net Architecture

U-Net Architecture

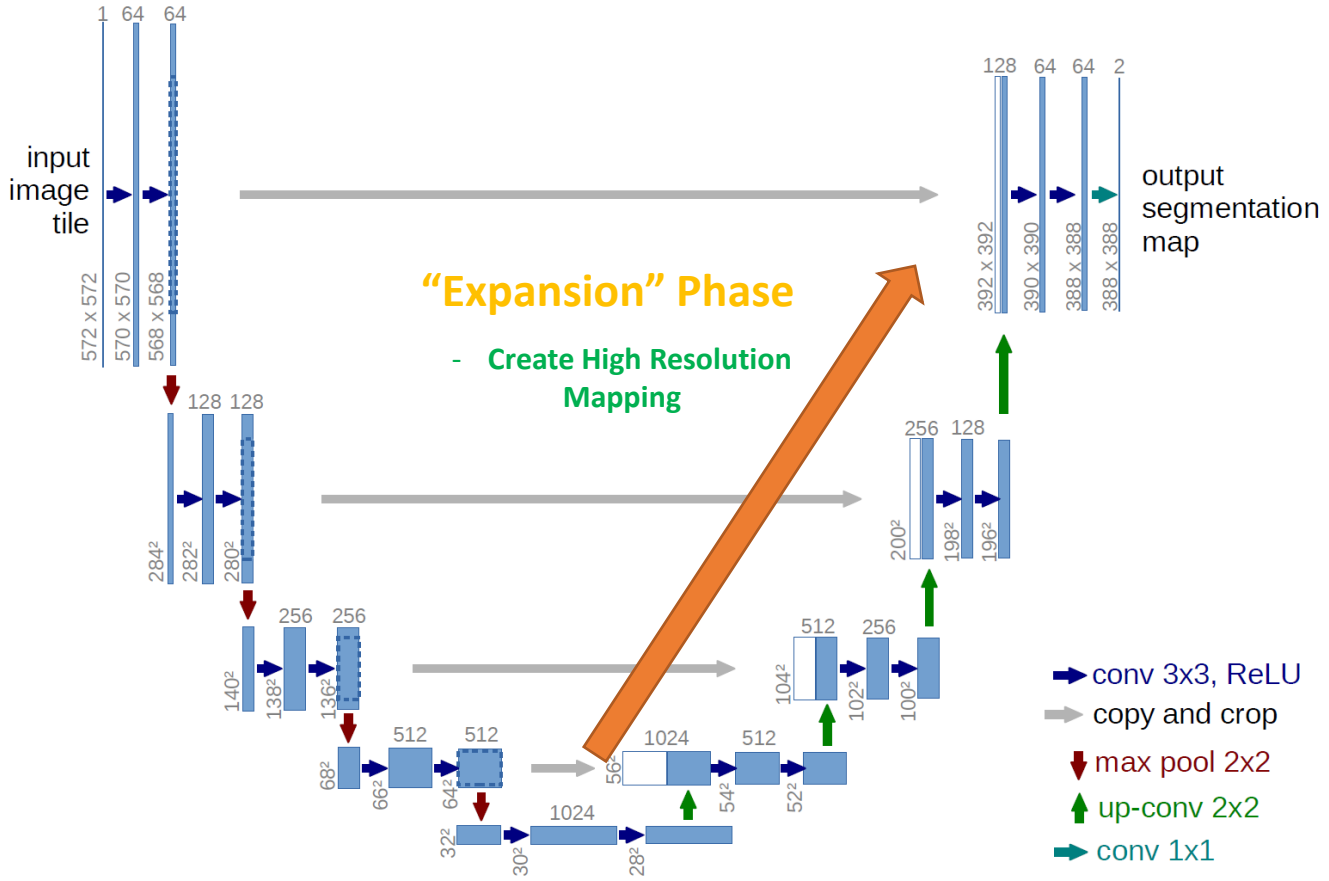


Ronneberger et al. (2015) U-net Architecture

Contracting Path (Encoding Path)

- UNET's contracting path uses convolutional layers and max pooling to reduce the spatial dimensions of the input image, capturing high-resolution, low-level features effectively.

U-Net Architecture

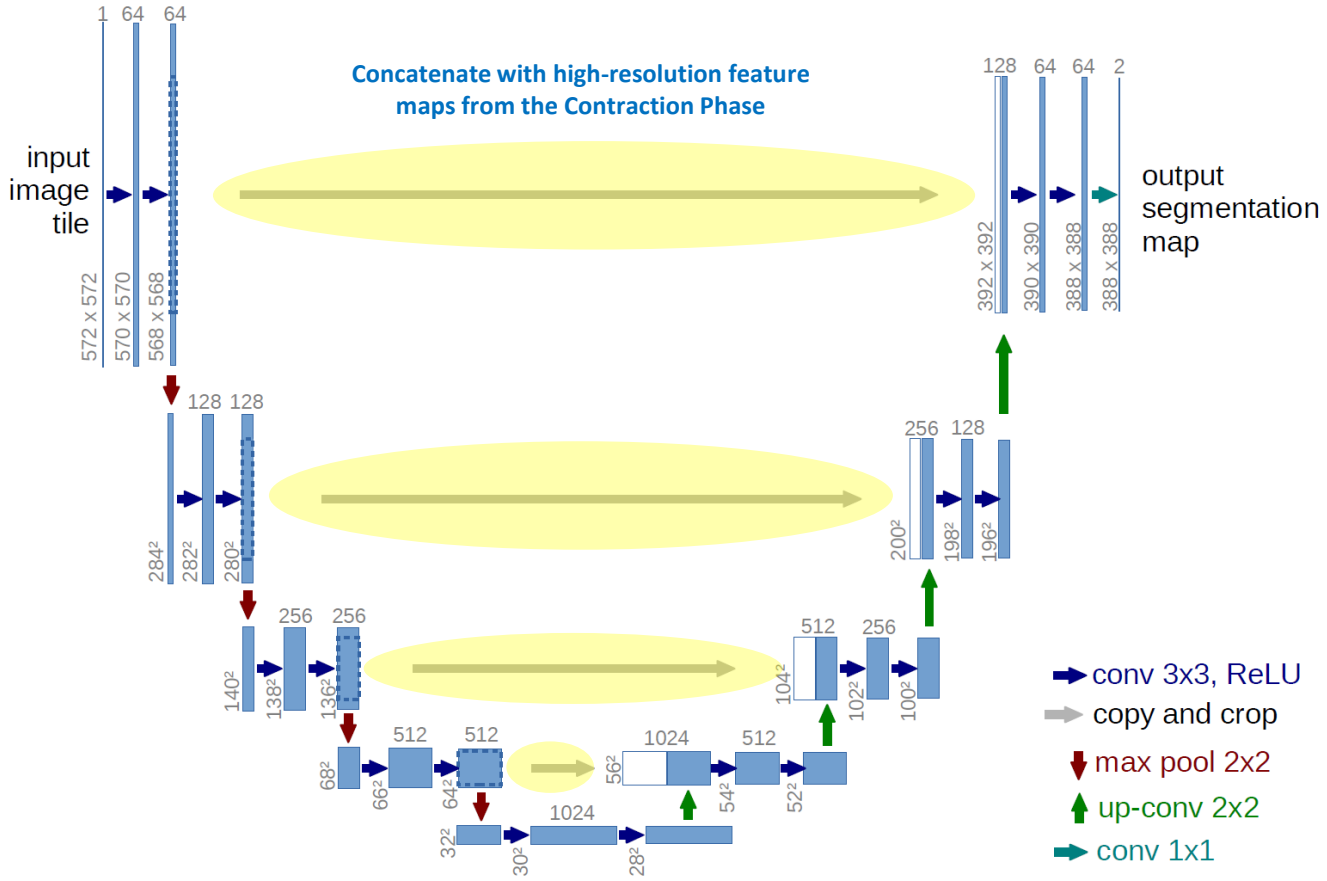


Ronneberger et al. (2015) U-net Architecture

Expanding Path (Decoding Path)

- Transposed convolutions in UNET’s expansion path upsample feature maps, increasing their spatial resolution to reconstruct a dense segmentation map

U-Net Architecture



Ronneberger et al. (2015) U-net Architecture

Purpose of Skip Connections

- Skip connections link layers in the encoding path (contracting path) to corresponding layers in the decoding path (expansion path) of the UNET architecture.
- Collecting Local and Global Information
  - By connecting matching layers, skip connections allow the network to use both local features (captured early in the network) and global features (captured deeper in the network).
  - Local features contain fine details important for accurate segmentation, while global features capture the overall structure of the image.
- Retention of Spatial Information
  - As the encoding path uses downsampling operations like max pooling, some spatial information is lost.
  - Skip connections help preserve this important spatial information by transferring feature maps directly from earlier layers to the corresponding layers in the decoding path.
- **Integration of Feature Maps**
  - In the decoding path, feature maps from skip connections are **concatenated** with the upsampled feature maps. This integration enriches the information available to the network and helps it make more accurate predictions.
  - The combined feature maps contain both high-resolution details and abstracted, high-level context.

Loss Function in UNET

- UNET frequently employs segmentation-friendly loss functions such as the Dice coefficient or cross-entropy loss.
- The **Dice Coefficient Loss**, or **Dice Loss**, is a popular loss function used in image segmentation tasks to measure the similarity between the predicted segmentation map and the ground truth. It is based on the **Dice Coefficient**, which is a measure of overlap between two sets and ranges from 0 (no overlap) to 1 (perfect overlap).
- The Dice Coefficient is calculated as:
- $Dice\ Coefficient = \frac{2*|X \cap Y|}{|X|+|Y|}$
- where:
  - X is the set of predicted pixels (segmentation output).
  - Y is the set of ground truth pixels.

Dice Coefficient Loss

- In the case of binary segmentation, this formula can be rewritten using sums:
  - $Dice\ coefficient = \frac{2\sum_i p_i g_i}{\sum_i p_i^2 + \sum_i g_i^2}$
  - p_i is the prediction at pixel i
  - g_i is the ground truthat pixel i
- The **Dice Loss** is simply 1− Dice Coefficient, since we want to minimize the loss during training.
- Dice Loss = 1- Dice coefficient

Dice loss calculation

- Ground truth: [1, 1, 0, 0, 1, 0, 1, 0]
- Predicted Segmentation: [1, 0, 0, 1, 1, 0, 1, 0]
- Intersection: [1, 0, 0, 0, 1, 0, 1, 0]
- Intersection count is 3
- Sum of ground truth = 4
- Sum of predicted segmentation = 4
- Dice coefficient = 2 * 3 / (4+4) = 6/8 = 3/4
- Dice loss = (1- ¾) = ¼

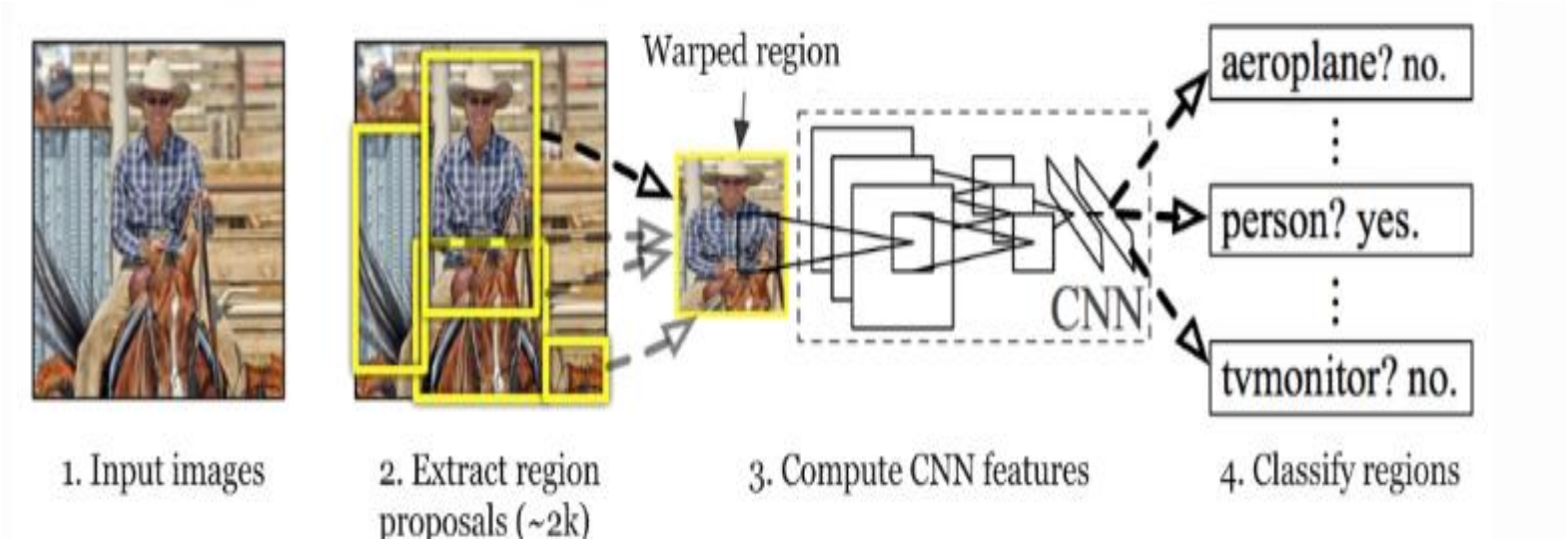
Object detection vs recognition

- Object Recognition
  - Also known as image classification, this task identifies the class of an object in an image. It doesn't provide localization information, but instead outputs class labels. Object recognition is used in applications like image tagging, content-based image retrieval, and visual search engines.
- Object Detection
  - This task identifies objects and provides their precise locations in an image or video. Object detection algorithms use machine learning or deep learning to produce results. They typically output bounding boxes around the identified objects. Object detection is used in applications like factory automation, product placement, and defect detection.

RCNN

- ***R-CNN (Region-based Convolutional Neural Network)*** was introduced by Ross Girshick et al. in 2014. R-CNN revolutionized object detection by combining the strengths of region proposal algorithms and deep learning, leading to remarkable improvements in detection accuracy and efficiency.
- The original RCNN model uses selective search to extract around 2,000 region proposals from the input image.
- Each region proposal is then warped into a fixed size and fed into a pre-trained CNN to extract features.
- These features are then passed to a classifier, such as SVM, to predict object labels, and a regression model refines the bounding box.

Example


Warped region
aeroplane? no.
person? yes.
tvmonitor? no.
CNN
1. Input images
2. Extract region proposals (~2k)
3. Compute CNN features
4. Classify regions

Key components

- **Region Proposals**
  - R-CNNs begin by generating *region proposals*, which are smaller sections of the image that may contain the objects we are searching for
  - The algorithm employs a method called *selective search*, a greedy approach that generates approximately 2,000 region proposals per image.
- **Selective Search**
  - ***Selective Search*** is a greedy algorithm that generates region proposals by combining smaller segmented region
  - Algorithm steps
    - **Generate Initial Segmentation**: The algorithm starts by performing an initial sub-segmentation of the input image.
    - **Combine Similar Regions**: It then recursively combines similar bounding boxes into larger ones. Similarities are evaluated based on factors such as color, texture, and region size.
    - **Generate Region Proposals**: Finally, these larger bounding boxes are used to create region proposals for object detection.
- **Input Preparation in R-CNN**
  - After generating the region proposals, these regions are warped into a uniform square shape to match the input dimensions required by the CNN model.

Selective Search

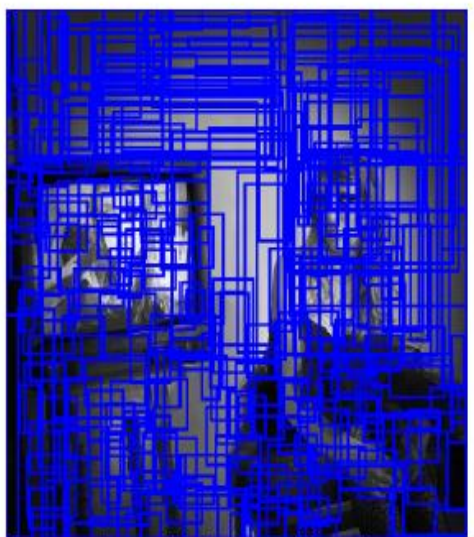
Step 1: Generate initial sub-segmentation

Goal: Generate many regions, each of which belongs to at most one object.

Using the method described by Felzenszwalb et al. from week 1 works well.

Input Image

Segmentation



Candidate objects

[P. F. Felzenszwalb and D. P. Huttenlocher. "Efficient Graph-Based Image Segmentation." IJCV, 59:167–181, 2004.]

Selective Search

Step 2: Recursively combine similar regions into larger ones.

Greedy algorithm:

1. From set of regions, choose two that are most similar.
2. Combine them into a single, larger region.
3. Repeat until only one region remains.

This yields a hierarchy of successively larger regions, just like we want.

Selective Search

Step 2: Recursively combine similar regions into larger ones.

Input Image | Initial Segmentation | After some iterations | After more iterations

Selective Search

Step 3: Use the generated regions to produce candidate object locations.

Input Image

Similarity

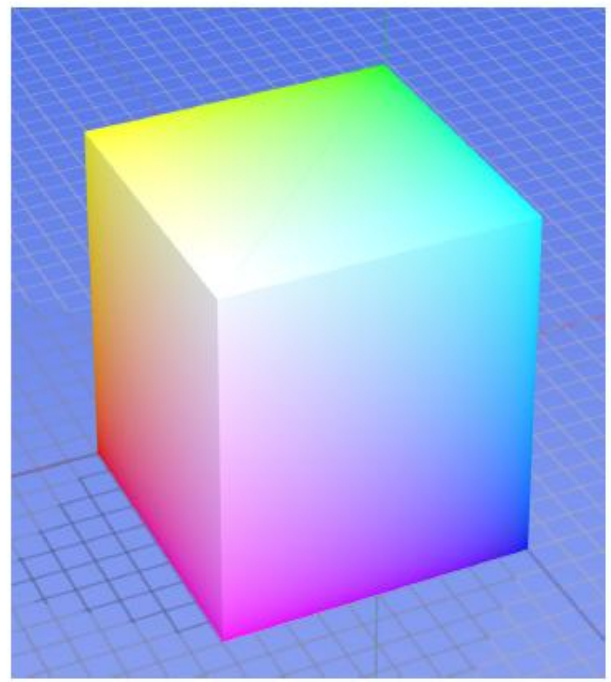
What do we mean by "**similarity**"?

Two-pronged approach:

1. Choose a **color space** that captures interesting things.
    a. Different color spaces have different invariants, and different responses to changes in color.
2. Choose a **similarity metric** for that space that captures everything we're interested: color, texture, size, and shape.

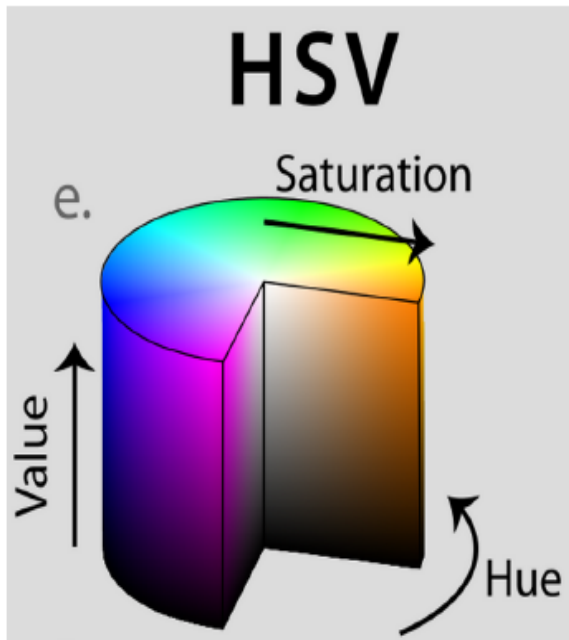
Similarity

RGB (red, green, blue) is a good baseline, but changes in illumination (shadows, light intensity) affect all three channels.



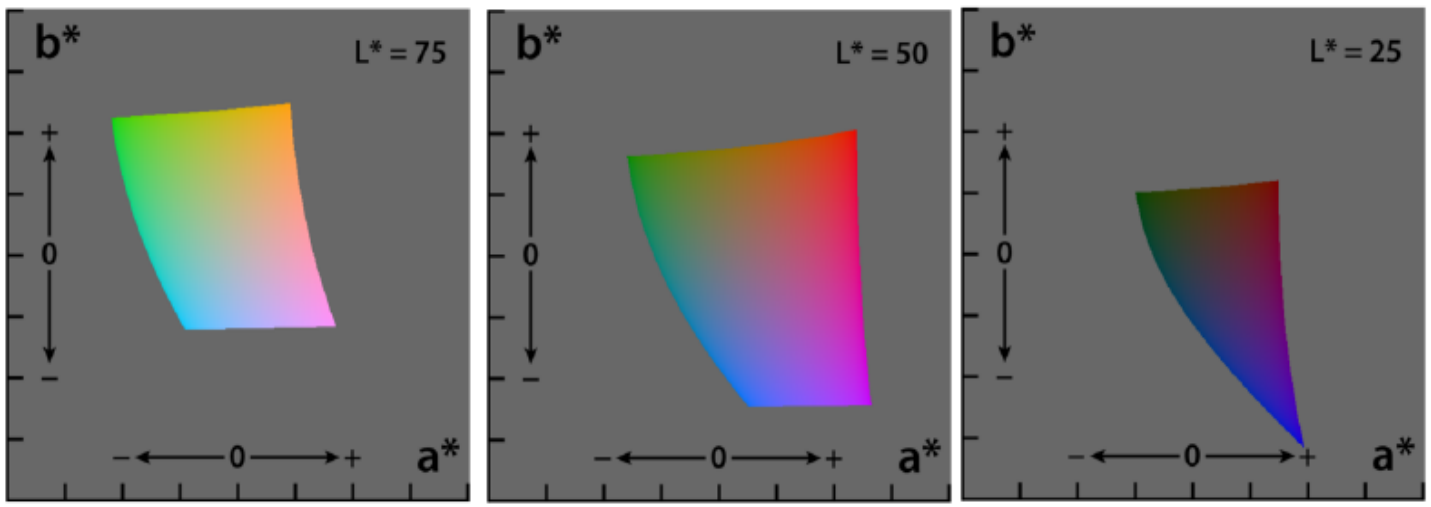
Similarity

HSV (hue, saturation, value) encodes color information in the hue channel, which is invariant to changes in lighting. Additionally, saturation is insensitive to shadows, and value is insensitive to brightness changes.



Similarity

Lab uses a lightness channel and two color channels (a and b). It's calibrated to be *perceptually uniform*. Like HSV, it's also somewhat invariant to changes in brightness and shadow.



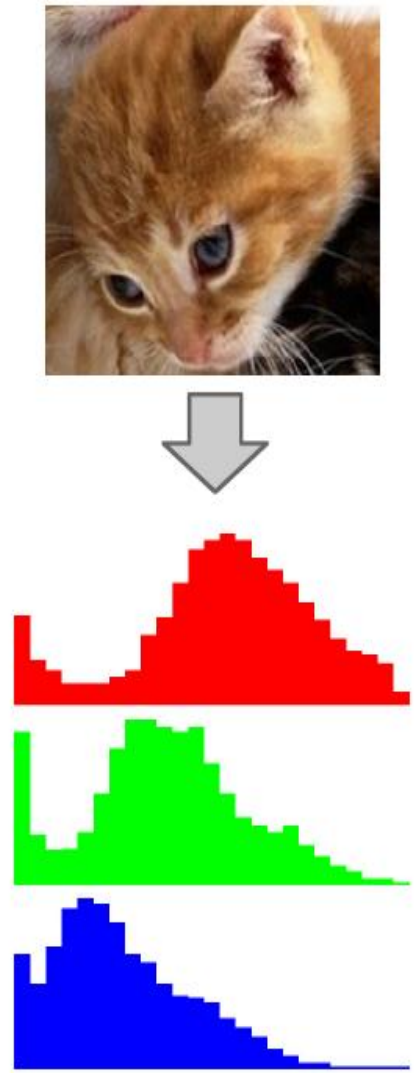
Similarity

Similarity Measures: Color Similarity

Create a color histogram *C* for each channel in region *r*.
In the paper, 25 bins were used, for 75 total dimensions.

We can measure similarity with histogram intersection:

$$s_{colour}(r_i, r_j) = \sum_{k=1}^{n} \min(c_i^k, c_j^k)$$

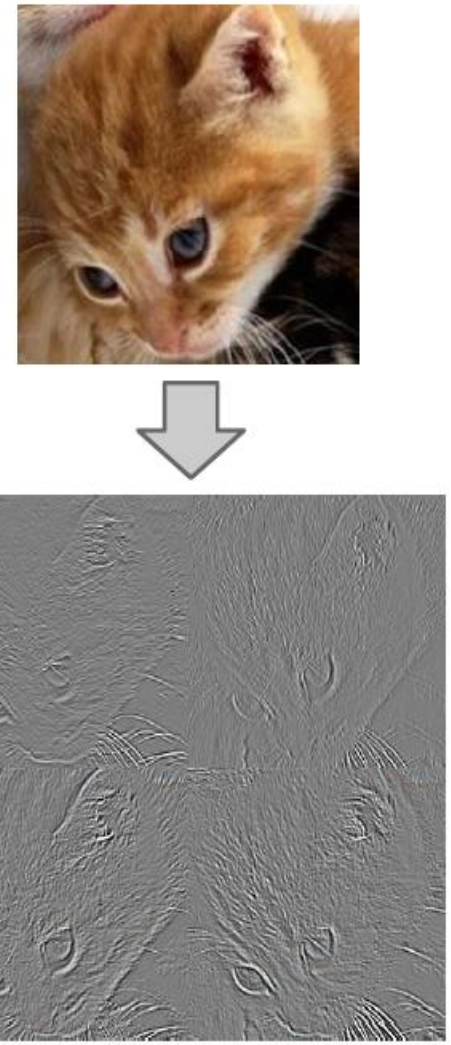


Similarity

Similarity Measures: Texture Similarity

Can measure textures with a HOG-like feature:

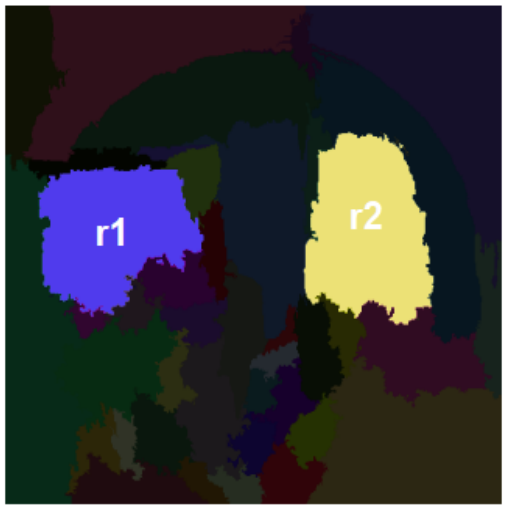
1. Extract gaussian derivatives of the image in 8 directions and for each channel.
2. Construct a 10-bin histogram for each, resulting in a 240-dimensional descriptor.



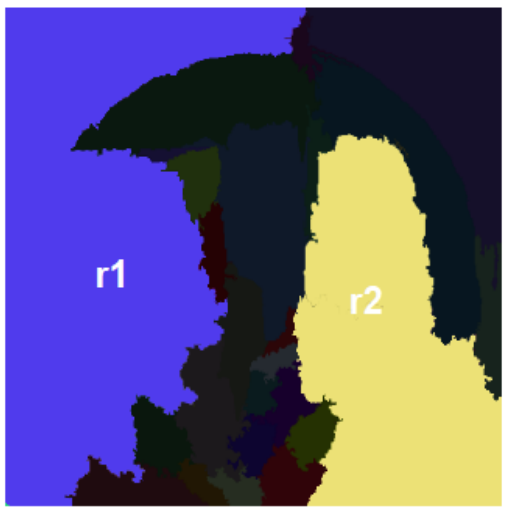
Similarity

Similarity Measures: Size Similarity

We want small regions to merge into larger ones, to create a balanced hierarchy.



Solution: Add a size component to our similarity metric, that ensures small regions are more similar to each other.



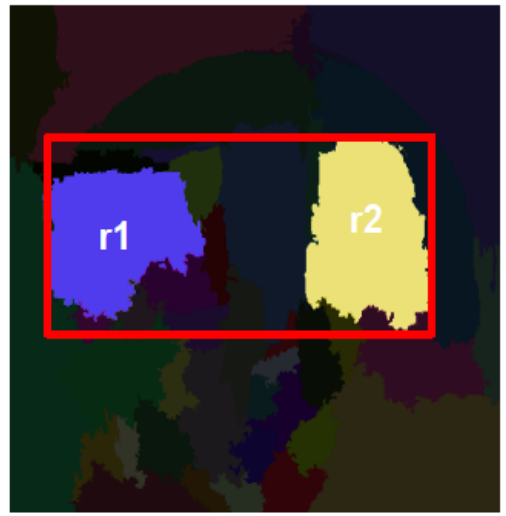
$$s_{size}(r_i, r_j) = 1 - \frac{\text{size}(r_i) + \text{size}(\text{r}_\text{j})}{\text{size}(im)}$$

Similarity

Similarity Measures: Shape Compatibility

We also want our merged regions to be cohesive, so we can add a measure of how well two regions "fit together".

$$fill(r_i, r_j) = 1 - \frac{\text{size}(BB_{ij}) - \text{size}(r_i) - \text{size}(r_i)}{\text{size}(im)}$$



Similarity

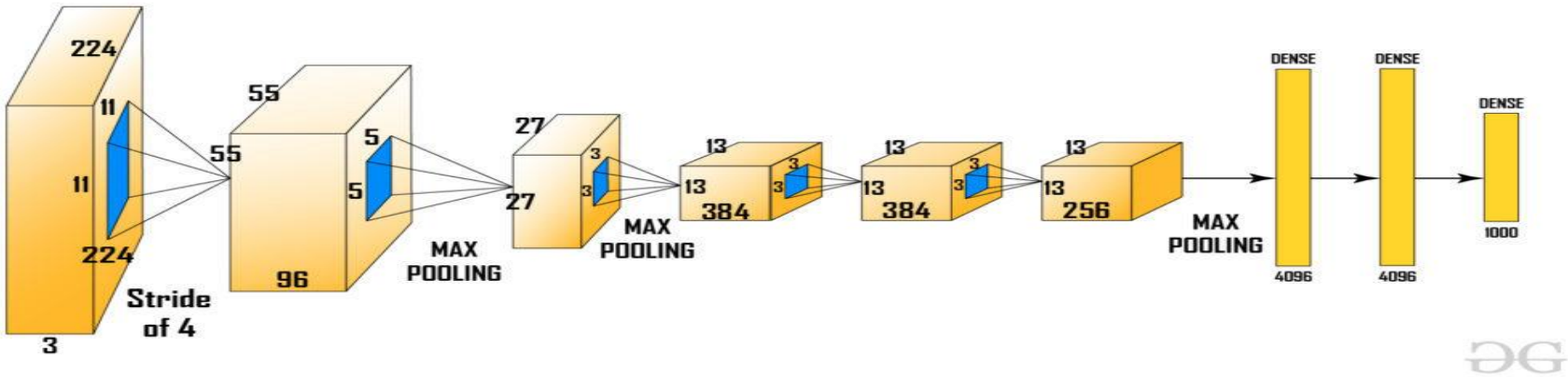
Final similarity metric:

We measure the similarity between two patches as a **linear combination** of the four given metrics:

$$s(r_i, r_j) = a_1 s_{colour}(r_i, r_j) + a_2 s_{texture}(r_i, r_j) + a_3 s_{size}(r_i, r_j) + a_4 s_{fill}(r_i, r_j),$$

Then, we can create a diverse collection of region-merging strategies by considering different weighted combinations in different color spaces.

Key Components



- The input size for AlexNet is (227, 227, 3), meaning each input image must be resized to these dimensions.
- Consequently, whether the region proposals are small or large, they need to be adjusted accordingly to fit the specified input size.
- From the above architecture, we remove the final softmax layer to obtain a (1, 4096) feature vector.
- This feature vector is then fed into both the Support Vector Machine (SVM) for classification and the bounding box regressor for improved localization.

Problems with R-CNN

- It still takes a huge amount of time to train the network as you would have to classify 2000 region proposals per image.
- It cannot be implemented real time as it takes around 47 seconds for each test image.
- The selective search algorithm is a fixed algorithm. Therefore, no learning is happening at that stage. This could lead to the generation of bad candidate region proposals.

Fast R-CNN

- The approach is similar to the R-CNN algorithm.
- But, instead of feeding the region proposals to the CNN, we feed the input image to the CNN to generate a convolutional feature map
- we identify the region of proposals and warp them into squares and by using a RoI pooling layer we reshape them into a fixed size
- From the RoI feature vector, we use a softmax layer to predict the class of the proposed region and also the offset values for the bounding box.

Faster R-CNN

- Similar to Fast R-CNN, the image is provided as an input to a convolutional network which provides a convolutional feature map
- Instead of using selective search algorithm on the feature map to identify the region proposals, a separate network is used to predict the region proposals.
- The predicted region proposals are then reshaped using a RoI pooling layer which is then used to classify the image within the proposed region and predict the offset values for the bounding boxes.

People Behind YOLO

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

Adapted from
https://www.cse.iitb.ac.in/~vkaushal/2017/01/11/yolo-real-time-object-detection/

Adapted from
https://www.cse.iitb.ac.in/~vkaushal/2017/01/11/yolo-real-time-object-detection/

YOLO's Claim to Fame

- Most accurate real-time detector
  - There are other more accurate ones, but they are not real-time
- Fastest object detector in the literature
  - Unbeaten!

Adapted from https://www.cse.iitb.ac.in/~vkaushal/2017/01/11/yolo-real-time-object-detection/

Need for RT Object Detection

- Autonomous driving
- Assistive devices
- General purpose responsive robotic systems

Evolution of Object Detection

- Haar - 1998
- SIFT – 1999
- Viola Jones Haar Cascades – 2001
- HOG – 2005
- SURF – 2006
- Region based segmentation and object detection – 2009
- DPM – 2010
- OverFeat – 2013
- SelectiveSearch – 2013
- DNN for Detection 2013
- DeCaf (Deep Convolutional Features) - 2014
- R-CNN – 2014
- Fast R-CNN, Faster R-CNN – 2015

How is YOLO different?

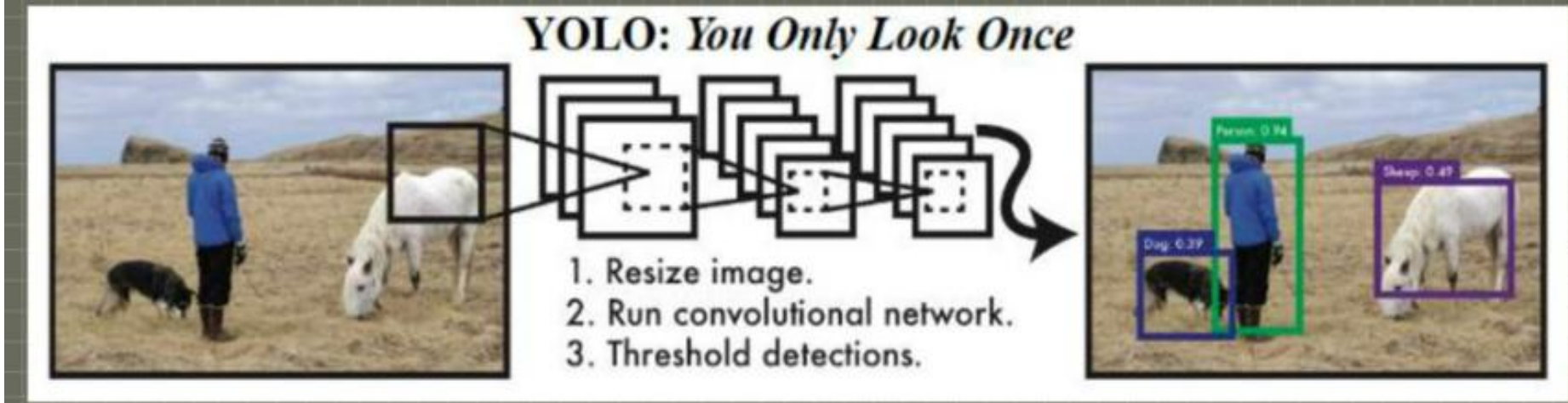
Prior Techniques

- Repurposes classifiers to perform detection
- Take a classifier for that object and evaluate it at various locations and scales in a test image (sliding window/region proposals)

Yolo

- A single regression problem (single neural network), straight from image pixels to bounding box coordinates and class probabilities
- Predictions directly from full images in one evaluation, about all classes

YOLO


YOLO: You Only Look Once
1. Resize image.
2. Run convolutional network.
3. Threshold detections.

How YOLO Works?

- Input image: S×S grid
  - If the center of an object falls into a grid cell, that grid cell is responsible for detecting that object

How YOLO Works?

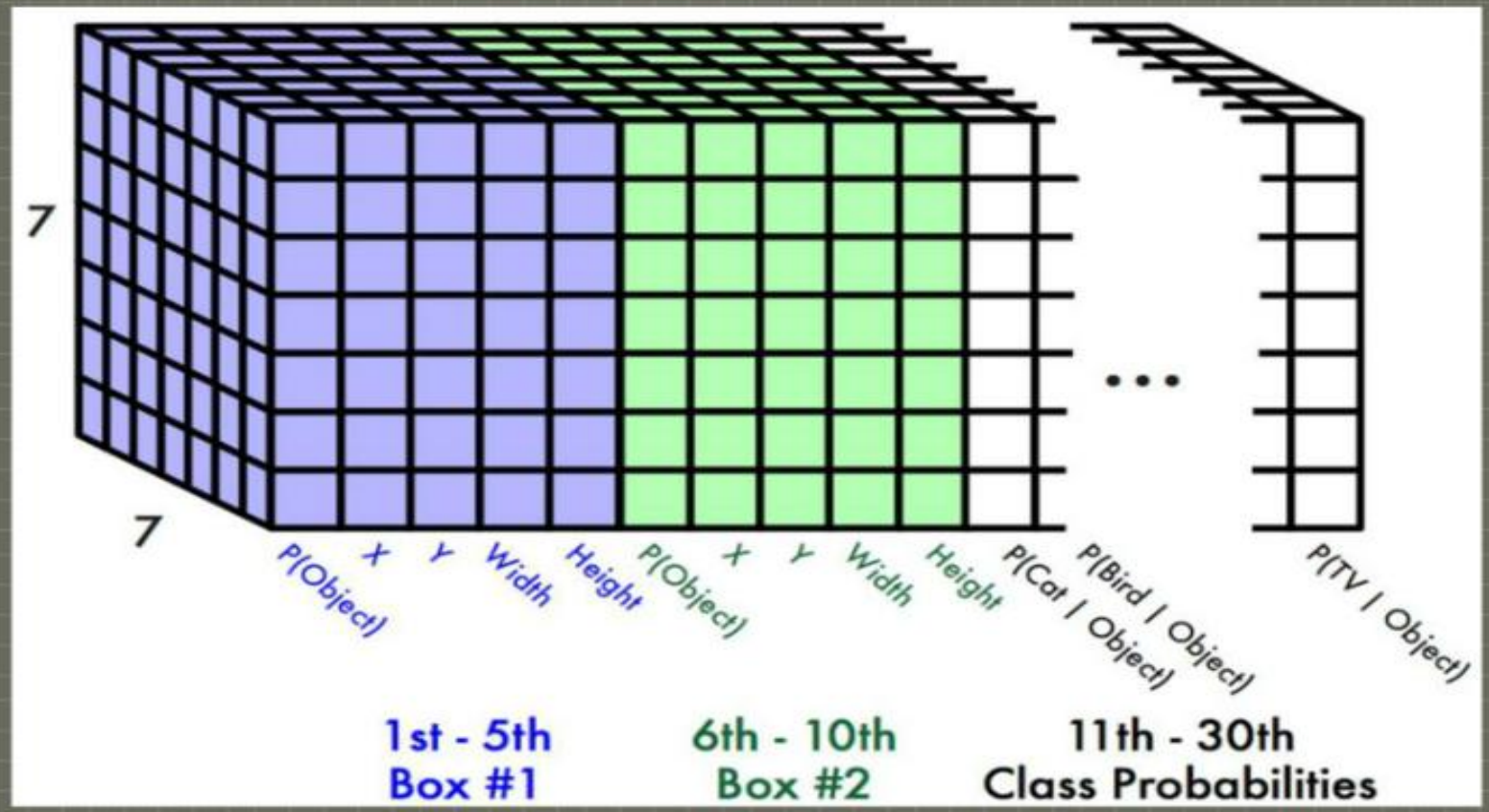
- Each grid cell predicts B bounding boxes and C conditional class probabilities $Pr(Class_i \mid Object)$
- Each bounding box consists of 5 predictions: x, y, w, h and confidence
  - x,y → center of box relative to bounds of grid cell [0, 1]
  - w,h → relative to whole image [0, 1]
  - Confidence = Pr(Object) * IOU between the predicted box and ground truth
- These predictions are encoded as S X S X (B*5 + C) tensor

How YOLO Works?

- Multiply the conditional class probabilities and the individual box confidence predictions to get class-specific confidence scores for each box
  - These scores encode both the probability of that class appearing in the box and how well the predicted box fits the object

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

Output Tensor



https://docs.google.com/presentation/d/1kAa7NOamBt4calBU9iHgT8a86RRHz9Yz2oh4-GTdX6M/edit?usp=sharing

Layers	filters	Size/stride	output	notes
Conv	64	$7 \times 7/2$	$224 \times 224 \times 64$	backbone
Maxpool		$2 \times 2/2$	$112 \times 112 \times 64$	
Conv	192	$3 \times 3/1$	$112 \times 112 \times 192$	
Maxpool		$2 \times 2/2$	$56 \times 56 \times 192$	
Conv	128	$1 \times 1/1$	$56 \times 56 \times 128$	
Conv	256	$3 \times 3/1$	$56 \times 56 \times 256$	
Conv	256	$1 \times 1/1$	$56 \times 56 \times 256$	
Conv	512	$3 \times 3/1$	$56 \times 56 \times 512$	
Maxpool		$2 \times 2/2$	$28 \times 28 \times 512$	
Conv	256	$1 \times 1/1$	$28 \times 28 \times 256$	
Conv	512	$3 \times 3/1$	$28 \times 28 \times 512$	
Conv	256	$1 \times 1/1$	$28 \times 28 \times 256$	
Conv	512	$3 \times 3/1$	$28 \times 28 \times 512$	
Conv	256	$1 \times 1/1$	$28 \times 28 \times 256$	
Conv	512	$3 \times 3/1$	$28 \times 28 \times 512$	
Conv	256	$1 \times 1/1$	$28 \times 28 \times 256$	
Conv	512	$3 \times 3/1$	$28 \times 28 \times 512$	
Conv	512	$1 \times 1/1$	$28 \times 28 \times 512$	
Conv	1024	$3 \times 3/1$	$28 \times 28 \times 1024$	
Maxpool		$2 \times 2/2$	$14 \times 14 \times 1024$	
Conv	512	$1 \times 1/1$	$14 \times 14 \times 512$	
Conv	1024	$3 \times 3/1$	$14 \times 14 \times 1024$	
Conv	512	$1 \times 1/1$	$14 \times 14 \times 512$	
Conv	1024	$3 \times 3/1$	$14 \times 14 \times 1024$	
Conv	1024	$3 \times 3/1$	$14 \times 14 \times 1024$	
Conv	1024	$3 \times 3/2$	$7 \times 7 \times 1024$	
Conv	1024	$3 \times 3/1$	$7 \times 7 \times 1024$	
Conv	1024	$3 \times 3/1$	$7 \times 7 \times 1024$	
FC			4096	neck
FC			1470	
reshape			$7 \times 7 \times 30$	head