# Bounding boxes

DEEP LEARNING FOR IMAGES WITH PYTORCH

**Michal Oleszak**
Machine Learning Engineer

# What is object recognition?

**Object recognition** identifies objects in images:

- Location of each object in an image (bounding box)
- Class label of each object

**Applications:** surveillance, medical diagnosis, traffic management, sports analytics

- In this video: annotation with bounding boxes
- In later videos: evaluation and models

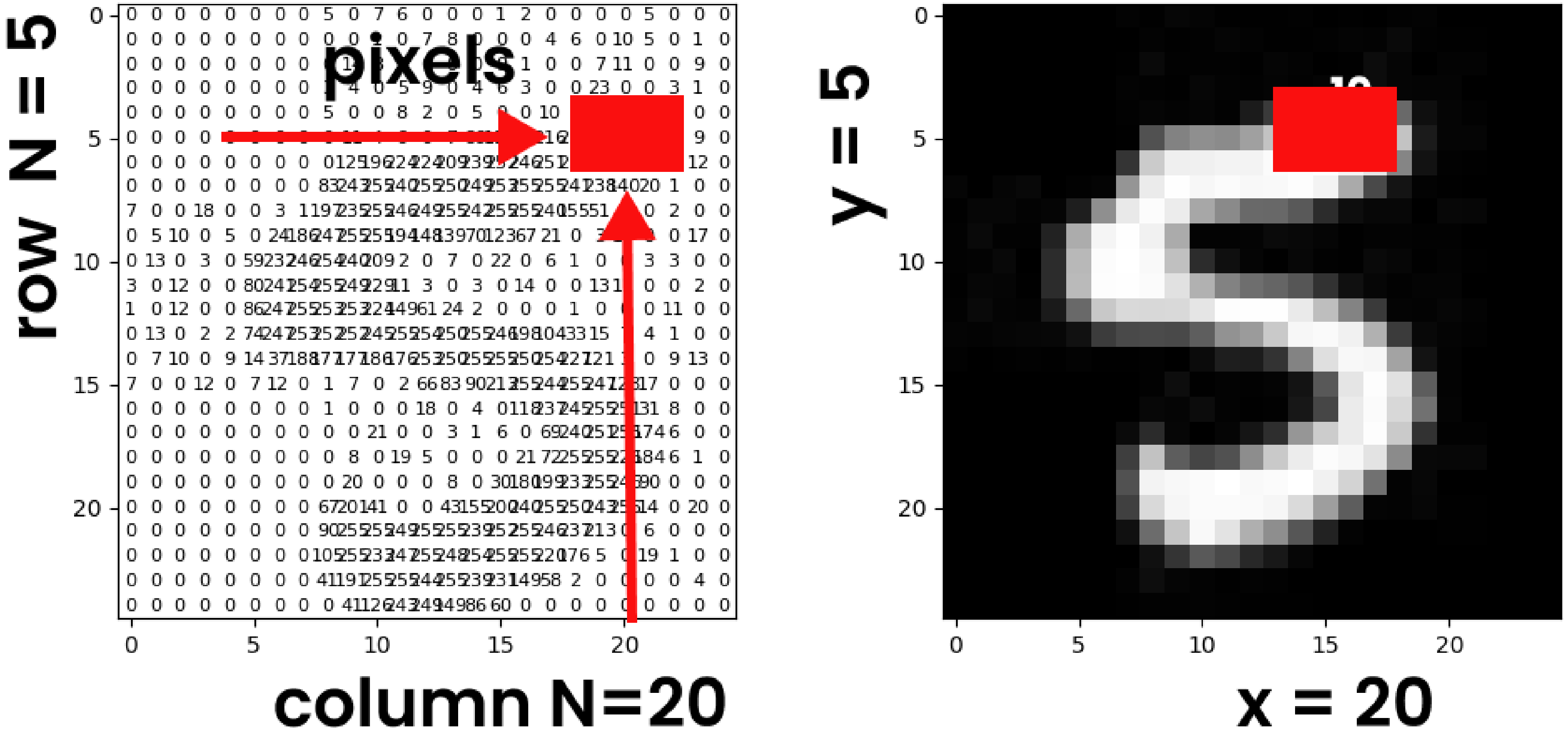
# Bounding box representation

- A rectangular box describing the object's spatial location
- Training data annotations & model outputs
- Ground truth bounding box: precise object location

# Bounding box representation

- A rectangular box describing the object's spatial location
- Training data annotations & model outputs
- Ground truth bounding box: precise object location
- Bounding box coordinates:
  - Top left and bottom right
  - Bounding box = (x1, y1, x2, y2)
  - x1 = x_min, x2 = x_max, ...

# Pixels and coordinates




- **Coordinates:** x - the column number, y - the row number
- **Origin:** (0, 0) - the top left corner

# Converting pixels to tensors

## Transforming with `ToTensor()`

- Tensor type:
  - torch.float
- Scaled tensor range:
  - [0.0, 1.0]

```
import torchvision.transforms as transforms
transform = transforms.Compose([
            transforms.Resize(224),
            transforms.ToTensor()
            ])
image_tensor = transform(image)
```

## Tranforming with `PILToTensor()`

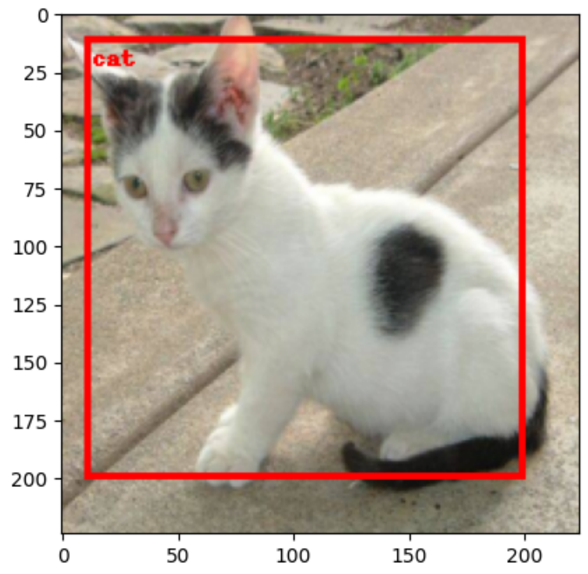
- Tensor type:
  - torch.uint8 (8-bit integer)
- Unscaled tensor range:
  - [0, 255]

```
import torchvision.transforms as transforms
transform = transforms.Compose([
            transforms.Resize(224),
            transforms.PILToTensor()
            ])
image_tensor = transform(image)
```

# Drawing the bounding box

```python
from torchvision.utils import draw_bounding_boxes

bbox = torch.tensor([x_min, y_min, x_max, y_max])
bbox = bbox.unsqueeze(0)
bbox_image = draw_bounding_boxes(
    image_tensor, bbox, width=3, colors="red"
)

transform = transforms.Compose([
    transforms.ToPILImage()
])
pil_image = transform(bbox_image)

import matplotlib.pyplot as plt
plt.imshow(pil_image)
```

- Import `draw_bounding_boxes`
- Collect coordinates into a tensor
- Unsqueeze to two dimensions
- Transform to image and plot



# Let's practice!

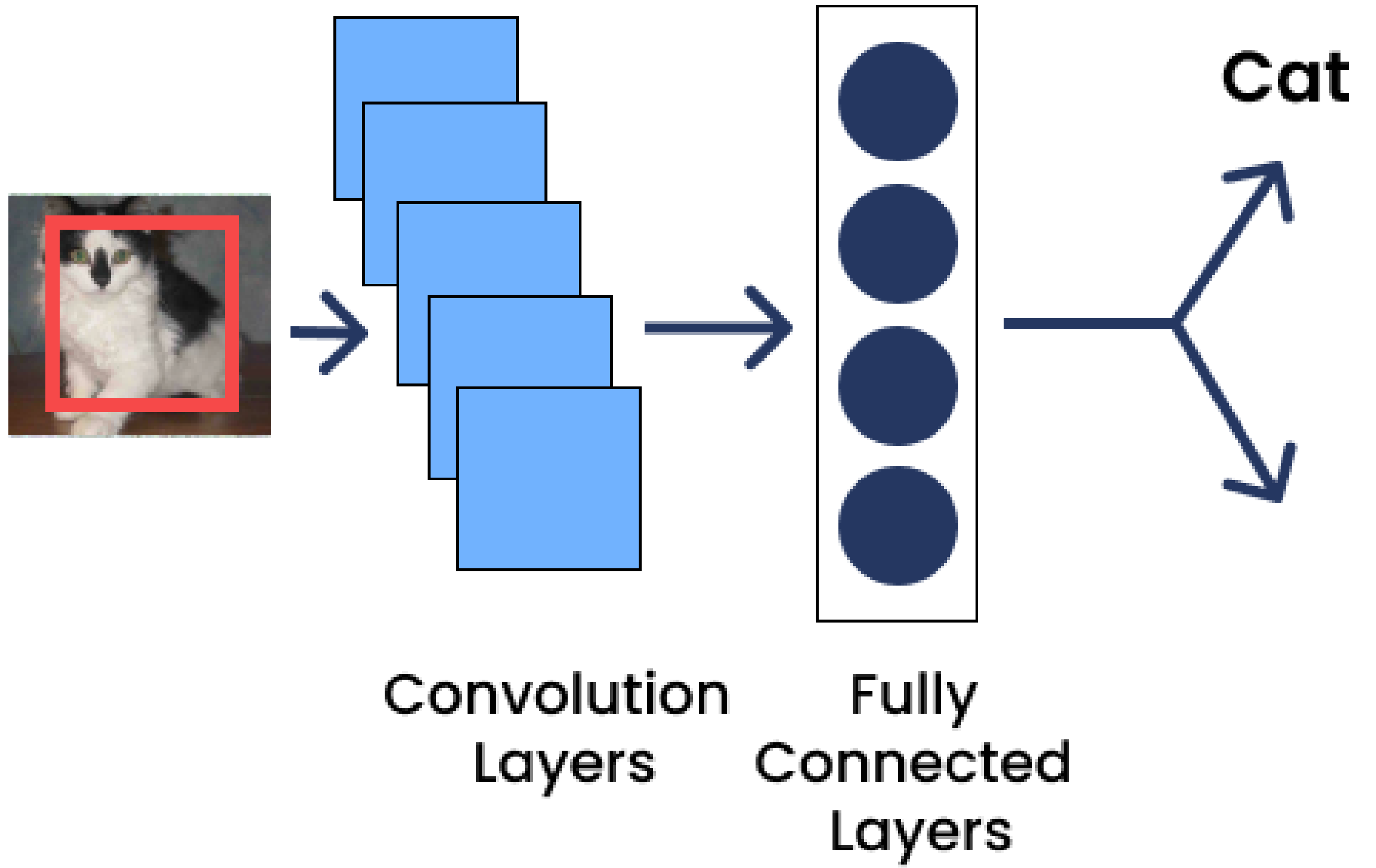
DEEP LEARNING FOR IMAGES WITH PYTORCH

# Evaluating object recognition models

DEEP LEARNING FOR IMAGES WITH PYTORCH

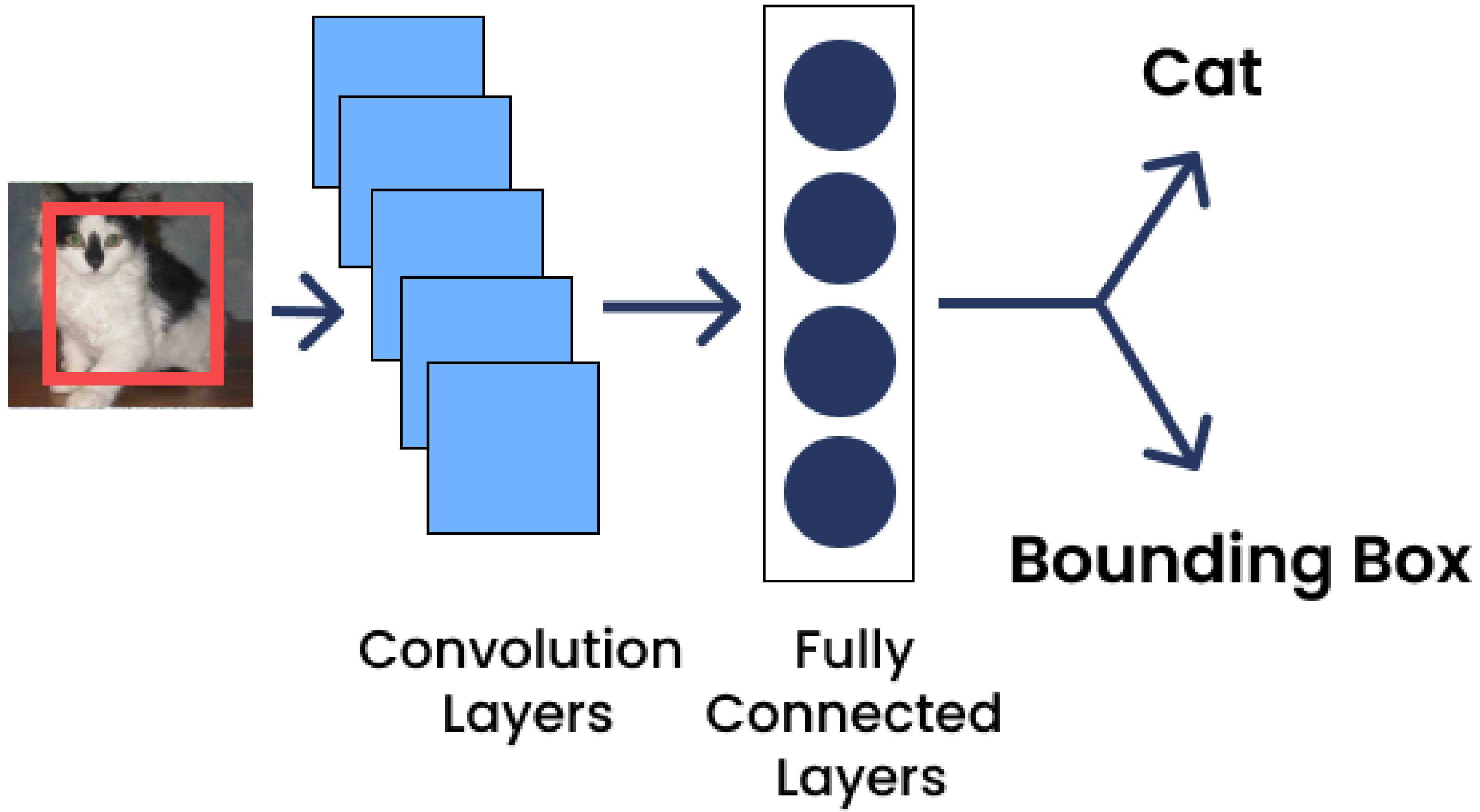
**Michal Oleszak**
Machine Learning Engineer

# Classification and localization




- Output 1: Classification (e.g., cat)

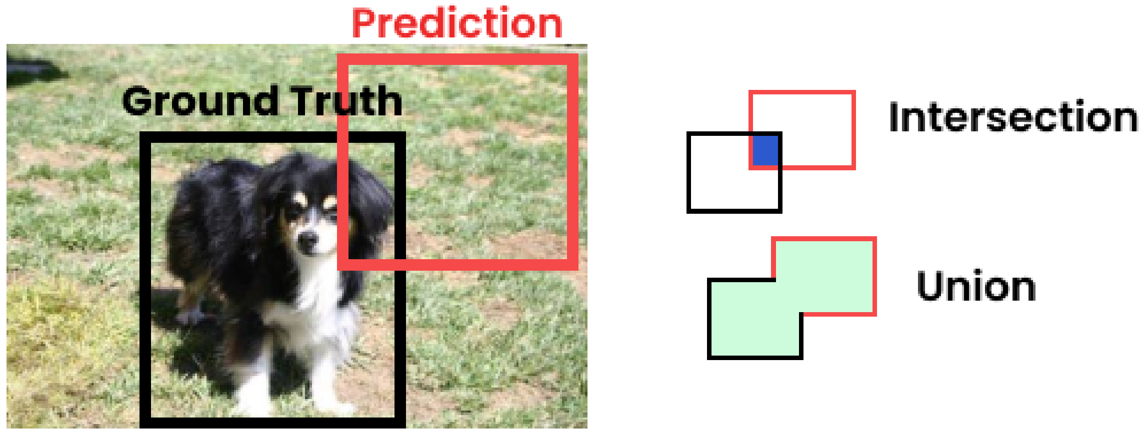
# Classification and localization




- Output 1: Classification (e.g., cat)
- Output 2: Bounding box regression [x1, y1, x2, y2]

# Intersection over union (IoU)

- **Object of interest:** object in image we want to detect (e.g., dog)
- **Ground truth box:** the accurate bounding box around the object of interest
- **Intersection over Union:** a metric to measure the overlap between two boxes




- IoU = Area of Intersection / Area of Union
  - IoU = 0 no overlap, IoU = 1 perfect overlap
  - IoU >0.5 is a good prediction

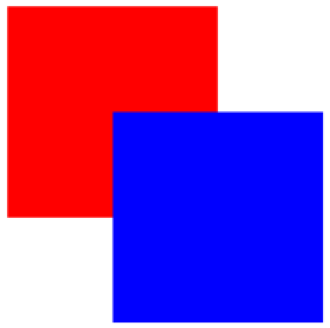
# IoU in PyTorch

```
bbox1 = [50, 50, 150, 150]
bbox2 = [100, 100, 200, 200]

bbox1 = torch.tensor(bbox1).unsqueeze(0)
bbox2 = torch.tensor(bbox2).unsqueeze(0)
```

```
from torchvision.ops import box_iou

iou = box_iou(bbox1, bbox2)
print(iou)
```

```
tensor([[0.1429]])
```

- Two sets of boxes (x1, y1, x2, y2)



- Convert vectors to 2-D tensors
- Calculate IoU

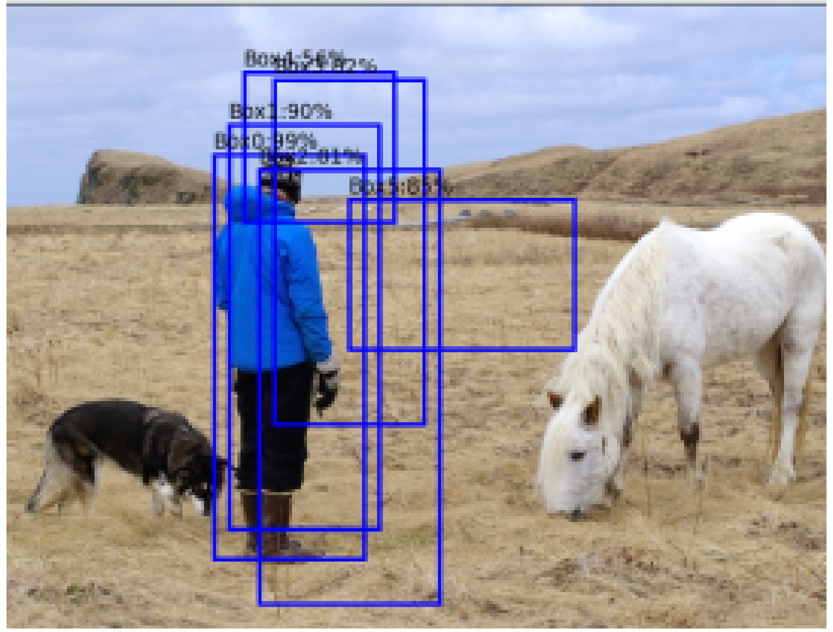
# Predicting bounding boxes

```python
model.eval()
with torch.no_grad():
    output = model(input_image)
print(output)
```

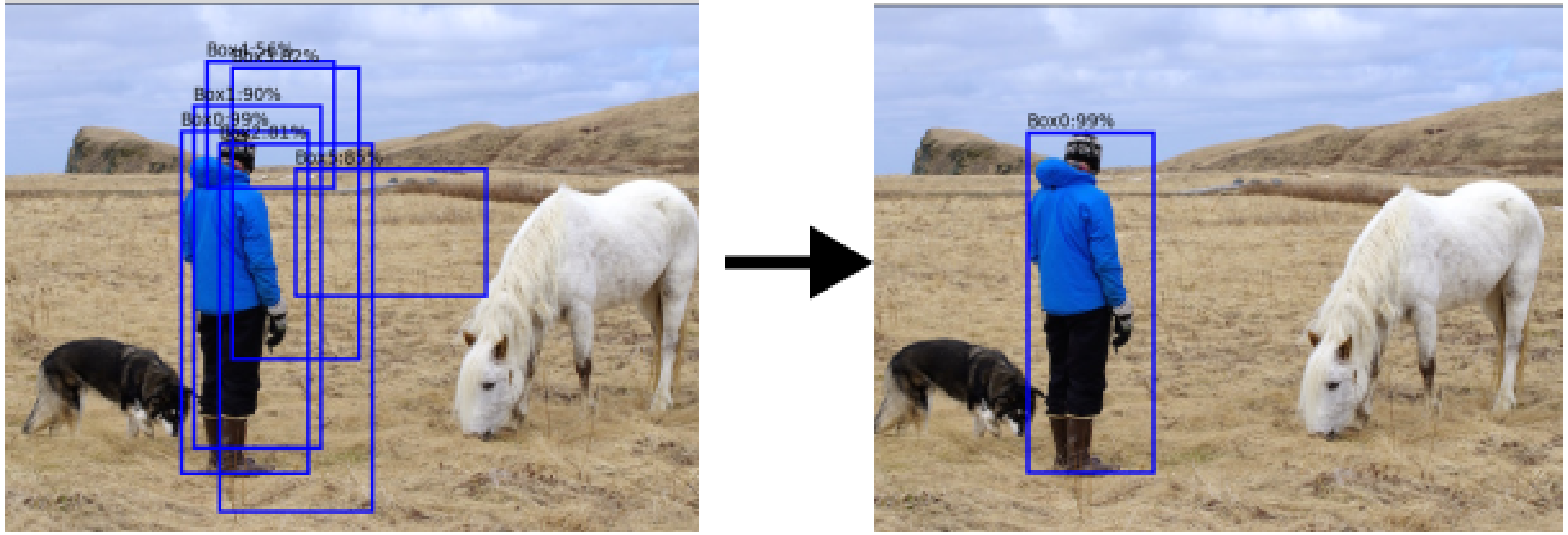
```
[{'boxes': tensor([[ 42.8553, 271.9481, 180.6003, 346.7082],
                  [191.6016,  80.4759, 247.8009, 387.5475], ....),
'scores': tensor([1.0000, 1.0000, 0.9998, ... ]),
'labels': tensor([18,  1, 20, 18, 18, 18 ...])
}]
```

```python
boxes = output[0]["boxes"]
scores = output[0]["scores"]
```

# Non-max suppression (NMS)



# Non-max suppression (NMS)




**Non-max suppression:** a common technique to select the most relevant bounding boxes

- **Non-max:** discarding boxes with low confidence score to contain an object
- **Suppression:** discarding boxes with low IoU

# Non-max suppression in PyTorch

```python
from torchvision.ops import nms

box_indices = nms(
    boxes=boxes,
    scores=scores,
    iou_threshold=0.5,
)
print(box_indices)
```

```
tensor([ 0,   1,   2,   8])
```

```python
filtered_boxes = boxes[box_indices]
```

- **Boxes:** tensors with the bounding box coordinates of the shape [N, 4]
- **Scores:** tensor with the confidence score for each box of the shape [N]
- **iou_threshold:** the threshold between 0.0 and 1.0
- Output: indices of filtered bounding boxes

# Let's practice!

DEEP LEARNING FOR IMAGES WITH PYTORCH

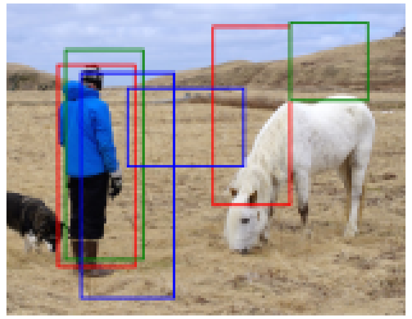
# Object detection using R-CNN

DEEP LEARNING FOR IMAGES WITH PYTORCH

**Michal Oleszak**
Machine Learning Engineer

# Region-based CNN family: R-CNN

R-CNN family: R-CNN, Fast-CNN, Faster CNN



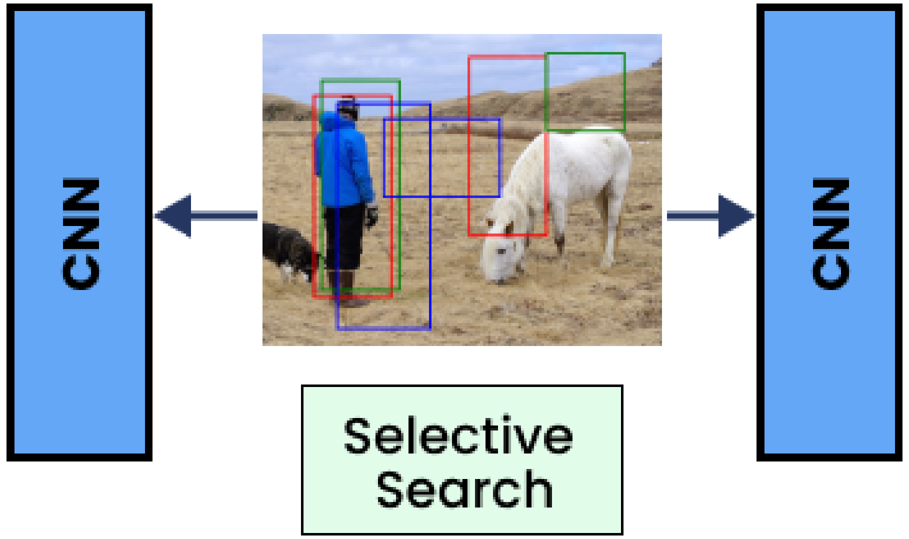
Selective Search

- Module 1: generation of region proposals

[1] Citation: Jason Brownlee. 2019. Deep Learning for Computer Vision.

# Region-based CNN family: R-CNN

R-CNN family: R-CNN, Fast-CNN, Faster CNN

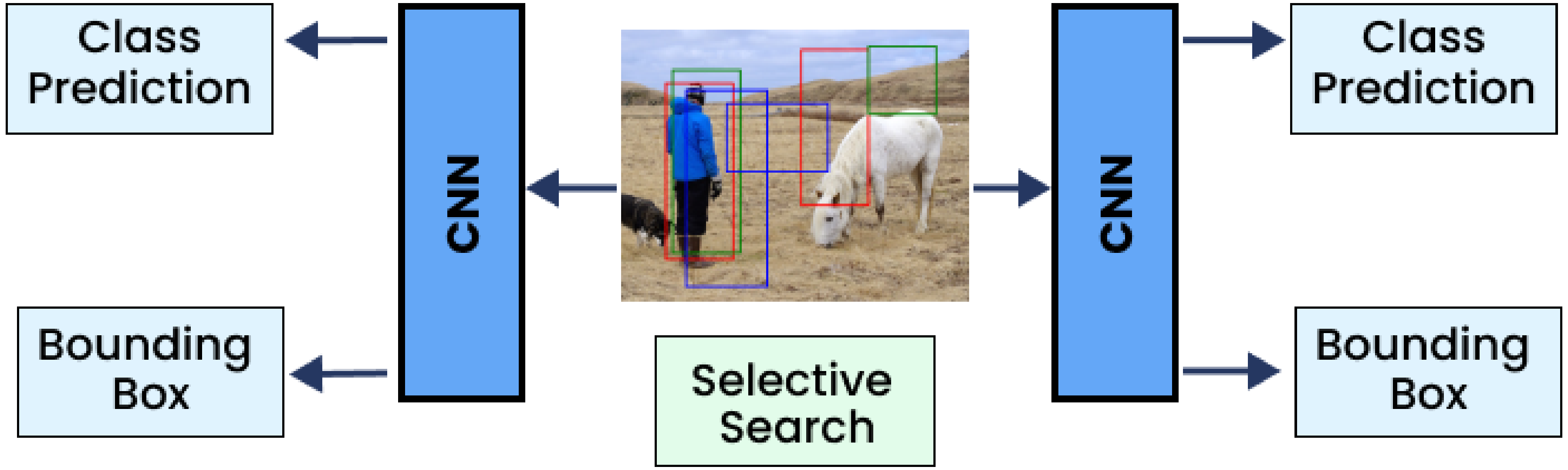



- Module 1: generation of region proposals
- Module 2: feature extraction (convolutional layers)

[1] Citation: Jason Brownlee. 2019. Deep Learning for Computer Vision.

# Region-based CNN family: R-CNN

R-CNN family: R-CNN, Fast-CNN, Faster CNN

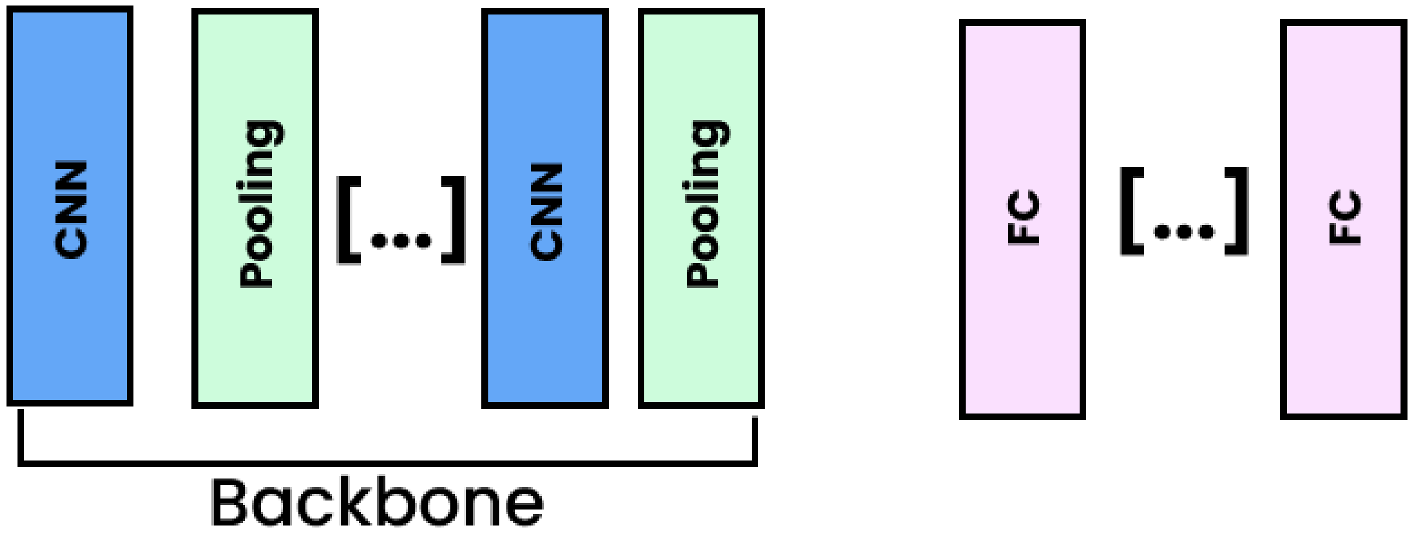



- Module 1: generation of region proposals
- Module 2: feature extraction (convolutional layers)
- Module 3: class and bounding box prediction

[1] Citation: Jason Brownlee. 2019. Deep Learning for Computer Vision.

# R-CNN: backbone

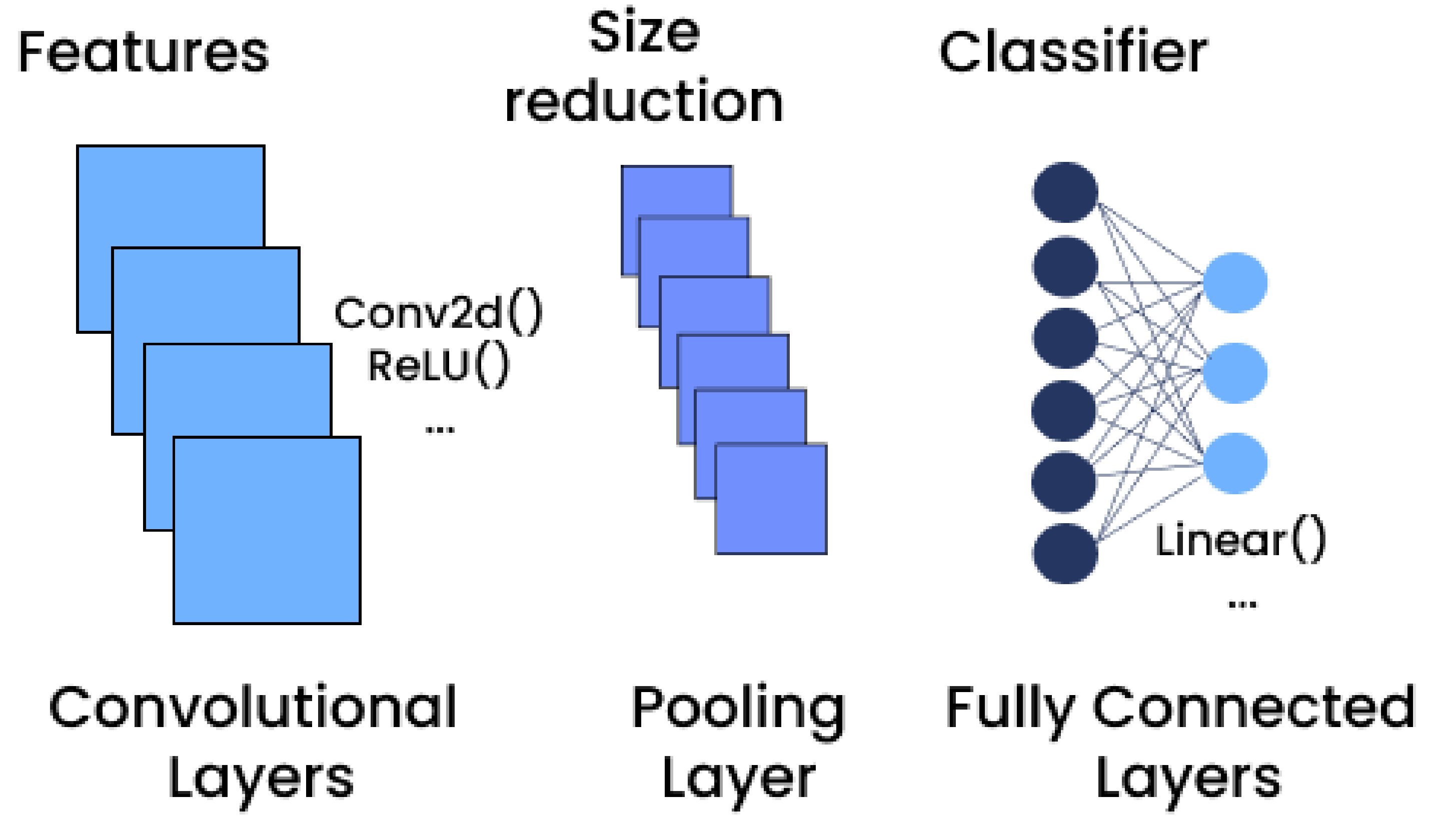
- Convolutional layers: pre-trained models
  - **Backbone:** the core CNN architecture responsible for feature extraction




- Convolutional & pooling layers
- Extract features for region proposals and object detection

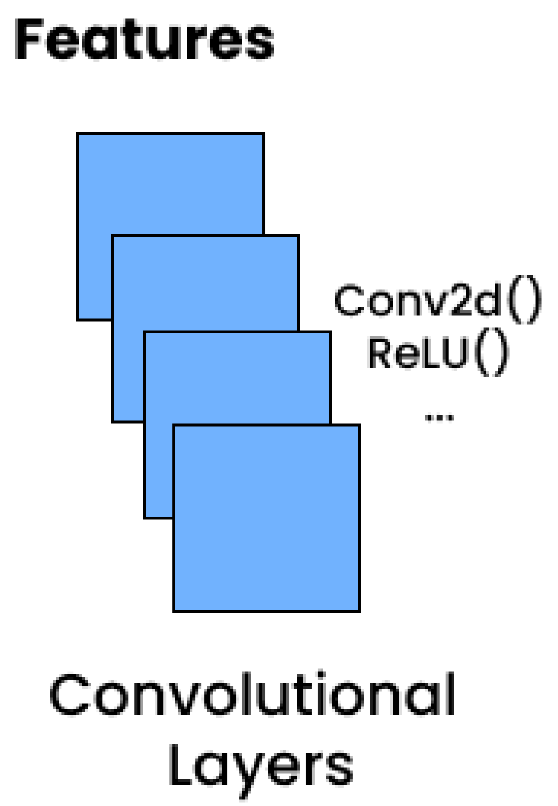
# R-CNN: backbone with PyTorch

```python
import torch.nn as nn
from torchvision.models import vgg16,
    VGG16_Weights

vgg = vgg16(weights=VGG16_Weights.DEFAULT)
```



# R-CNN: backbone with PyTorch

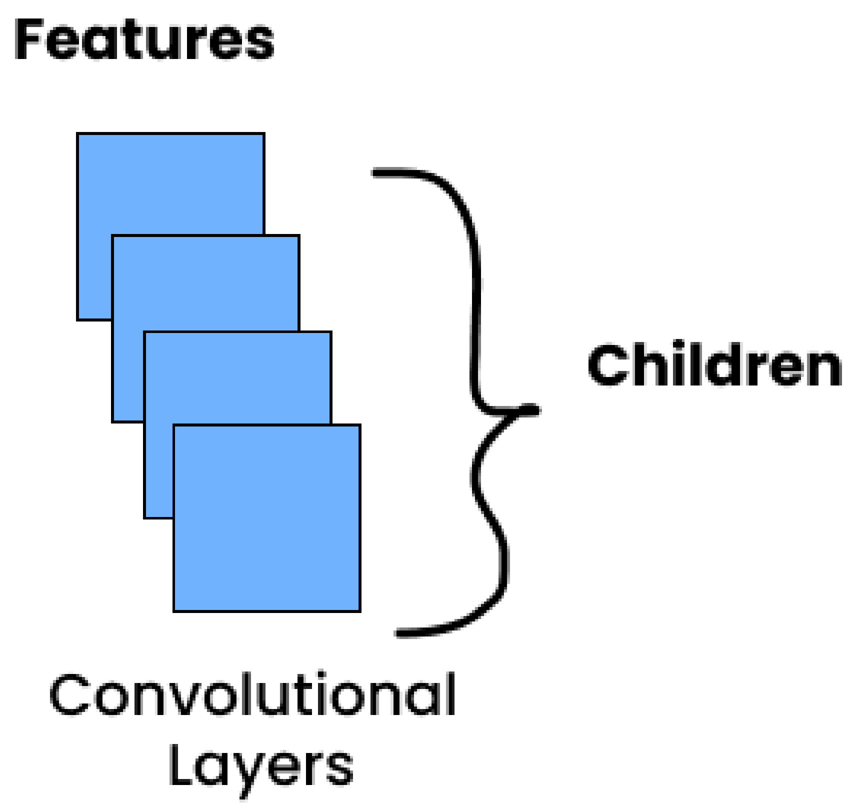
```python
import torch.nn as nn
from torchvision.models import vgg16,
    VGG16_Weights

vgg = vgg16(weights=VGG16_Weights.DEFAULT)
```




- `.features` : only convolutional layers

# R-CNN: backbone with PyTorch

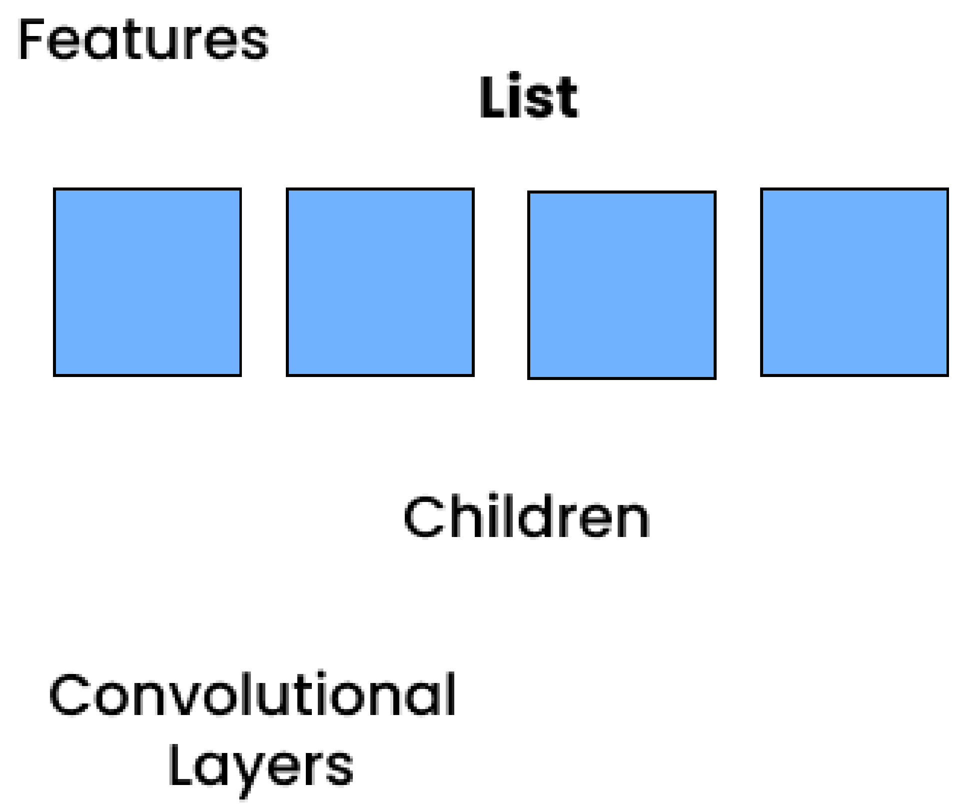
```python
import torch.nn as nn
from torchvision.models import vgg16,
    VGG16_Weights

vgg = vgg16(weights=VGG16_Weights.DEFAULT)
```




- `.features` : only convolutional layers
- `.children()` : all layers from block

# R-CNN: backbone with PyTorch

```
import torch.nn as nn
from torchvision.models import vgg16,
    VGG16_Weights

vgg = vgg16(weights=VGG16_Weights.DEFAULT)

backbone = nn.Sequential(
    *list(vgg.features.children())
)
```




- `nn.Sequential(*list())` : all sub-layers are placed into a sequential block as a list
  - `*` : unpacks the elements from the list

- `.features` : only convolutional layers
- `.children()` : all layers from block

# R-CNN: classifier layer

- Extract backbone's output size

```
input_dimension = nn.Sequential(*list(
    vgg_backbone.classifier.children())
)[0].in_features
```

- Create a new classifier

```
classifier = nn.Sequential(
    nn.Linear(input_dimension, 512),
    nn.ReLU(),
    nn.Linear(512, num_classes),
)
```

# R-CNN: box regressor layer

- Sits on top of the backbone
- 4 outputs for the 4 box coordinates

```
box_regressor = nn.Sequential(
    nn.Linear(input_dimension, 32),
    nn.ReLU(),
    nn.Linear(32, 4),
)
```

# Putting it all together: object detection model

```
class ObjectDetectorCNN(nn.Module):
    def __init__(self):
        super(ObjectDetectorCNN, self).__init__()
        vgg = vgg16(weights=VGG16_Weights.DEFAULT)
        self.backbone = nn.Sequential(*list(vgg.features.children()))
        input_features = nn.Sequential(*list(vgg.classifier.children()))[0].in_features
        self.classifier = nn.Sequential(
            nn.Linear(input_features, 512),
            nn.ReLU(),
            nn.Linear(512, 2),
        )
        self.box_regressor = nn.Sequential(
            nn.Linear(input_features, 32),
            nn.ReLU(),
            nn.Linear(32, 4),
        )
```

# Putting it all together: object detection model

```python
class ObjectDetector(nn.Module):
    (...)

    def forward(self, x):
        features = self.backbone(x)
        bboxes = self.regressor(features)
        classes = self.classifier(features)
        return bboxes, classes
```

# Running object recognition

1. Load and transform the image
2. `unsqueeze()` the image to add the batch dimension
3. Pass the image tensor to the model
4. Run Non-Max Suppression ( `nms()` ) over model's output
5. `draw_bounding_boxes()` on top of the image

# Let's practice!

DEEP LEARNING FOR IMAGES WITH PYTORCH

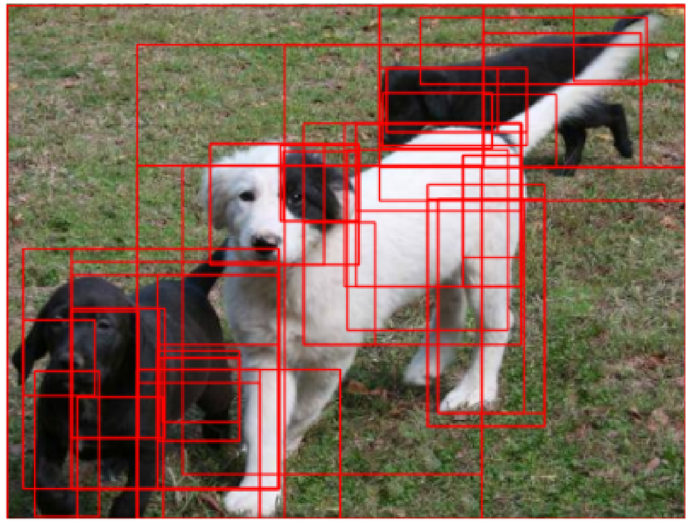
# Region network proposals with Faster R-CNN

DEEP LEARNING FOR IMAGES WITH PYTORCH

**Michal Oleszak**
Machine Learning Engineer

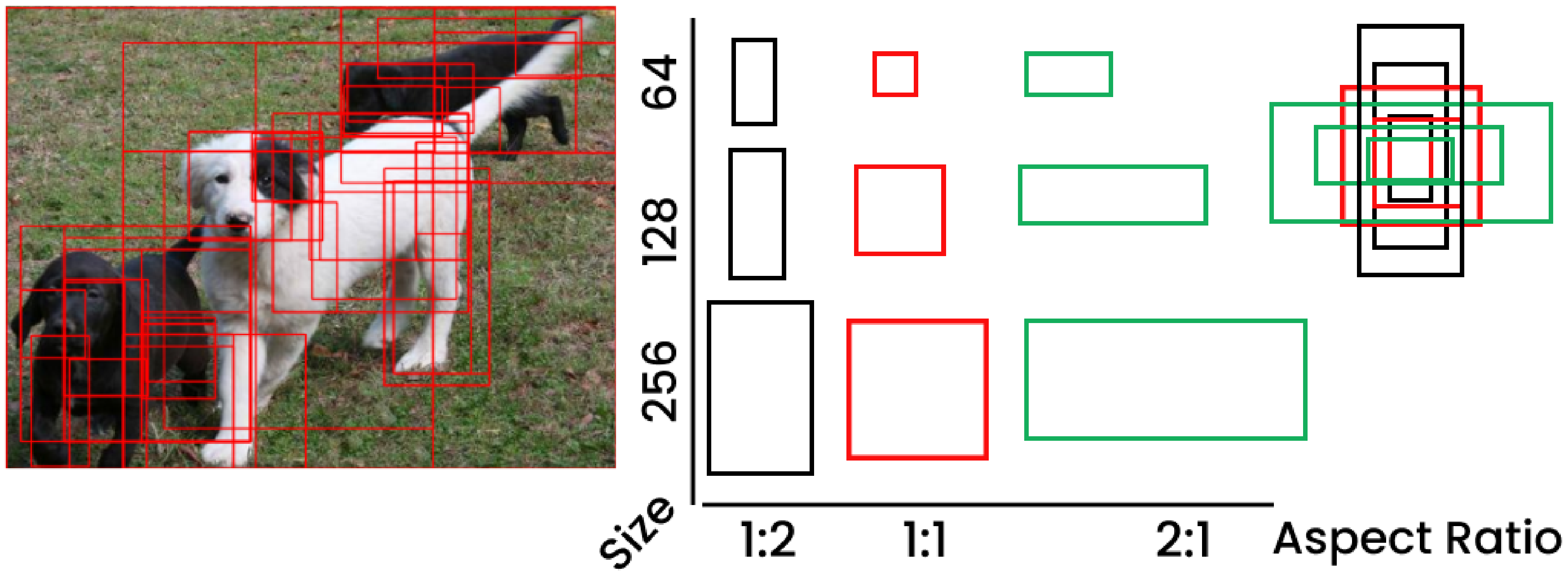
# Regions and anchor boxes

- **Region:** a smaller area of the image that could contain objects of interest, grouped by visual characteristics



# Regions and anchor boxes

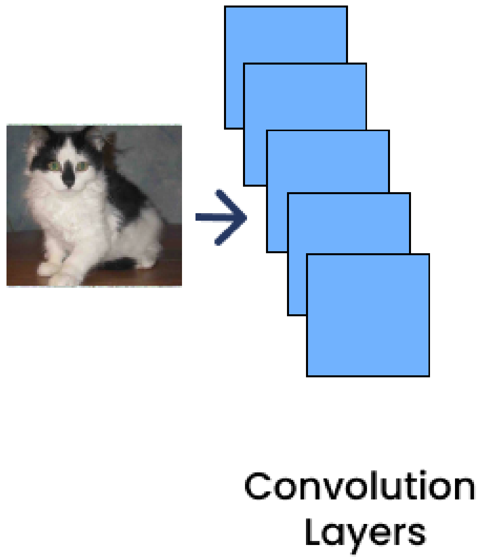
- **Region:** a smaller area of the image that could contain objects of interest, grouped by visual characteristics




- **Anchor box:** predefined bounding box templates of different sizes and shapes

# Faster R-CNN model

**Faster R-CNN:** an advanced version of R-CNN

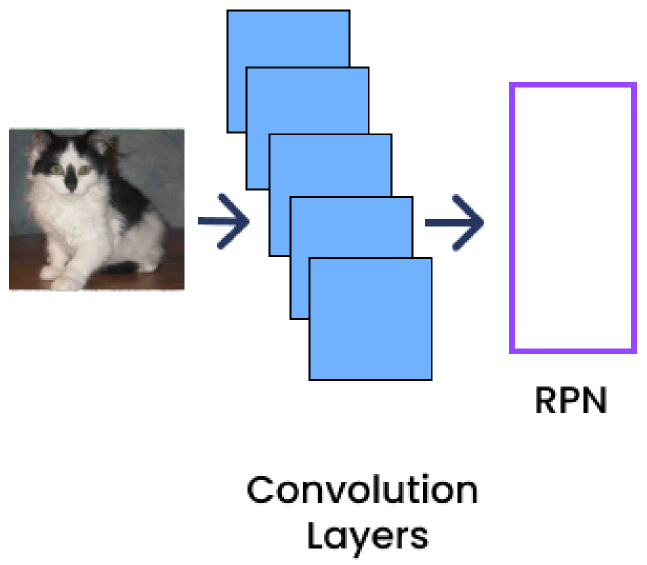



- Backbone (convolutional layers)

[1] Edward Raff. 2022. Inside Deep Learning.

# Faster R-CNN model

**Faster R-CNN:** an advanced version of R-CNN

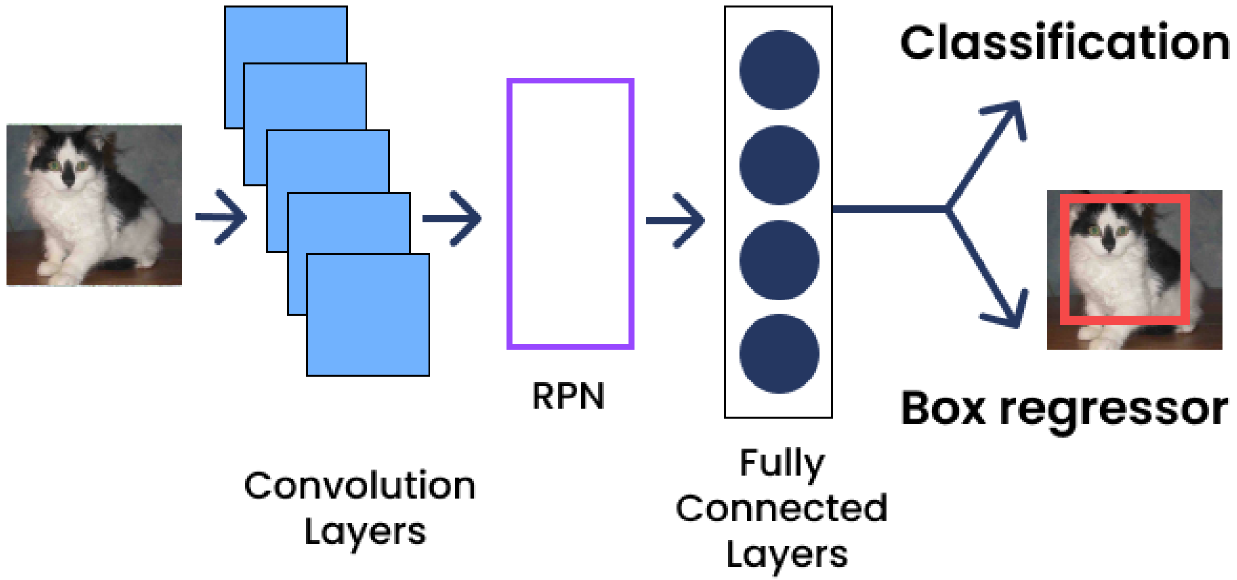



- Backbone (convolutional layers)
- Region proposal network (RPN) for bounding box proposals

[1] Edward Raff. 2022. Inside Deep Learning.

# Faster R-CNN model

**Faster R-CNN:** an advanced version of R-CNN

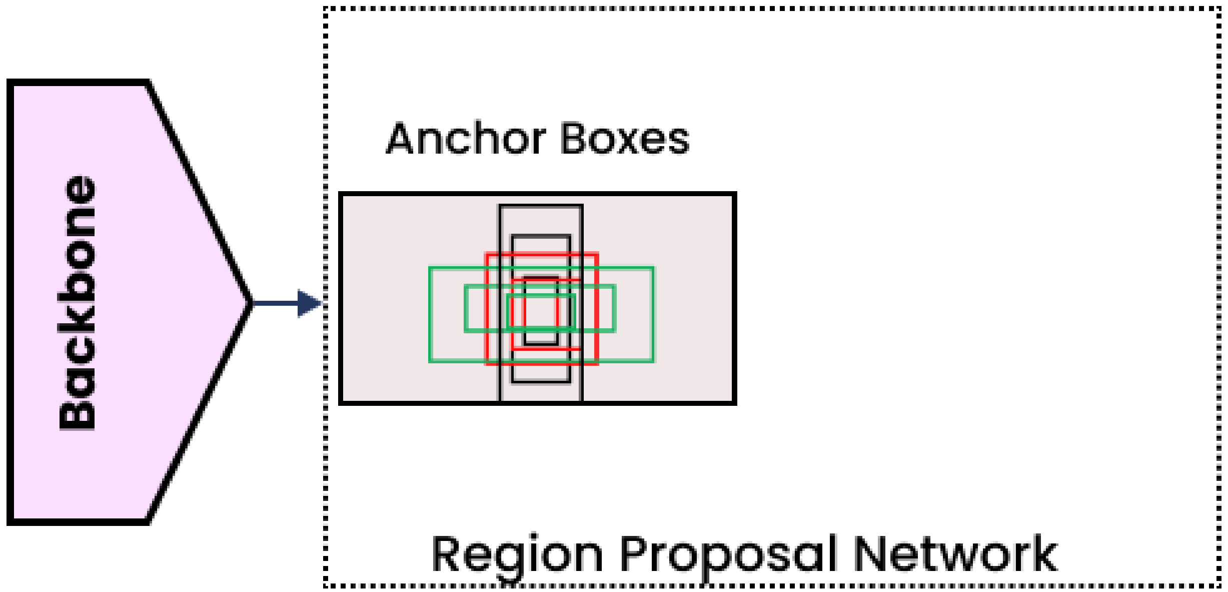



- Convolution layers (backbone): feature maps
- Region proposal network (RPN): bounding box proposals
- Classifier and regressor to produce predictions

[1] Edward Raff. 2022. Inside Deep Learning.

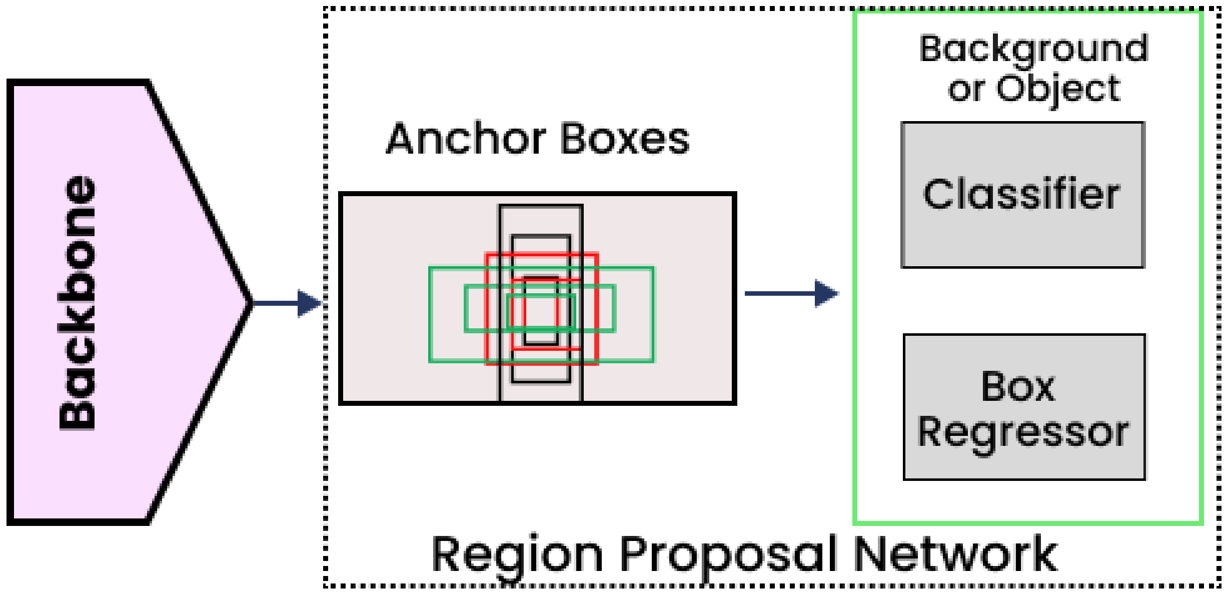
# Region proposal network (RPN)

# Region proposal network (RPN)



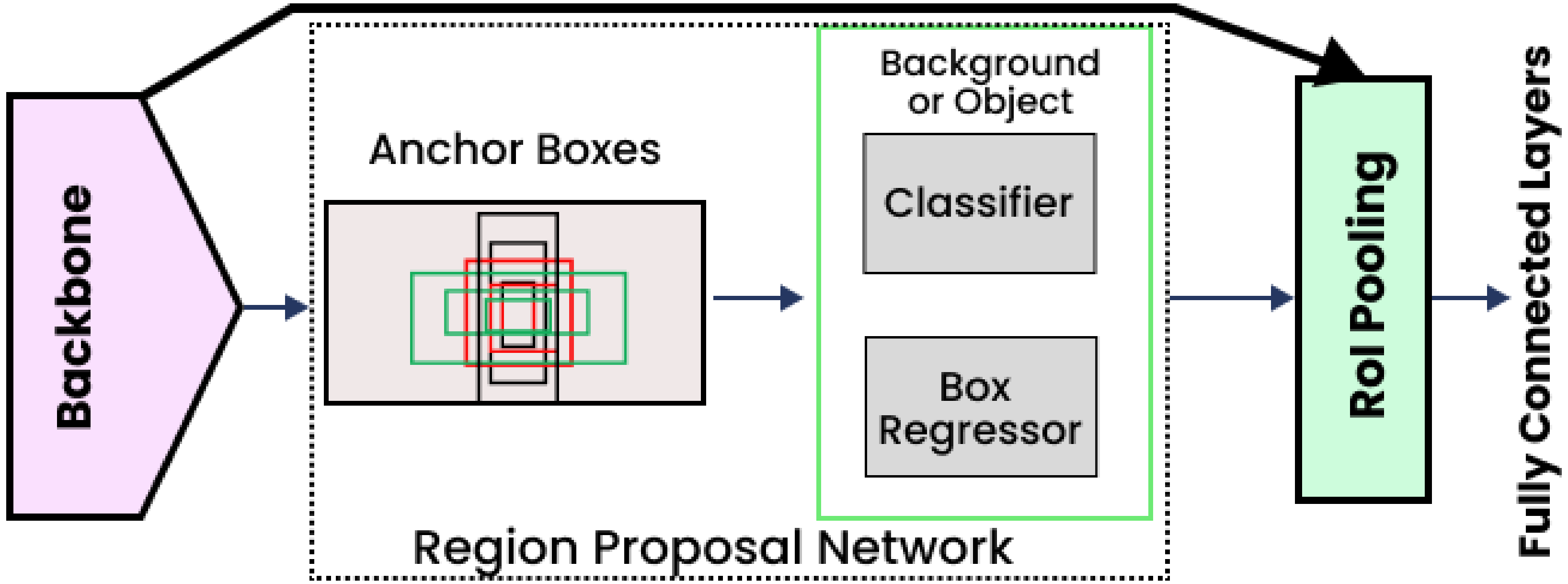

- Anchor generator:
  - Generate a set of anchor boxes of different sizes and aspect ratios

# Region proposal network (RPN)




- Anchor generator:
  - Generate a set of anchor boxes of different sizes and aspect ratios
- Classifier and regressor:
  - Predict if the box contains an object and provide coordinates

# Region proposal network (RPN)




- Anchor generator:
  - Generate a set of anchor boxes of different sizes and aspect ratios
- Classifier and regressor:
  - Predict if the box contains an object and provide coordinates
- Region of interest (RoI) pooling:
  - Resize the RPN proposal to a fixed size for fully connected layers

# RPN in PyTorch

```
from torchvision.models.detection.rpn import AnchorGenerator

anchor_generator = AnchorGenerator(
    sizes=((32, 64, 128),),
    aspect_ratios=((0.5, 1.0, 2.0),),
)
```

```
from torchvision.ops import MultiScaleRoIAlign

roi_pooler = MultiScaleRoIAlign(
    featmap_names=["0"],
    output_size=7,
    sampling_ratio=2,
)
```

# Fast R-CNN loss functions

- RPN classification loss:
  - region contains object or not
  - binary cross-entropy
  - `rpn_cls_criterion = nn.BCEWithLogitsLoss()`
- R-CNN classification loss:
  - multiple object classes
  - cross-entropy
  - `rcnn_cls_criterion = nn.CrossEntropyLoss()`
- RPN box regression loss:
  - bounding box coordinates
  - mean squared error
  - `rpn_reg_criterion = nn.MSELoss()`
- R-CNN box regression loss:
  - bounding box coordinates
  - mean squared error
  - `rcnn_reg_criterion = nn.MSELoss()`

# Faster R-CNN in PyTorch

```
from torchvision.models.detection import FasterRCNN

backbone = torchvision.models.mobilenet_v2(weights="DEFAULT").features
backbone.out_channels = 1280

model = FasterRCNN(
        backbone=backbone,
        num_classes=num_classes,
        rpn_anchor_generator=anchor_generator,
        box_roi_pool=roi_pooler,
)
```

# Faster R-CNN in PyTorch

Load pre-trained Faster R-CNN

```
from torchvision.models.detection.faster_rcnn import FastRCNNPredictor
model = torchvision.models.detection.fasterrcnn_resnet50_fpn(weights="DEFAULT")
```

Define number of classes and classifier input sise

```
num_classes = 2
in_features = model.roi_heads.box_predictor.cls_score.in_features
```

Replace model's classifier with a one with the desired number of classes

```
model.roi_heads.box_predictor = FastRCNNPredictor(in_features, num_classes)
```

# Let's practice!

DEEP LEARNING FOR IMAGES WITH PYTORCH