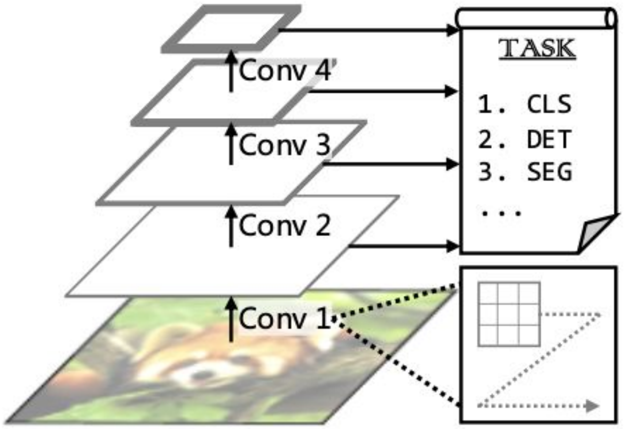
Bottleneck Transformers for Visual Recognition

A Self-attention model for vision

Will, Tin, Tarunyaa

Premise: Out with CNNs

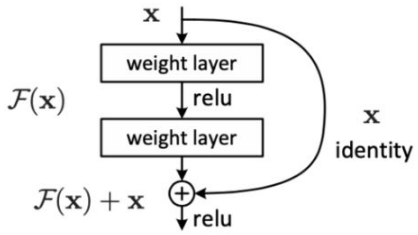
Better alternatives to pure CNNs for CV such as object detection, image classification



(a) CNNs: VGG [53], ResNet [21], *etc.*

➔ **ResNet: Residual Neural Network**

Intermediate input added to the output of a aries of convolutional blocks to enable scaling.



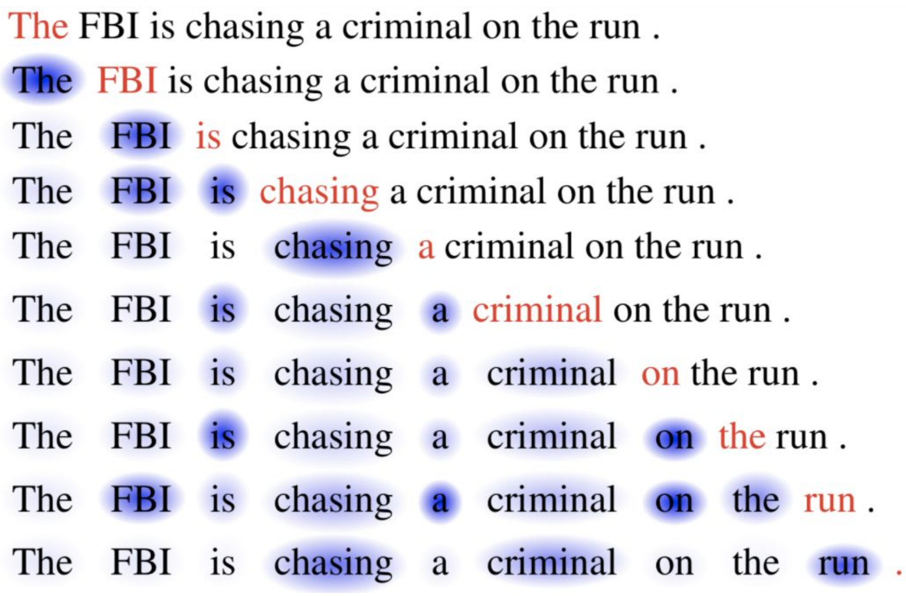
They capture local patterns information but they fail to understand long-term dependencies + require many layers.

Premise: Out with CNNs

Self-attention **to model long-term dependencies**

- ➔ Longer-term memory than RNNs & LSTMs
- ➔ NLP to Image processing
- ➔ GPT, BERT

They capture long term dependencies and don't require as many layers.

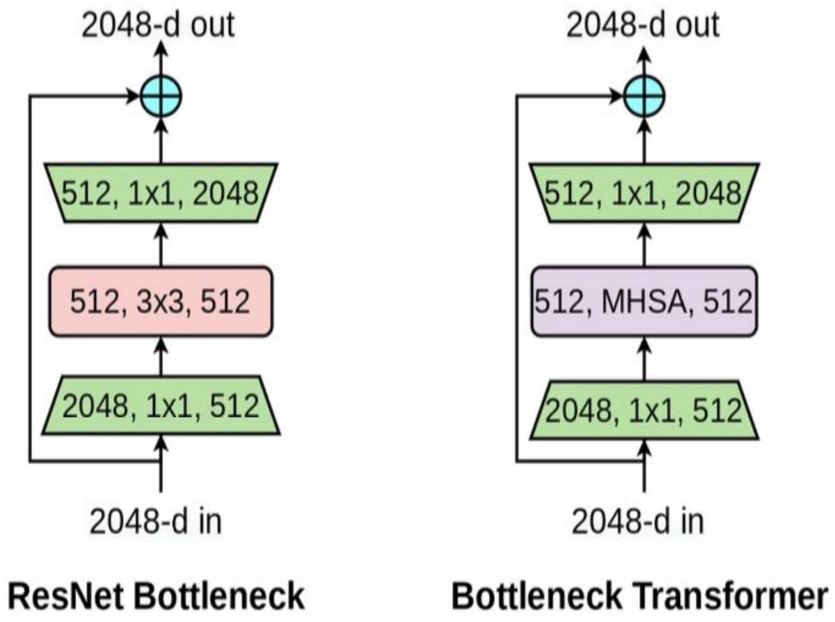


Proposal: Using Self-Attention in Vision

HYBRID SOLUTION: Replace spatial convolutional layers w/ multi-head self attention layer

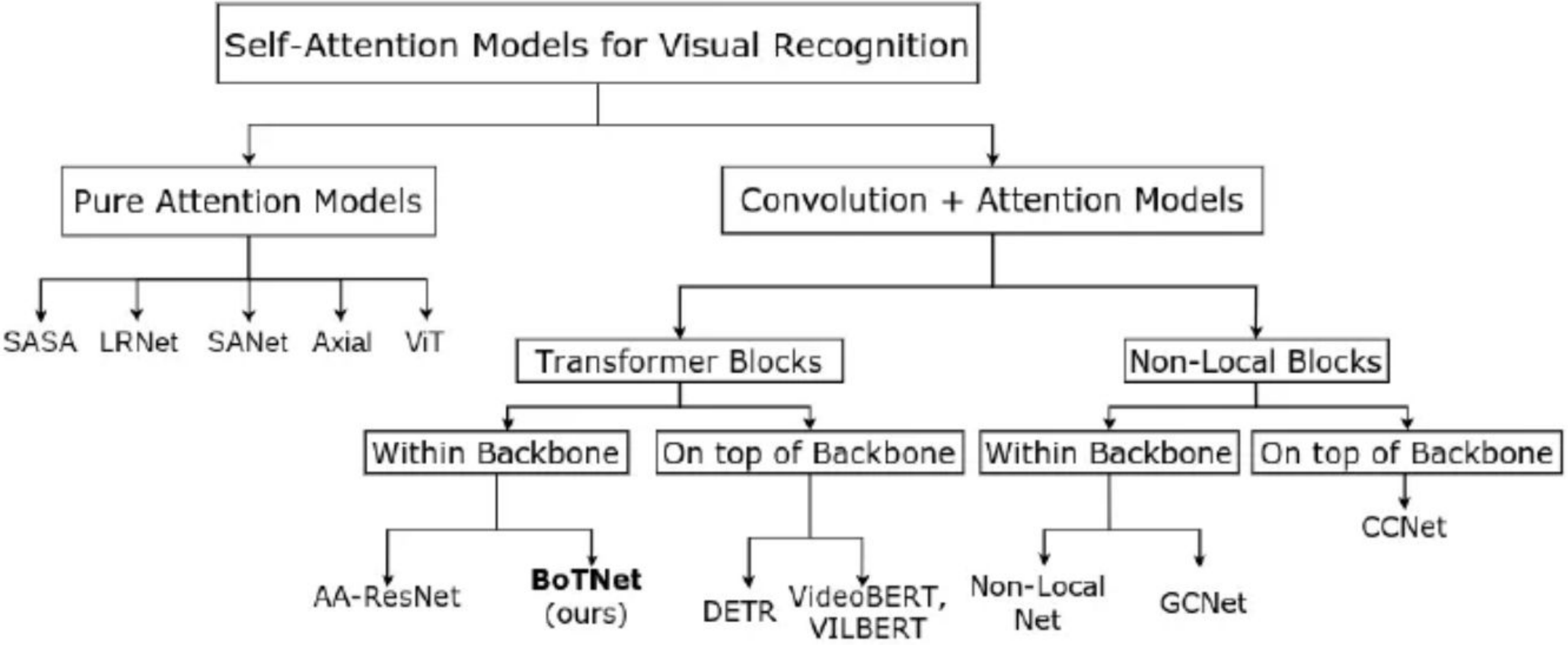
- ➔ Use convolutions to deal with large images efficiently; do spatial downsampling
- ➔ Letting global self-attention work on small resolutions

Avoids processing large images w/ self-attention since its memory & computation required scales quadratically w/ spatial dimensions



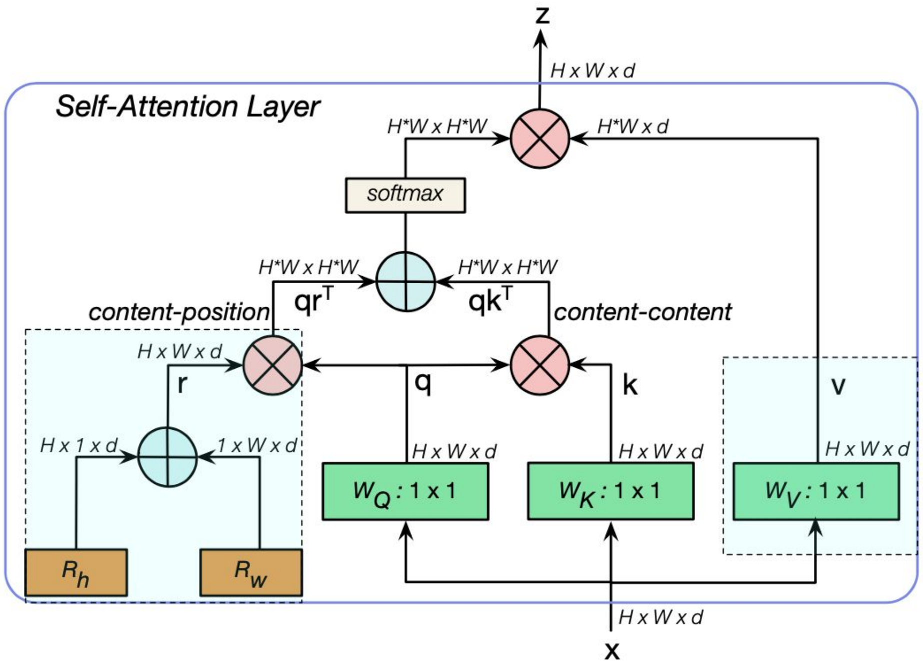
Proposal: Using Self-Attention in Vision

HYBRID SOLUTION: Replace spatial convolutional layers w/ multi-head self attention layer



Method: Positional Encoding

Making the attention-operation *position aware*



➔ Global attention is performed on a 2D feature map

➔ Split relative position encodings, *Rh* and *Rw*, for height and width respectively

Method: Model Architecture

Low relative overhead

stage	output	ResNet-50	BoTNet-50
c1	512 × 512	7×7, 64, stride 2	7×7, 64, stride 2
c2	256 × 256	3×3 max pool, stride 2 $\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	3×3 max pool, stride 2 $\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$
c3	128 × 128	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$
c4	64 × 64	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 6$
c5	32 × 32	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ \text{MHSA}, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$
# params.		**25.5**$\times10^6$	**20.8**$\times10^6$
M.Adds		**85.4**$\times10^9$	**102.98**$\times10^9$
TPU steptime		**786.5** ms	**1032.66** ms

- ➔ Only difference is the use MHSA layer in c5
- ➔ BoT50 has only 1.2 x multiple-adds and 1.3 x training overheads with 1.2x fewer parameters.

Results: Comparison w/ ResNet on Coco

BoT50 is better than R50 and R101, competitive with R152

Backbone	AP^{bb}	AP^{mk}
R50	42.1	37.7
BoT50	43.6 (+ **1.5**)	38.9 (+ **1.2**)
R101	43.3	38.4
BoT101	45.5 (+ **2.2**)	40.4 (+ **2.0**)
R152	44.2	39.1
BoT152	46.0 (+ **1.8**)	40.6 (+ **1.5**)

Relative positional encoding boosts performance

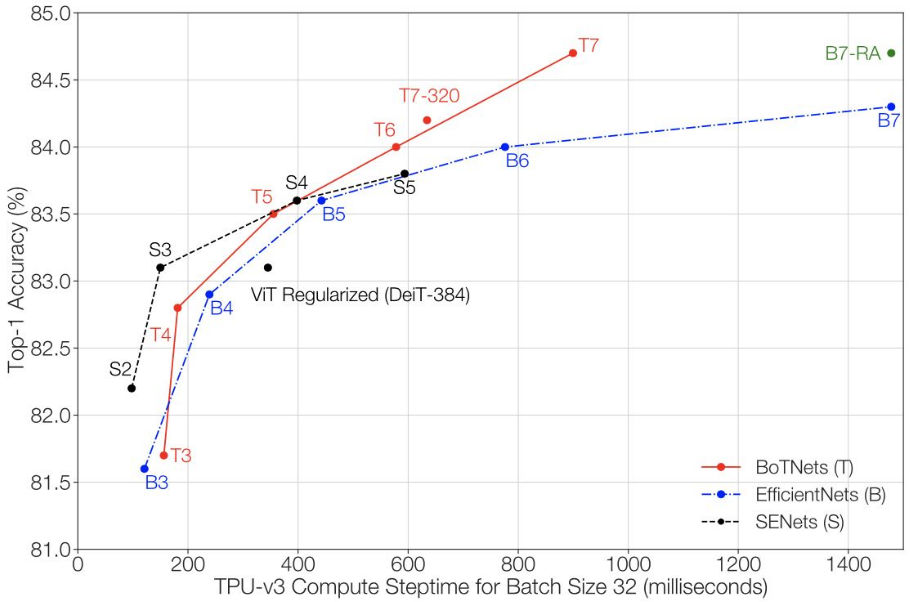
Backbone	Att. Type	AP^{bb}	AP^{mk}
R50	-	42.1	37.7
BoT50	qk^T	42.7 (+ **0.6**)	38.3 (+ **0.6**)
BoT50	qr_{relative}^T	43.1 (+ **1.0**)	38.4 (+ **0.7**)
BoT50	$qk^T + qr_{\text{relative}}^T$	43.6 (+ **1.5**)	38.9 (+ **1.2**)
BoT50	$qk^T + qr_{\text{abs}}^T$	42.5 (+ **0.4**)	38.1 (+ **0.4**)

Surpasses previous best published model on ResNet

Results: Comparison w/ ResNet on Coco

BoTNet benefits from training on larger images

Backbone	res	AP^{bb}	AP^{mk}
R50	1280	44.0	39.5
BoT50	1024	45.9 (+ **1.9**)	40.7 (+ **1.2**)
BoT50	1280	46.1 (+ **2.1**)	41.2 (+ **1.8**)
R101	1280	46.4	41.2
BoT101	1024	47.4 (+ **1.0**)	42.0 (+ **0.8**)
BoT101	1280	47.9 (+ **1.5**)	42.4 (+ **1.2**)



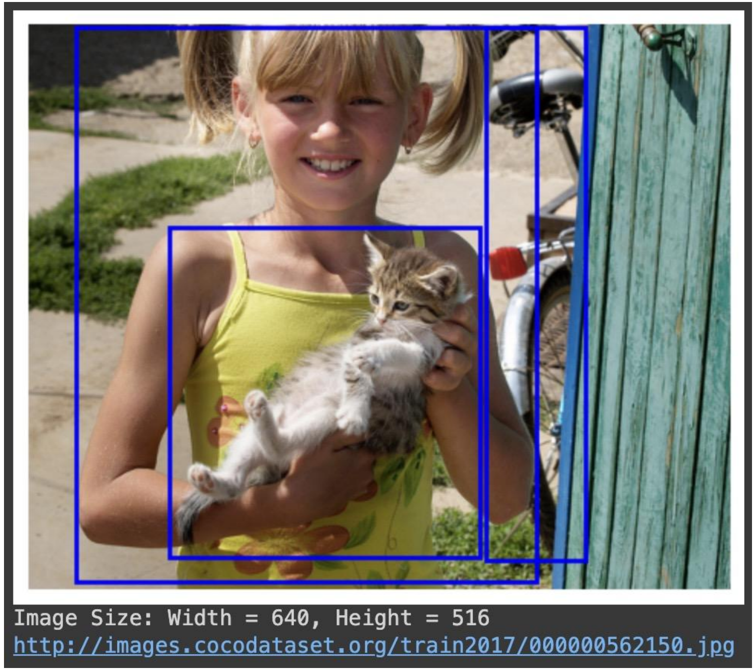
Scales well w/ larger images

COCO dataset

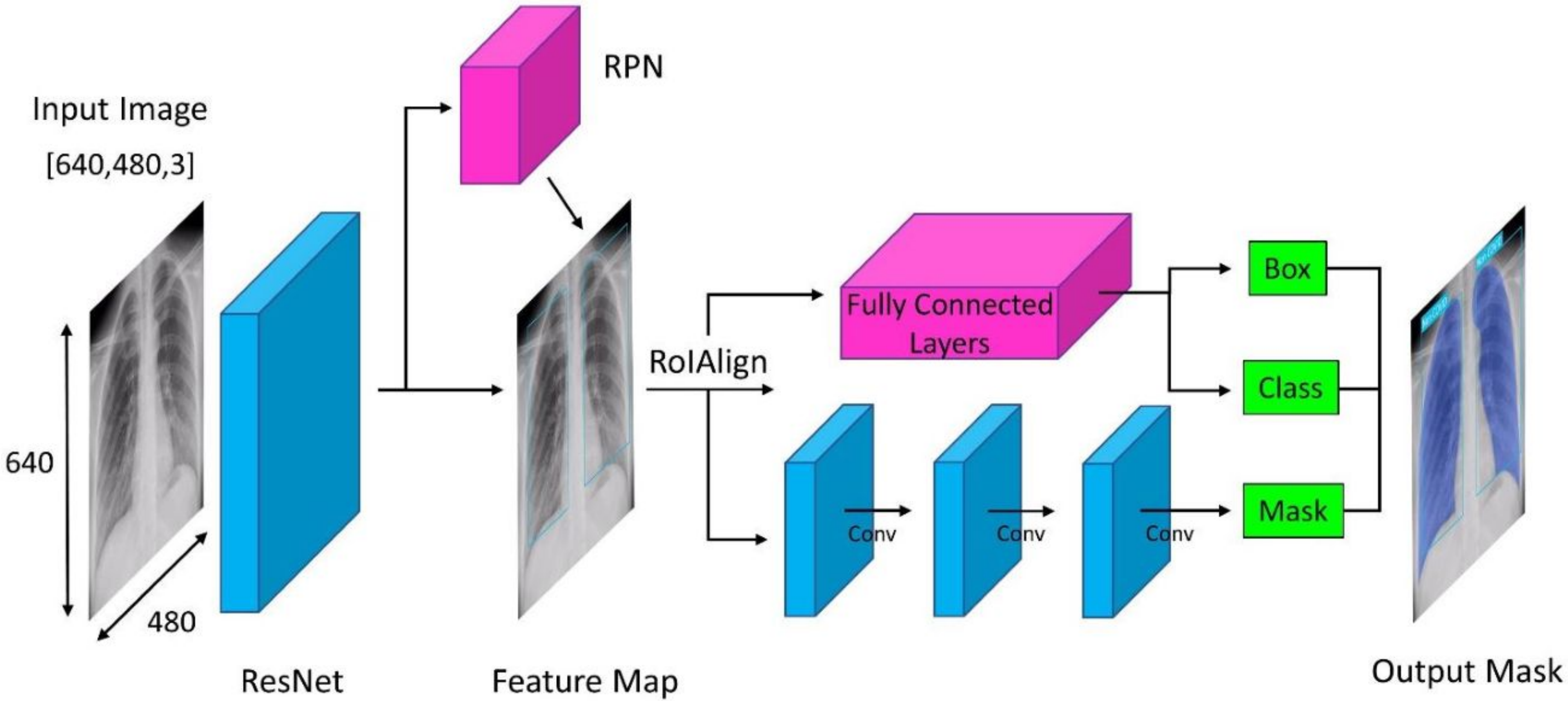
```
annotation{
  "id"              : int,
  "image_id"        : int,
  "category_id"     : int,
  "segmentation"    : RLE or [polygon],
  "area"            : float,
  "bbox"            : [x,y,width,height],
  "iscrowd"         : 0 or 1,
}

categories[{
  "id"              : int,
  "name"            : str,
  "supercategory"   : str,
}]
```

- annotations
  - captions_train2017.json
  - captions_val2017.json
  - instances_train2017.json
  - instances_val2017.json
  - person_keypoints_train2017.json
  - person_keypoints_val2017.json
- resized_train2017
- sample_data
- train2017
  - 000000000009.jpg
  - 000000000025.jpg
  - 000000000030.jpg
  - 000000000034.jpg
  - 000000000036.jpg
  - 000000000042.jpg
  - 000000000049.jpg

Instance segmentation



Object detection

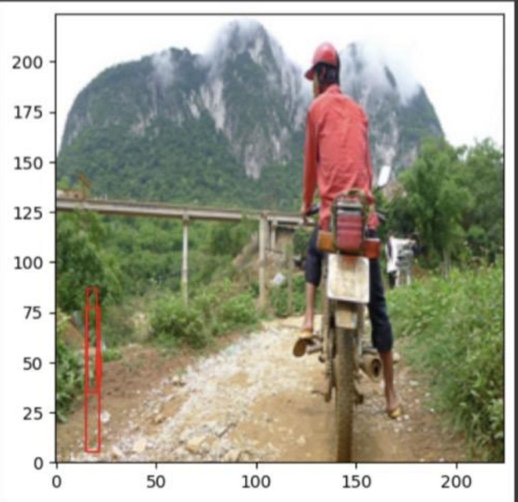


Mask R-CNN Architecture

MMDetection

```python
import torch.nn as nn
from torchvision.models.detection import MaskRCNN
from torchvision.models.detection.rpn import AnchorGenerator
from torchvision.ops import MultiScaleRoIAlign


backbone = ResNet(Bottleneck, [3, 4, 6, 3], resolution=(224, 224), heads=4)

# you are effectively informing the rest of the MaskRCNN model about the shape
# of the tensors it will receive from the backbone. This ensures that subsequent
# layers can be correctly configured to work with these tensors.
backbone.out_channels = 2048

anchor_generator = AnchorGenerator(sizes=((32, 64, 128, 256, 512),), aspect_ratios=((0.5, 1.0, 2.0),))

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

mask_roi_pooler = MultiScaleRoIAlign(featmap_names=['0'], output_size=14, sampling_ratio=2)

# Define the model
model = MaskRCNN(backbone, num_classes=91,  # COCO has 80 classes + background
                 rpn_anchor_generator=anchor_generator,
                 box_roi_pool=roi_pooler,
                 mask_roi_pool=mask_roi_pooler)
```

(2, 224, 224, 3)



Our Code

- Image Classification on CIFAR
- Does slightly better than CNN and uses fewer parameters
- Exploration of the architecture
- Here

Thank you!