Table of Contents

MODELS AND PRE-TRAINED WEIGHTS

The `torchvision.models` subpackage contains definitions of models for addressing different tasks, including: image classification, pixelwise semantic segmentation, object detection, instance segmentation, person keypoint detection, video classification, and optical flow.

General information on pre-trained weights

TorchVision offers pre-trained weights for every provided architecture, using the PyTorch `torch.hub` . Instancing a pre-trained model will download its weights to a cache directory. This directory can be set using the *TORCH_HOME* environment variable. See `torch.hub.load_state_dict_from_url()` for details.

• NOTE

The pre-trained models provided in this library may have their own licenses or terms and conditions derived from the dataset used for training. It is your responsibility to determine whether you have permission to use the models for your use case.

• NOTE

Backward compatibility is guaranteed for loading a serialized `state_dict` to the model created using old PyTorch version. On the contrary, loading entire saved models or serialized `ScriptModules` (serialized using older versions of PyTorch) may not preserve the historic behaviour. Refer to the following documentation

Initializing pre-trained models

As of v0.13, TorchVision offers a new Multi-weight support API for loading different weights to the existing model builder methods:

```
from torchvision.models import resnet50, ResNet50_Weights

# Old weights with accuracy 76.130%
resnet50(weights=ResNet50_Weights.IMAGENET1K_V1)

# New weights with accuracy 80.858%
resnet50(weights=ResNet50_Weights.IMAGENET1K_V2)

# Best available weights (currently alias for IMAGENET1K_V2)
# Note that these weights may change across versions
resnet50(weights=ResNet50_Weights.DEFAULT)

# Strings are also supported
resnet50(weights="IMAGENET1K_V2")

# No weights - random initialization
resnet50(weights=None)
```

Migrating to the new API is very straightforward. The following method calls between the 2 APIs are all equivalent:

```
from torchvision.models import resnet50, ResNet50_Weights

# Using pretrained weights:
resnet50(weights=ResNet50_Weights.IMAGENET1K_V1)
resnet50(weights="IMAGENET1K_V1")
resnet50(pretrained=True)  # deprecated
resnet50(True)  # deprecated

# Using no weights:
resnet50(weights=None)
resnet50()
resnet50(pretrained=False)  # deprecated
resnet50(False)  # deprecated
```

Note that the `pretrained` parameter is now deprecated, using it will emit warnings and will be removed on v0.15.

Using the pre-trained models

Before using the pre-trained models, one must preprocess the image (resize with right resolution/interpolation, apply inference transforms, rescale the values etc). There is no standard way to do this as it depends on how a given model was trained. It can vary across model families, variants or even weight versions. Using the correct preprocessing method is critical and failing to do so may lead to decreased accuracy or incorrect outputs.

All the necessary information for the inference transforms of each pre-trained model is provided on its weights documentation. To simplify inference, TorchVision bundles the necessary preprocessing transforms into each model weight. These are accessible via the `weight.transforms` attribute:

```
# Initialize the Weight Transforms
weights = ResNet50_Weights.DEFAULT
preprocess = weights.transforms()

# Apply it to the input image
img_transformed = preprocess(img)
```

Some models use modules which have different training and evaluation behavior, such as batch normalization. To switch between these modes, use `model.train()` or `model.eval()` as appropriate. See `train()` or `eval()` for details.

```
# Initialize model
weights = ResNet50_Weights.DEFAULT
model = resnet50(weights=weights)

# Set model to eval mode
model.eval()
```

Using models from Hub

Most pre-trained models can be accessed directly via PyTorch Hub without having TorchVision installed:

```
import torch

# Option 1: passing weights param as string
model = torch.hub.load("pytorch/vision", "resnet50", weights="IMAGENET1K_V2")

# Option 2: passing weights param as enum
weights = torch.hub.load("pytorch/vision", "get_weight", weights="ResNet50_Weights.IMAGENET1K_V2")
model = torch.hub.load("pytorch/vision", "resnet50", weights=weights)
```

The only exception to the above are the detection models included on `torchvision.models.detection`. These models require TorchVision to be installed because they depend on custom C++ operators.

Classification

The following classification models are available, with or without pre-trained weights:

- AlexNet
- ConvNeXt
- DenseNet
- EfficientNet
- EfficientNetV2
- GoogLeNet
- Inception V3
- MNASNet
- MobileNet V2
- MobileNet V3
- RegNet
- ResNet
- ResNeXt
- ShuffleNet V2
- SqueezeNet
- SwinTransformer
- VGG
- VisionTransformer
- Wide ResNet

Here is an example of how to use the pre-trained image classification models:

```
from torchvision.io import read_image
from torchvision.models import resnet50, ResNet50_Weights

img = read_image("test/assets/encode_jpeg/grace_hopper_517x606.jpg")

# Step 1: Initialize model with the best available weights
weights = ResNet50_Weights.DEFAULT
model = resnet50(weights=weights)
model.eval()

# Step 2: Initialize the inference transforms
preprocess = weights.transforms()

# Step 3: Apply inference preprocessing transforms
batch = preprocess(img).unsqueeze(0)

# Step 4: Use the model and print the predicted category
prediction = model(batch).squeeze(0).softmax(0)
class_id = prediction.argmax().item()
score = prediction[class_id].item()
category_name = weights.meta["categories"][class_id]
print(f"{category_name}: {100 * score:.1f}%")
```

The classes of the pre-trained model outputs can be found at `weights.meta["categories"]`.

Table of all available classification weights

Accuracies are reported on ImageNet-1K using single crops:

Weight	Acc@1	Acc@5	Params	Recipe
AlexNet_Weights.IMAGENET1K_V1	56.522	79.066	61.1M	link
ConvNeXt_Base_Weights.IMAGENET1K_V1	84.062	96.87	88.6M	link
ConvNeXt_Large_Weights.IMAGENET1K_V1	84.414	96.976	197.8M	link
ConvNeXt_Small_Weights.IMAGENET1K_V1	83.616	96.65	50.2M	link
ConvNeXt_Tiny_Weights.IMAGENET1K_V1	82.52	96.146	28.6M	link
DenseNet121_Weights.IMAGENET1K_V1	74.434	91.972	8.0M	link
DenseNet161_Weights.IMAGENET1K_V1	77.138	93.56	28.7M	link
DenseNet169_Weights.IMAGENET1K_V1	75.6	92.806	14.1M	link
DenseNet201_Weights.IMAGENET1K_V1	76.896	93.37	20.0M	link
EfficientNet_B0_Weights.IMAGENET1K_V1	77.692	93.532	5.3M	link
EfficientNet_B1_Weights.IMAGENET1K_V1	78.642	94.186	7.8M	link
EfficientNet_B1_Weights.IMAGENET1K_V2	79.838	94.934	7.8M	link
EfficientNet_B2_Weights.IMAGENET1K_V1	80.608	95.31	9.1M	link
EfficientNet_B3_Weights.IMAGENET1K_V1	82.008	96.054	12.2M	link
EfficientNet_B4_Weights.IMAGENET1K_V1	83.384	96.594	19.3M	link
EfficientNet_B5_Weights.IMAGENET1K_V1	83.444	96.628	30.4M	link
EfficientNet_B6_Weights.IMAGENET1K_V1	84.008	96.916	43.0M	link
EfficientNet_B7_Weights.IMAGENET1K_V1	84.122	96.908	66.3M	link
EfficientNet_V2_L_Weights.IMAGENET1K_V1	85.808	97.788	118.5M	link
EfficientNet_V2_M_Weights.IMAGENET1K_V1	85.112	97.156	54.1M	link
EfficientNet_V2_S_Weights.IMAGENET1K_V1	84.228	96.878	21.5M	link
GoogLeNet_Weights.IMAGENET1K_V1	69.778	89.53	6.6M	link
Inception_V3_Weights.IMAGENET1K_V1	77.294	93.45	27.2M	link
MNASNet0_5_Weights.IMAGENET1K_V1	67.734	87.49	2.2M	link
MNASNet0_75_Weights.IMAGENET1K_V1	71.18	90.496	3.2M	link
MNASNet1_0_Weights.IMAGENET1K_V1	73.456	91.51	4.4M	link
MNASNet1_3_Weights.IMAGENET1K_V1	76.506	93.522	6.3M	link
MobileNet_V2_Weights.IMAGENET1K_V1	71.878	90.286	3.5M	link
MobileNet_V2_Weights.IMAGENET1K_V2	72.154	90.822	3.5M	link
MobileNet_V3_Large_Weights.IMAGENET1K_V1	74.042	91.34	5.5M	link
MobileNet_V3_Large_Weights.IMAGENET1K_V2	75.274	92.566	5.5M	link
MobileNet_V3_Small_Weights.IMAGENET1K_V1	67.668	87.402	2.5M	link
RegNet_X_16GF_Weights.IMAGENET1K_V1	80.058	94.944	54.3M	link
RegNet_X_16GF_Weights.IMAGENET1K_V2	82.716	96.196	54.3M	link
RegNet_X_1_6GF_Weights.IMAGENET1K_V1	77.04	93.44	9.2M	link
RegNet_X_1_6GF_Weights.IMAGENET1K_V2	79.668	94.922	9.2M	link
RegNet_X_32GF_Weights.IMAGENET1K_V1	80.622	95.248	107.8M	link
RegNet_X_32GF_Weights.IMAGENET1K_V2	83.014	96.288	107.8M	link
RegNet_X_3_2GF_Weights.IMAGENET1K_V1	78.364	93.992	15.3M	link
RegNet_X_3_2GF_Weights.IMAGENET1K_V2	81.196	95.43	15.3M	link
RegNet_X_400MF_Weights.IMAGENET1K_V1	72.834	90.95	5.5M	link
RegNet_X_400MF_Weights.IMAGENET1K_V2	74.864	92.322	5.5M	link
RegNet_X_800MF_Weights.IMAGENET1K_V1	75.212	92.348	7.3M	link
RegNet_X_800MF_Weights.IMAGENET1K_V2	77.522	93.826	7.3M	link
RegNet_X_8GF_Weights.IMAGENET1K_V1	79.344	94.686	39.6M	link

RegNet_X_8GF_Weights.IMAGENET1K_V2	81.682	95.628	39.6M	link
RegNet_Y_128GF_Weights.IMAGENET1K_SWAG_E2E_V1	88.228	98.682	644.8M	link
RegNet_Y_128GF_Weights.IMAGENET1K_SWAG_LINEAR_V1	86.068	97.844	644.8M	link
RegNet_Y_16GF_Weights.IMAGENET1K_V1	80.424	95.24	83.6M	link
RegNet_Y_16GF_Weights.IMAGENET1K_V2	82.886	96.328	83.6M	link
RegNet_Y_16GF_Weights.IMAGENET1K_SWAG_E2E_V1	86.012	98.054	83.6M	link
RegNet_Y_16GF_Weights.IMAGENET1K_SWAG_LINEAR_V1	83.976	97.244	83.6M	link
RegNet_Y_1_6GF_Weights.IMAGENET1K_V1	77.95	93.966	11.2M	link
RegNet_Y_1_6GF_Weights.IMAGENET1K_V2	80.876	95.444	11.2M	link
RegNet_Y_32GF_Weights.IMAGENET1K_V1	80.878	95.34	145.0M	link
RegNet_Y_32GF_Weights.IMAGENET1K_V2	83.368	96.498	145.0M	link
RegNet_Y_32GF_Weights.IMAGENET1K_SWAG_E2E_V1	86.838	98.362	145.0M	link
RegNet_Y_32GF_Weights.IMAGENET1K_SWAG_LINEAR_V1	84.622	97.48	145.0M	link
RegNet_Y_3_2GF_Weights.IMAGENET1K_V1	78.948	94.576	19.4M	link
RegNet_Y_3_2GF_Weights.IMAGENET1K_V2	81.982	95.972	19.4M	link
RegNet_Y_400MF_Weights.IMAGENET1K_V1	74.046	91.716	4.3M	link
RegNet_Y_400MF_Weights.IMAGENET1K_V2	75.804	92.742	4.3M	link
RegNet_Y_800MF_Weights.IMAGENET1K_V1	76.42	93.136	6.4M	link
RegNet_Y_800MF_Weights.IMAGENET1K_V2	78.828	94.502	6.4M	link
RegNet_Y_8GF_Weights.IMAGENET1K_V1	80.032	95.048	39.4M	link
RegNet_Y_8GF_Weights.IMAGENET1K_V2	82.828	96.33	39.4M	link
ResNeXt101_32X8D_Weights.IMAGENET1K_V1	79.312	94.526	88.8M	link
ResNeXt101_32X8D_Weights.IMAGENET1K_V2	82.834	96.228	88.8M	link
ResNeXt101_64X4D_Weights.IMAGENET1K_V1	83.246	96.454	83.5M	link
ResNeXt50_32X4D_Weights.IMAGENET1K_V1	77.618	93.698	25.0M	link
ResNeXt50_32X4D_Weights.IMAGENET1K_V2	81.198	95.34	25.0M	link
ResNet101_Weights.IMAGENET1K_V1	77.374	93.546	44.5M	link
ResNet101_Weights.IMAGENET1K_V2	81.886	95.78	44.5M	link
ResNet152_Weights.IMAGENET1K_V1	78.312	94.046	60.2M	link
ResNet152_Weights.IMAGENET1K_V2	82.284	96.002	60.2M	link
ResNet18_Weights.IMAGENET1K_V1	69.758	89.078	11.7M	link
ResNet34_Weights.IMAGENET1K_V1	73.314	91.42	21.8M	link
ResNet50_Weights.IMAGENET1K_V1	76.13	92.862	25.6M	link
ResNet50_Weights.IMAGENET1K_V2	80.858	95.434	25.6M	link
ShuffleNet_V2_X0_5_Weights.IMAGENET1K_V1	60.552	81.746	1.4M	link
ShuffleNet_V2_X1_0_Weights.IMAGENET1K_V1	69.362	88.316	2.3M	link
ShuffleNet_V2_X1_5_Weights.IMAGENET1K_V1	72.996	91.086	3.5M	link
ShuffleNet_V2_X2_0_Weights.IMAGENET1K_V1	76.23	93.006	7.4M	link
SqueezeNet1_0_Weights.IMAGENET1K_V1	58.092	80.42	1.2M	link
SqueezeNet1_1_Weights.IMAGENET1K_V1	58.178	80.624	1.2M	link
Swin_B_Weights.IMAGENET1K_V1	83.582	96.64	87.8M	link
Swin_S_Weights.IMAGENET1K_V1	83.196	96.36	49.6M	link
Swin_T_Weights.IMAGENET1K_V1	81.474	95.776	28.3M	link
VGG11_BN_Weights.IMAGENET1K_V1	70.37	89.81	132.9M	link
VGG11_Weights.IMAGENET1K_V1	69.02	88.628	132.9M	link
VGG13_BN_Weights.IMAGENET1K_V1	71.586	90.374	133.1M	link
VGG13_Weights.IMAGENET1K_V1	69.928	89.246	133.0M	link
VGG16_BN_Weights.IMAGENET1K_V1	73.36	91.516	138.4M	link
VGG16_Weights.IMAGENET1K_V1	71.592	90.382	138.4M	link
VGG16_Weights.IMAGENET1K_FEATURES	nan	nan	138.4M	link
VGG19_BN_Weights.IMAGENET1K_V1	74.218	91.842	143.7M	link
VGG19_Weights.IMAGENET1K_V1	72.376	90.876	143.7M	link
ViT_B_16_Weights.IMAGENET1K_V1	81.072	95.318	86.6M	link
ViT_B_16_Weights.IMAGENET1K_SWAG_E2E_V1	85.304	97.65	86.9M	link
ViT_B_16_Weights.IMAGENET1K_SWAG_LINEAR_V1	81.886	96.18	86.6M	link
ViT_B_32_Weights.IMAGENET1K_V1	75.912	92.466	88.2M	link
ViT_H_14_Weights.IMAGENET1K_SWAG_E2E_V1	88.552	98.694	633.5M	link
ViT_H_14_Weights.IMAGENET1K_SWAG_LINEAR_V1	85.708	97.73	632.0M	link
ViT_L_16_Weights.IMAGENET1K_V1	79.662	94.638	304.3M	link
ViT_L_16_Weights.IMAGENET1K_SWAG_E2E_V1	88.064	98.512	305.2M	link
ViT_L_16_Weights.IMAGENET1K_SWAG_LINEAR_V1	85.146	97.422	304.3M	link
ViT_L_32_Weights.IMAGENET1K_V1	76.972	93.07	306.5M	link
Wide_ResNet101_2_Weights.IMAGENET1K_V1	78.848	94.284	126.9M	link
Wide_ResNet101_2_Weights.IMAGENET1K_V2	82.51	96.02	126.9M	link
Wide_ResNet50_2_Weights.IMAGENET1K_V1	78.468	94.086	68.9M	link
Wide_ResNet50_2_Weights.IMAGENET1K_V2	81.602	95.758	68.9M	link

Quantized models

The following architectures provide support for INT8 quantized models, with or without pre-trained weights:

- Quantized GoogLeNet
- Quantized InceptionV3
- Quantized MobileNet V2
- Quantized MobileNet V3
- Quantized ResNet
- Quantized ResNeXt
- Quantized ShuffleNet V2

Here is an example of how to use the pre-trained quantized image classification models:

```
from torchvision.io import read_image
from torchvision.models.quantization import resnet50, ResNet50_QuantizedWeights

img = read_image("test/assets/encode_jpeg/grace_hopper_517x606.jpg")

# Step 1: Initialize model with the best available weights
weights = ResNet50_QuantizedWeights.DEFAULT
model = resnet50(weights=weights, quantize=True)
model.eval()

# Step 2: Initialize the inference transforms
preprocess = weights.transforms()

# Step 3: Apply inference preprocessing transforms
batch = preprocess(img).unsqueeze(0)

# Step 4: Use the model and print the predicted category
prediction = model(batch).squeeze(0).softmax(0)
class_id = prediction.argmax().item()
score = prediction[class_id].item()
category_name = weights.meta["categories"][class_id]
print(f"{category_name}: {100 * score}%")
```

The classes of the pre-trained model outputs can be found at `weights.meta["categories"]`.

Table of all available quantized classification weights

Accuracies are reported on ImageNet-1K using single crops:

Weight	Acc@1	Acc@5	Params	Recipe
GoogLeNet_QuantizedWeights.IMAGENET1K_FBGEMM_V1	69.826	89.404	6.6M	link
Inception_V3_QuantizedWeights.IMAGENET1K_FBGEMM_V1	77.176	93.354	27.2M	link
MobileNet_V2_QuantizedWeights.IMAGENET1K_QNNPACK_V1	71.658	90.15	3.5M	link
MobileNet_V3_Large_QuantizedWeights.IMAGENET1K_QNNPACK_V1	73.004	90.858	5.5M	link
ResNeXt101_32X8D_QuantizedWeights.IMAGENET1K_FBGEMM_V1	78.986	94.48	88.8M	link
ResNeXt101_32X8D_QuantizedWeights.IMAGENET1K_FBGEMM_V2	82.574	96.132	88.8M	link
ResNeXt101_64X4D_QuantizedWeights.IMAGENET1K_FBGEMM_V1	82.898	96.326	83.5M	link
ResNet18_QuantizedWeights.IMAGENET1K_FBGEMM_V1	69.494	88.882	11.7M	link
ResNet50_QuantizedWeights.IMAGENET1K_FBGEMM_V1	75.92	92.814	25.6M	link
ResNet50_QuantizedWeights.IMAGENET1K_FBGEMM_V2	80.282	94.976	25.6M	link
ShuffleNet_V2_X0_5_QuantizedWeights.IMAGENET1K_FBGEMM_V1	57.972	79.78	1.4M	link
ShuffleNet_V2_X1_0_QuantizedWeights.IMAGENET1K_FBGEMM_V1	68.36	87.582	2.3M	link
ShuffleNet_V2_X1_5_QuantizedWeights.IMAGENET1K_FBGEMM_V1	72.052	90.7	3.5M	link
ShuffleNet_V2_X2_0_QuantizedWeights.IMAGENET1K_FBGEMM_V1	75.354	92.488	7.4M	link

Semantic Segmentation

WARNING

The segmentation module is in Beta stage, and backward compatibility is not guaranteed.

The following semantic segmentation models are available, with or without pre-trained weights:

- DeepLabV3
- FCN
- LRASPP

Here is an example of how to use the pre-trained semantic segmentation models:

```python
from torchvision.io.image import read_image
from torchvision.models.segmentation import fcn_resnet50, FCN_ResNet50_Weights
from torchvision.transforms.functional import to_pil_image

img = read_image("gallery/assets/dog1.jpg")

# Step 1: Initialize model with the best available weights
weights = FCN_ResNet50_Weights.DEFAULT
model = fcn_resnet50(weights=weights)
model.eval()

# Step 2: Initialize the inference transforms
preprocess = weights.transforms()

# Step 3: Apply inference preprocessing transforms
batch = preprocess(img).unsqueeze(0)

# Step 4: Use the model and visualize the prediction
prediction = model(batch)["out"]
normalized_masks = prediction.softmax(dim=1)
class_to_idx = {cls: idx for (idx, cls) in enumerate(weights.meta["categories"])}
mask = normalized_masks[0, class_to_idx["dog"]]
to_pil_image(mask).show()
```

The classes of the pre-trained model outputs can be found at `weights.meta["categories"]`. The output format of the models is illustrated in Semantic segmentation models.

Table of all available semantic segmentation weights

All models are evaluated a subset of COCO val2017, on the 20 categories that are present in the Pascal VOC dataset:

Weight	Mean IoU	pixelwise Acc	Params	Recipe
`DeepLabV3_MobileNet_V3_Large_Weights.COCO_WITH_VOC_LABELS_V1`	60.3	91.2	11.0M	link
`DeepLabV3_ResNet101_Weights.COCO_WITH_VOC_LABELS_V1`	67.4	92.4	61.0M	link
`DeepLabV3_ResNet50_Weights.COCO_WITH_VOC_LABELS_V1`	66.4	92.4	42.0M	link
`FCN_ResNet101_Weights.COCO_WITH_VOC_LABELS_V1`	63.7	91.9	54.3M	link
`FCN_ResNet50_Weights.COCO_WITH_VOC_LABELS_V1`	60.5	91.4	35.3M	link
`LRASPP_MobileNet_V3_Large_Weights.COCO_WITH_VOC_LABELS_V1`	57.9	91.2	3.2M	link

Object Detection, Instance Segmentation and Person Keypoint Detection

The pre-trained models for detection, instance segmentation and keypoint detection are initialized with the classification models in torchvision. The models expect a list of `Tensor[C, H, W]`. Check the constructor of the models for more information.

- WARNING

  The detection module is in Beta stage, and backward compatibility is not guaranteed.

Object Detection

The following object detection models are available, with or without pre-trained weights:

- Faster R-CNN
- FCOS
- RetinaNet
- SSD
- SSDlite

Here is an example of how to use the pre-trained object detection models:

```python
from torchvision.io.image import read_image
from torchvision.models.detection import fasterrcnn_resnet50_fpn_v2, FasterRCNN_ResNet50_FPN_V2_Weights
from torchvision.utils import draw_bounding_boxes
from torchvision.transforms.functional import to_pil_image

img = read_image("test/assets/encode_jpeg/grace_hopper_517x606.jpg")

# Step 1: Initialize model with the best available weights
weights = FasterRCNN_ResNet50_FPN_V2_Weights.DEFAULT
model = fasterrcnn_resnet50_fpn_v2(weights=weights, box_score_thresh=0.9)
model.eval()

# Step 2: Initialize the inference transforms
preprocess = weights.transforms()

# Step 3: Apply inference preprocessing transforms
batch = [preprocess(img)]

# Step 4: Use the model and visualize the prediction
prediction = model(batch)[0]
labels = [weights.meta["categories"][i] for i in prediction["labels"]]
box = draw_bounding_boxes(img, boxes=prediction["boxes"],
                          labels=labels,
                          colors="red",
                          width=4, font_size=30)
im = to_pil_image(box.detach())
im.show()
```

The classes of the pre-trained model outputs can be found at `weights.meta["categories"]`. For details on how to plot the bounding boxes of the models, you may refer to Instance segmentation models.

Table of all available Object detection weights

Box MAPs are reported on COCO val2017:

Weight	Box MAP	Params	Recipe
`FCOS_ResNet50_FPN_Weights.COCO_V1`	39.2	32.3M	link
`FasterRCNN_MobileNet_V3_Large_320_FPN_Weights.COCO_V1`	22.8	19.4M	link
`FasterRCNN_MobileNet_V3_Large_FPN_Weights.COCO_V1`	32.8	19.4M	link
`FasterRCNN_ResNet50_FPN_V2_Weights.COCO_V1`	46.7	43.7M	link
`FasterRCNN_ResNet50_FPN_Weights.COCO_V1`	37	41.8M	link
`RetinaNet_ResNet50_FPN_V2_Weights.COCO_V1`	41.5	38.2M	link
`RetinaNet_ResNet50_FPN_Weights.COCO_V1`	36.4	34.0M	link
`SSD300_VGG16_Weights.COCO_V1`	25.1	35.6M	link
`SSDLite320_MobileNet_V3_Large_Weights.COCO_V1`	21.3	3.4M	link

Instance Segmentation

The following instance segmentation models are available, with or without pre-trained weights:

- Mask R-CNN

For details on how to plot the masks of the models, you may refer to Instance segmentation models.

Table of all available Instance segmentation weights

Box and Mask MAPs are reported on COCO val2017:

Weight	Box MAP	Mask MAP	Params	Recipe
`MaskRCNN_ResNet50_FPN_V2_Weights.COCO_V1`	47.4	41.8	46.4M	link
`MaskRCNN_ResNet50_FPN_Weights.COCO_V1`	37.9	34.6	44.4M	link

Keypoint Detection

The following person keypoint detection models are available, with or without pre-trained weights:

- Keypoint R-CNN

The classes of the pre-trained model outputs can be found at `weights.meta["keypoint_names"]`. For details on how to plot the bounding boxes of the models, you may refer to Visualizing keypoints.

Table of all available Keypoint detection weights

Box and Keypoint MAPs are reported on COCO val2017:

Weight	Box MAP	Keypoint MAP	Params	Recipe
`KeypointRCNN_ResNet50_FPN_Weights.COCO_LEGACY`	50.6	61.1	59.1M	link
`KeypointRCNN_ResNet50_FPN_Weights.COCO_V1`	54.6	65	59.1M	link

Video Classification

• WARNING

The video module is in Beta stage, and backward compatibility is not guaranteed.

The following video classification models are available, with or without pre-trained weights:

- Video ResNet

Here is an example of how to use the pre-trained video classification models:

```
from torchvision.io.video import read_video
from torchvision.models.video import r3d_18, R3D_18_Weights

vid, _, _ = read_video("test/assets/videos/v_SoccerJuggling_g23_c01.avi", output_format="TCHW")
vid = vid[:32]  # optionally shorten duration

# Step 1: Initialize model with the best available weights
weights = R3D_18_Weights.DEFAULT
model = r3d_18(weights=weights)
model.eval()

# Step 2: Initialize the inference transforms
preprocess = weights.transforms()

# Step 3: Apply inference preprocessing transforms
batch = preprocess(vid).unsqueeze(0)

# Step 4: Use the model and print the predicted category
prediction = model(batch).squeeze(0).softmax(0)
label = prediction.argmax().item()
score = prediction[label].item()
category_name = weights.meta["categories"][label]
print(f"{category_name}: {100 * score}%")
```

The classes of the pre-trained model outputs can be found at `weights.meta["categories"]`.

Table of all available video classification weights

Accuracies are reported on Kinetics-400 using single crops for clip length 16:

Weight	Acc@1	Acc@5	Params	Recipe
MC3_18_Weights.KINETICS400_V1	53.9	76.29	11.7M	link
R2Plus1D_18_Weights.KINETICS400_V1	57.5	78.81	31.5M	link
R3D_18_Weights.KINETICS400_V1	52.75	75.45	33.4M	link

Optical Flow

The following Optical Flow models are available, with or without pre-trained

- RAFT

‹ Previous Next ›

Docs
Access comprehensive developer documentation for PyTorch
View Docs

Tutorials
Get in-depth tutorials for beginners and advanced developers
View Tutorials

Resources
Find development resources and get your questions answered
View Resources

PyTorch
Get Started
Features
Ecosystem
Blog

Resources
Tutorials
Docs
Discuss
Github Issues

Stay Connected