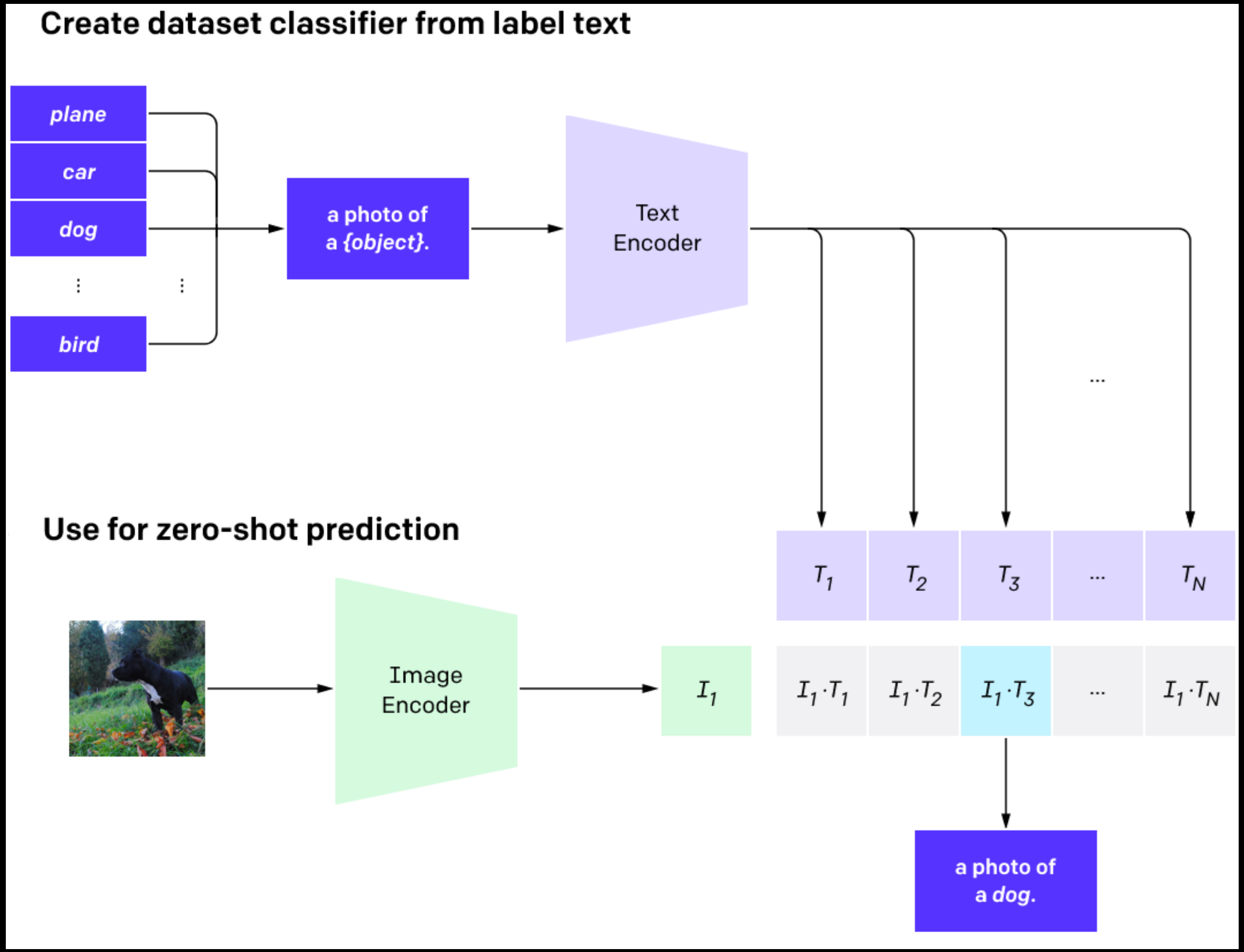
AIL 862

Lecture 19



Create dataset classifier from label text
plane
car
dog
bird
a photo of a {object}.
Text Encoder
Use for zero-shot prediction
Image Encoder
T_1
T_2
T_3
...
T_N
I_1
$I_1 \cdot T_1$
$I_1 \cdot T_2$
$I_1 \cdot T_3$
$I_1 \cdot T_N$
a photo of a dog.

CLIP

Some discussion on performance

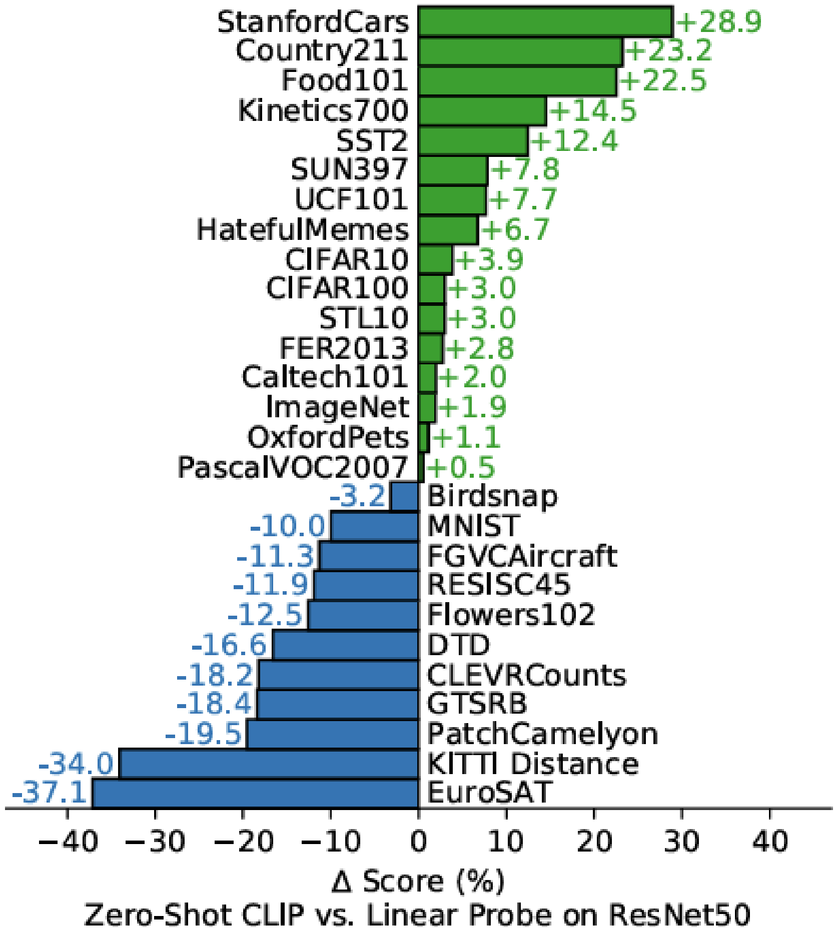


Figure 5. **Zero-shot CLIP is competitive with a fully supervised baseline.** Across a 27 dataset eval suite, a zero-shot CLIP classifier outperforms a fully supervised linear classifier fitted on ResNet-50 features on 16 datasets, including ImageNet.

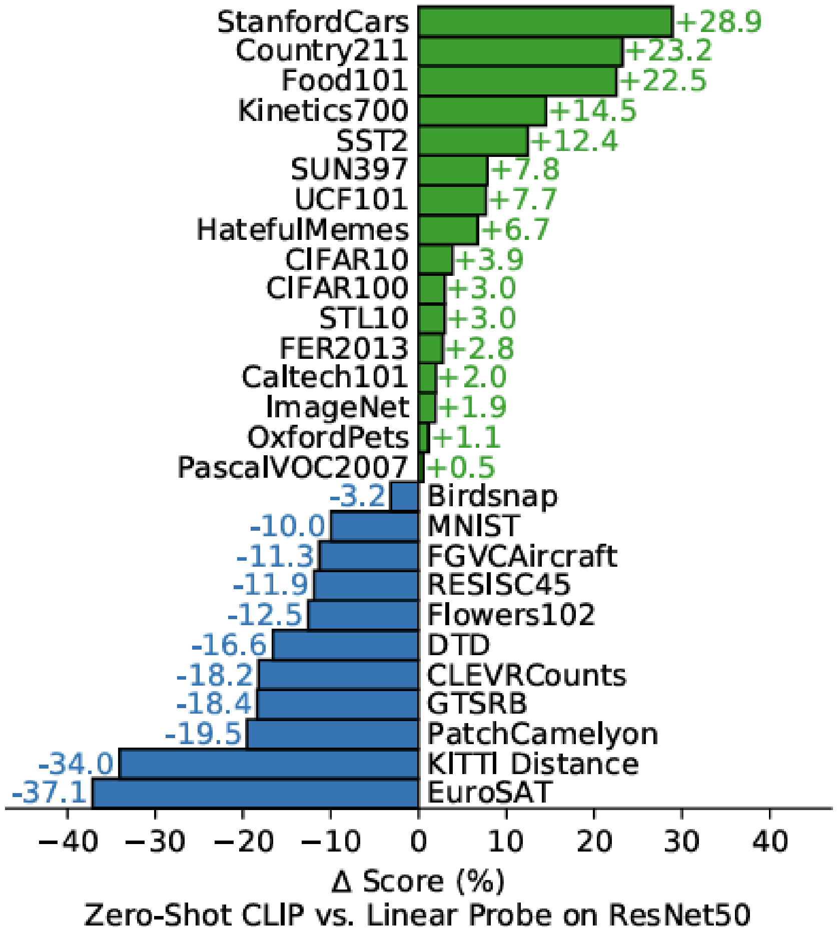
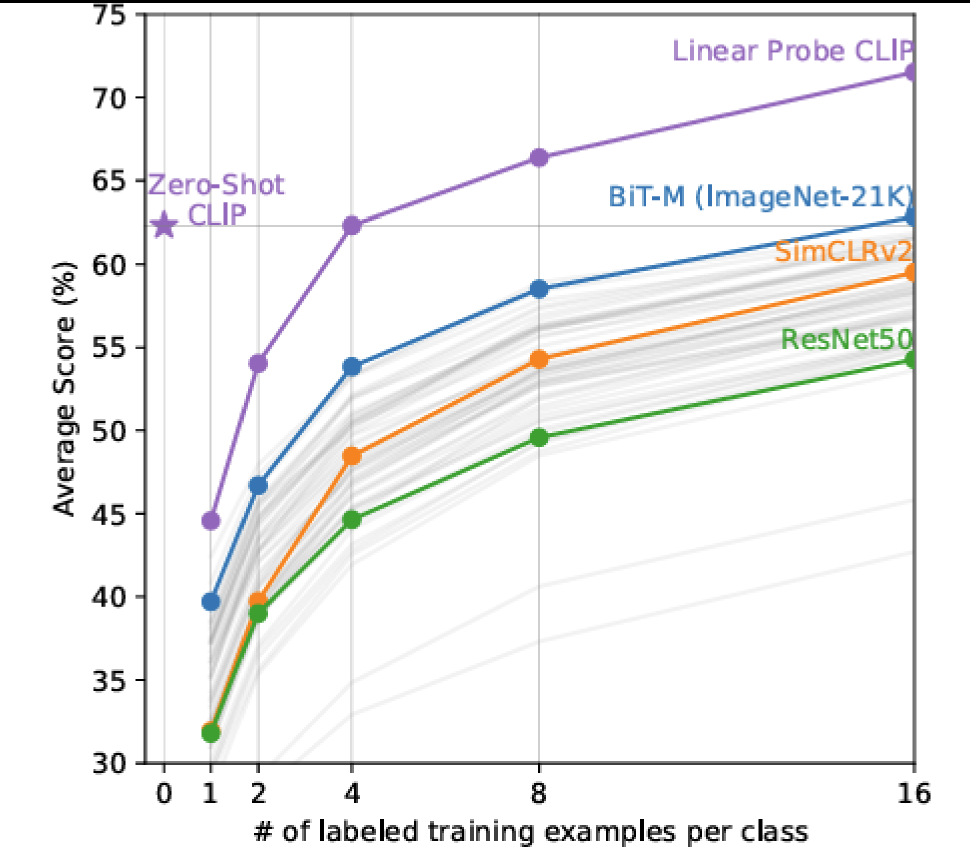


Figure 5. **Zero-shot CLIP is competitive with a fully supervised baseline.** Across a 27 dataset eval suite, a zero-shot CLIP classifier outperforms a fully supervised linear classifier fitted on ResNet-50 features on 16 datasets, including ImageNet.

Read the last name



Linear Probe CLIP
Zero-Shot
CLIP
BiT-M (ImageNet-21K)
SimCLRv2
ResNet50
Average Score (%)
# of labeled training examples per class
75
70
65
60
55
50
45
40
35
30
0
1
2
4
8
16

Use full sentence

- A photo of {label}
- A satellite photo of {label}

Ranking with CLIP

Enhancing GAN with CLIP

Partly based on - Generating images from caption and vice versa via CLIP-Guided Generative Latent Space Search

ROI proposal and CLIP cascade

ROI proposal and CLIP cascade

- ✓ ROI proposal generation from a given image
- ✓ Cropped segments are passed to CLIP
- ✓ CLIP compares these segments with text prompts describing the target

Partly motivated from - Test-Time Adaptation with SaLIP: A Cascade of SAM and CLIP for Zero-shot Medical Image Segmentation

RSClip

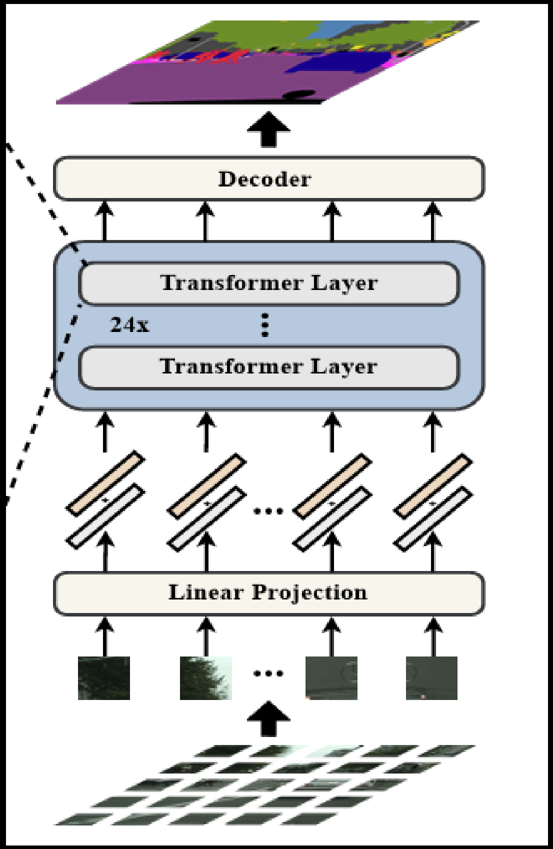
Algorithm 1 Curriculum learning for zero-shot RSSC.

```
Initialize the CLIP model using weights trained on WIT for the WebImageText dataset.
for iteration r={1,2,...R} do
    (1) Predict classification probabilities for all unlabeled samples using Eq. (1) and Eq. (2).
    (2) Select top-K_r samples with the highest probabilities for each class as pseudo labels according to Eq. (3) and Eq. (4).
    (3) Retrain the CLIP model using pseudo labels.
    (4) Update the CLIP model.
end for
```

RS-CLIP: Zero shot remote sensing scene classification via contrastive vision-language supervision

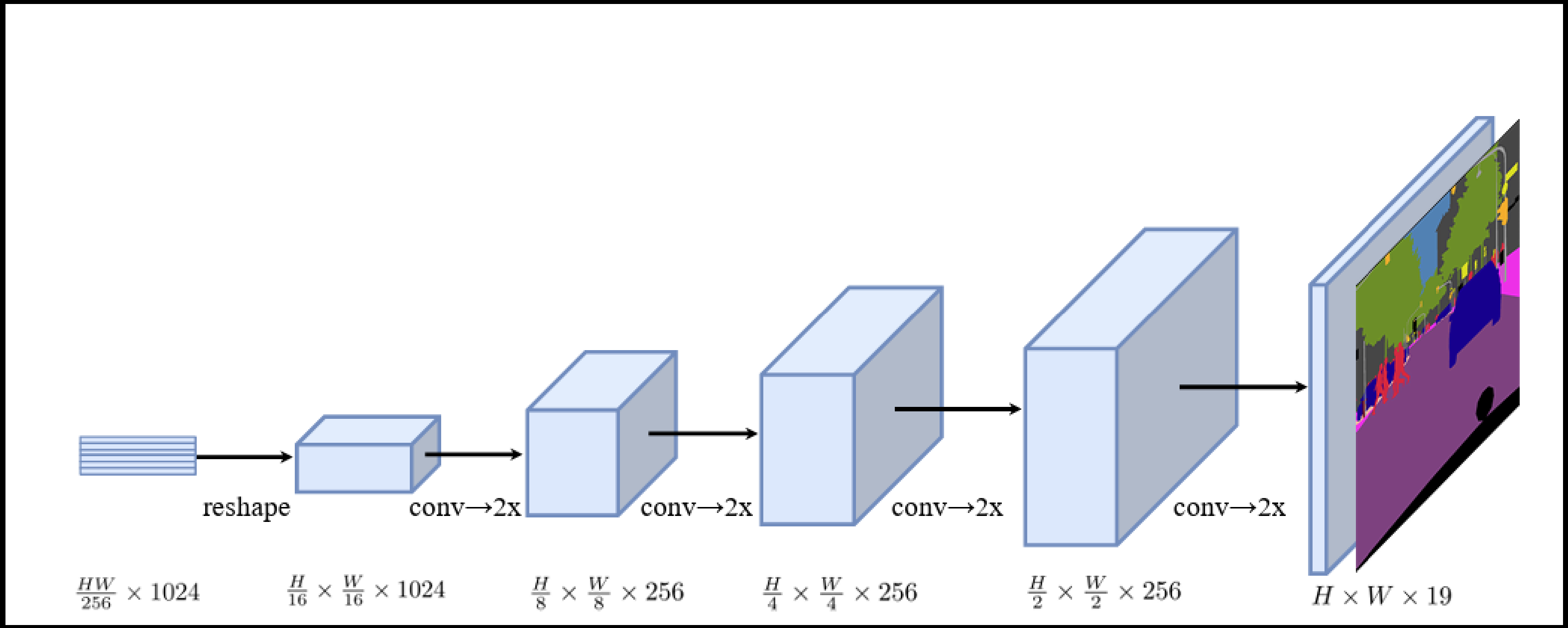
Global context

ViT for semantic segmentation



Rethinking Semantic Segmentation from a Sequence-to-Sequence Perspective with Transformers, 2021

Decoder from previous slide



Rethinking Semantic Segmentation from a Sequence-to-Sequence Perspective with Transformers, 2021

U-Net recap

Importance of context at multiple scales

Image sequentialization

Patch-embedding

CNN-transformer hybrid encoder

- CNN is first used as a feature extractor to generate a feature map for the input. Patch embedding is applied to patches extracted from the CNN feature map instead of from raw images.

- It allows us to leverage the intermediate high resolution CNN feature maps in the decoding path

TransUNet: Transformers Make Strong Encoders for Medical Image Segmentation, 2021

Cascaded upsampler

Look at code

```python
class TransUNet(nn.Module):
    def __init__(self, img_dim, in_channels, out_channels, head_num, mlp_dim, block_num, patch_dim, class_num):
        super().__init__()

        self.encoder = Encoder(img_dim, in_channels, out_channels,
                               head_num, mlp_dim, block_num, patch_dim)

        self.decoder = Decoder(out_channels, class_num)

    def forward(self, x):
        x, x1, x2, x3 = self.encoder(x)
        x = self.decoder(x, x1, x2, x3)

        return x
```

https://github.com/mkara44/transunet_pytorch/blob/main/utils/transunet.py

Look at code

```python
class TransUNet(nn.Module):
    def __init__(self, img_dim, in_channels, out_channels, head_num, mlp_dim, block_num, patch_dim, class_num):
        super().__init__()

        self.encoder = Encoder(img_dim, in_channels, out_channels,
                               head_num, mlp_dim, block_num, patch_dim)

        self.decoder = Decoder(out_channels, class_num)

    def forward(self, x):
        x, x1, x2, x3 = self.encoder(x)
        x = self.decoder(x, x1, x2, x3)

        return x
```

https://github.com/mkara44/transunet_pytorch/blob/main/utils/transunet.py

```python
class Encoder(nn.Module):
    def __init__(self, img_dim, in_channels, out_channels, head_num, mlp_dim, block_num, patch_dim):
        super().__init__()

        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=7, stride=2, padding=3, bias=False)
        self.norm1 = nn.BatchNorm2d(out_channels)
        self.relu = nn.ReLU(inplace=True)

        self.encoder1 = EncoderBottleneck(out_channels, out_channels * 2, stride=2)
        self.encoder2 = EncoderBottleneck(out_channels * 2, out_channels * 4, stride=2)
        self.encoder3 = EncoderBottleneck(out_channels * 4, out_channels * 8, stride=2)

        self.vit_img_dim = img_dim // patch_dim
        self.vit = ViT(self.vit_img_dim, out_channels * 8, out_channels * 8,
                       head_num, mlp_dim, block_num, patch_dim=1, classification=False)

        self.conv2 = nn.Conv2d(out_channels * 8, 512, kernel_size=3, stride=1, padding=1)
        self.norm2 = nn.BatchNorm2d(512)

    def forward(self, x):
        x = self.conv1(x)
        x = self.norm1(x)
        x1 = self.relu(x)

        x2 = self.encoder1(x1)
        x3 = self.encoder2(x2)
        x = self.encoder3(x3)

        x = self.vit(x)
        x = rearrange(x, "b (x y) c -> b c x y", x=self.vit_img_dim, y=self.vit_img_dim)

        x = self.conv2(x)
        x = self.norm2(x)
        x = self.relu(x)

        return x, x1, x2, x3
```

```python
class EncoderBottleneck(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1, base_width=64):
        super().__init__()

        self.downsample = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=stride, bias=False),
            nn.BatchNorm2d(out_channels)
        )

        width = int(out_channels * (base_width / 64))

        self.conv1 = nn.Conv2d(in_channels, width, kernel_size=1, stride=1, bias=False)
        self.norm1 = nn.BatchNorm2d(width)

        self.conv2 = nn.Conv2d(width, width, kernel_size=3, stride=2, groups=1, padding=1, dilation=1, bias=False)
        self.norm2 = nn.BatchNorm2d(width)

        self.conv3 = nn.Conv2d(width, out_channels, kernel_size=1, stride=1, bias=False)
        self.norm3 = nn.BatchNorm2d(out_channels)

        self.relu = nn.ReLU(inplace=True)

    def forward(self, x):
        x_down = self.downsample(x)

        x = self.conv1(x)
        x = self.norm1(x)
        x = self.relu(x)

        x = self.conv2(x)
        x = self.norm2(x)
        x = self.relu(x)

        x = self.conv3(x)
        x = self.norm3(x)
        x = x + x_down
        x = self.relu(x)

        return x
```

```
class Decoder(nn.Module):
    def __init__(self, out_channels, class_num):
        super().__init__()

        self.decoder1 = DecoderBottleneck(out_channels * 8, out_channels * 2)
        self.decoder2 = DecoderBottleneck(out_channels * 4, out_channels)
        self.decoder3 = DecoderBottleneck(out_channels * 2, int(out_channels * 1 / 2))
        self.decoder4 = DecoderBottleneck(int(out_channels * 1 / 2), int(out_channels * 1 / 8))

        self.conv1 = nn.Conv2d(int(out_channels * 1 / 8), class_num, kernel_size=1)

    def forward(self, x, x1, x2, x3):
        x = self.decoder1(x, x3)
        x = self.decoder2(x, x2)
        x = self.decoder3(x, x1)
        x = self.decoder4(x)
        x = self.conv1(x)

        return x
```

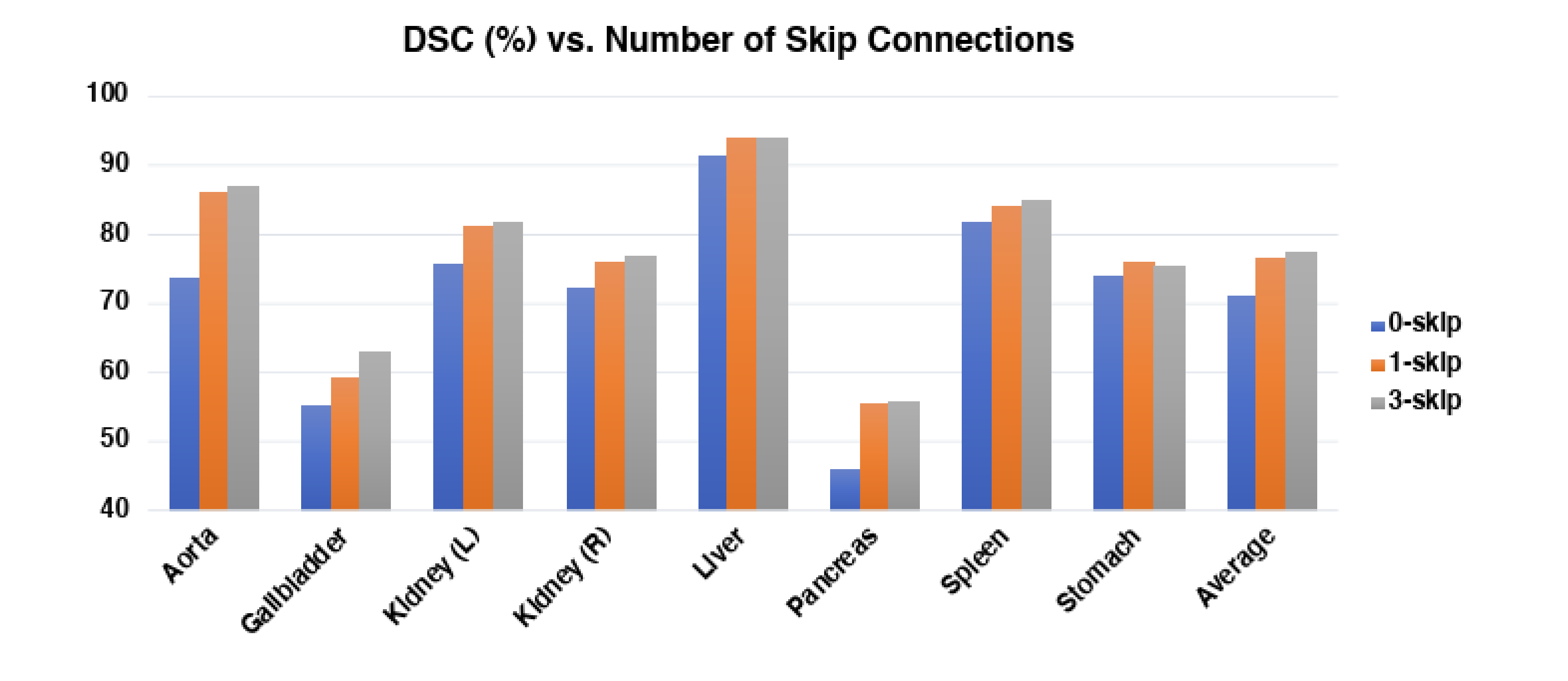
```python
class DecoderBottleneck(nn.Module):
    def __init__(self, in_channels, out_channels, scale_factor=2):
        super().__init__()

        self.upsample = nn.Upsample(scale_factor=scale_factor, mode='bilinear', align_corners=True)
        self.layer = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True),
            nn.Conv2d(out_channels, out_channels, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True)
        )

    def forward(self, x, x_concat=None):
        x = self.upsample(x)

        if x_concat is not None:
            x = torch.cat([x_concat, x], dim=1)

        x = self.layer(x)
        return x
```

Performance

Framework		Average	
Encoder	Decoder	DSC ↑	HD ↓
V-Net [9]		68.81	-
DARR [5]		69.77	-
R50	U-Net [12]	74.68	36.87
R50	AttnUNet [13]	75.57	36.97
ViT [4]	None	61.50	39.61
ViT [4]	CUP	67.86	36.11
R50-ViT [4]	CUP	71.29	32.87
TransUNet		**77.48**	**31.69**

On Synapse multi-organ dataset

Performance: role of skip connection



Patch size

Patch size	Seq_length	Average DSC
32	49	76.99
16	196	77.48
8	784	**77.83**

Performance: model scaling

Table 4: Ablation study on the model scale.

Model scale	Average DSC	Aorta	Gallbladder	Kidney (L)	Kidney (R)	Liver	Pancreas	Spleen	Stomach
Base	77.48	87.23	63.13	81.87	77.02	94.08	55.86	85.08	75.62
Large	78.52	87.42	63.92	82.17	80.19	94.47	57.64	87.42	74.90