CSC413 Programming Assignment 2: Convolutional Neural Networks

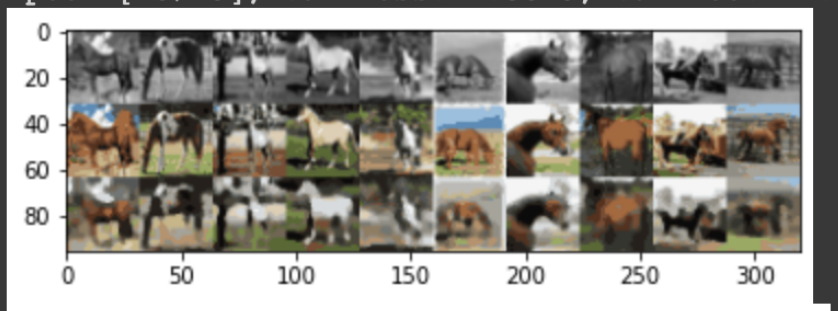
Part A: Pooling and Upsampling

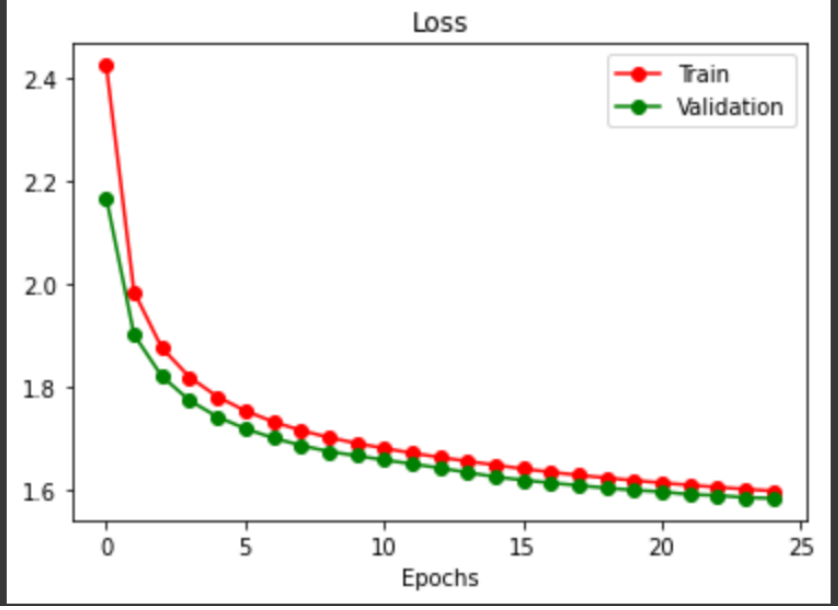
Q1. Code for model PoolUpsampleNet

```
class PoolUpsampleNet(nn.Module):
    def __init__(self, kernel, num_filters, num_colours, num_in_channels):
        super().__init__()

        # Useful parameters
        padding = kernel // 2

        self.layer1 = nn.Sequential(
            nn.Conv2d(in_channels=num_in_channels, out_channels=num_filters,
                      kernel_size=kernel, padding=padding),
            nn.MaxPool2d(kernel_size=2),
            nn.BatchNorm2d(num_features=num_filters),
            nn.ReLU(),
        )

        self.layer2 = nn.Sequential(
            nn.Conv2d(in_channels=num_filters, out_channels=num_filters * 2,
                      kernel_size=kernel, padding=padding),
            nn.MaxPool2d(kernel_size=2),
            nn.BatchNorm2d(num_features=num_filters * 2),
            nn.ReLU(),
        )

        self.layer3 = nn.Sequential(
            nn.Conv2d(in_channels=num_filters * 2, out_channels=num_filters,
                      kernel_size=kernel, padding=padding),
            nn.Upsample(scale_factor=2),
            nn.BatchNorm2d(num_features=num_filters),
            nn.ReLU(),
        )

        self.layer4 = nn.Sequential(
            nn.Conv2d(in_channels=num_filters, out_channels=num_colours,
                      kernel_size=kernel, padding=padding),
            nn.Upsample(scale_factor=2),
            nn.BatchNorm2d(num_features=num_colours),
            nn.ReLU(),
        )

        self.layer5 = nn.Conv2d(in_channels=num_colours, out_channels=num_colours,
                                kernel_size=kernel, padding=padding)

    def forward(self, x):
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)
        x = self.layer5(x)
        return x
```

Q2. Visualizations

```
Epoch [25/25], Loss: 1.5987, Time (s): 31
Epoch [25/25], Val Loss: 1.5845, Val Acc: 41.3%, Time(s): 31.75
```





Q2. Commentary

As the above output shows, the validation accuracy of this model is 41.3%, which means about 41.3% area of the images are correctly colored. This seems a good result but obviously could have further improvement.

Q3. Assume the kernel size is k with the image size of 32x32

	# of Weights	# of Outputs	# of connections
Conv1	NIC * NF * k^2 + NF	NF * 32^2	k^2 *NIC * NF * 32^2
MaxPool1	0	NF * 16^2	NF * 32^2
BatchNorm1	2NF	NF * 16^2	NF * 16^2
Conv2	2NF^2 * k^2 + 2NF	2NF * 16^2	k^2 * NF * 2NF * 16^2
MaxPool2	0	2NF * 8^2	2NF * 16^2
BatchNorm2	4NF	2NF * 8^2	2NF * 8^2
Conv3	2NF^2 * k^2 +NF	NF * 8^2	k^2 * 2NF * NF *8^2
UpSample1	0	NF * 16^2	NF * 16^2
BatchNorm3	2NF	NF * 16^2	NF * 16^2
Conv4	NF * NC * k^2 +NC	NC * 16^2	k^2 * NF * NC * 16^2
UpSample2	0	NC * 32^2	NC * 32^2
BatchNorm4	2NC	NC * 32^2	NC * 32^2
Conv5	NC^2 * k^2 + NC	NC * 32^2	k^2 * NC^2 * 32^2

Total number of weights:
NIC * NF * k^2 + NF + 2NF + 2NF^2 * k^2 + 2NF + 4NF + 2 NF^2 * k^2 + NF +2NF + NF * NC * k^2 + NC + 2NC + NC^2 * k^2 + NC
= k^2 * (NIC * NF) + 12NF + 4k^2 * NF^2 + k^2 * (NF * NC) + 4NC + k^2 * NC^2
= k^2 * (NIC * NF + 4NF^2 + NF * NC + NC^2) + 12NF + 4NC

Total number of outputs:
NF * 32^2 + NF * 16^2 + NF * 16^2 + 2NF * 16^2 + 2NF * 8^2 + 2NF * 8^2 + NF * 8^2 + NF * 16^2 + NF * 16^2 + NC * 16^2 + NC * 32^2 + NC * 32^2 + NC * 32^2
= 2880NF + 3328NC

Total number of connections:
k^2 * NIC * NF * 32^2 + NF * 32^2 + NF * 16^2 + k^2 * NF * 2NF * 16^2 + 2NF * 16^2 + 2NF * 8^2 + k^2 * 2NF * NF *8^2 + NF * 16^2 + NF * 16^2 + k^2 * NF * NC * 16^2 + NC * 32^2 + NC * 32^2 + k^2 * NC^2 * 32^2
= 1024k^2 * (NIC * NF) + 2432NF + 640k^2 * NF^2 + 256k^2 * (NF *NC) + 2048NC + 1024k^2 * NC^2
= k^2 * (1024(NIC * NF) + 640NF^2 + 256(NF *NC) + 1024NC^2) + 2432NF + 2048NC

When each input dimension (width/height) is doubled, the number of weights will not change, while the other two numbers will be multiplied by 4.

Total number of weights:
= k^2 * (NIC * NF + 4NF^2 + NF * NC + NC^2) + 12NF + 4NC

Total number of outputs:
= 4 * (2880NF + 3328NC)

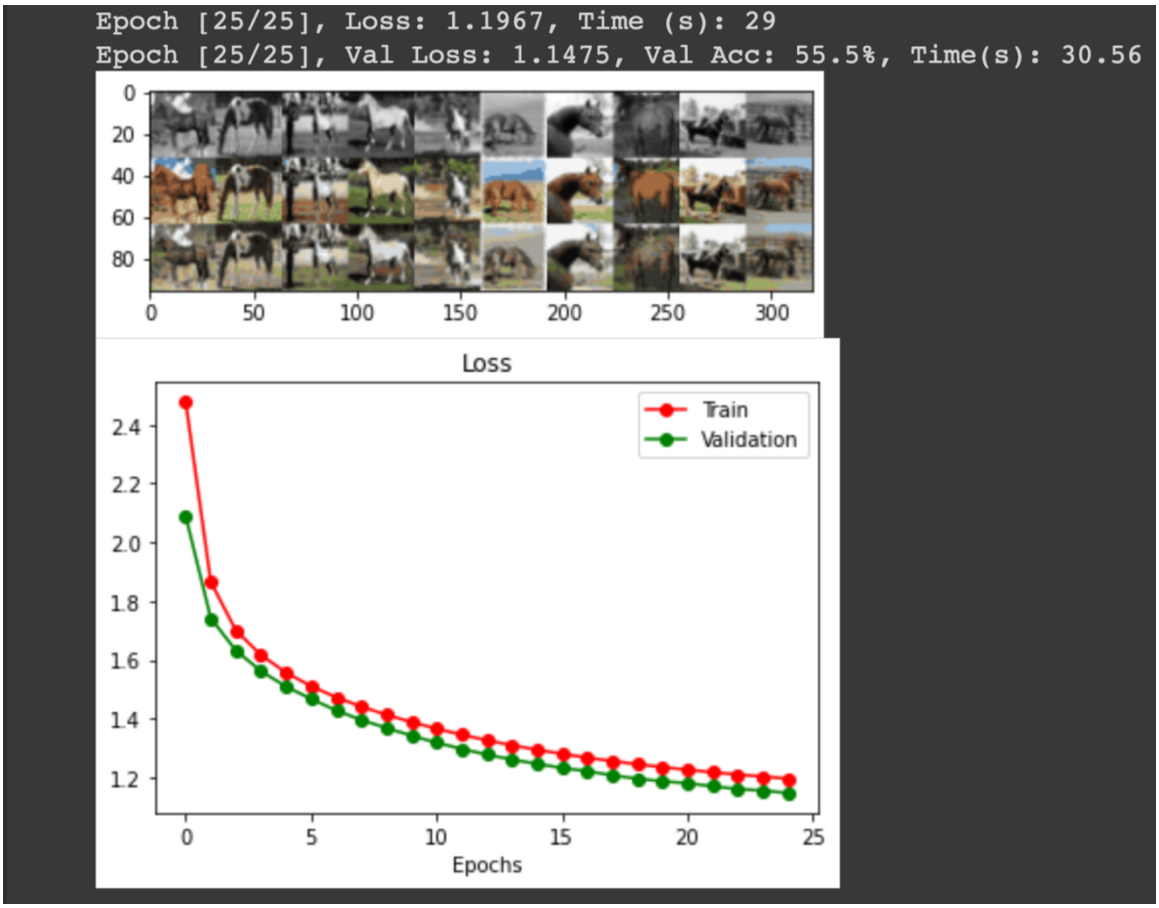
Total number of connections:
= 4 * {k^2 * (1024(NIC * NF) + 640NF^2 + 256(NF *NC) + 1024NC^2) + 2432NF + 2048NC}

Part B: Strided and Transposed Dilated Convolutions

Q1. Code for model ConvTransposeNet

```
class ConvTransposeNet(nn.Module):
    def __init__(self, kernel, num_filters, num_colours, num_in_channels):
        super().__init__()

        # Useful parameters
        stride = 2
        padding = kernel // 2
        output_padding = 1

        self.layer1 = nn.Sequential(
            nn.Conv2d(in_channels=num_in_channels, out_channels=num_filters,
                      kernel_size=kernel, stride=2, padding=1),
            nn.BatchNorm2d(num_features=num_filters),
            nn.ReLU(),
        )

        self.layer2 = nn.Sequential(
            nn.Conv2d(in_channels=num_filters, out_channels=num_filters * 2,
                      kernel_size=kernel, stride=2, padding=1),
            nn.BatchNorm2d(num_features=num_filters * 2),
            nn.ReLU(),
        )

        self.layer3 = nn.Sequential(
            nn.ConvTranspose2d(in_channels=num_filters * 2, out_channels=num_filters,
                               kernel_size=kernel, stride=2, dilation=1,
                               padding=1, output_padding=1),
            nn.BatchNorm2d(num_features=num_filters),
            nn.ReLU(),
        )

        self.layer4 = nn.Sequential(
            nn.ConvTranspose2d(in_channels=num_filters, out_channels=num_colours,
                               kernel_size=kernel, stride=2, dilation=1,
                               padding=1, output_padding=1),
            nn.BatchNorm2d(num_features=num_colours),
            nn.ReLU(),
        )

        self.layer5 = nn.Conv2d(in_channels=num_colours, out_channels=num_colours,
                                kernel_size=kernel, padding=padding)

    def forward(self, x):
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)
        x = self.layer5(x)
        return x
```

Q2. Plot figure (training/validation curves)



Q3. Compare model results

The model ConvTransposeNet performs better than the model in Part A, as the above output shows, the validation loss of 1.1475 is lower than 1.5845 from Part A and the validation accuracy of 55.5% is higher than 41.3% from Part A. Some possible reasons include:

1. This model uses two ConvTranspose2d layers instead of the nearest neighbour interpolation Upsample layers.
2. This model does not use the Maxpool layers but use a stride of 2 during convolution.

Q4

For kernel size of 4, padding = 1 and output_padding = 0.
For kernel size of 5, padding = 2 and output_padding = 1.

Q5

As the batch size increases, the validation loss increases and the final image output quality becomes worse, with a fixed number of epochs.

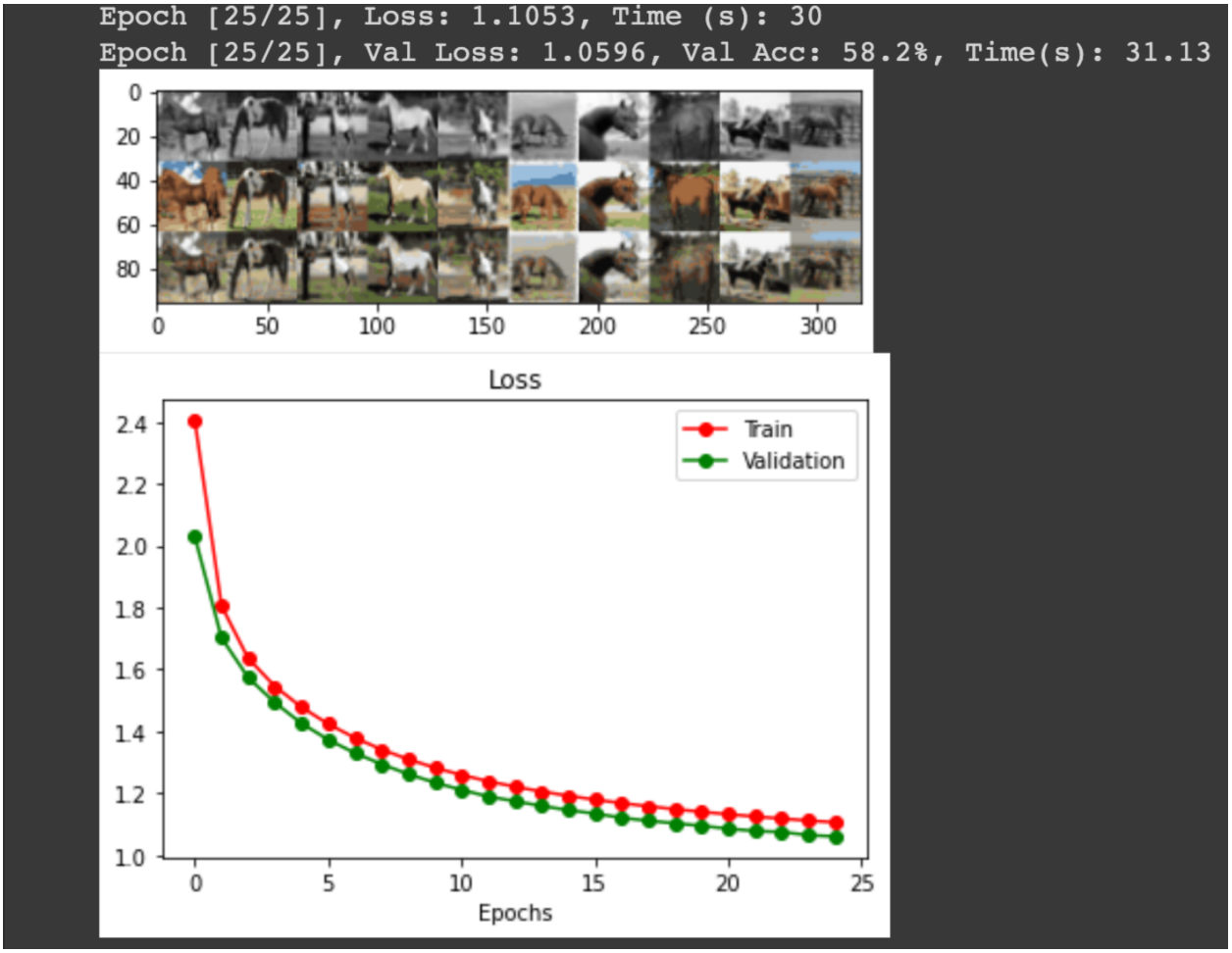
Part C: Skip Connections

Q1. Code for model UNet

```
class UNet(nn.Module):
    def __init__(self, kernel, num_filters, num_colours, num_in_channels):
        super().__init__()

        # Useful parameters
        stride = 2
        padding = kernel // 2
        output_padding = 1

        self.layer1 = nn.Sequential(
            nn.Conv2d(in_channels=num_in_channels, out_channels=num_filters,
                      kernel_size=kernel, stride=2, padding=1),
            nn.BatchNorm2d(num_features=num_filters),
            nn.ReLU(),
        )

        self.layer2 = nn.Sequential(
            nn.Conv2d(in_channels=num_filters, out_channels=num_filters * 2,
                      kernel_size=kernel, stride=2, padding=1),
            nn.BatchNorm2d(num_features=num_filters * 2),
            nn.ReLU(),
        )

        self.layer3 = nn.Sequential(
            nn.ConvTranspose2d(in_channels=num_filters * 2, out_channels=num_filters,
                               kernel_size=kernel, stride=2, dilation=1,
                               padding=1, output_padding=1),
            nn.BatchNorm2d(num_features=num_filters),
            nn.ReLU(),
        )
```

```
        self.layer4 = nn.Sequential(
            nn.ConvTranspose2d(in_channels=num_filters * 2, out_channels=num_colours,
                               kernel_size=kernel, stride=2, dilation=1,
                               padding=1, output_padding=1),
            nn.BatchNorm2d(num_features=num_colours),
            nn.ReLU(),
        )

        self.layer5 = nn.Conv2d(in_channels=num_in_channels + num_colours,
                                out_channels=num_colours,
                                kernel_size=kernel, padding=padding)

    def forward(self, x):
        x1 = self.layer1(x)
        x2 = self.layer2(x1)
        x3 = self.layer3(x2)
        x3 = torch.cat([x3, x1], dim=1)
        x4 = self.layer4(x3)
        x4 = torch.cat([x4, x], dim=1)
        x5 = self.layer5(x4)
        return x5
```

Q2. Plot figure (training/validation curves)



Q3. Compare model results

1. The validation loss decreases from 1.1475 to 1.0596 (about 7.66% decrement), and the validation accuracy increases from 55.5% to 58.2% (about 4.86% increment). Thus, this model is better than the previous one.
2. Skip connections does improve the validation loss and accuracy and the final image output quality.
3. Here are two possible reasons why skip connections might improve the performance of our CNN models:
    a. Skip connections add additional trainable parameters to the model.
    b. Without skip connections, only the most essential information is retained to the final output layer. However, after adding skip connections, the model could utilize more information, including those has been lost in the pooling or strided convolution layers.

Part D.1: Fine-tuning from pre-trained models for object detection

```
125     # Freeze
126     freeze = [f'model.{x}.' for x in range(freeze)]  # layers to freeze
127     for k, v in model.named_parameters():
128         # --- YOUR CODE GOES HERE ---
129         if any(x in k for x in freeze):
130           v.requires_grad = False
131         else:
132           v.requires_grad = True
133         # ----------------------------
```

Part D.2: Implement the classification loss

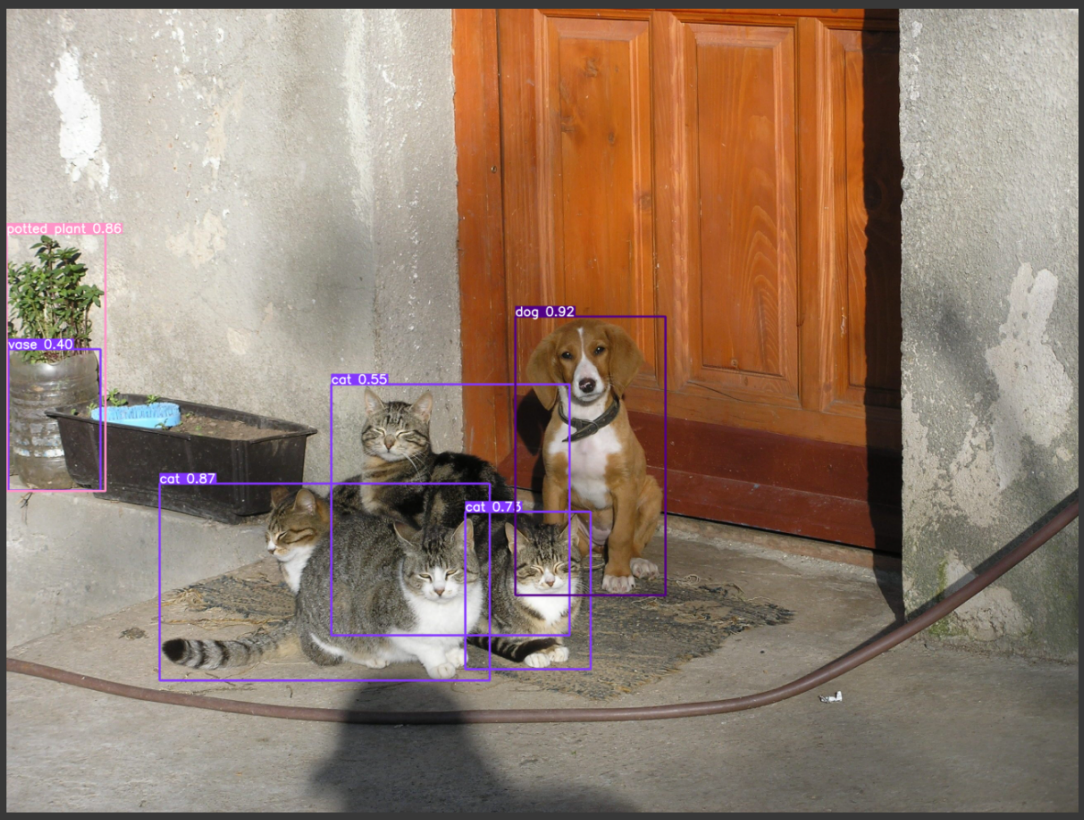
Q1. Code for classification loss in utils/loss.py

```
 95         # Define loss criteria
 96         # --- YOUR CODE GOES HERE ---
 97         BCEcls = nn.BCEWithLogitsLoss(pos_weight=torch.tensor([h['cls_pw']], device=device))
 98         # ----------------------------
 99         BCEobj = nn.BCEWithLogitsLoss(pos_weight=torch.tensor([h['obj_pw']], device=device))
100
```

```
145                 # Classification
146                 if self.nc > 1:  # cls loss (only if multiple classes)
147                     t = torch.full_like(ps[:, 5:], self.cn, device=device)  # targets
148                     t[range(n), tcls[i]] = self.cp
149                     # --- YOUR CODE GOES HERE ---
150                     lcls += self.BCEcls(ps[:, 5:], t)
151                     # ----------------------------
152
```

Q2. Visualization of predictions

```
display.Image(filename='runs/detect/exp/Cats_and_dog.jpg', width=600)
```



```
display.Image(filename='runs/detect/exp/bus.jpg', width=600)
```