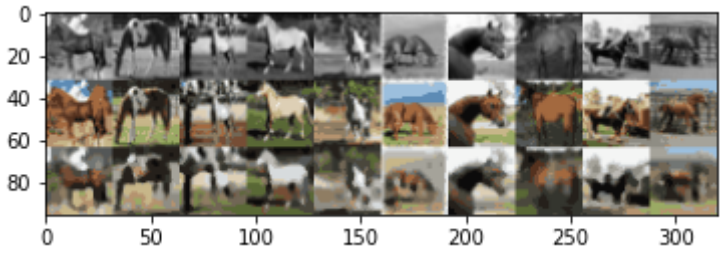
Programming Assignment 2

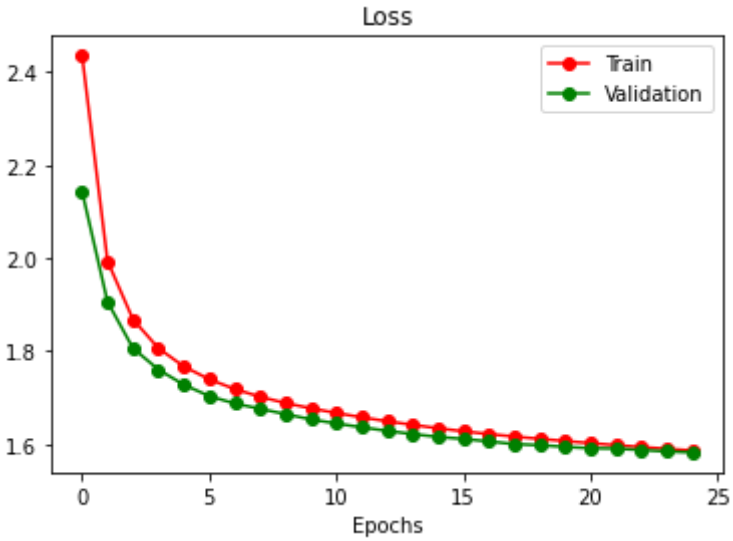
Part A: Pooling and Upsampling

A.1 Implementation of PoolUpsampleNet

```
[7]
    class PoolUpsampleNet(nn.Module):
        def __init__(self, kernel, num_filters, num_colours, num_in_channels):
            super().__init__()

            # Useful parameters
            padding = kernel // 2

            ############### YOUR CODE GOES HERE ###############
            self.firstLayer = nn.Sequential(
                nn.Conv2d(num_in_channels, num_filters, kernel_size=kernel, padding=padding),
                nn.MaxPool2d(2),
                nn.BatchNorm2d(num_filters),
                nn.ReLU()
            )

            self.secondLayer = nn.Sequential(
                nn.Conv2d(num_filters, 2*num_filters, kernel_size=kernel, padding=padding),
                nn.MaxPool2d(2),
                nn.BatchNorm2d(2*num_filters),
                nn.ReLU()
            )

            self.thirdLayer = nn.Sequential(
                nn.Conv2d(2*num_filters, num_filters, kernel_size=kernel, padding=padding),
                nn.Upsample(scale_factor=2),
                nn.BatchNorm2d(num_filters),
                nn.ReLU()
            )

            self.fourthLayer = nn.Sequential(
                nn.Conv2d(num_filters, num_colours, kernel_size=kernel, padding=padding),
                nn.Upsample(scale_factor=2),
                nn.BatchNorm2d(num_colours),
                nn.ReLU()
            )

            self.lastLayer = nn.Conv2d(num_colours, num_colours, kernel, padding=padding)
            ###################################################

        def forward(self, x):
            ############### YOUR CODE GOES HERE ###############
            first = self.firstLayer(x)
            second = self.secondLayer(first)
            third = self.thirdLayer(second)
            fourth = self.fourthLayer(third)
            return self.lastLayer(fourth)
            ###################################################
```

A.2 Training Result





The shown figure is the result obtained from the main training loop of PoolUpsampleNet. It does not look good to me as the images are quite blurry with greyscale pixels present. Also, after 25 epoch, accuracy is around 41.4%, which is quite low.

A.3

Assume the kernel size is k

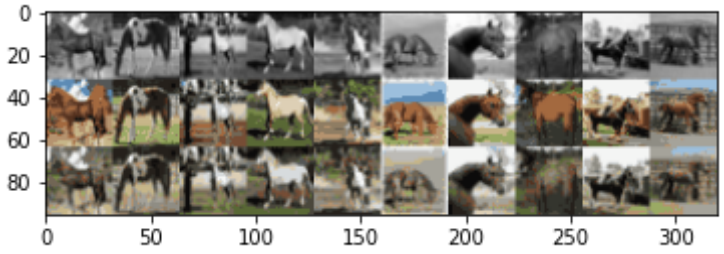
- when each input dimension (width/height) is not doubled (original input)
  - number of weights =
  - number of outputs =
  - number f connections =
- when each input dimension (width/height) is doubled
  - number of weights =
  - number of outputs =
  - number f connections =

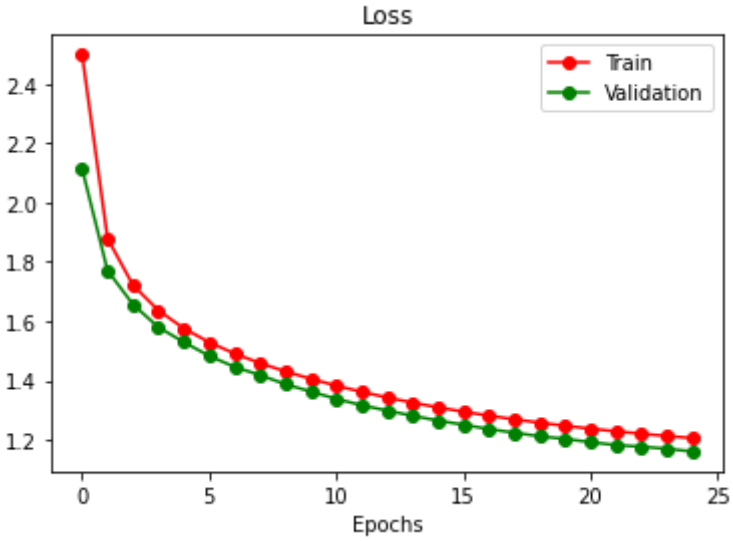
Part B: Strided and Transposed Dilated Convolutions

B.1 Implementation of ConvTransposeNet

```python
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

        ############### YOUR CODE GOES HERE ###############
        self.firstLayer = nn.Sequential(
            nn.Conv2d(num_in_channels, num_filters, kernel_size=kernel, padding=padding, stride=stride),
            nn.BatchNorm2d(num_filters),
            nn.ReLU()
        )

        self.secondLayer = nn.Sequential(
            nn.Conv2d(num_filters, 2*num_filters, kernel_size=kernel, padding=padding, stride=stride),
            nn.BatchNorm2d(2*num_filters),
            nn.ReLU()
        )

        self.thirdLayer = nn.Sequential(
            nn.ConvTranspose2d(2*num_filters, num_filters, kernel_size=kernel, padding=padding, stride=stride,
                               output_padding=output_padding),
            nn.BatchNorm2d(num_filters),
            nn.ReLU()
        )

        self.fourthLayer = nn.Sequential(
            nn.ConvTranspose2d(num_filters,num_colours,kernel_size=kernel, padding=padding, stride=stride,
                               output_padding=output_padding),
            nn.BatchNorm2d(num_colours),
            nn.ReLU()
        )

        self.lastLayer = nn.Conv2d(num_colours, num_colours, kernel_size=kernel, padding=padding)
        ####################################################

    def forward(self, x):
        ############### YOUR CODE GOES HERE ###############
        first = self.firstLayer(x)
        second = self.secondLayer(first)
        third = self.thirdLayer(second)
        fourth = self.fourthLayer(third)
        return self.lastLayer(fourth)
        ####################################################
```

B.2





B.3

The trained result seems better from what we got from Part A because 1) the image seems less blurry and 2) the validation accuracy increases from 41.4% to 54.7%.

B.4

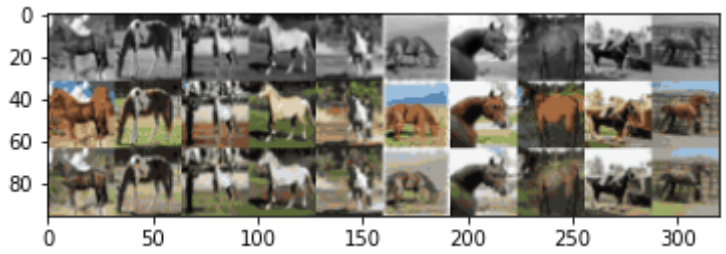
B.5

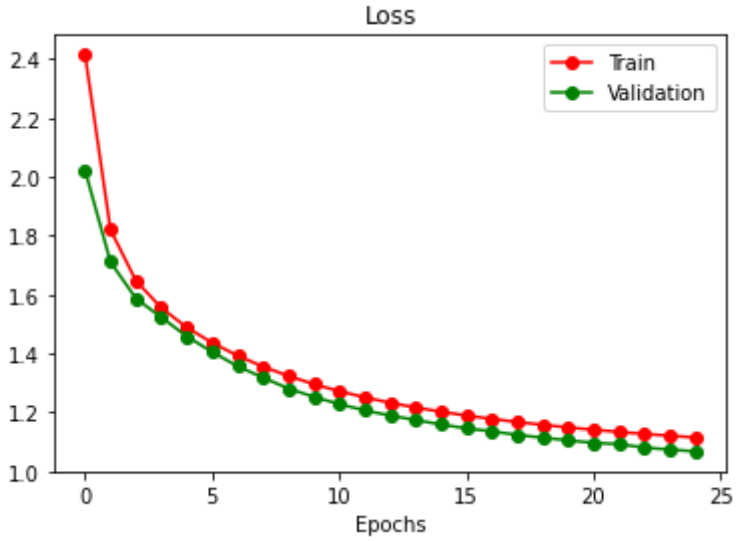
Part C: Skip Connections

C.1 Implementation of UNet

```python
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

        ############### YOUR CODE GOES HERE ###############
        self.firstLayer = nn.Sequential(
            nn.Conv2d(num_in_channels, num_filters, kernel_size=kernel, padding=padding, stride=stride),
            nn.BatchNorm2d(num_filters),
            nn.ReLU()
        )

        self.secondLayer = nn.Sequential(
            nn.Conv2d(num_filters, 2*num_filters, kernel_size=kernel, padding=padding, stride=stride),
            nn.BatchNorm2d(2*num_filters),
            nn.ReLU()
        )

        self.thirdLayer = nn.Sequential(
            nn.ConvTranspose2d(2*num_filters, num_filters, kernel_size=kernel, padding=padding, stride=stride,
                               output_padding=output_padding),
            nn.BatchNorm2d(num_filters),
            nn.ReLU()
        )

        self.fourthLayer = nn.Sequential(
            nn.ConvTranspose2d(2*num_filters,num_colours,kernel_size=kernel, padding=padding, stride=stride,
                               output_padding=output_padding),
            nn.BatchNorm2d(num_colours),
            nn.ReLU()
        )

        self.lastLayer = nn.Conv2d(num_in_channels + num_colours, num_colours, kernel_size=kernel,
                                   padding=padding)
        ###################################################

    def forward(self, x):
        ############### YOUR CODE GOES HERE ###############
        first = self.firstLayer(x)
        second = self.secondLayer(first)
        third = self.thirdLayer(second)
        fourth = self.fourthLayer(torch.cat([third, first], 1))
        return self.lastLayer(torch.cat([fourth, x], 1))
        ###################################################
```

C.2





C.3

Part D: Object Detection

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

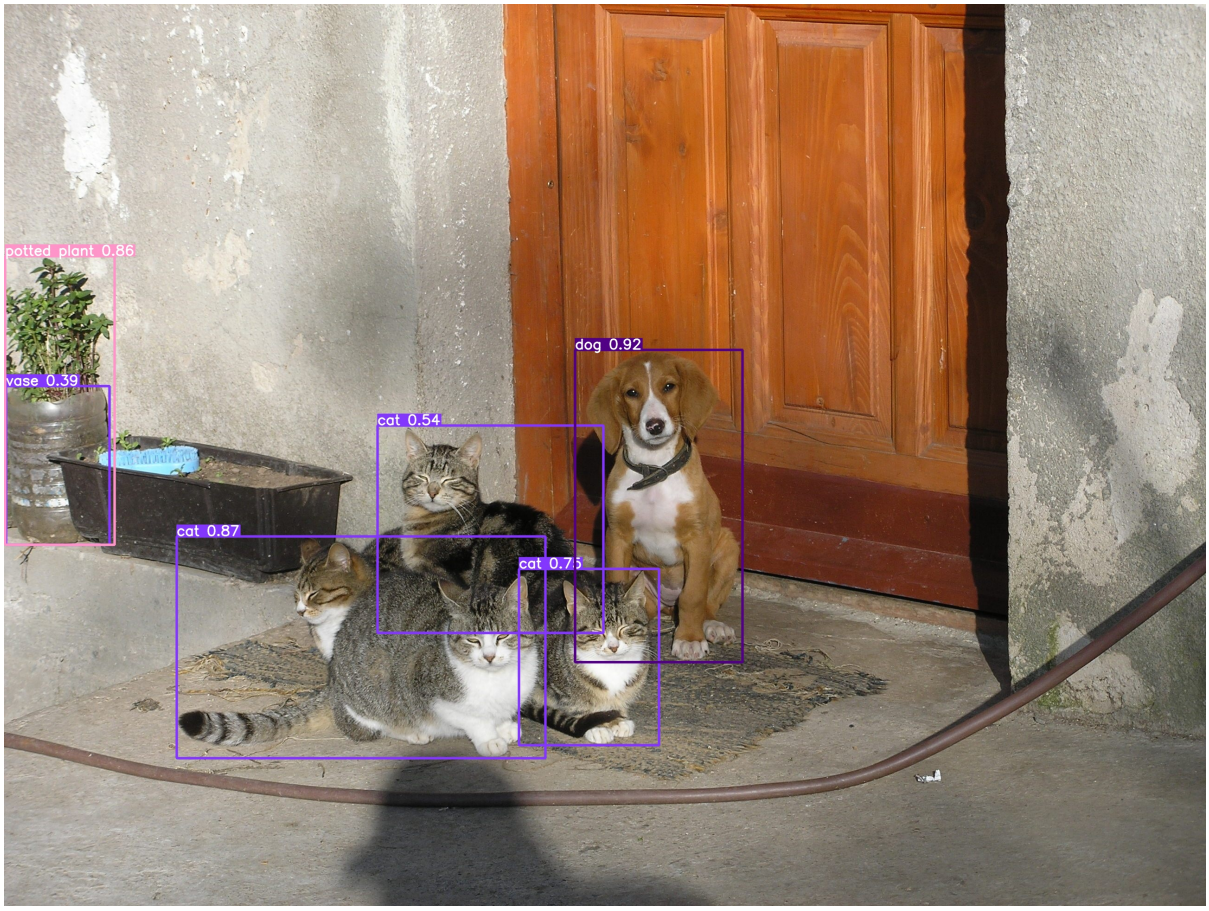
```
# Freeze
freeze = [f'model.{x}.' for x in range(freeze)]  # layers to fr
for k, v in model.named_parameters():
    # --- YOUR CODE GOES HERE ---
    if any(x in k for x in freeze):
        v.requires_grad = False
    else:
        v.requires_grad = True
    # ----------------------------
```

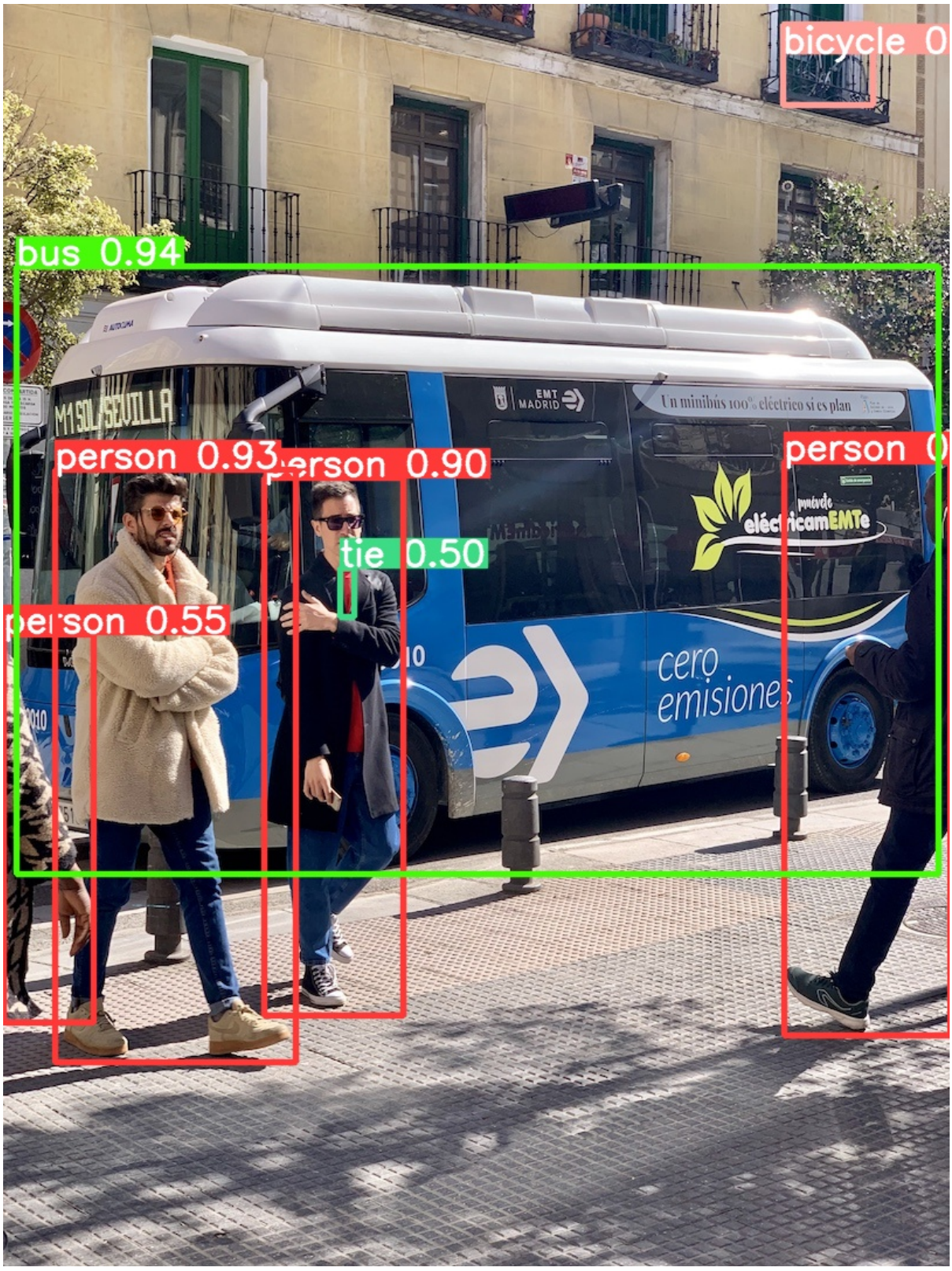
D.2 Implement the classification loss

D.2.1

```
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
101         # Class label smoothing https://arxiv.org/pdf/1902.04103.pdf eqn 3
102         self.cp, self.cn = smooth_BCE(eps=h.get('label_smoothing', 0.0))  # positive, negative BCE ta
103
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
```

D.2.2




bicycle 0
bus 0.94
M1 SOL/SEVILLA
EMT MADRID
Un minibús 100% eléctrico sí es plan
person 0.93
person 0.90
person 0
tie 0.50
person 0.55
cero emisiones