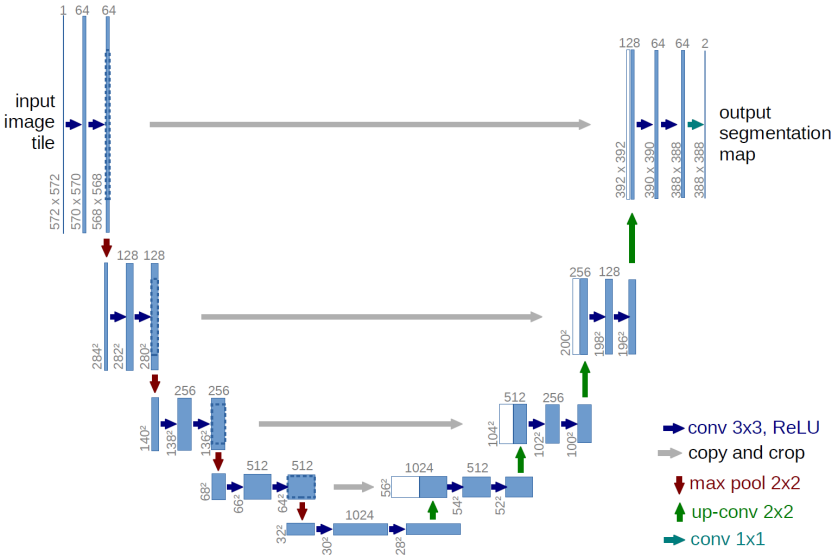
U - Net:



The name U - Net comes from its encoding- decoding architecture that looks like a U.

Intuition:

The above image shows U-Net turning a 572×572 image into a smaller 388×388 segmented map. It shrinks the image to capture features then upsamples to restore size using skip connections to keep details. The output labels each pixel as object or background.

Architecture:

1. Contracting Path (Encoder):
Uses small filters (3×3 pixels) to scan the image and find features.
Apply an activation function called ReLU to add non-linearity help the model to learn better.
Uses max pooling (2×2 filters) to shrink the image size while keeping important information. This helps the network focus on bigger features.

2. Bottleneck:
The middle of the “U” where the most compressed and abstract information is stored. It links the encoder and decoder.

3. Expansive Path (Decoder):
Uses upsampling i.e increasing image size to get back the original image size.
Combines information from the encoder using “skip connections.” These connections help the decoder get spatial details that might have been lost when shrinking the image
Uses convolution layers again to clean up and refine the output.

4. **Skip Connections for Precision**: Skip connections help preserve spatial accuracy by bringing forward detailed features from earlier layers. These are especially useful when the model needs to distinguish boundaries in segmentation tasks.

Implementation:

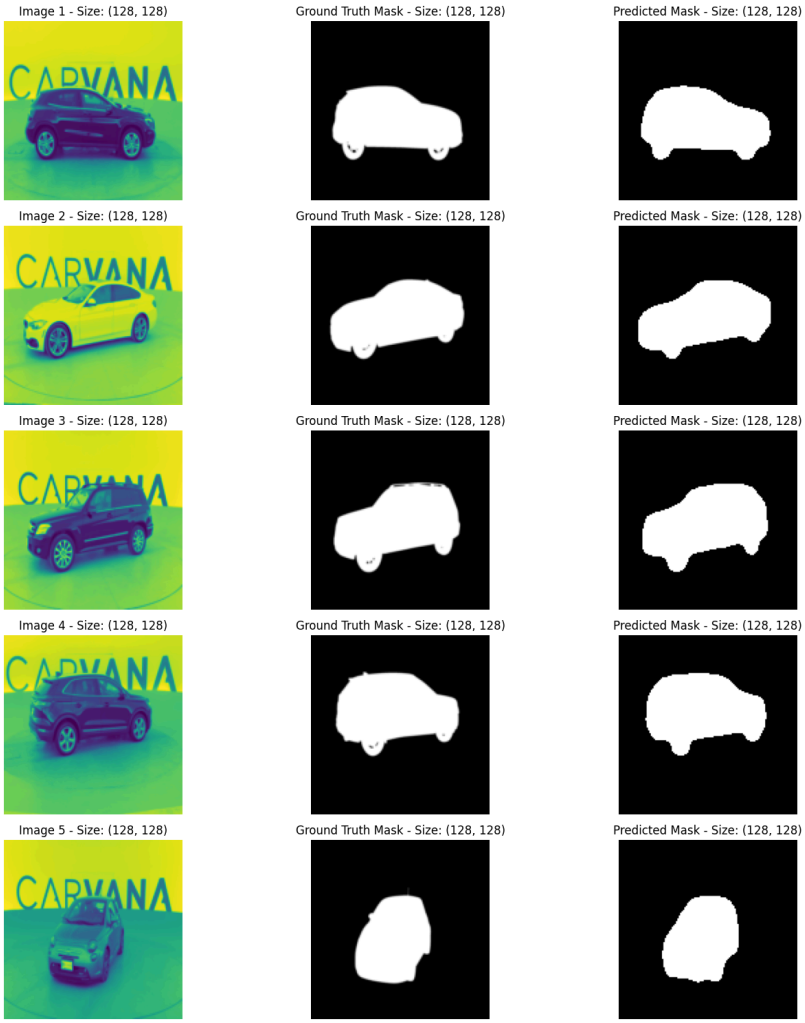
We implemented the model only using pytorch's nn.module, creating a class called U_net and defining the blocks.

```
class UNet(nn.Module):
    def __init__(self):
        super(UNet, self).__init__()

        self.max_pool_2x2 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.down_conv_1 = double_conv(1, 64)
        self.down_conv_2 = double_conv(64, 128)
        self.down_conv_3 = double_conv(128, 256)
        self.down_conv_4 = double_conv(256, 512)
        self.down_conv_5 = double_conv(512, 1024)

        self.up_trans_1 = nn.ConvTranspose2d(in_channels=1024, out_channels=512, kernel_size=2, stride=2)
        self.up_conv_1 = double_conv(1024, 512)
        self.up_trans_2 = nn.ConvTranspose2d(in_channels=512, out_channels=256, kernel_size=2, stride=2)
        self.up_conv_2 = double_conv(512, 256)
        self.up_trans_3 = nn.ConvTranspose2d(in_channels=256, out_channels=128, kernel_size=2, stride=2)
        self.up_conv_3 = double_conv(256, 128)
        self.up_trans_4 = nn.ConvTranspose2d(in_channels=128, out_channels=64, kernel_size=2, stride=2)
        self.up_conv_4 = double_conv(128, 64)

        self.out = nn.Conv2d(in_channels=64, out_channels=1, kernel_size=1)

        self.features = {}

    def forward(self, image):
        x1 = self.down_conv_1(image)
        self.features['enc1'] = x1
        x2 = self.max_pool_2x2(x1)
        x3 = self.down_conv_2(x2)
        self.features['enc2'] = x3
        x4 = self.max_pool_2x2(x3)
        x5 = self.down_conv_3(x4)
        self.features['enc3'] = x5
        x6 = self.max_pool_2x2(x5)
        x7 = self.down_conv_4(x6)
        self.features['enc4'] = x7
        x8 = self.max_pool_2x2(x7)
        x9 = self.down_conv_5(x8)
        self.features['bottleneck'] = x9

        x = self.up_trans_1(x9)
        y = crop_img(x7, x)
        x = self.up_conv_1(torch.cat([x, y], 1))
        self.features['dec4'] = x

        x = self.up_trans_2(x)
        y = crop_img(x5, x)
        x = self.up_conv_2(torch.cat([x, y], 1))
        self.features['dec3'] = x

        x = self.up_trans_3(x)
        y = crop_img(x3, x)
        x = self.up_conv_3(torch.cat([x, y], 1))
        self.features['dec2'] = x

        x = self.up_trans_4(x)
        y = crop_img(x1, x)
        x = self.up_conv_4(torch.cat([x, y], 1))
        self.features['dec1'] = x

        x = self.out(x)
        self.features['output'] = x
        return x
```

We trained it on the carvana dataset (80:20 split for train and test data), for 20 epochs, Adam Optimizer and Binary Cross Entropy as the loss function.

Finally we achieved an accuracy of : Epoch 20/20, :Test Loss: 0.0238, Test Accuracy: 0.9670

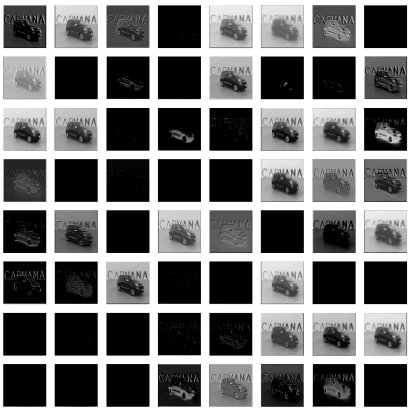
Visualizing Random Outputs against ground truth:

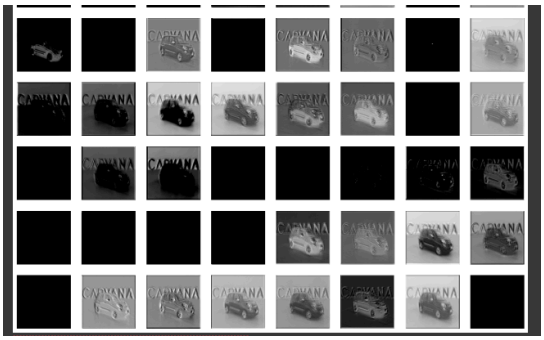


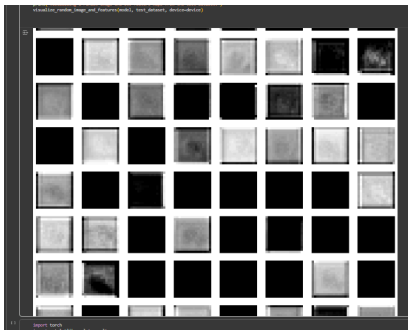
We had good results, further improvements can be made using more skip connections and a deeper network.

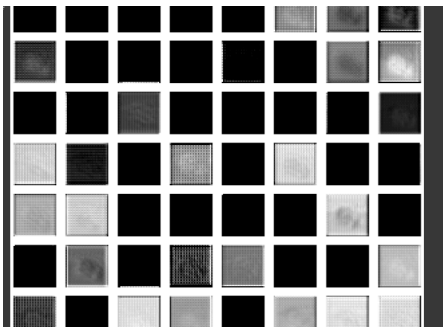
Additionally I also proceeded to visualise each and every block in the U Net Architecture:

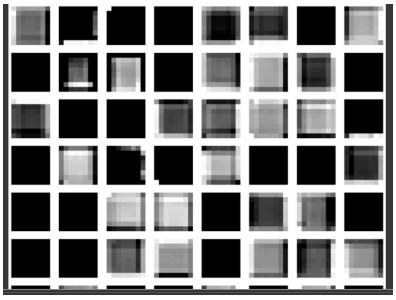
enc1











bottleneck

Inference / Analysis of U-Net Layer Outputs on the Carvana Dataset

The layer-wise outputs of the U-Net model trained on the Carvana dataset reveal how the network progressively learns and reconstructs spatial and semantic information for accurate car mask segmentation.

Encoder (Contracting Path):

- The initial convolutional layers capture low-level spatial features such as edges, color gradients, and basic textures of the car and background.
- As depth increases, the feature maps become more abstract and sparse, with higher layers encoding object boundaries, reflections, and car body shapes.
- The pooling operations reduce spatial resolution but expand the receptive field, allowing the model to capture contextual information such as the overall car outline.

Bottleneck:

- At the deepest part of the network, the feature maps represent high-level semantic concepts — distinguishing the car from background regions like shadows or the driveway.
- These representations are dense and compact, encoding class-level information rather than spatial detail.

Decoder (Expanding Path):

- The upsampling and concatenation (skip connections) restore spatial detail lost during downsampling.
- Skip connections successfully preserve fine-grained edge information, crucial for pixel-accurate segmentation of car contours and mirror edges.
- The later decoder layers progressively refine the segmentation map, turning coarse car outlines into precise binary masks that align well with the original image.

Final Output Layer:

- The sigmoid output produces a binary mask, where pixel intensities close to 1 correspond to the car region.
- Qualitatively, the masks are smooth, continuous, and show strong boundary adherence, indicating the network's effective learning of both local and global car features.

General Observation:

- Layer visualizations confirm that the U-Net's encoder-decoder symmetry, combined with skip connections, leads to rich hierarchical feature representation.
- Early layers are texture-focused, middle layers are structure-focused, and late layers integrate both for precise segmentation.

Summary

From the layer-wise outputs, it can be inferred that:

- The encoder layers specialize in extracting contextual and semantic car features.
- The decoder layers specialize in recovering spatial precision.
- The U-Net architecture effectively balances detail preservation and semantic abstraction, resulting in accurate car mask predictions for the Carvana dataset.