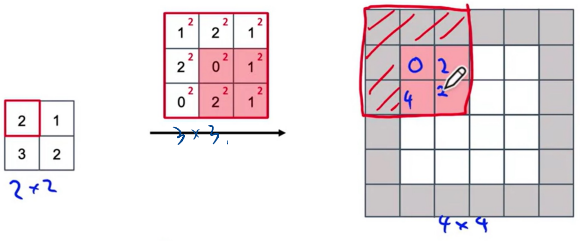
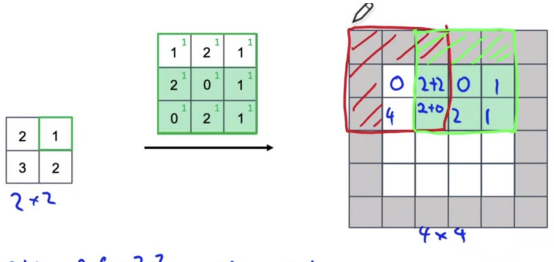
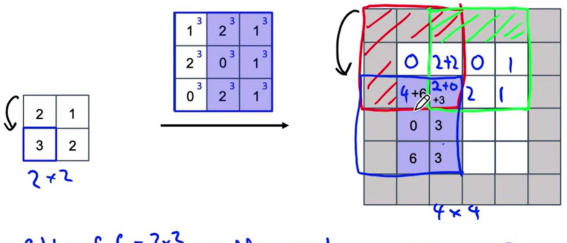
Transpose Convolution



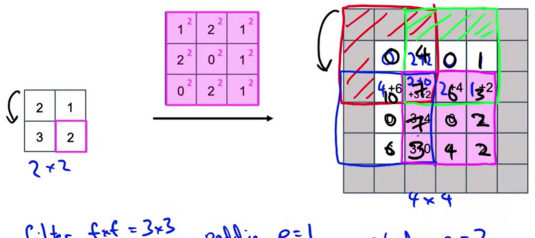
filter f×f = 3×3 padding p=1 strida s=2



filter f×f = 3×3 padding p=1 strida s=2

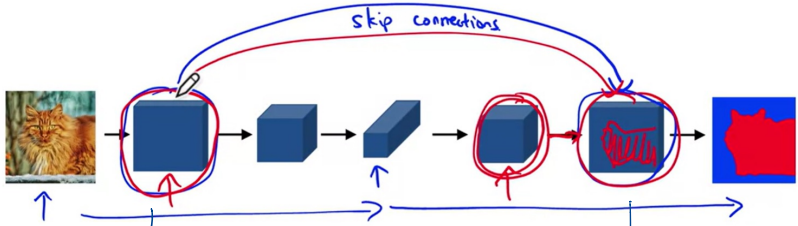


filter f×f = 3×3 padding p=1 strida s=2



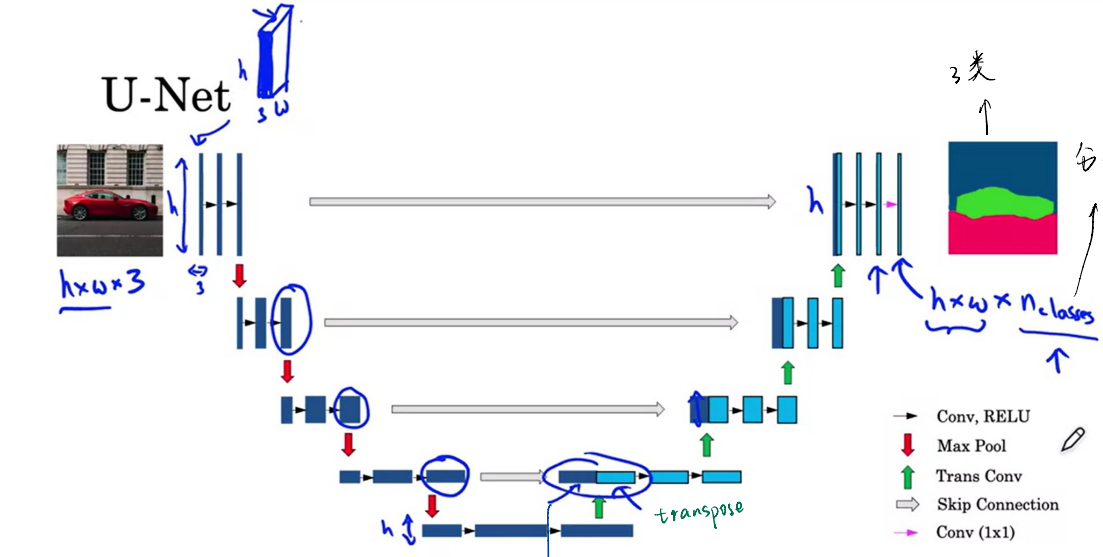
filter f×f = 3×3 padding p=1 strida s=2

Deep Learning for Semantic Segmentation



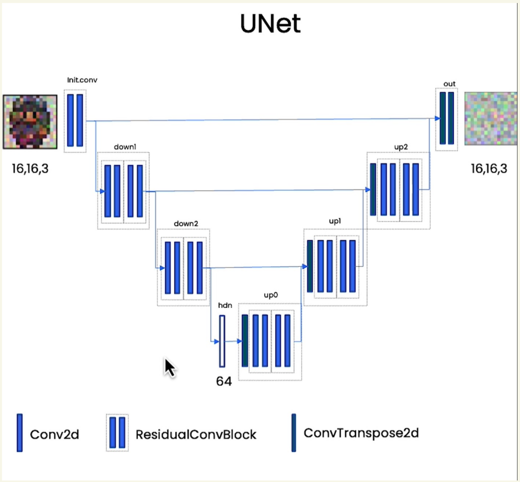
分辨率很高
细粒度低的特征信息

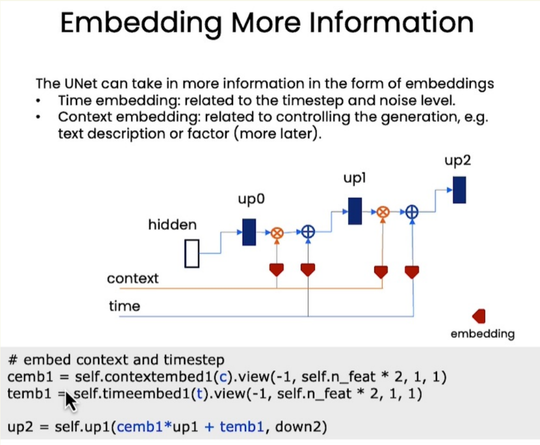
更高级上下文
+更详细的细节



[Ronneberger et al., 2015, U-Net: Convolutional Networks for Biomedical Image Segmentation] Andrew Ng

Project 1 用 Tensorflow





setting things up.

```python
class ContextUnet(nn.Module):
    def __init__(self, in_channels, n_feat=256, n_cfeat=10, height=28):  # cfeat - context features
        super(ContextUnet, self).__init__()

        # number of input channels, number of intermediate feature maps and number of classes
        self.in_channels = in_channels
        self.n_feat = n_feat
        self.n_cfeat = n_cfeat
        self.h = height  #assume h == w. must be divisible by 4, so 28,24,20,16...

        # Initialize the initial convolutional layer
        self.init_conv = ResidualConvBlock(in_channels, n_feat, is_res=True)

        # Initialize the down-sampling path of the U-Net with two levels
        self.down1 = UnetDown(n_feat, n_feat)        # down1 #[10, 256, 8, 8]
        self.down2 = UnetDown(n_feat, 2 * n_feat)    # down2 #[10, 256, 4,  4]

         # original: self.to_vec = nn.Sequential(nn.AvgPool2d(7), nn.GELU())
        self.to_vec = nn.Sequential(nn.AvgPool2d((4)), nn.GELU())

        # Embed the timestep and context labels with a one-layer fully connected neural network
        self.timeembed1 = EmbedFC(1, 2*n_feat)
        self.timeembed2 = EmbedFC(1, 1*n_feat)
        self.contextembed1 = EmbedFC(n_cfeat, 2*n_feat)
        self.contextembed2 = EmbedFC(n_cfeat, 1*n_feat)

        # Initialize the up-sampling path of the U-Net with three levels
        self.up0 = nn.Sequential(
            nn.ConvTranspose2d(2 * n_feat, 2 * n_feat, self.h//4, self.h//4), # up-sample
            nn.GroupNorm(8, 2 * n_feat), # normalize
            nn.ReLU(),
        )
        self.up1 = UnetUp(4 * n_feat, n_feat)
        self.up2 = UnetUp(2 * n_feat, n_feat)

        # Initialize the final convolutional layers to map to the same number of channels as the input image
        self.out = nn.Sequential(
            nn.Conv2d(2 * n_feat, n_feat, 3, 1, 1), # reduce number of feature maps   #in_channels, out_channels, kernel_size,
            nn.GroupNorm(8, n_feat), # normalize
            nn.ReLU(),
            nn.Conv2d(n_feat, self.in_channels, 3, 1, 1), # map to same number of channels as input
        )
```

这里 Pytorch 版 diffusion model 中 预测噪音.

```python
def forward(self, x, t, c=None):
    """
    x : (batch, n_feat, h, w) : input image
    t : (batch, n_cfeat)      : time step
    c : (batch, n_classes)    : context label
    """
    # x is the input image, c is the context label, t is the timestep, context_mask says which samples to block the context on

    # pass the input image through the initial convolutional layer
    x = self.init_conv(x)
    # pass the result through the down-sampling path
    down1 = self.down1(x)       #[10, 256, 8, 8]
    down2 = self.down2(down1)   #[10, 256, 4, 4]

    # convert the feature maps to a vector and apply an activation
    hiddenvec = self.to_vec(down2)

    # mask out context if context_mask == 1
    if c is None:
        c = torch.zeros(x.shape[0], self.n_cfeat).to(x)

    # embed context and timestep
    cemb1 = self.contextembed1(c).view(-1, self.n_feat * 2, 1, 1)     # (batch, 2*n_feat, 1,1)
    temb1 = self.timeembed1(t).view(-1, self.n_feat * 2, 1, 1)
    cemb2 = self.contextembed2(c).view(-1, self.n_feat, 1, 1)
    temb2 = self.timeembed2(t).view(-1, self.n_feat, 1, 1)
    #print(f"uunet forward: cemb1 {cemb1.shape}. temb1 {temb1.shape}, cemb2 {cemb2.shape}. temb2 {temb2.shape}")


    up1 = self.up0(hiddenvec)
    up2 = self.up1(cemb1*up1 + temb1, down2)  # add and multiply embeddings
    up3 = self.up2(cemb2*up2 + temb2, down1)
    out = self.out(torch.cat((up3, x), 1))
    return out
```