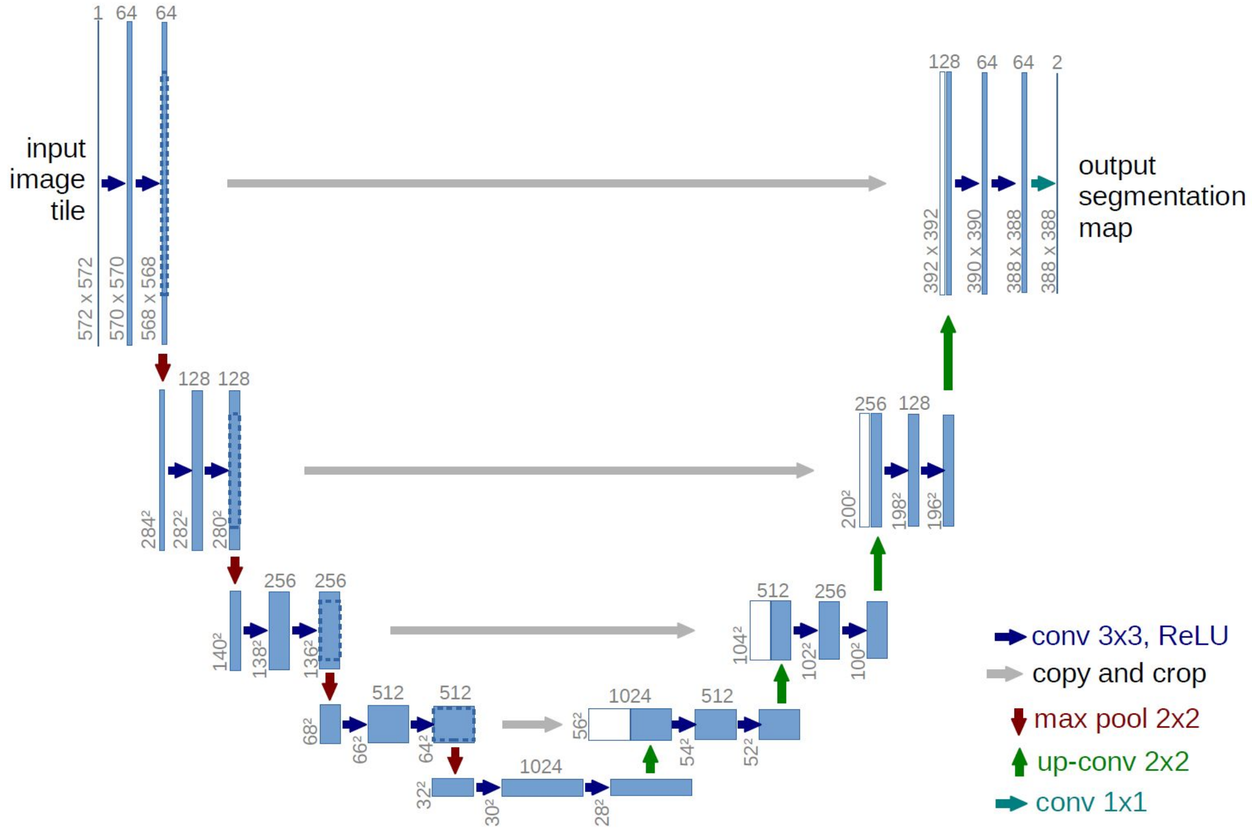
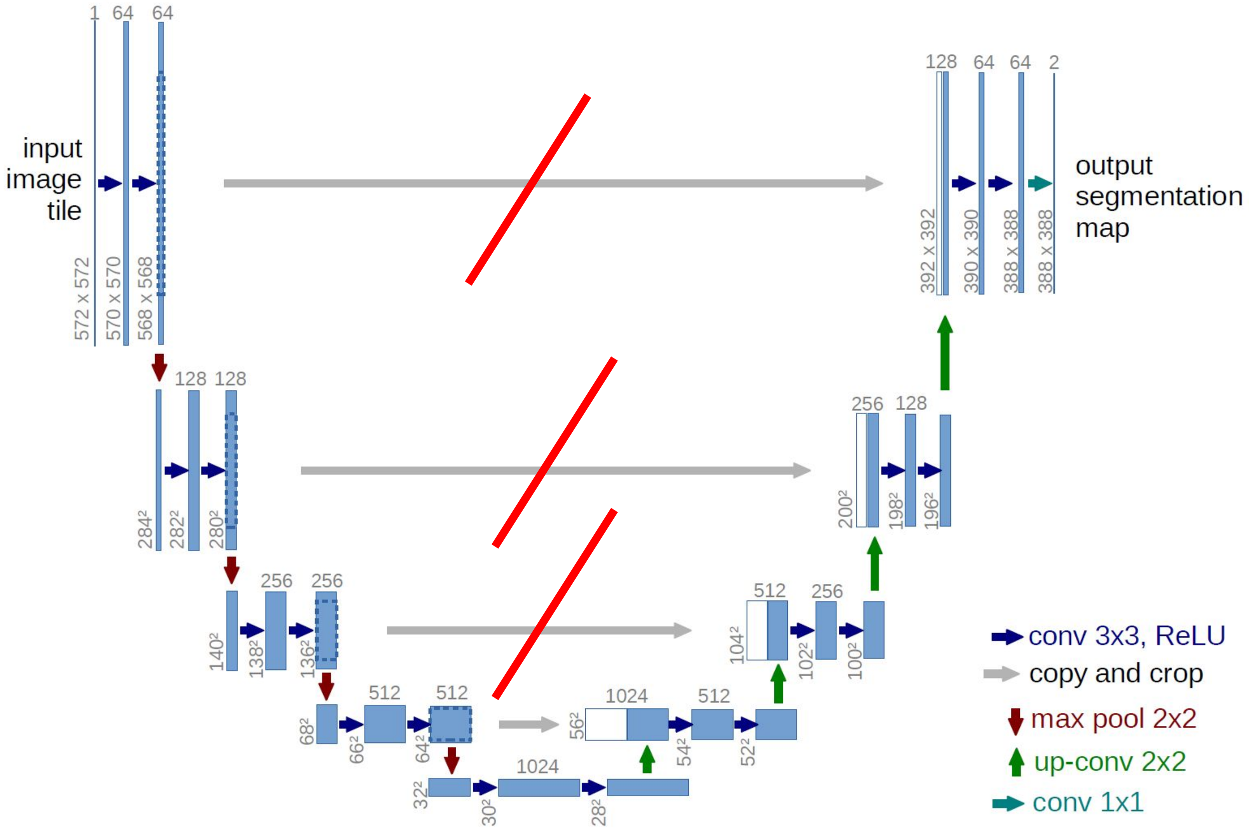
ResNet50 based Image Compression

нейро сжиматель

Изначальная идея (поток мыслей)



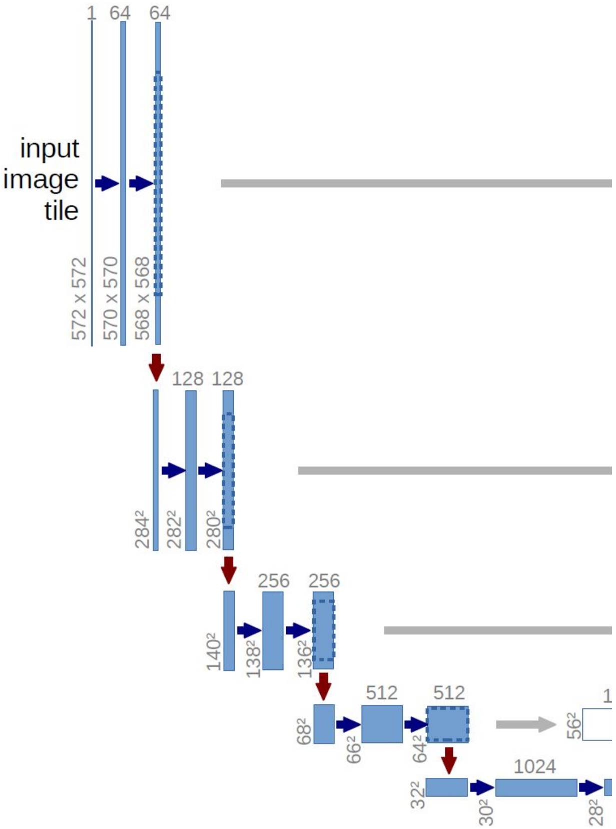
Изначальная идея (не очень умно)



Изначальная идея (не очень умно)

```
Old Forward
Pre-pool layer_0 shape: torch.Size([2, 3, 512, 512]) total: 786432
Pre-pool layer_1 shape: torch.Size([2, 64, 256, 256]) total: 4194304
Pre-pool layer_2 shape: torch.Size([2, 256, 128, 128]) total: 4194304
Pre-pool layer_3 shape: torch.Size([2, 512, 64, 64]) total: 2097152
Pre-pool layer_4 shape: torch.Size([2, 1024, 32, 32]) total: 1048576
Latent shape: torch.Size([2, 1024, 16, 16]) total: 262144
Up block key: layer_4 shape: torch.Size([2, 1024, 32, 32]) total: 1048576
Up block key: layer_3 shape: torch.Size([2, 512, 64, 64]) total: 2097152
Up block key: layer_2 shape: torch.Size([2, 256, 128, 128]) total: 4194304
Up block key: layer_1 shape: torch.Size([2, 64, 256, 256]) total: 4194304
Up block key: layer_0 shape: torch.Size([2, 3, 512, 512]) total: 786432
Out shape:  torch.Size([2, 3, 512, 512])
```

Изначальная идея №2



Дальше Upscale или ConvTranspose2D

Датасет

https://www.kaggle.com/datasets/rhtsingh/130k-images-512x512-universal-image-embeddings

130k Images (512x512) - Universal Image Embeddings

Метрики

1. MSE
2. PSNR
3. Latent space Entropy (!)
4. Perceptual loss (VGG16)

```
# MSE
def mse_loss(result, target):
    result = result.view(result.size(0), -1)
    target = target.view(target.size(0), -1)
    mses = F.mse_loss(result, target, reduction='none').mean(dim=1)
    return torch.mean(mses)

# PSNR
def psnr(result, target):
    result = result.view(result.size(0), -1)
    maxes = torch.max(result, dim = 1)[0] ** 2
    target = target.view(target.size(0), -1)
    mses = F.mse_loss(result, target, reduction='none').mean(dim=1)
    psnrs = 10.0 * torch.log10(maxes.to(torch.float32) /
mses.to(torch.float32)).to(selected_dtype)
    return torch.mean(psnrs)
```

```
def latent_entropy_aprox(result):
    unique, counts = torch.unique(result, return_counts=True)
    probabilities = counts.float() / result.numel()
    entropy = -torch.sum(probabilities * torch.log2(probabilities + 1e-9))  #
    return entropy.to(selected_dtype)
# Perseptual loss
class VGGFeatures(nn.Module):
    def __init__(self, feature_layers = {3, 6, 11, 16}):
        super(VGGFeatures, self).__init__()
        vgg = torchvision.models.vgg11(weights="VGG11_Weights.DEFAULT")
        vgg.eval()
        for param in vgg.parameters():
            param.requires_grad = False
        self.vgg = vgg
        self.feature_layers = feature_layers
    def forward(self, x):
        features = []
        for i, layer in enumerate(self.vgg.features):
            x = layer(x)
            if i in self.feature_layers:
                features.append(x)
        return features
vgg_features = VGGFeatures().to(device)
def vgg_perceptual_loss(result, target, vgg_to_use=vgg_features):
    result = vgg_to_use(result)
    target = vgg_to_use(target)
    return sum(F.mse_loss(orig, decomp) for orig, decomp in zip(result, target))
```

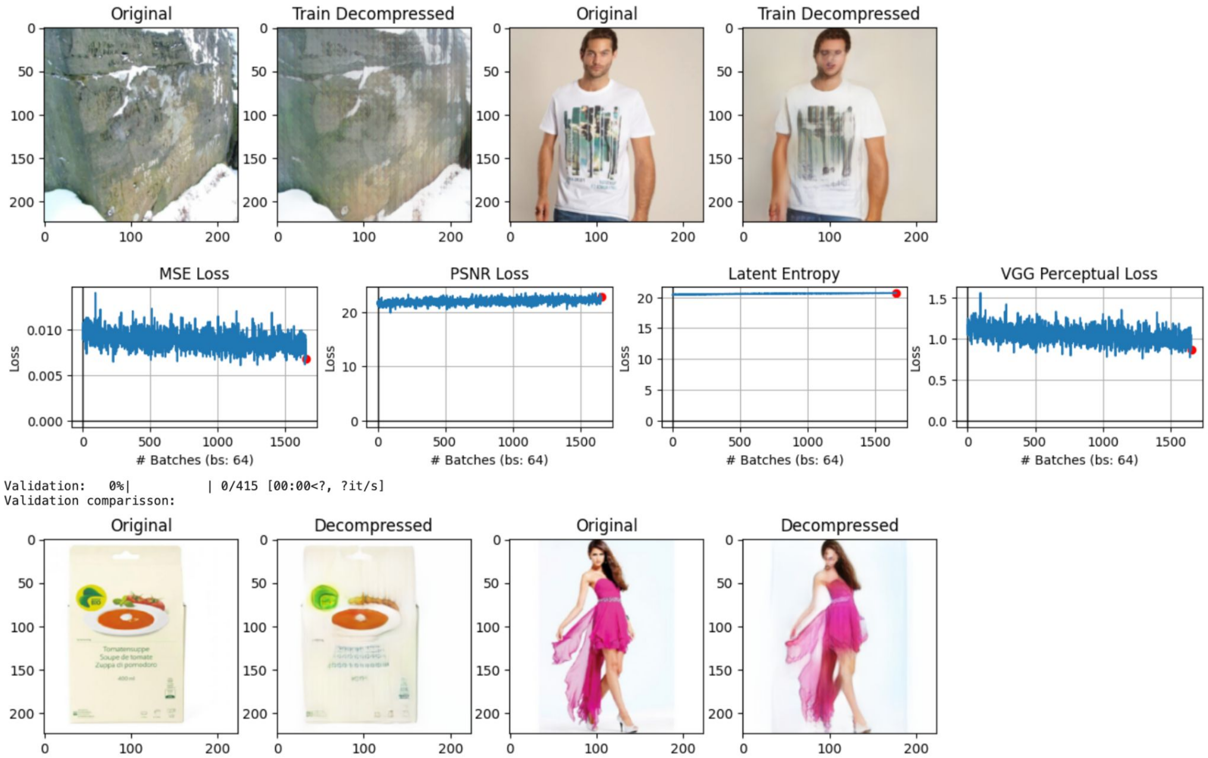
Метрики

1. MSE (обучаем только на ней)

Считаем все, запоминаем

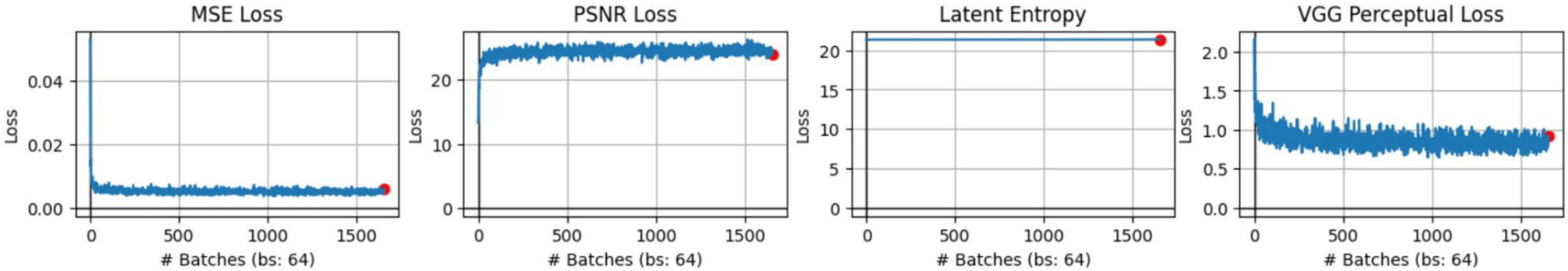
Потом расставляем коэф. на основе собранных данных

2. PSNR
3. Latent space Entropy (!)
4. Perceptual loss (VGG16)



```
Validation:   0%|          | 0/415 [00:00<?, ?it/s]
Validation comparisson:
```

Казалось бы успех…

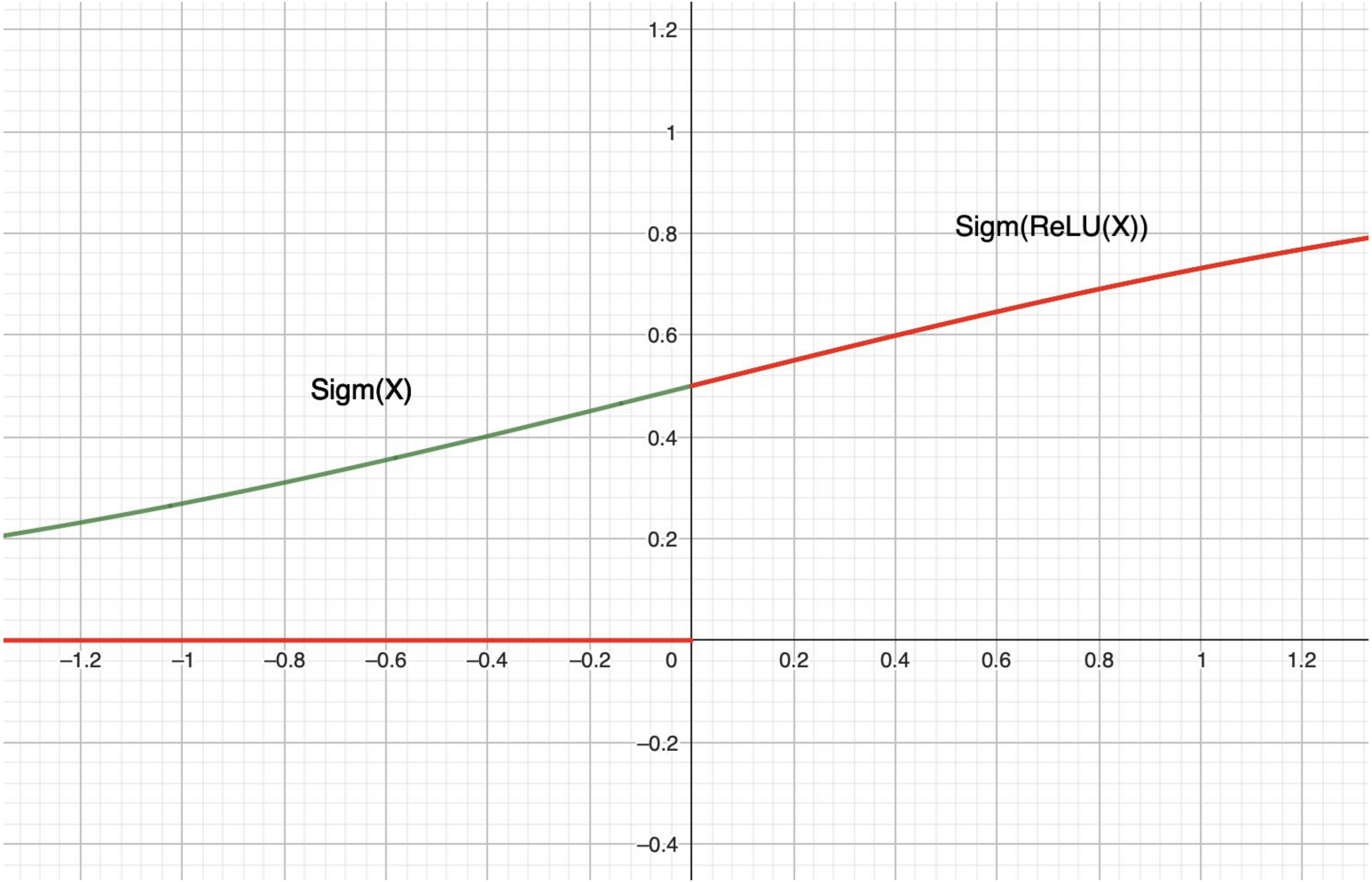


Но что-то пошло не так

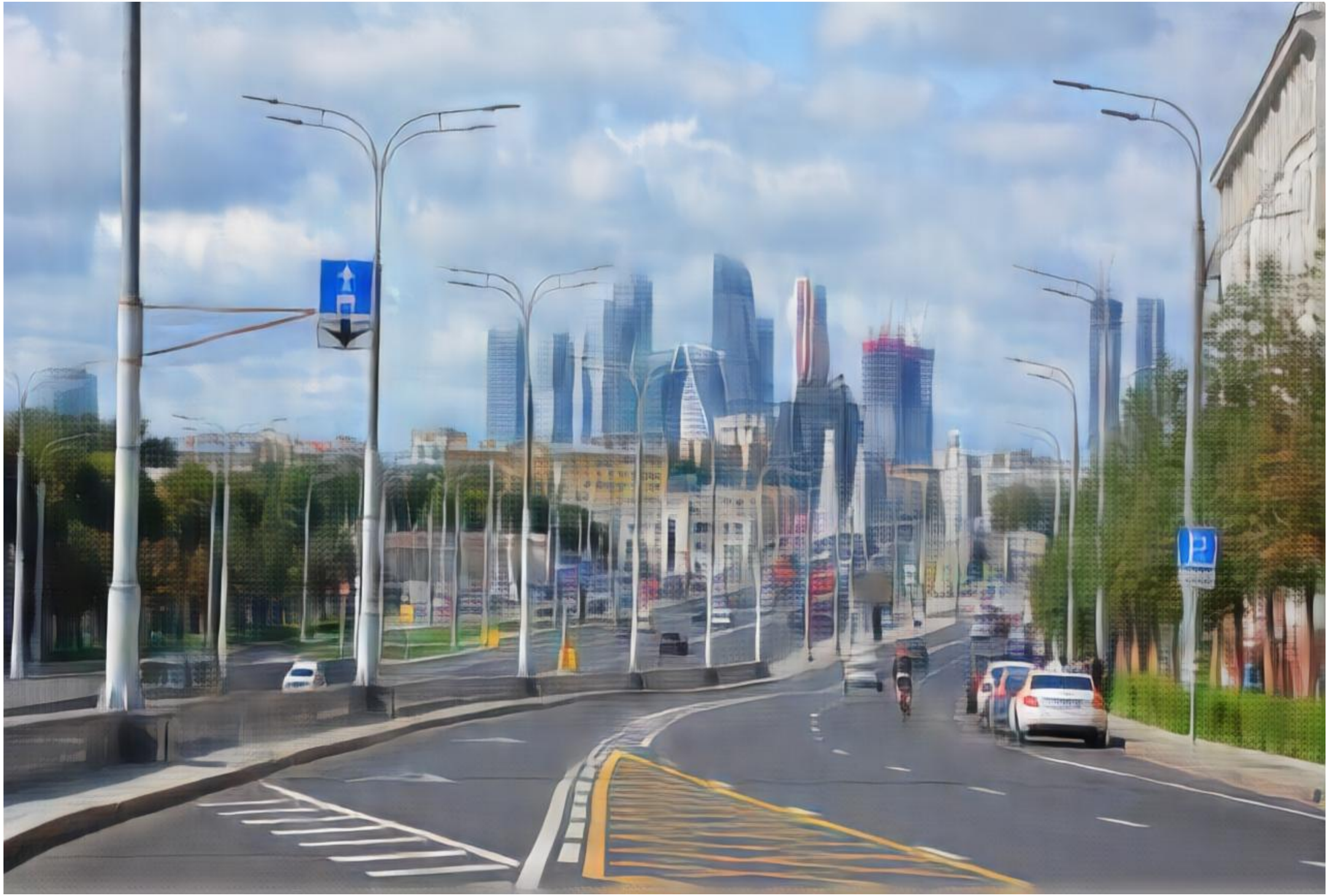
Промежуточное представление должно быть **[0, 1)**

Функция активации: **sigmoid**

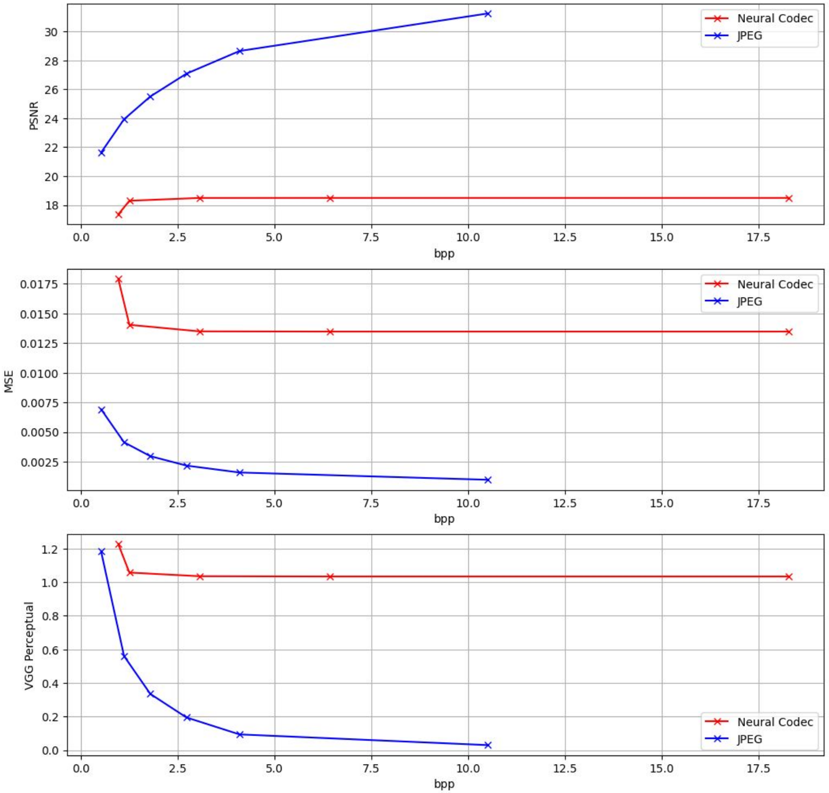
Однако перед этим шло **ReLU…**


Sigm(ReLU(X))
Sigm(X)
1.2
1
0.8
0.6
0.4
0.2
−0.2
−0.4
−1.2
−1
−0.8
−0.6
−0.4
−0.2
0
0.2
0.4
0.6
0.8
1
1.2

P


P

Результаты (Сравнение с JPEG)


PSNR
MSE
VGG Perceptual
bpp
Neural Codec
JPEG