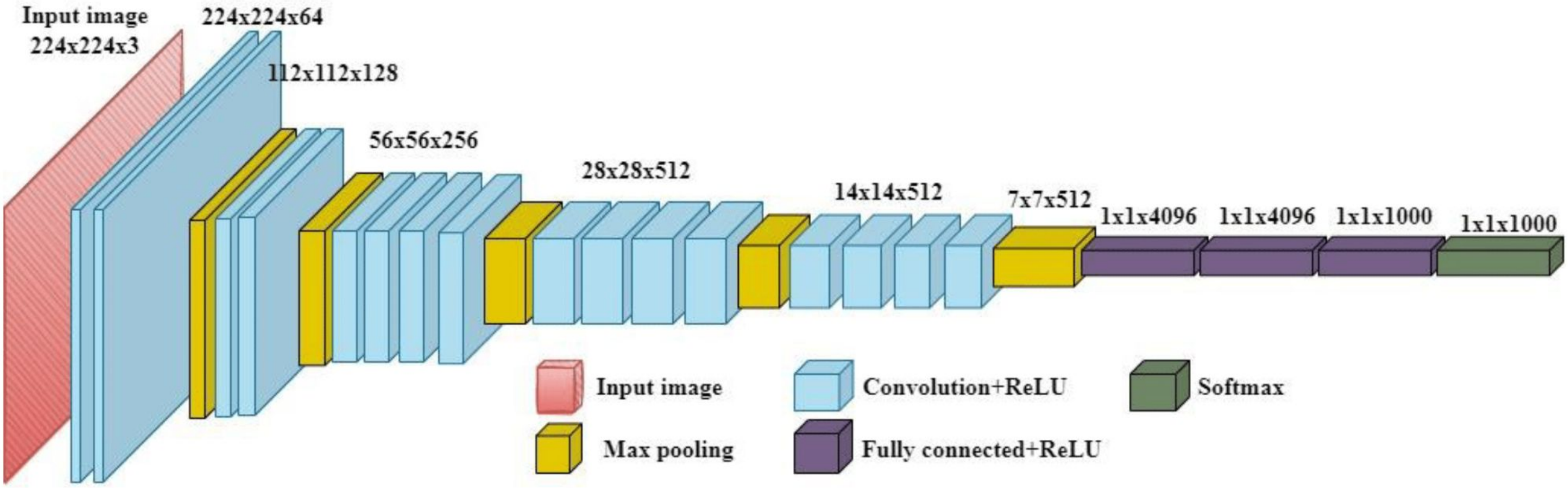
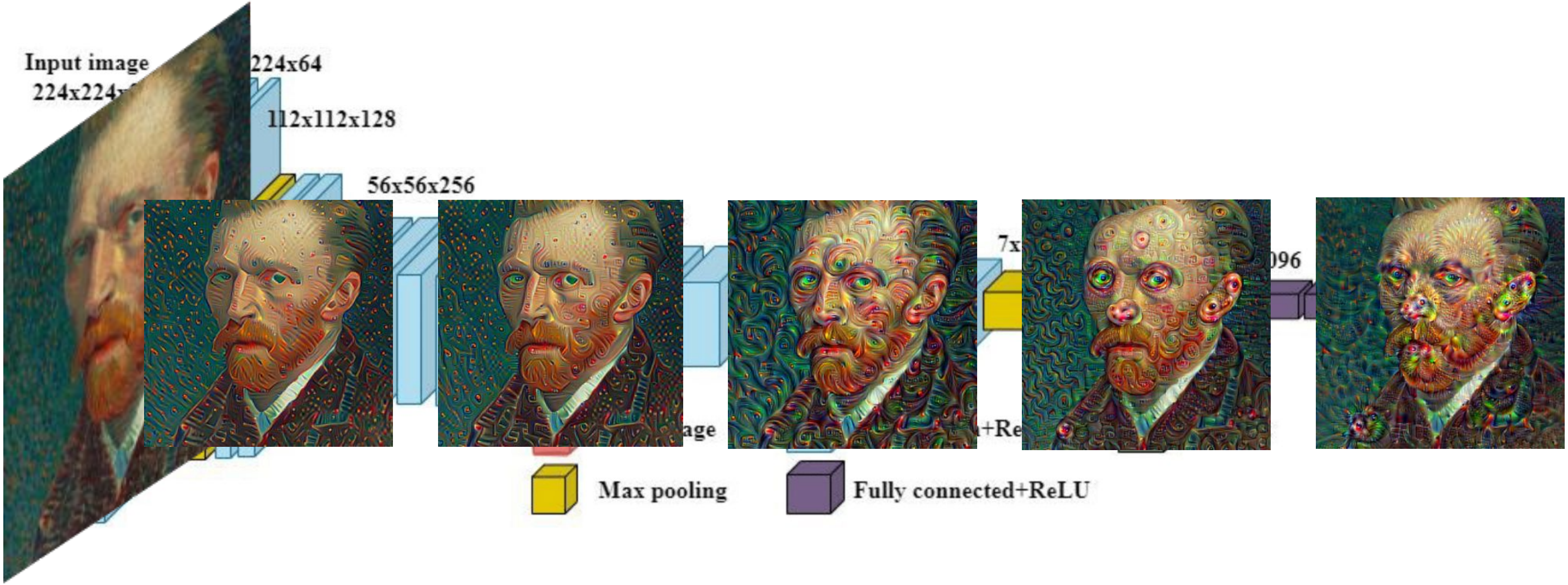
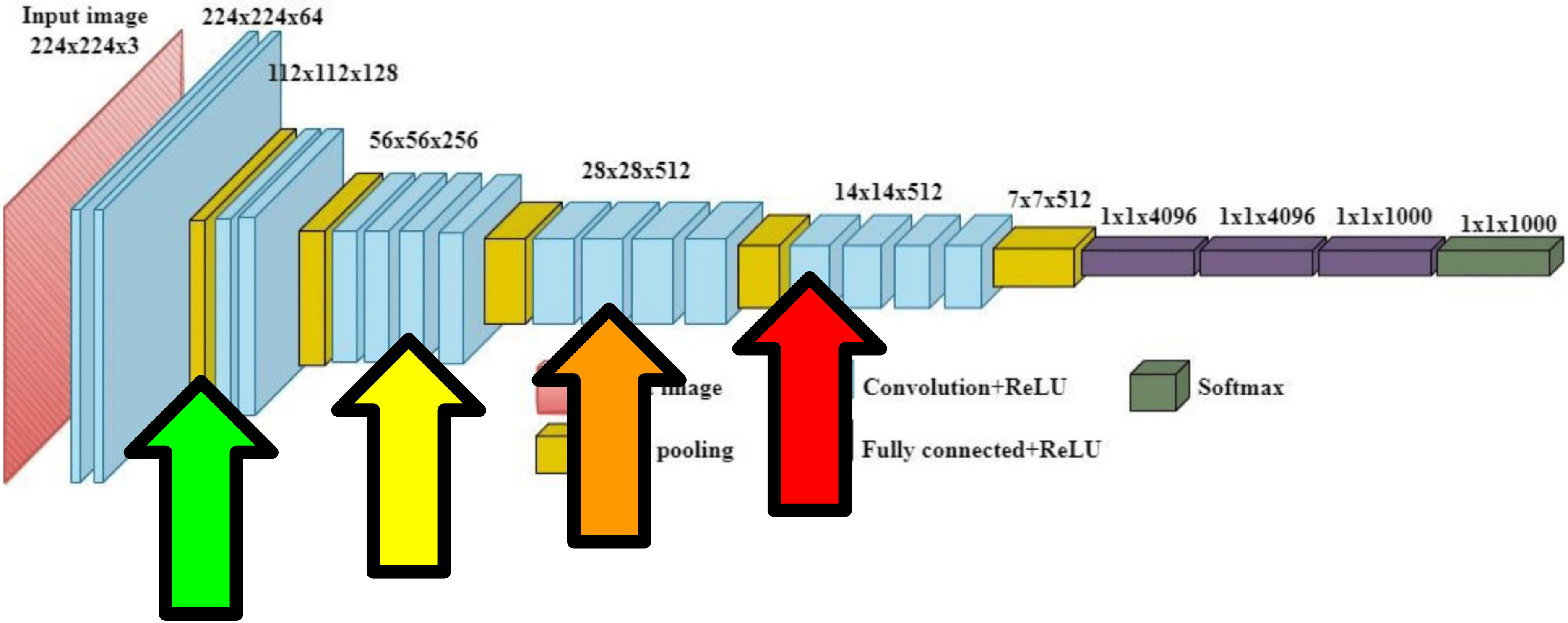
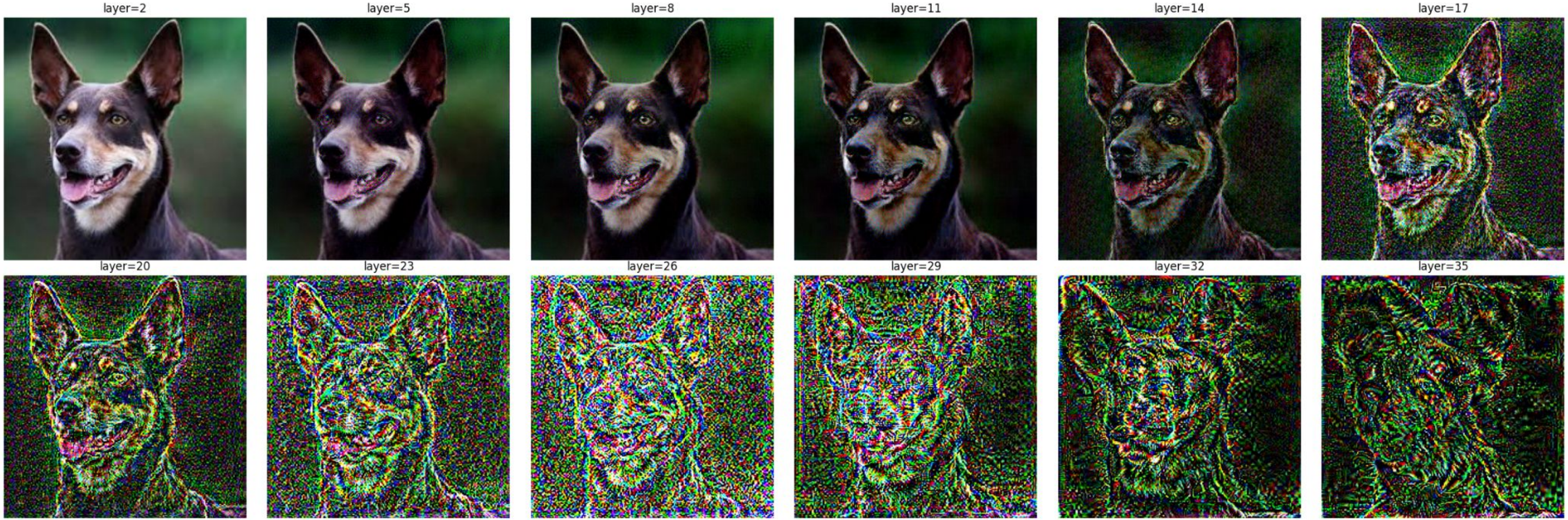
Image classifier

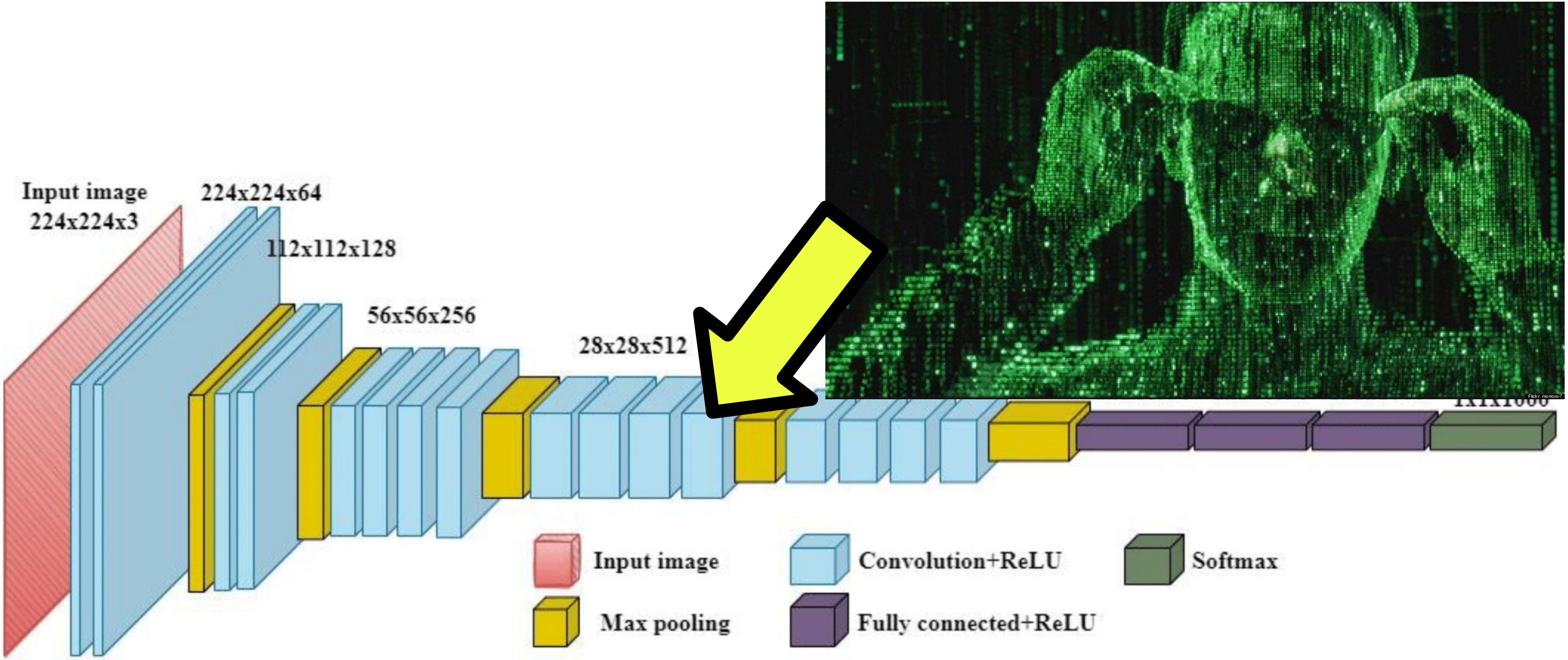


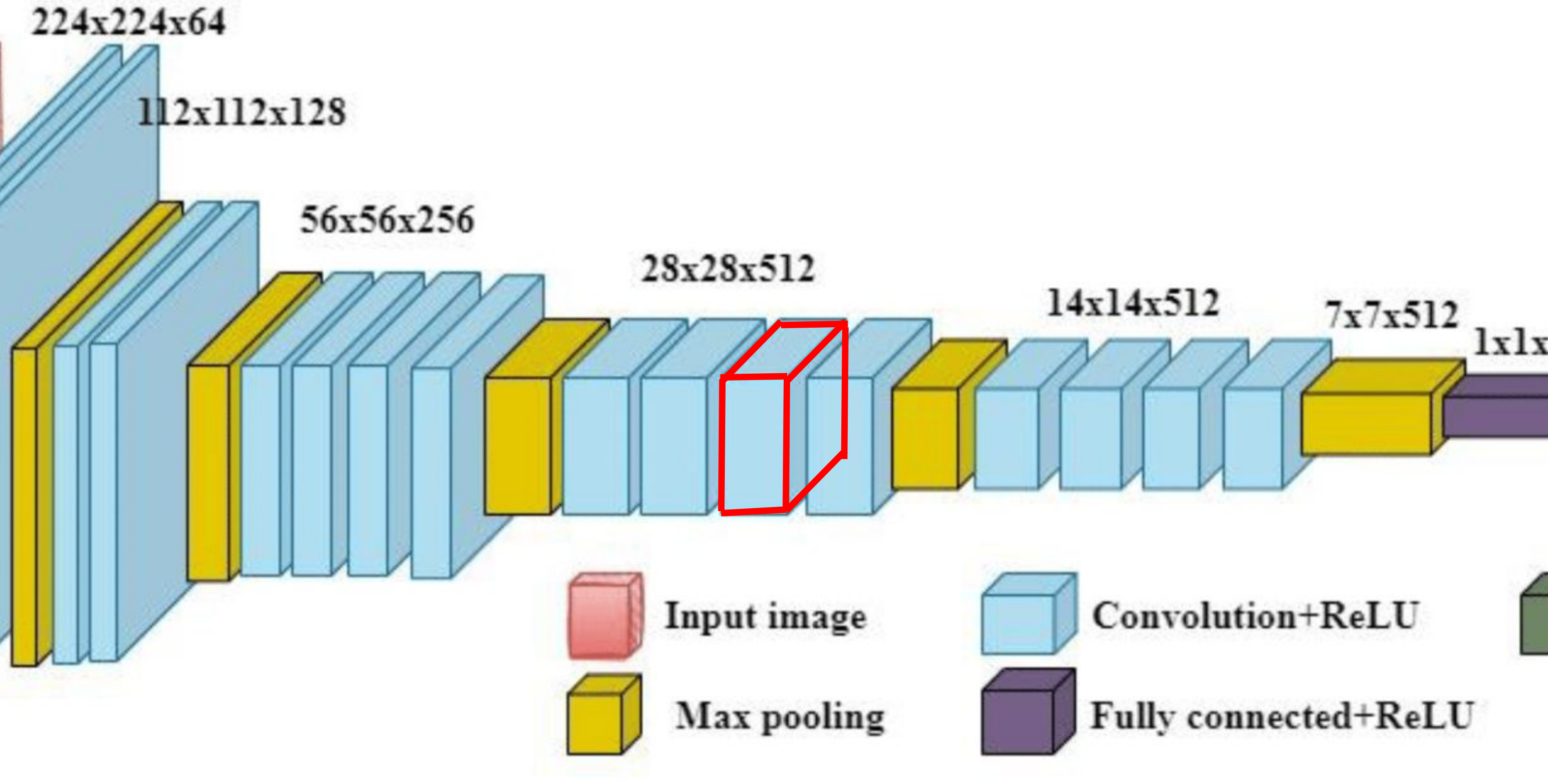

Input image
224x224x
224x64
112x112x128
56x56x256
age
7x
+Re
096
Max pooling
Fully connected+ReLU

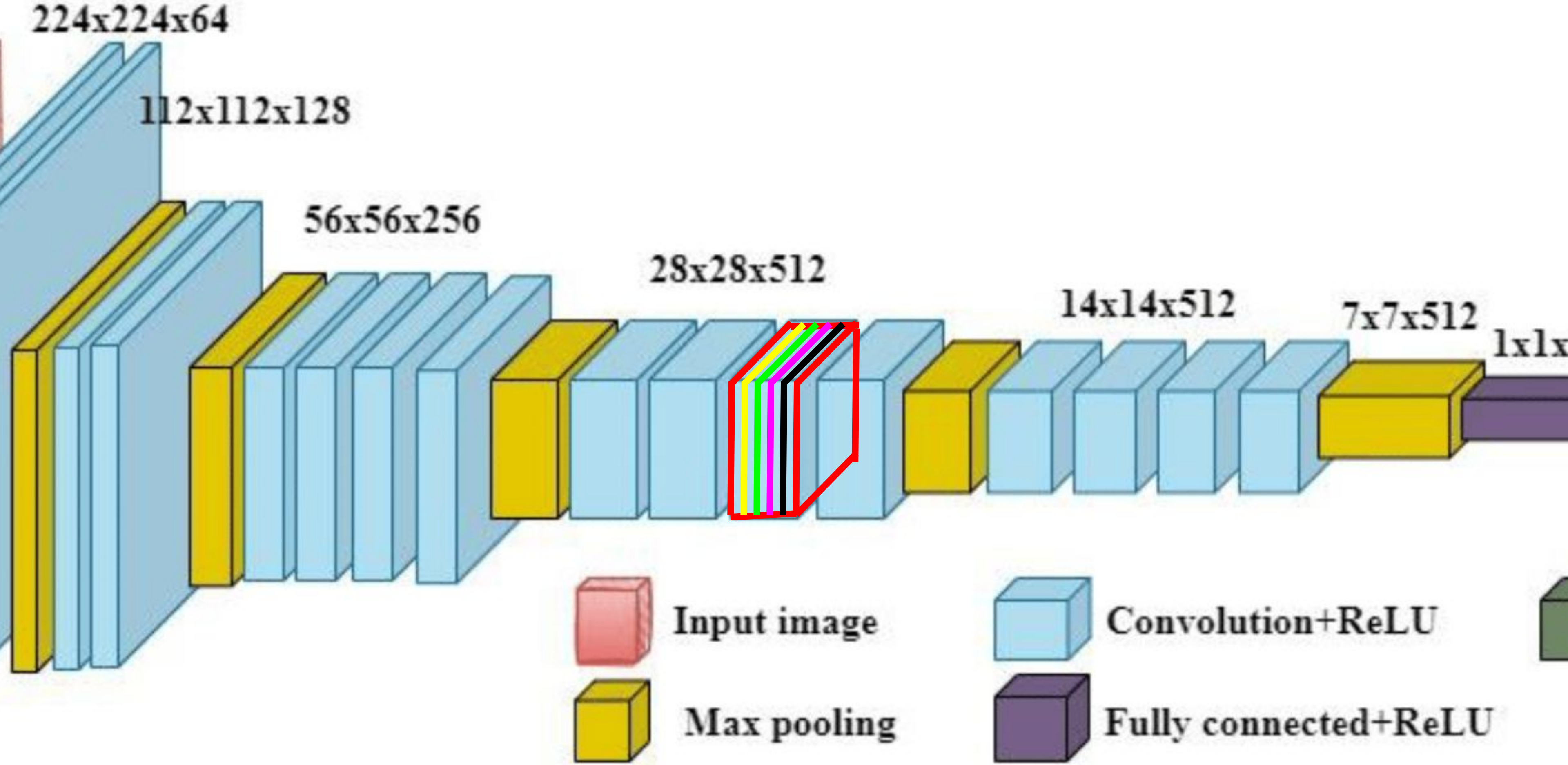
compression



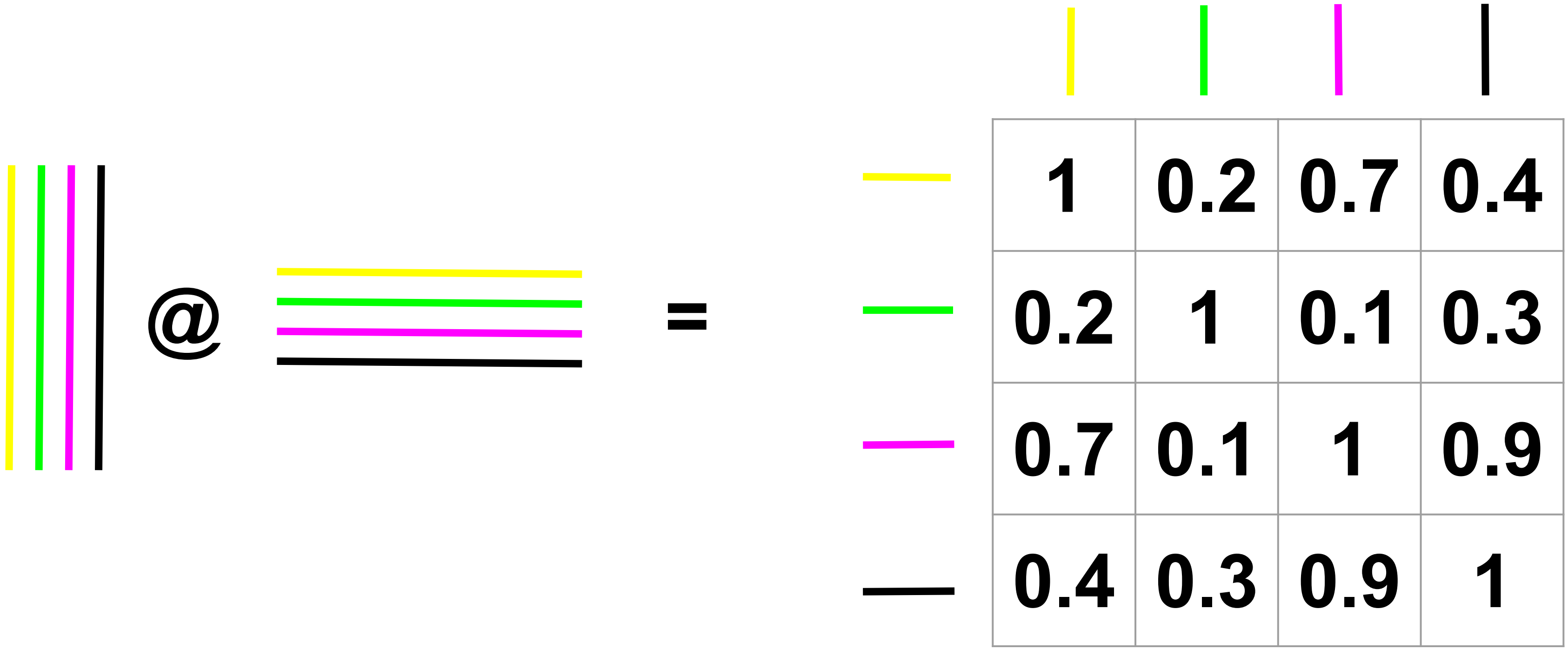

layer=2
layer=5
layer=8
layer=11
layer=14
layer=17
layer=20
layer=23
layer=26
layer=29
layer=32
layer=35


Input image
224x224x3
224x224x64
112x112x128
56x56x256
28x28x512
1x1x1000
Input image
Convolution+ReLU
Softmax
Max pooling
Fully connected+ReLU


224x224x64
112x112x128
56x56x256
28x28x512
14x14x512
7x7x512
1x1x
Input image
Convolution+ReLU
Max pooling
Fully connected+ReLU


224x224x64
112x112x128
56x56x256
28x28x512
14x14x512
7x7x512
1x1x
Input image
Convolution+ReLU
Max pooling
Fully connected+ReLU

Gram matrix (aka. covariance aka. correlation)

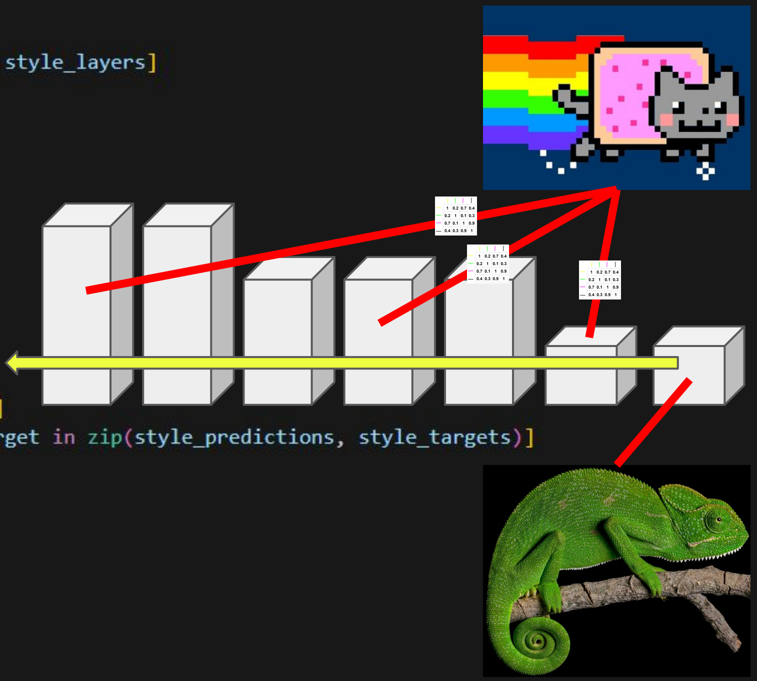


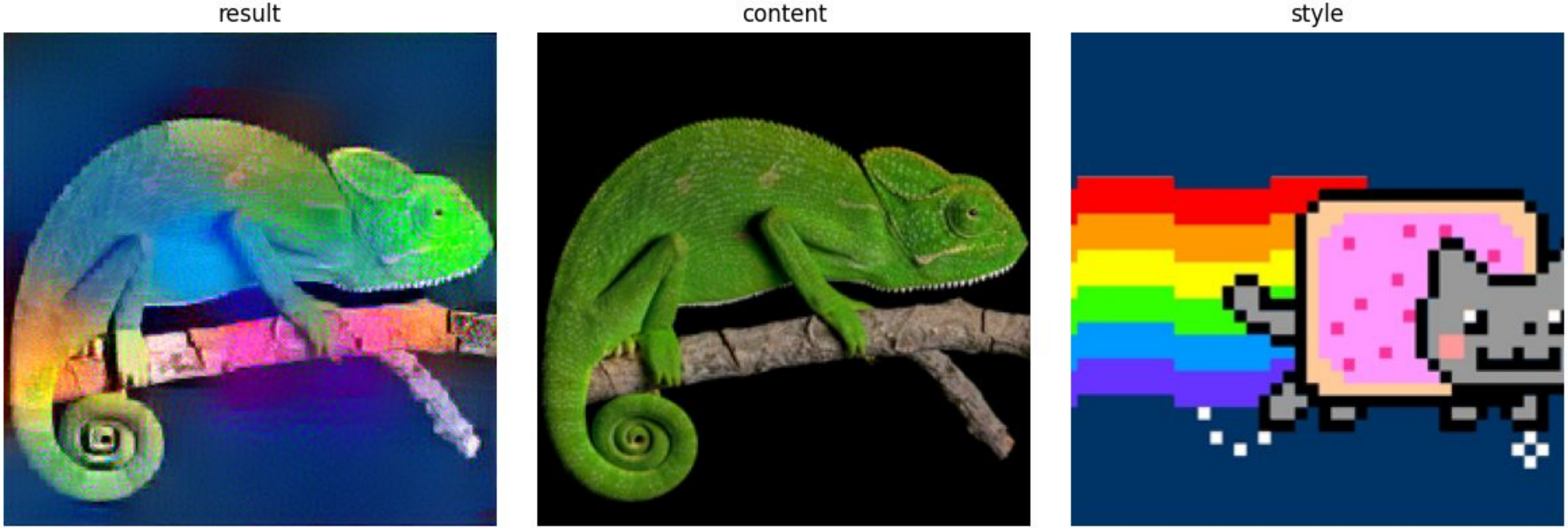
	🟨	🟩	🟪	⬛
🟨	1	0.2	0.7	0.4
🟩	0.2	1	0.1	0.3
🟪	0.7	0.1	1	0.9
⬛	0.4	0.3	0.9	1

```python
style_layers = [1, 6, 11, 20, 29]
content_layer = 28

def style_transfer(start, content, style, scaler=1., content_layer=content_layer, style_layers=style_layers, epochs=10, m=vgg_hooked, mem=vgg_mem):
    content_activations = save_activations(content)
    style_activations = save_activations(style)
    # move to device
    start = copy.deepcopy(start.detach()).to(device).requires_grad_()
    content_target = content_activations[content_layer].to(device)
    style_targets = [gram(style_activations[layer]).to(device) for layer in style_layers]
    m = m.to(device)
    m.eval()
    # optimizer
    optimizer = torch.optim.LBFGS([start])
    for epoch in tqdm(range(epochs)):
        def closure():
            m(start)
            # content loss
            content_predicted = mem[content_layer]
            content_loss = F.mse_loss(content_predicted, content_target)
            # style loss
            style_predictions = [gram(mem[layer]) for layer in style_layers]
            style_losses = [F.mse_loss(predicted, target) for predicted, target in zip(style_predictions, style_targets)]
            style_loss = torch.stack(style_losses).sum()
            # merge losses
            loss = content_loss + scaler * style_loss
            optimizer.zero_grad()
            loss.backward()
            return loss

        optimizer.step(closure)
        start.data = torch.clip(start, 0.0, 1.0).data
        if epoch % log_every == log_every - 1:
            print(epoch)
    return start
```

✓ 0.0s

Python

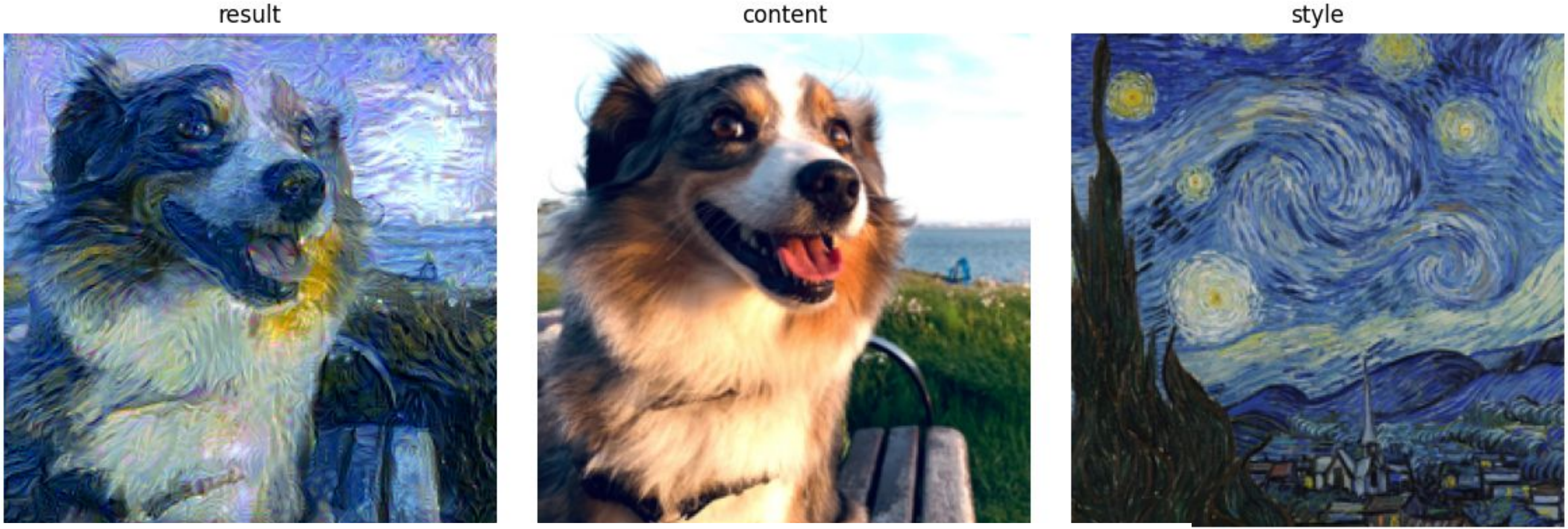


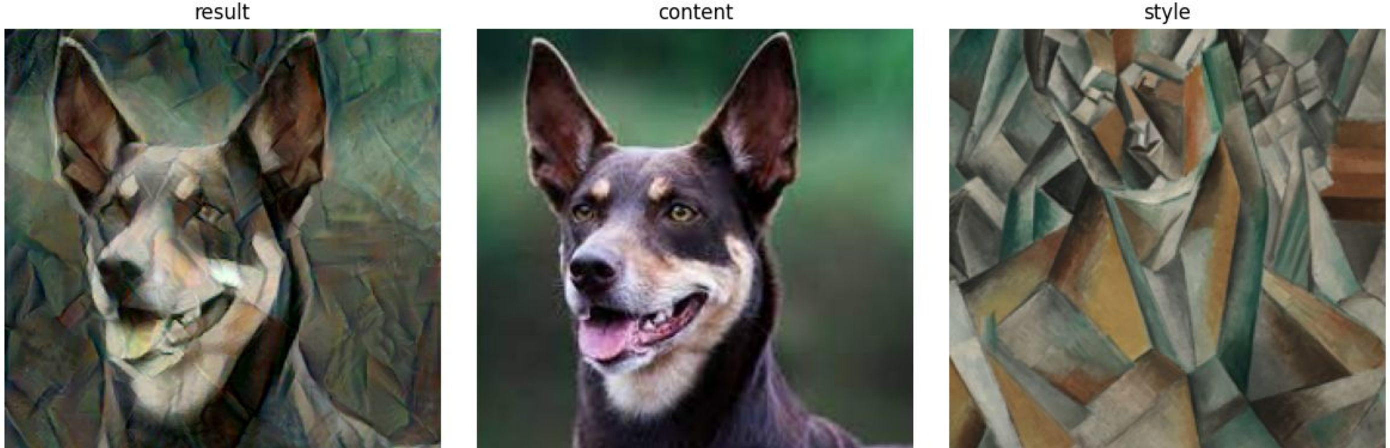


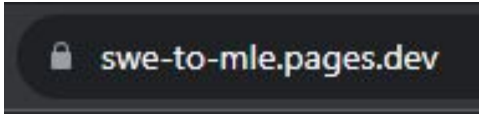
result

content

style







swe-to-mle.pages.dev