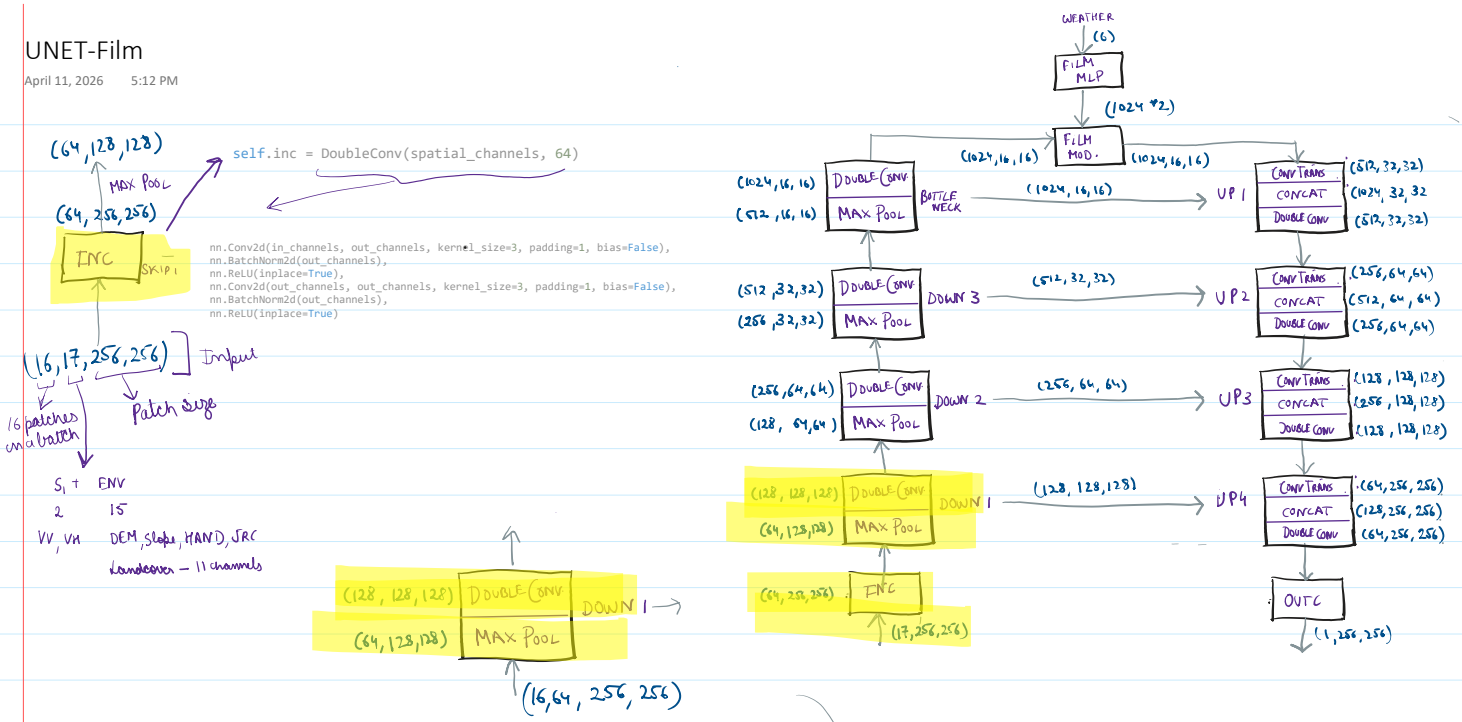
UNET-Film

April 11, 2026 5:12 PM



MAX POOLING nn.MaxPool2d(2)

$$Y_{c,i,j} = \max_{m \in \{0,1\}, n \in \{0,1\}} X_{c,2i+m,2j+n}$$

Shrunk each feature map by sliding 2×2 window, passing highest number

$\frac{256}{2}, \frac{256}{2}$

Output (16, 64, 128, 128)

Conv 2D nn.Conv2d(in_channels=17, out_channels=64, kernel_size=3, padding=1, bias=False)

$$Y_{c_{out},i,j} = \sum_{c=0}^{c_{in}-1} \sum_{m=0}^{k_h-1} \sum_{n=0}^{k_w-1} X_{c,i+m,j+n} K_{c_{out},c,m,n}$$

64 filters of 3×3 across 128×128 image in 17 layers

Output (16, 64, 128, 128)

16 patches in 1 batch; 64 feature maps; 256×256

BatchNorm2D nn.BatchNorm2d(64)

$$\hat{X}_c = \frac{X_c - \mu_c}{\sqrt{\sigma_c^2 + \epsilon}}$$

For each of 64 filters, calculate μ (mean) & σ^2 (variance) across 16 patches

Output (16, 64, 128, 128)

ReLU nn.ReLU(inplace=True)

$$Y_c = \gamma_c \hat{X}_c + \beta_c$$

Every number passes through ReLU, only +ve numbers pass, -ve numbers → 0

Output (16, 64, 128, 128)

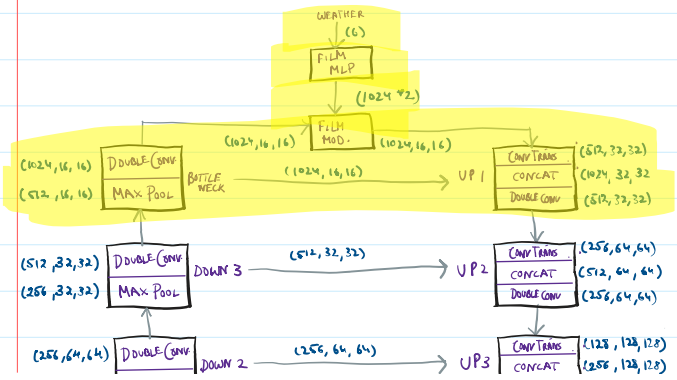
× 2 (DOUBLE CONV)

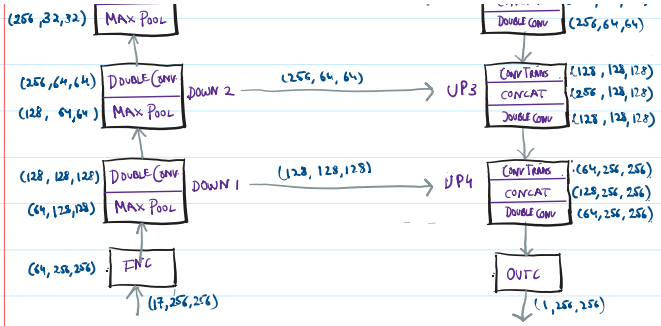
ENCODER BLOCK

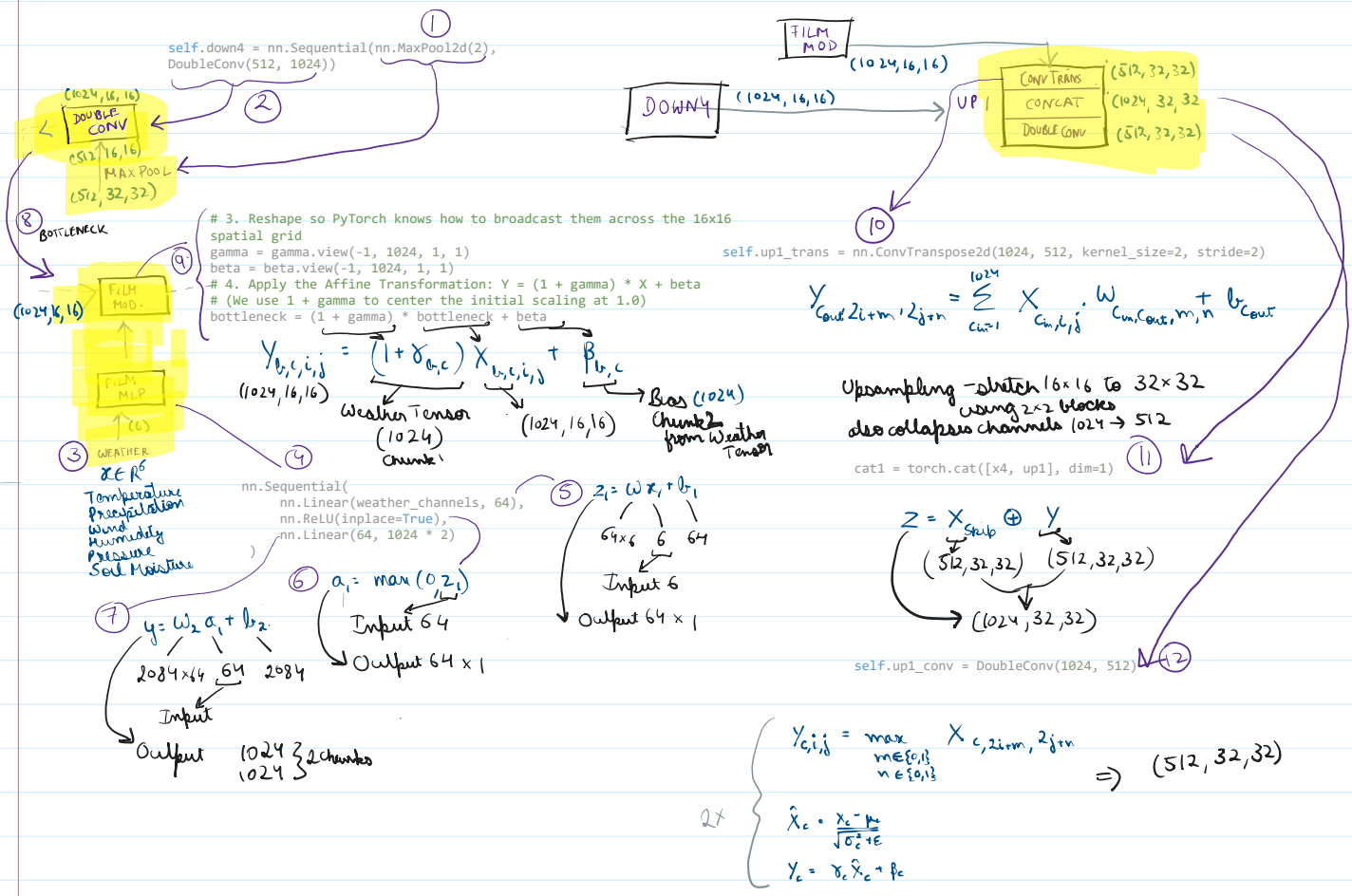
We're increasing features

& reducing the size of each feature patch

⇒ Granular feature maps




(256,32,32)
MAX POOL
(256,64,64)
DOUBLE CONV
(128, 64, 64)
MAX POOL
DOWN 2
(256, 64, 64)
(128, 128, 128)
DOUBLE CONV
(64, 128, 128)
MAX POOL
DOWN 1
(128, 128, 128)
(64, 256, 256)
INC
(17, 256, 256)
DOUBLE CONV
(256, 64, 64)
UP3
CONV TRANS
(128, 128, 128)
CONCAT
(256, 128, 128)
DOUBLE CONV
(128, 128, 128)
UP4
CONV TRANS
(64, 256, 256)
CONCAT
(128, 256, 256)
DOUBLE CONV
(64, 256, 256)
OUTC
(1, 256, 256)


self.down4 = nn.Sequential(nn.MaxPool2d(2), DoubleConv(512, 1024))
DOUBLE CONV
MAX POOL
(1024,16,16)
(512,16,16)
(512,32,32)
DOWN4
(1024,16,16)
FILM MOD
(1024,16,16)
UP1
CONV TRANS
CONCAT
DOUBLE CONV
(512,32,32)
(1024,32,32)
(512,32,32)
BOTTLENECK
FILM MOD.
FILM MLP
(6)
WEATHER
Temperature
Precipitation
Wind
Humidity
Pressure
Soil Moisture
# 3. Reshape so PyTorch knows how to broadcast them across the 16x16 spatial grid
gamma = gamma.view(-1, 1024, 1, 1)
beta = beta.view(-1, 1024, 1, 1)
# 4. Apply the Affine Transformation: Y = (1 + gamma) * X + beta
# (We use 1 + gamma to center the initial scaling at 1.0)
bottleneck = (1 + gamma) * bottleneck + beta
Weather Tensor (1024)
Chunk 1
Chunk 2 from Weather Tensor
Bias (1024)
(1024,16,16)
nn.Sequential(
nn.Linear(weather_channels, 64),
nn.ReLU(inplace=True),
nn.Linear(64, 1024 * 2)
)
Input 64
Output 64 x 1
Input 6
Output 64 x 1
2084x64 64 2084
Input
Output 1024 1024 2 chunks
self.up1_trans = nn.ConvTranspose2d(1024, 512, kernel_size=2, stride=2)
Upsampling - stretch 16x16 to 32x32 using 2x2 blocks also collapses channels 1024 → 512
cat1 = torch.cat([x4, up1], dim=1)
(512,32,32)
(512,32,32)
(1024,32,32)
self.up1_conv = DoubleConv(1024, 512)
=> (512,32,32)
2x