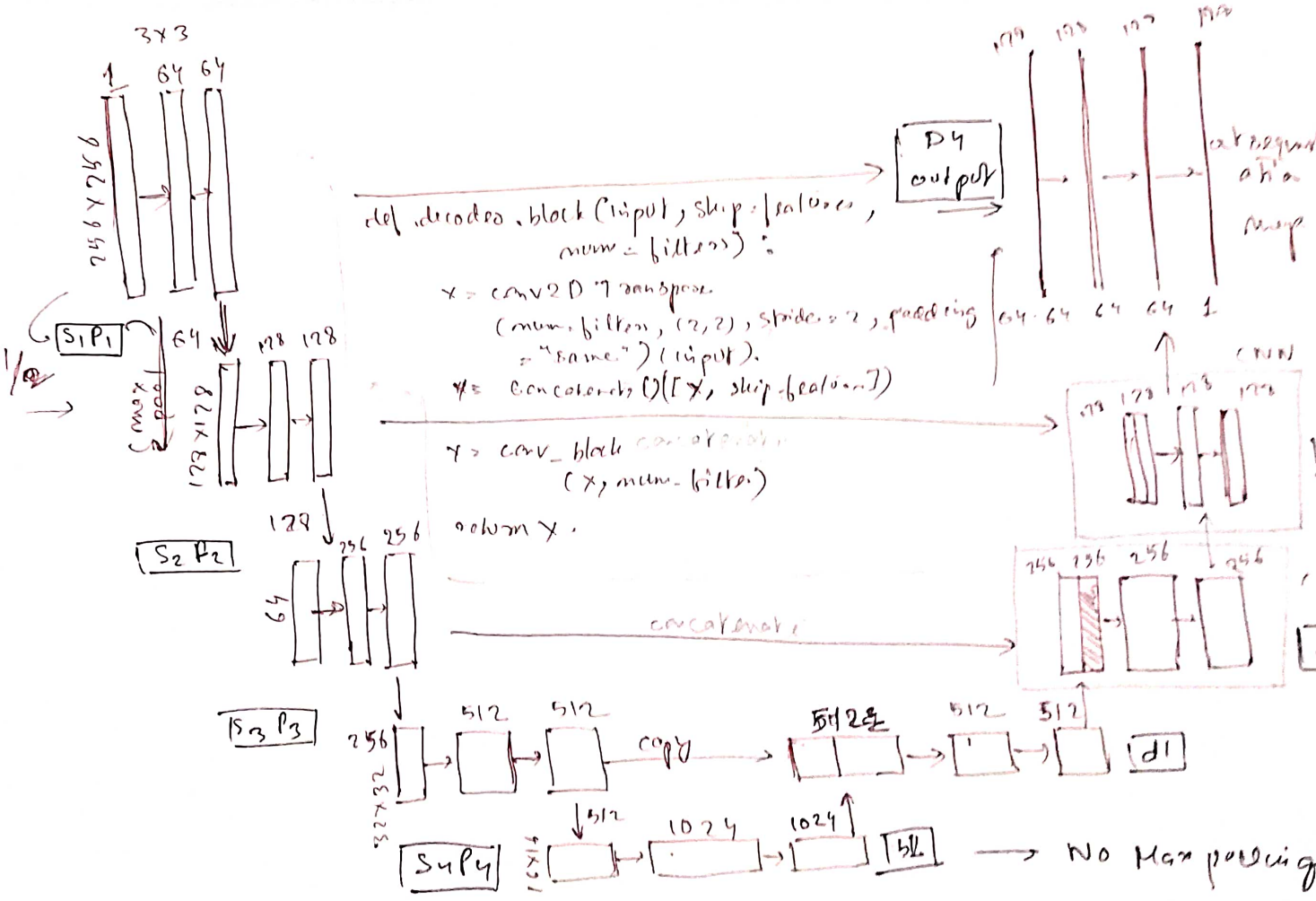
Putting together, UNET using functional API

for conv_encoder and decoder respectively.

```
inputs = Input(input_shape)
s1, p1 = encoder_block(inputs, 64)
s2, p2 = encoder_block(p1, 128)
s3, p3 = encoder_block(p2, 256)
s4, p4 = encoder_block(p3, 512)

b1 = conv_block(p4, 1024)
d1 = decoder_block(b1, s4, 512)
d2 = decoder_block(d1, s3, 256)
d3 = decoder_block(d2, s2, 128)
d4 = decoder_block(d3, s1, 64)
```

outputs → Conv2D(1, 1, padding="same", activation="sigmoid")(d4)

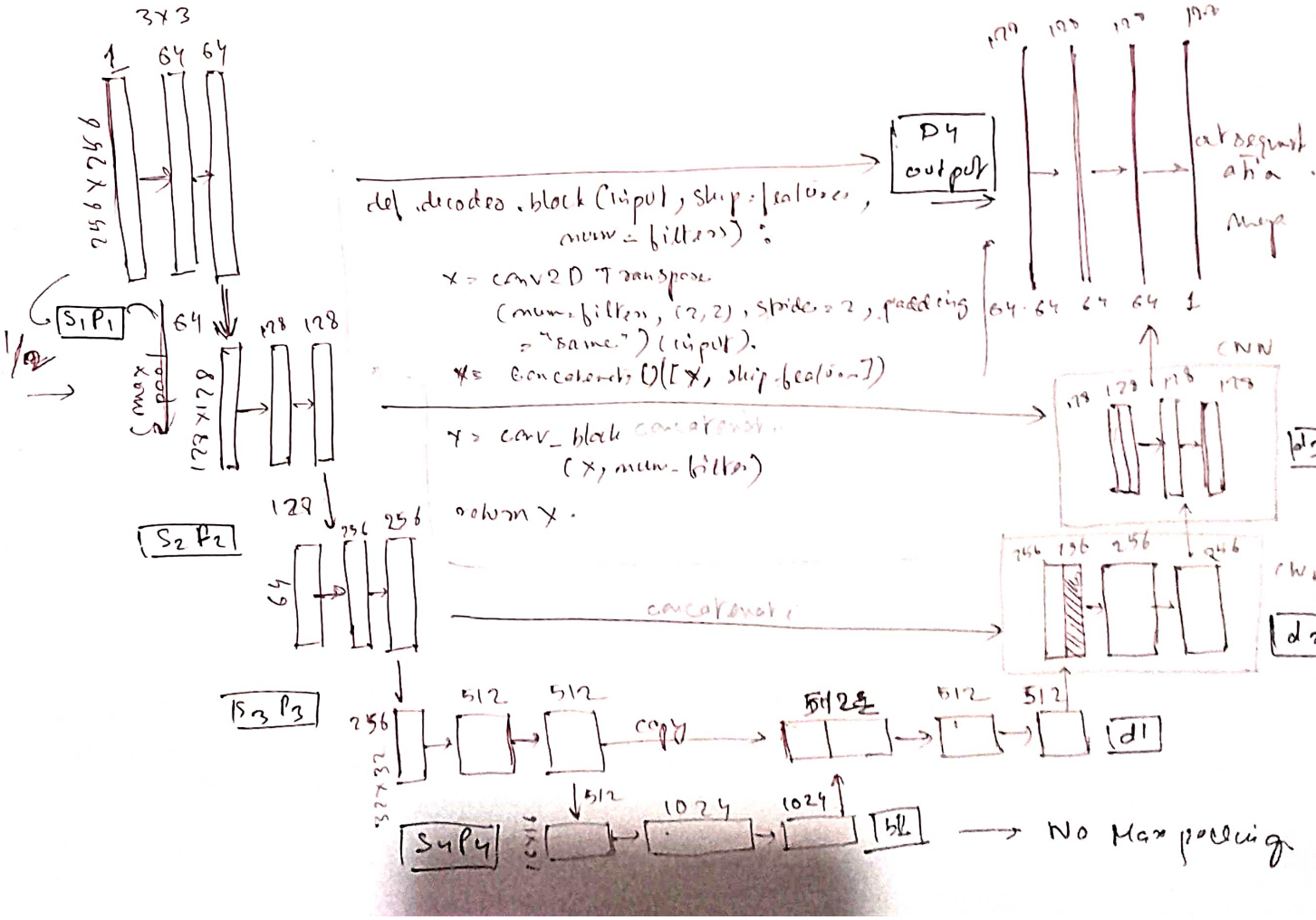


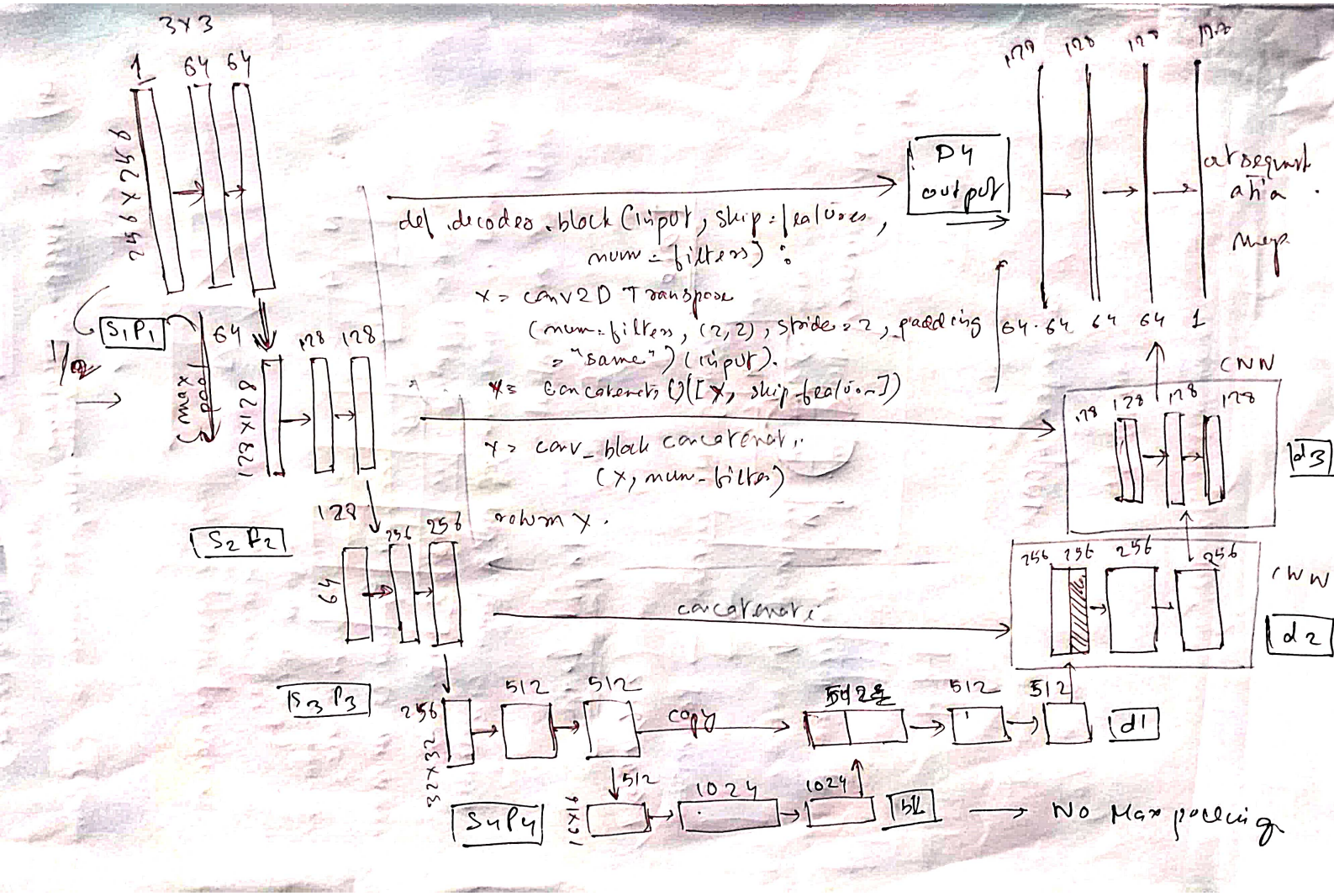
Putting together, UNET using functional code blocks for conv, encoder and decoder respectively.

```
Inputs = Input(input_shape)
s1, p1 = encoder_block(inputs, 64)
s2, p2 = encoder_block(p1, 128)
s3, p3 = encoder_block(p2, 256)
s4, p4 = encoder_block(p3, 512)

b1 = conv_block(p4, 1024)
d1 = decoder_block(b1, s4, 512)
d2 = decoder_block(d1, s3, 256)
d3 = decoder_block(d2, s2, 128)
d4 = decoder_block(d3, s1, 64).
```

output → Conv2D(1, 1, padding="same", activation="sigmoid")(d4)




3x3
1
64
64
256x256
S1 P1
max pool
64
128 128
128x128
S2 P2
128
256
256
64
S3 P3
256
512
512
32x32
copy
S4 P4
512
1024
1024
B L
No Max pooling
def decoder_block (input, skip_features, num_filters):
x = Conv2D Transpose (num_filters, (2,2), strides = 2, padding = "same") (input).
x = Concatenate()([x, skip_features])
x = conv_block concatenate (x, num_filters)
return x.
D4 output
128 128 128 128
64 64 64 64 1
activation map
CNN
d3
256 256 256 256
CNN
d2
concatenate
512 512
d1

```
inputs = Input(input_shape)

s1, p1 = encoder_block(inputs, 64)

s2, p2 = encoder_block(p1, 128)

s3, p3 = encoder_block(p2, 256)

s4, p4 = encoder_block(p3, 512)

b1 = conv_block(p4, 1024)

d1 = decoder_block(b1, s4, 512)

d2 = decoder_block(d1, s3, 256)

d3 = decoder_block(d2, s2, 128)

d4 = decoder_block(d3, s1, 64).
```

output → Conv2D(1, 1, padding="same", activation="sigmoid")(d4)