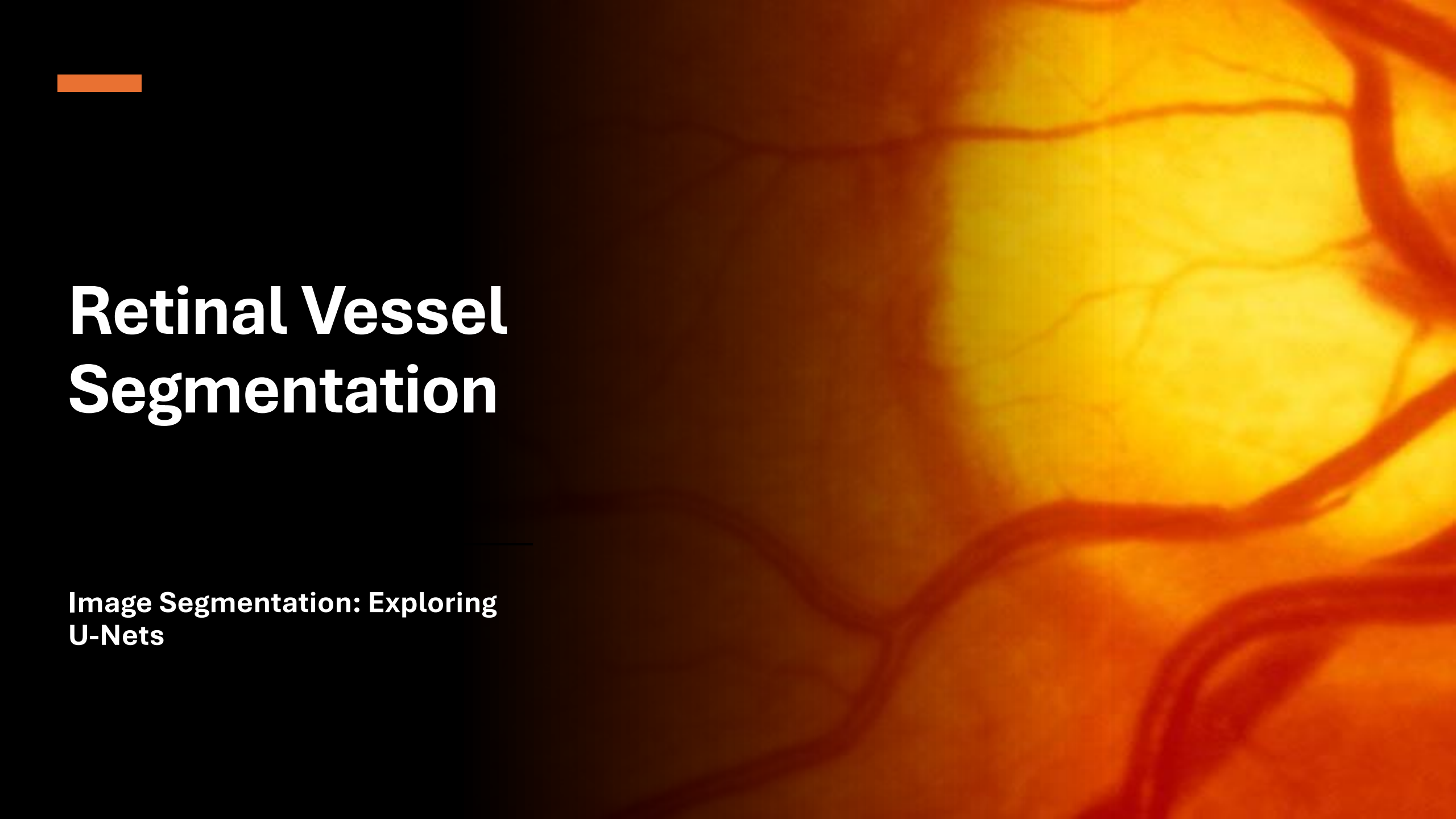
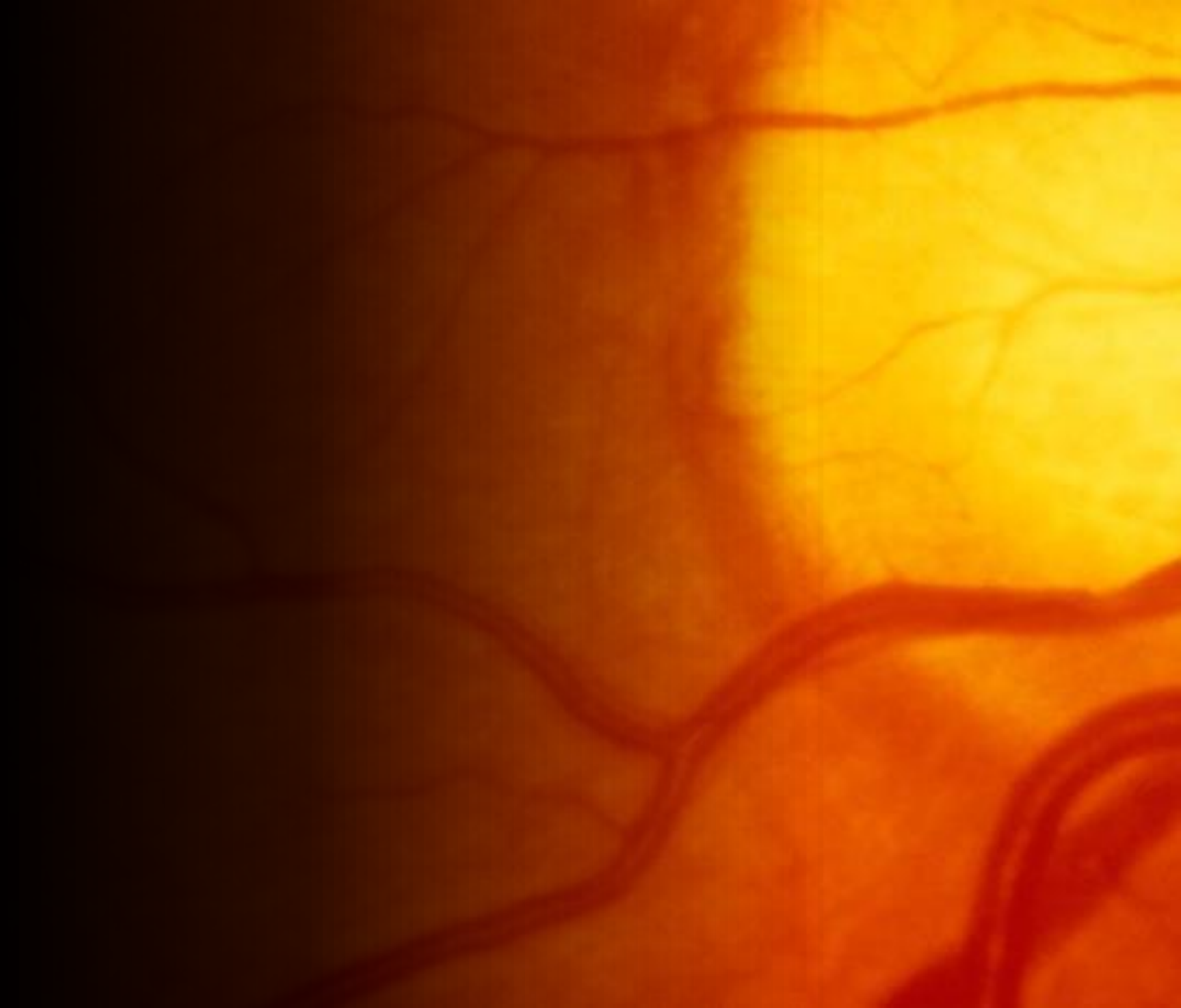
Retinal Vessel Segmentation

Image Segmentation: Exploring U-Nets



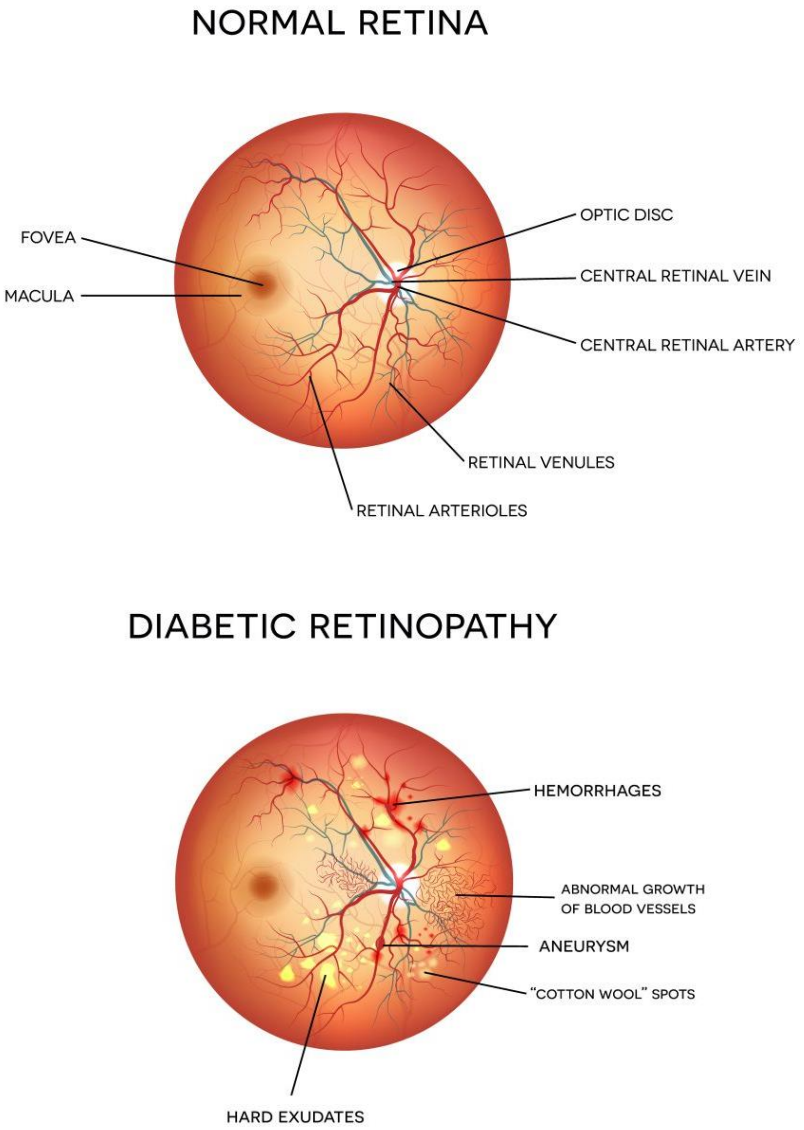
Content

- DRIVE dataset
- UNet applications
- Architectures and findings
- Final results and visualizations



Why?

- Early detection of diseases
  - diabetic retinopathy, hypertension, …
- Assesment overall vascular health
- Non-invasive
  - Reduced risk of complications compared to surgical methods
- Cost-effective
  - Reduced cost compared to surgical methods

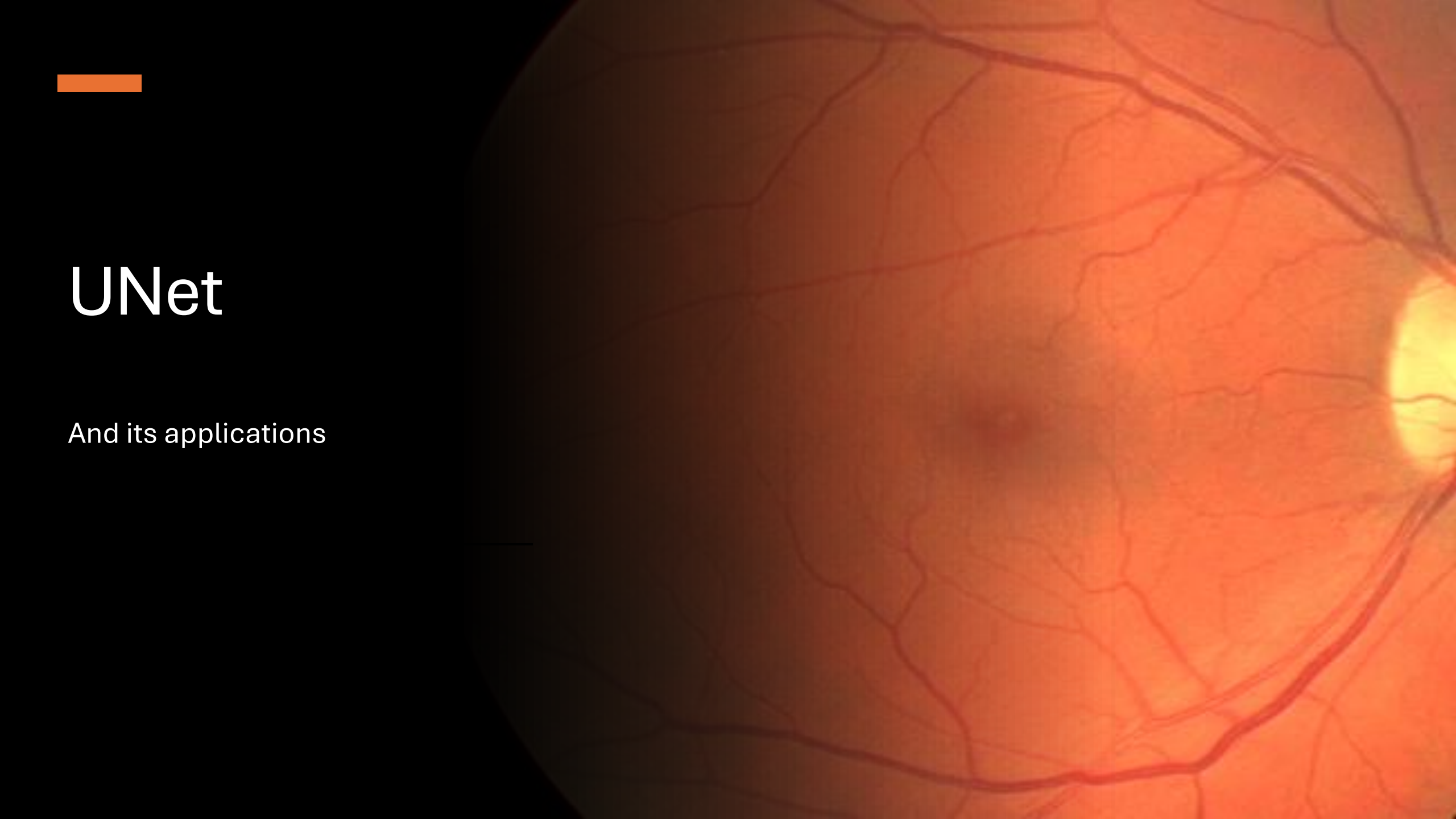


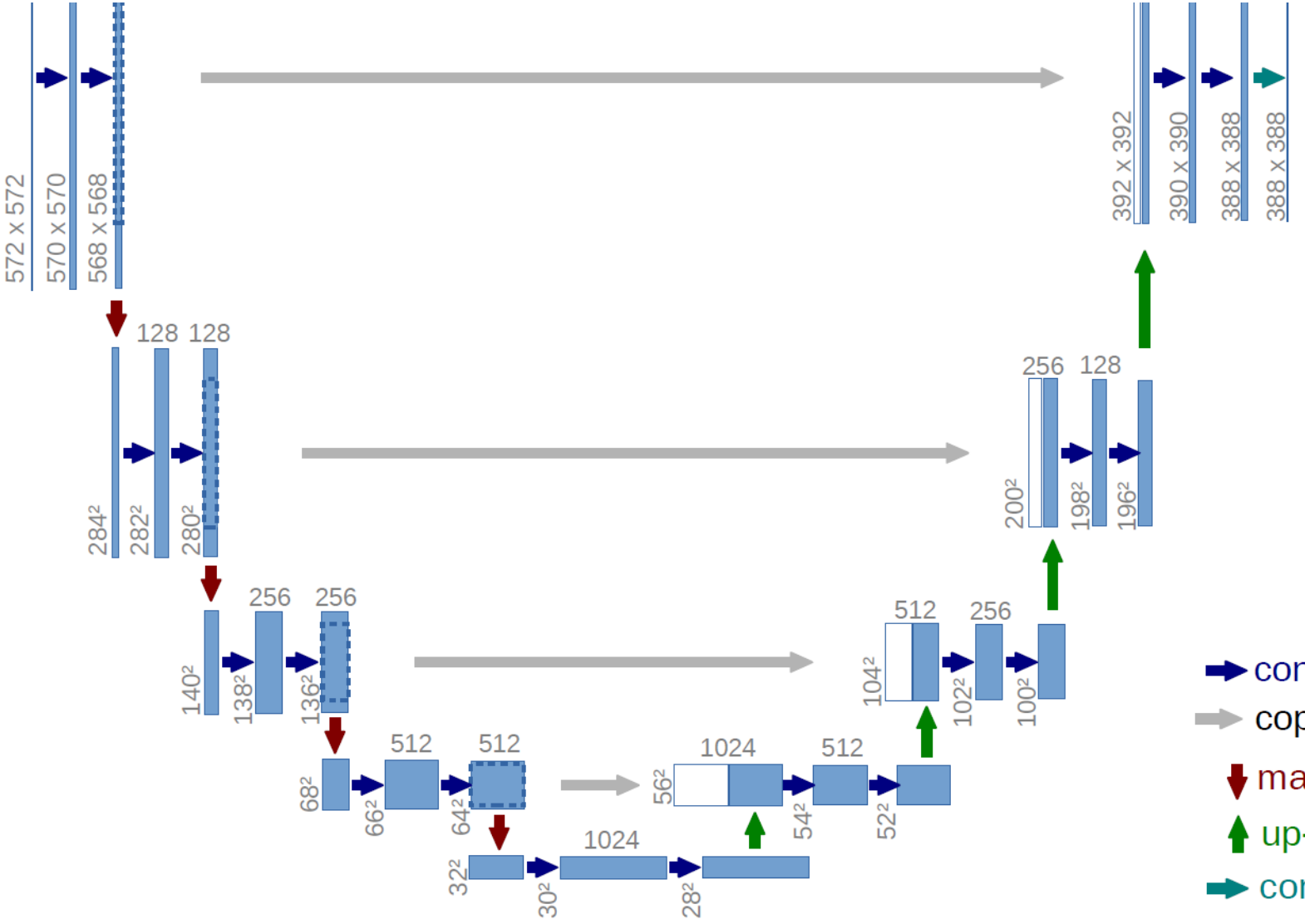
- Goal:
  - Segmentation of retinal blood vessels

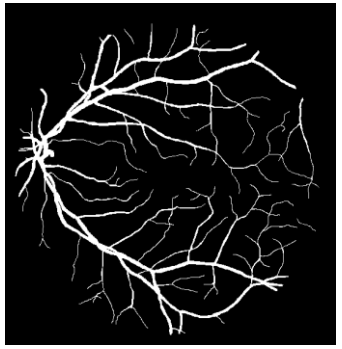
- Challenges:
  - Only 20 labeled images
  - Highly imbalanced data
  - Thick vessels <-> fine vessels
  - Noise

UNet

And its applications




572 x 572
570 x 570
568 x 568
128
128
284²
282²
280²
256
256
140²
138²
136²
512
512
68²
66²
64²
1024
32²
30²
28²
1024
512
56²
54²
52²
512
256
104²
102²
100²
256
128
200²
198²
196²
392 x 392
390 x 390
388 x 388
388 x 388
conv 3x3, ReLU
copy and crop
max pool 2x2
up-conv 2x2
conv 1x1



U-Net applications

Image Segmentation

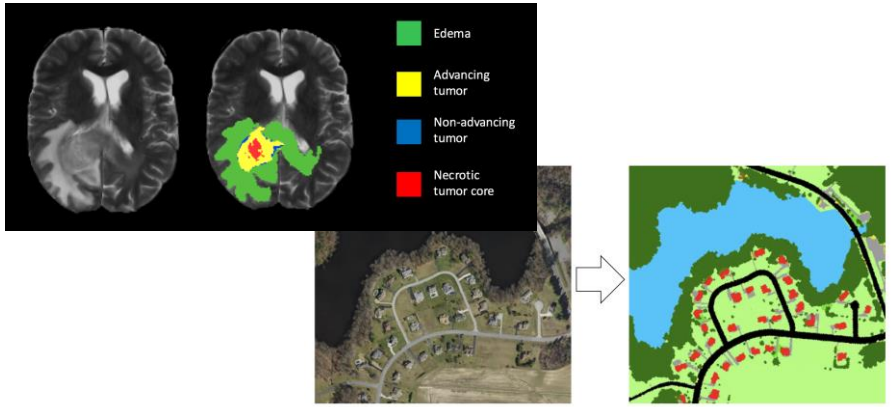


Image Denoising

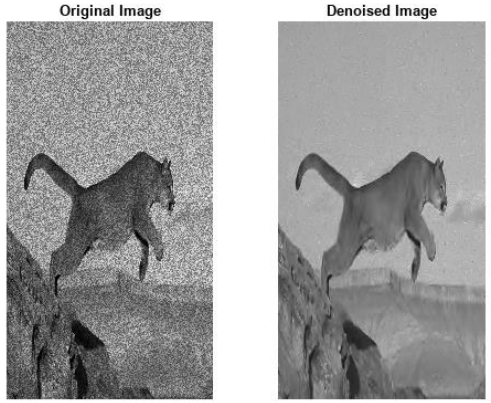


Image Upscaling

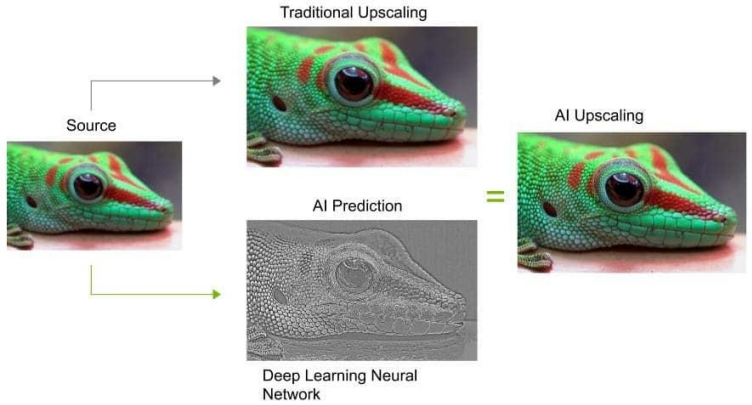


Image Generation

Data augmentation

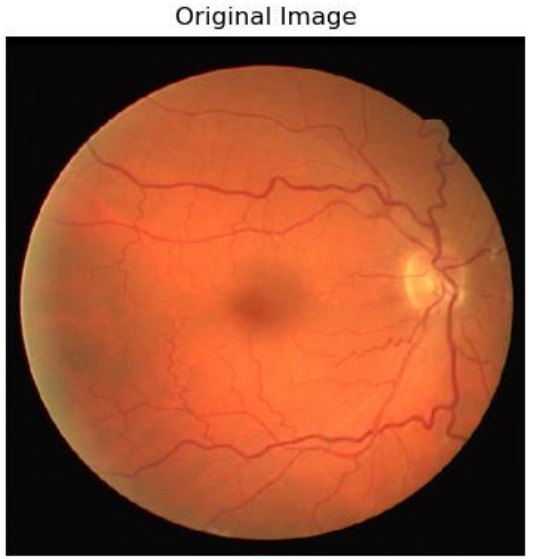
Increased dataset diversity
Prevents overfitting
Enhanced feature learning

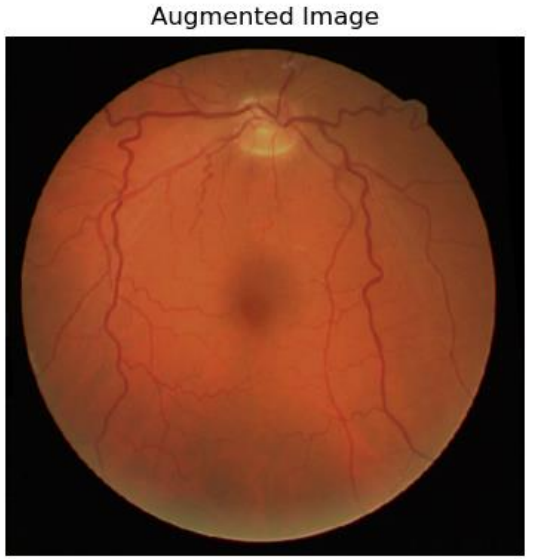
Horizontal Flip

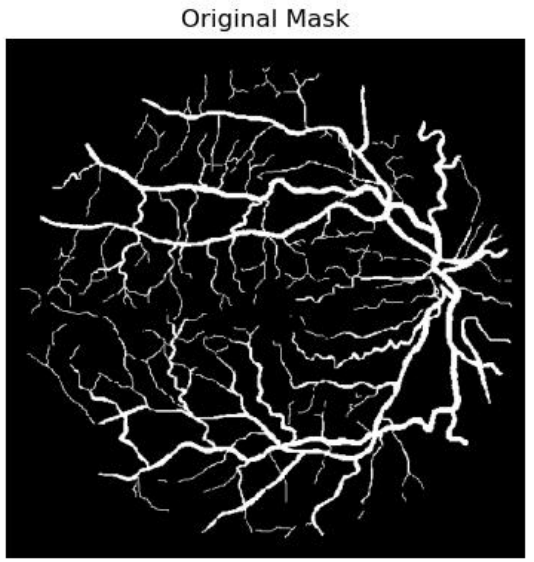
Vertical Flip

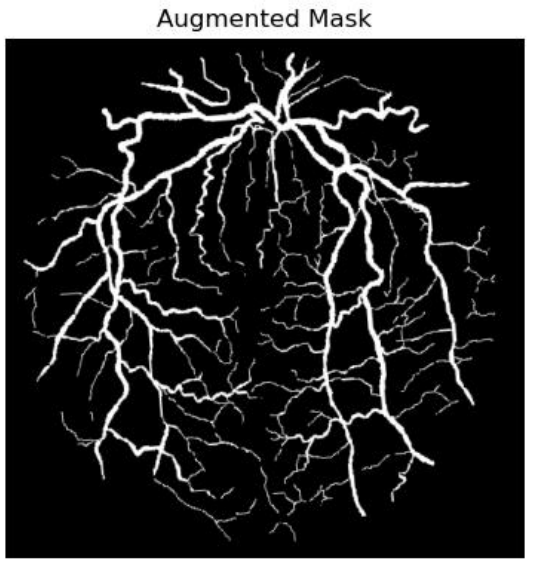
Rotation

Brightness Adjustment







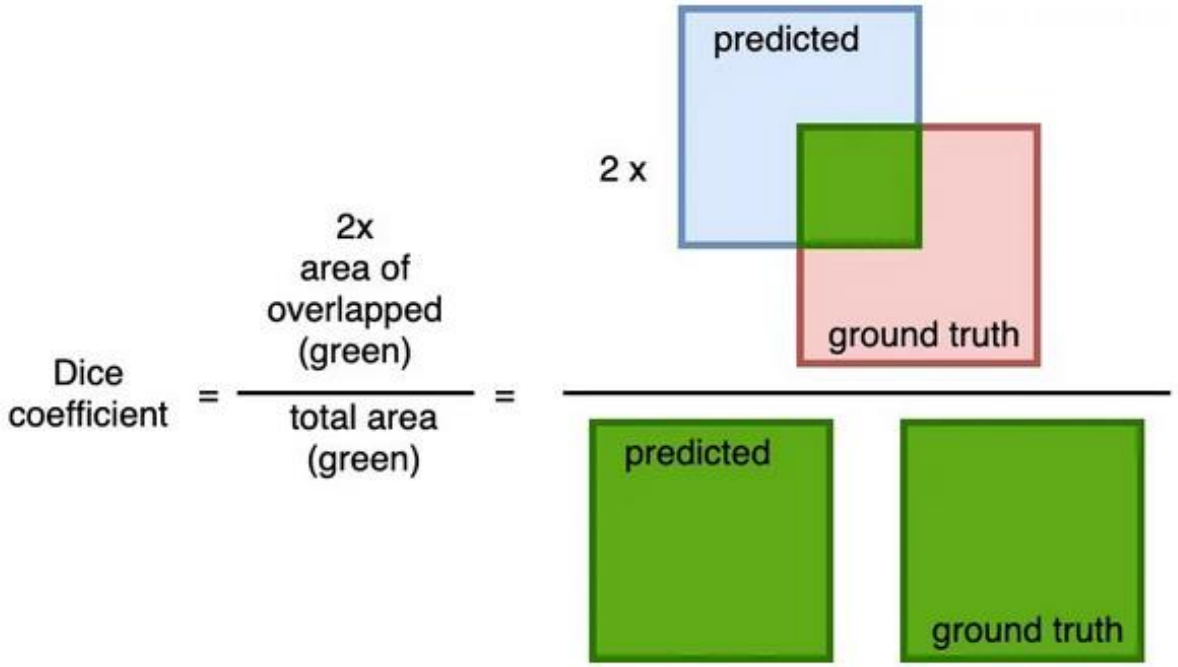


DICE loss

Solves imbalance problem by increasing weight of overlap

(high imbalance typical for medical imaging)

Quantification of the similarity between the two sets

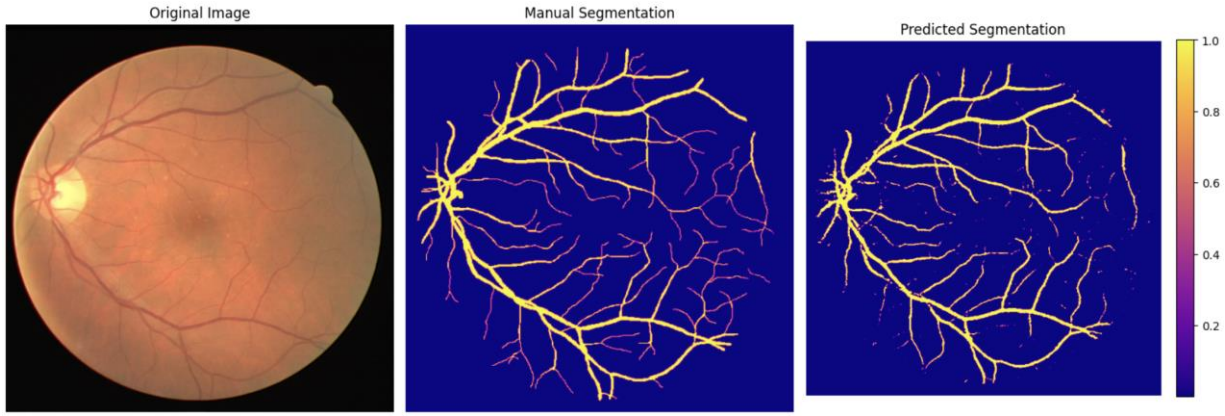


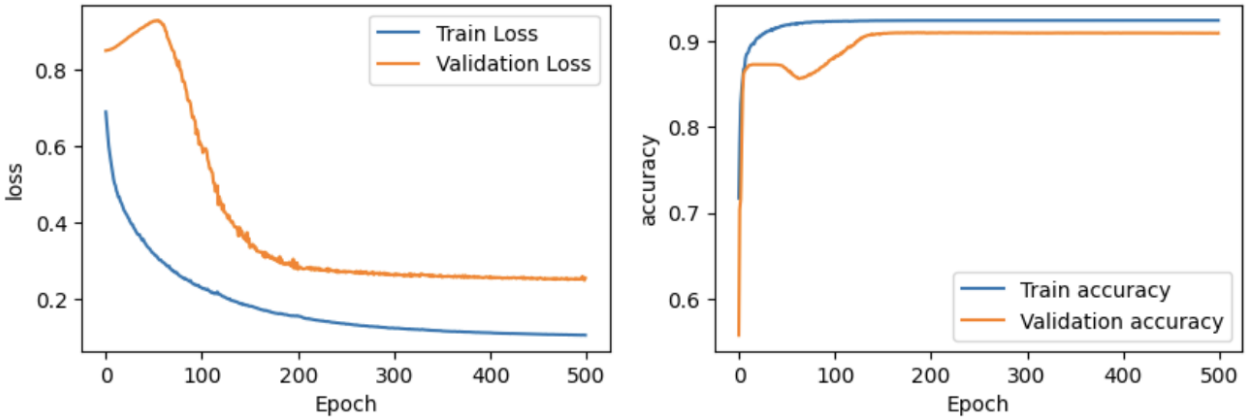
Dice loss = 1 – Dice coefficient

U-NET: Baseline Results

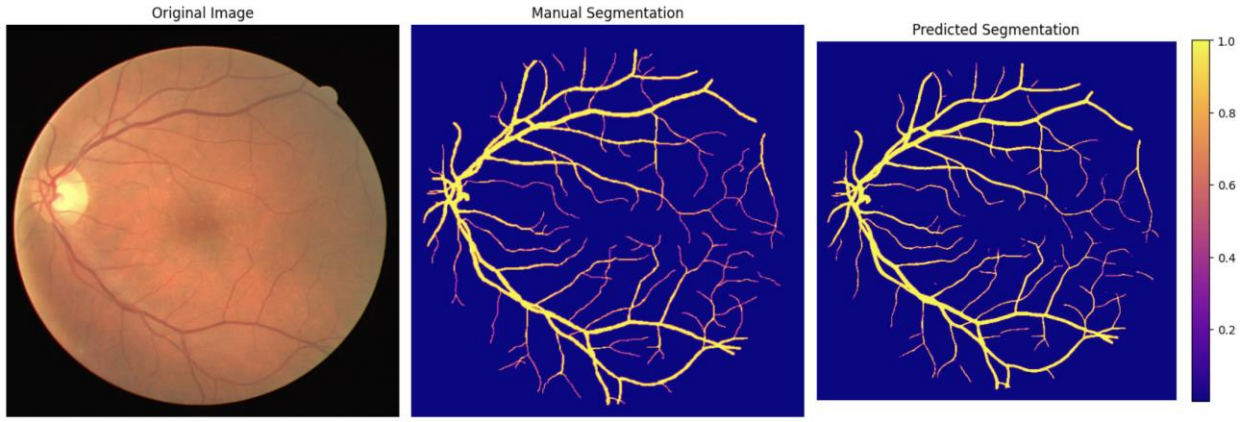
U-Net: 31,039,425 trainable parameters

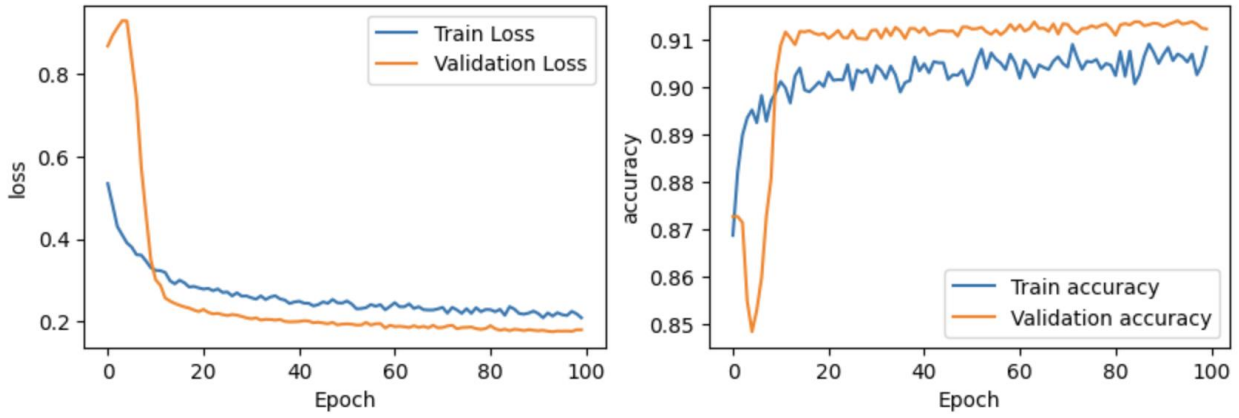
Original Dataset
- Validation DICE: 0,7499





Augmented Data
- Validation DICE: 0,8246



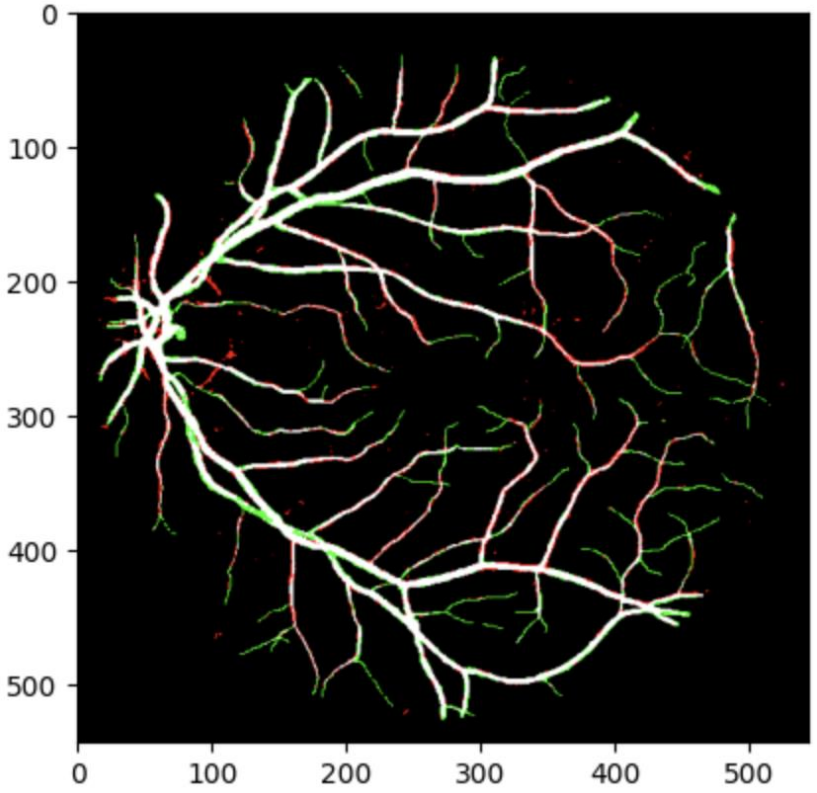


U-NET: Baseline Results

U-Net: 31,039,425 trainable parameters

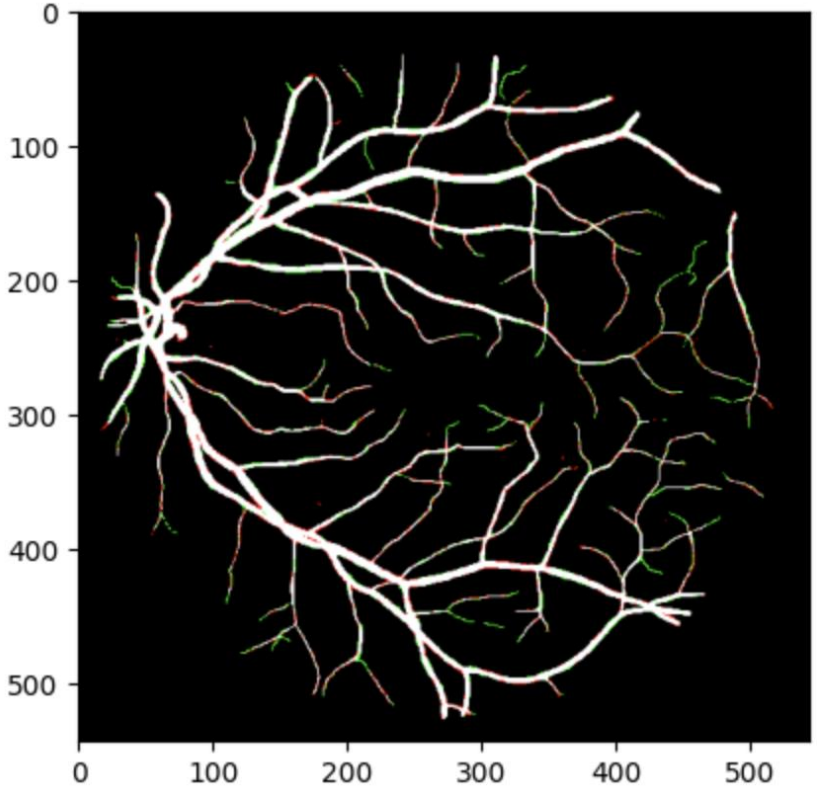
Original Dataset

- Validation DICE: 0,7499



Augmented Data

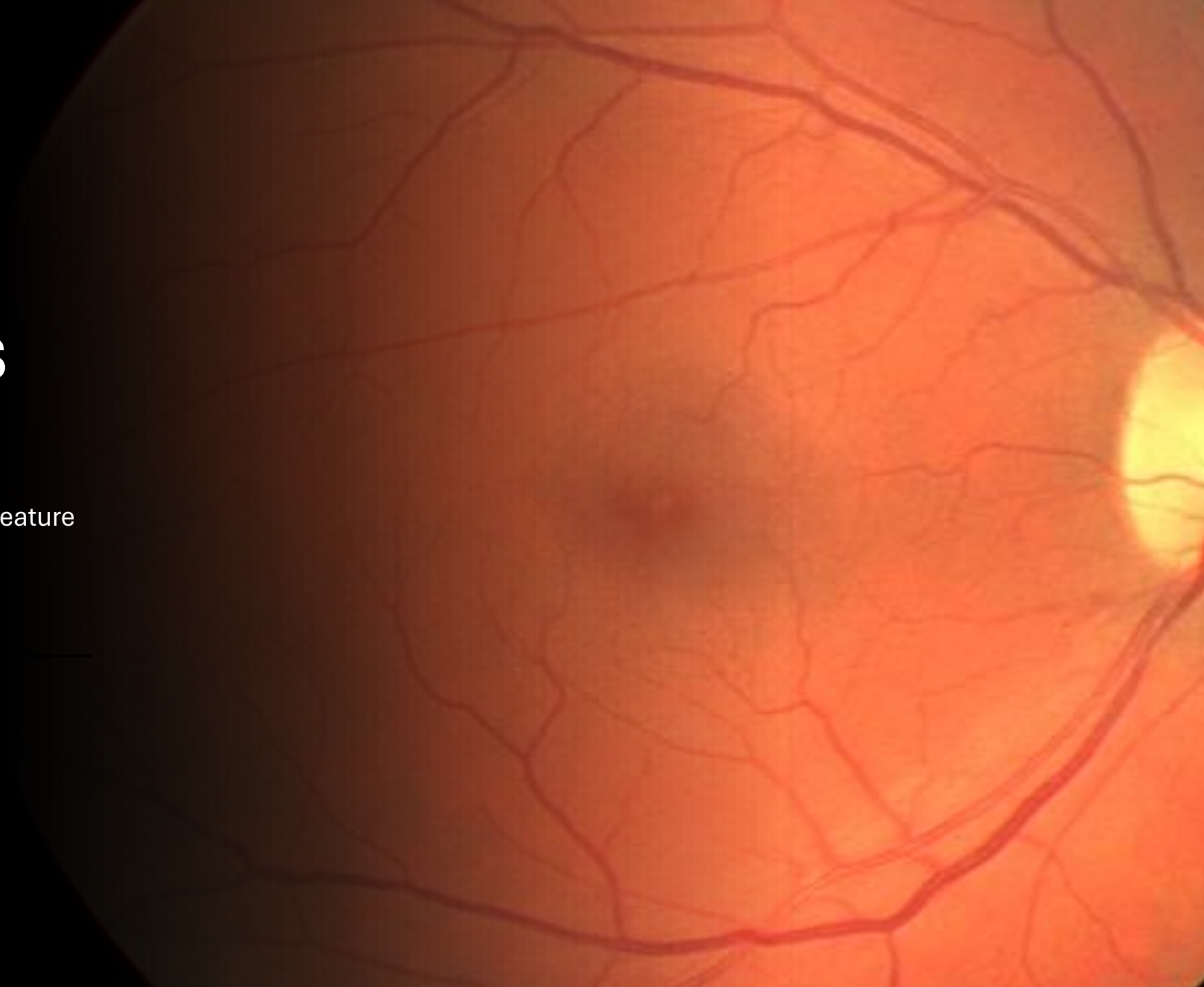
- Validation DICE: 0,8246



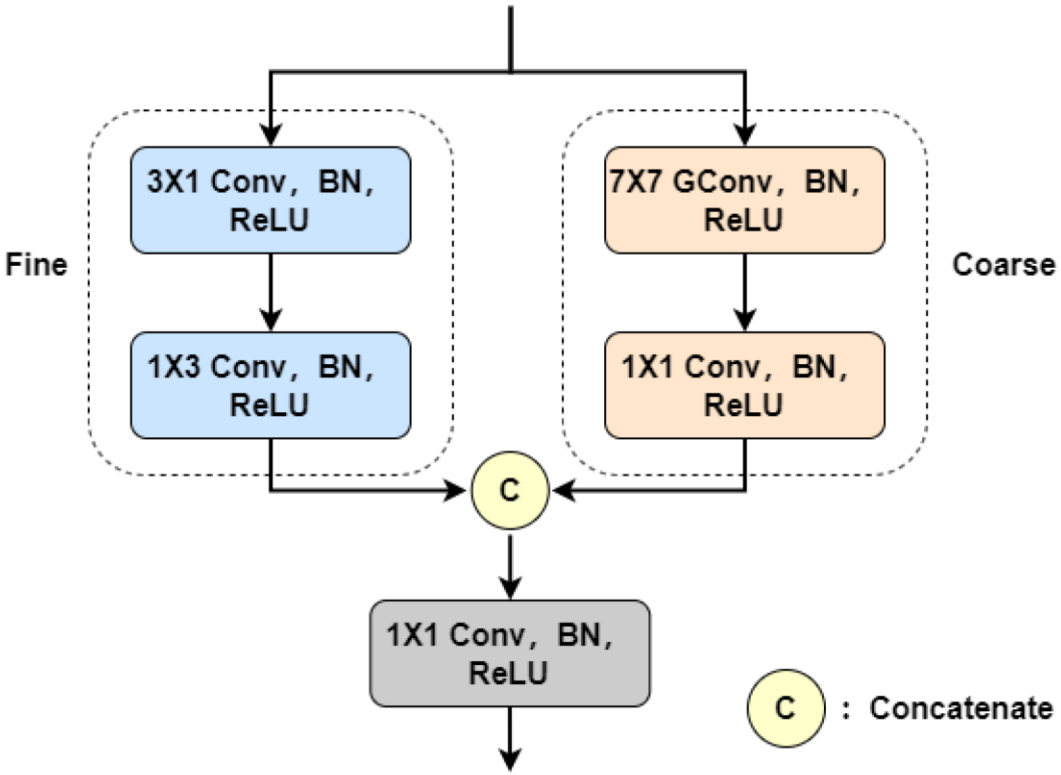
TP	
TN	
FP	
FN	

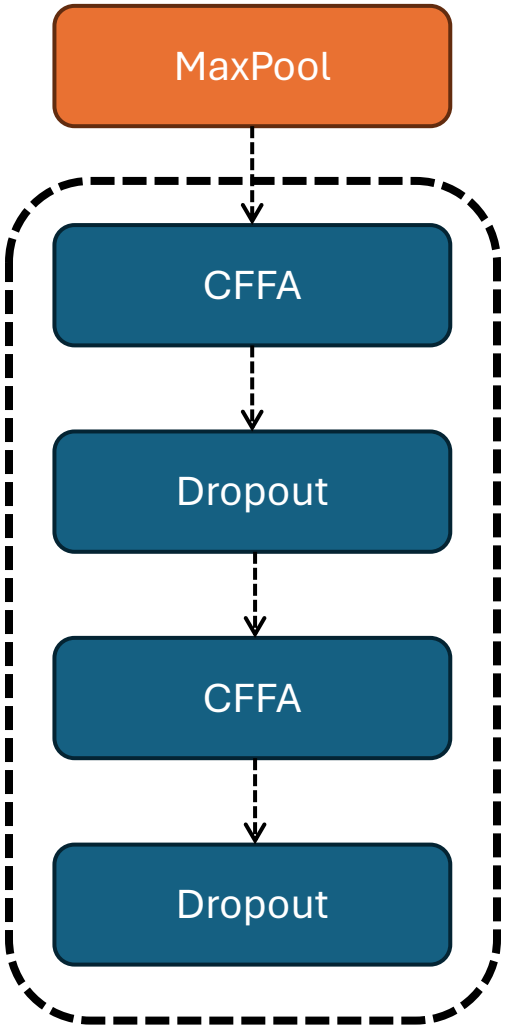
Advanced Architectures

- CBAM-UNet (With Coarse Fine Feature Aggregation)
- CMP-UNet
- Deformable UNet



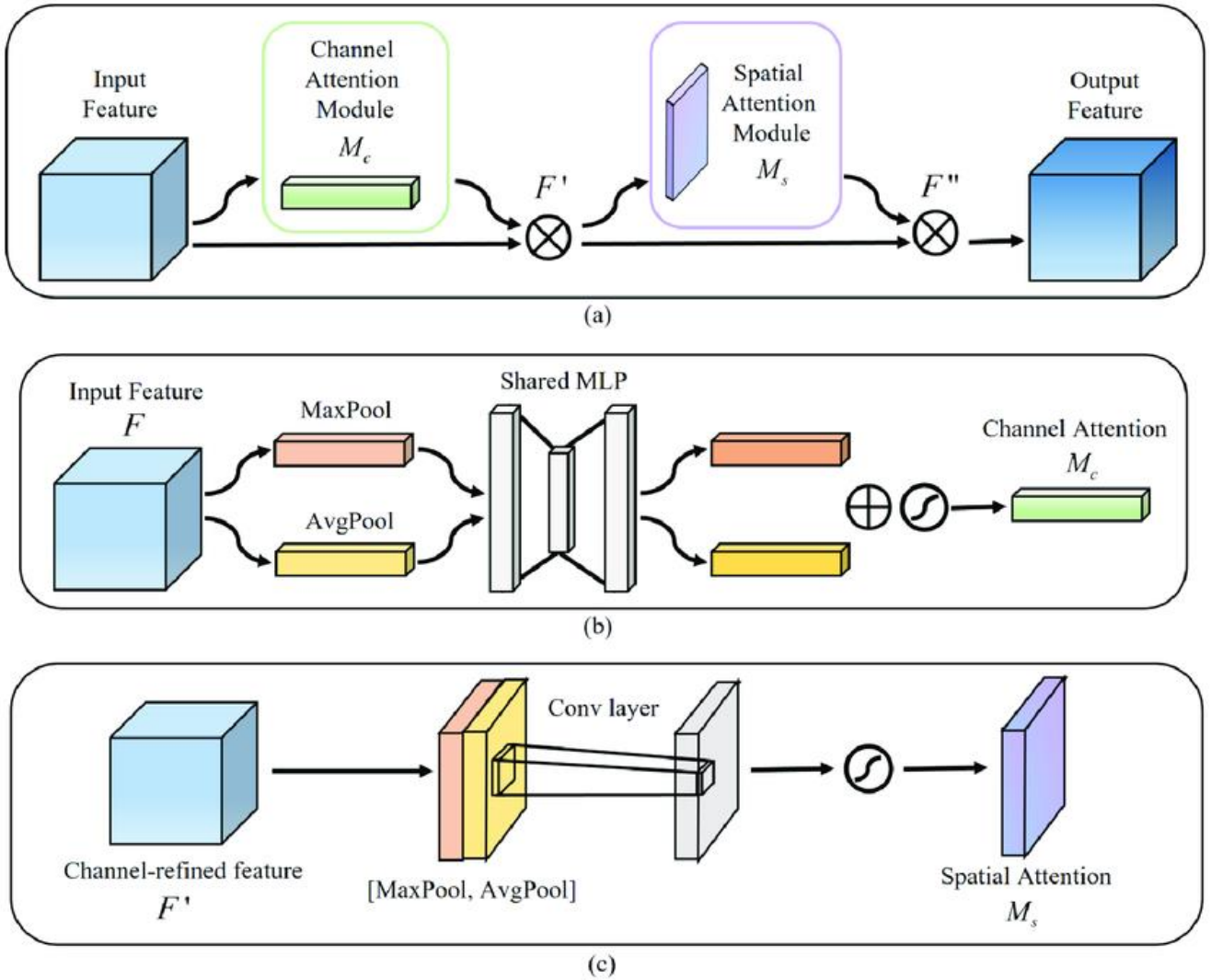
Down: CFFA


Fine
3X1 Conv, BN, ReLU
1X3 Conv, BN, ReLU
7X7 GConv, BN, ReLU
1X1 Conv, BN, ReLU
Coarse
C
1X1 Conv, BN, ReLU
C : Concatenate


MaxPool
CFFA
Dropout
CFFA
Dropout

```python
def CFFA(input, num_filters):
    fine = Conv2D(num_filters, (3, 1), padding="same")(input)
    fine = BatchNormalization()(fine)
    fine = Activation("relu")(fine)

    fine = Conv2D(num_filters, (1, 3), padding="same")(fine)
    fine = BatchNormalization()(fine)
    fine = Activation("relu")(fine)


    coarse = Conv2D(num_filters, 7, padding="same")(input)
    coarse = BatchNormalization()(coarse)
    coarse = Activation("relu")(coarse)

    coarse = Conv2D(num_filters, 1, padding="same")(coarse)
    coarse = BatchNormalization()(coarse)
    coarse = Activation("relu")(coarse)

    x = Concatenate()([fine, coarse])
    x = Conv2D(num_filters, 1, padding="same")(x)
    x = BatchNormalization()(x)
    x = Activation("relu")(x)
    return x
```

Bridge: Convolutional Block Attention Module

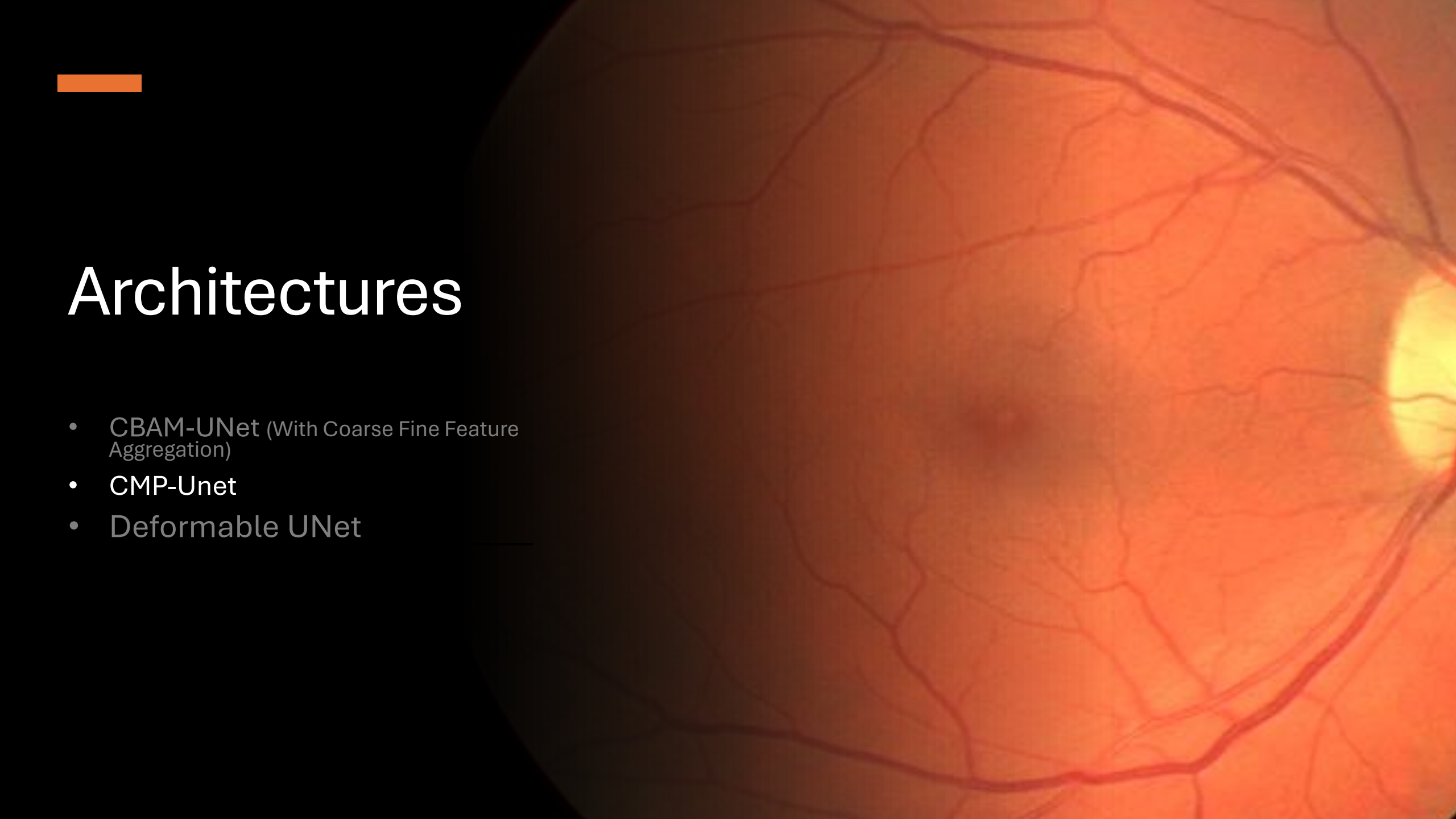


```python
def channel_attention(input_feature, ratio=8):
    channel = input_feature.shape[-1]
    shared_layer_one = Dense(channel//ratio,
                             activation='relu',
                             kernel_initializer='he_normal',
                             use_bias=True,
                             bias_initializer='zeros')
    shared_layer_two = Dense(channel,
                             kernel_initializer='he_normal',
                             use_bias=True,
                             bias_initializer='zeros')
    avg_pool = GlobalAveragePooling2D()(input_feature)
    avg_pool = Reshape((1,1,channel))(avg_pool)
    assert avg_pool.shape[1:] == (1,1,channel)
    avg_pool = shared_layer_one(avg_pool)
    assert avg_pool.shape[1:] == (1,1,channel//ratio)
    avg_pool = shared_layer_two(avg_pool)
    assert avg_pool.shape[1:] == (1,1,channel)

    max_pool = GlobalMaxPooling2D()(input_feature)
    max_pool = Reshape((1,1,channel))(max_pool)
    assert max_pool.shape[1:] == (1,1,channel)
    max_pool = shared_layer_one(max_pool)
    assert max_pool.shape[1:] == (1,1,channel//ratio)
    max_pool = shared_layer_two(max_pool)
    assert max_pool.shape[1:] == (1,1,channel)

    cbam_feature = Add()([avg_pool,max_pool])
    cbam_feature = Activation('sigmoid')(cbam_feature)
    return Multiply()([input_feature, cbam_feature])
```

```python
def spatial_attention(input_feature):
    kernel_size = 7
    channel = input_feature.shape[-1]
    cbam_feature = input_feature
    output_shape = (input_feature.shape[1], input_feature.shape[2], 1)


    avg_pool = Lambda(lambda x: tf.keras.backend.mean(x, axis=3, keepdims=True), output_shape)(cbam_feature)
    assert avg_pool.shape[-1] == 1
    max_pool = Lambda(lambda x: tf.keras.backend.max(x, axis=3, keepdims=True), output_shape)(cbam_feature)
    assert max_pool.shape[-1] == 1
    concat = Concatenate(axis=3)([avg_pool, max_pool])
    assert concat.shape[-1] == 2
    cbam_feature = Conv2D(filters=1,
                          kernel_size=kernel_size,
                          strides=1,
                          padding='same',
                          activation='sigmoid',
                          kernel_initializer='he_normal',
                          use_bias=False)(concat)
    assert cbam_feature.shape[-1] == 1

    return Multiply()([input_feature, cbam_feature])
```

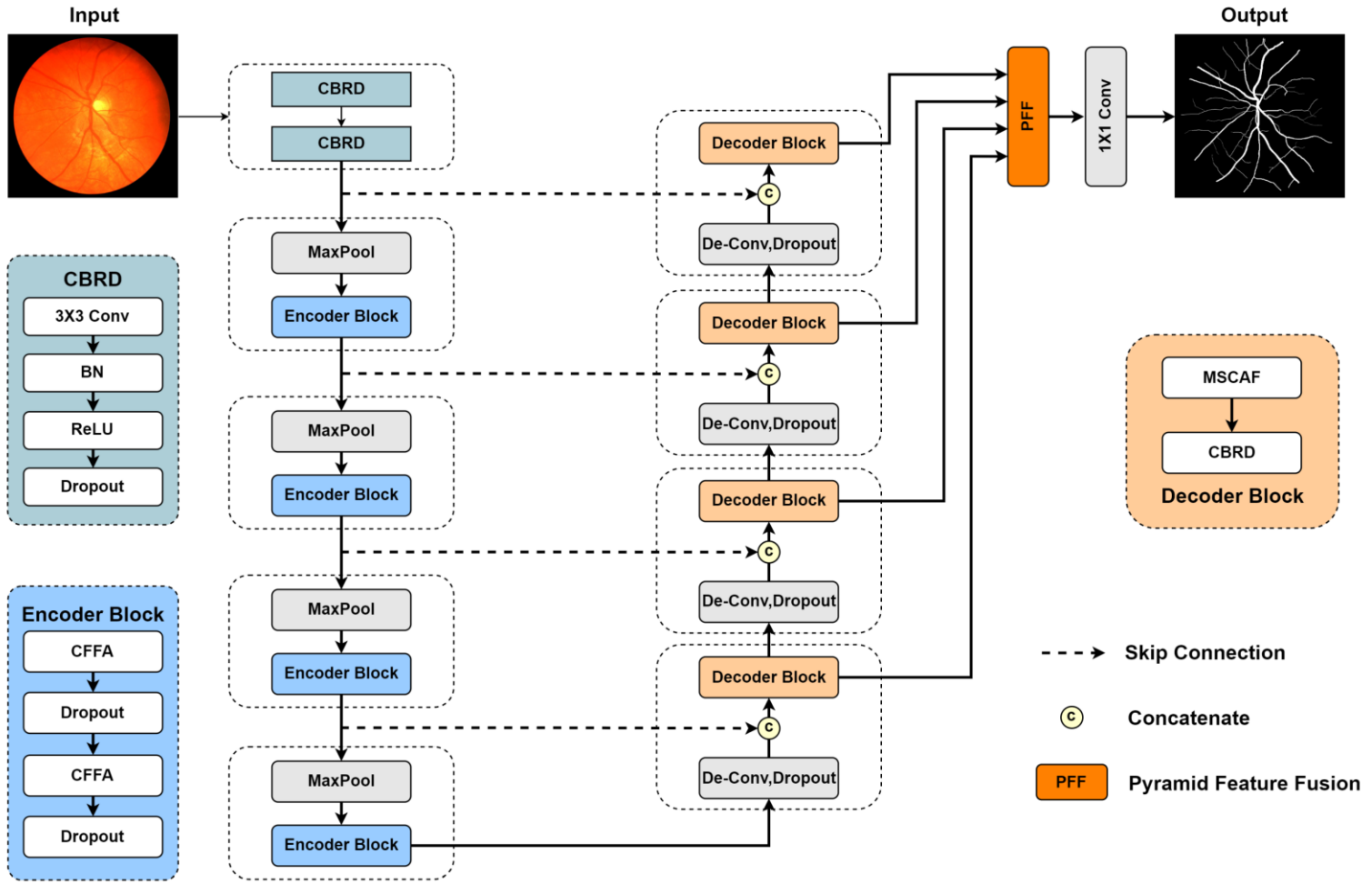
Architectures

- CBAM-UNet (With Coarse Fine Feature Aggregation)
- CMP-Unet
- Deformable UNet

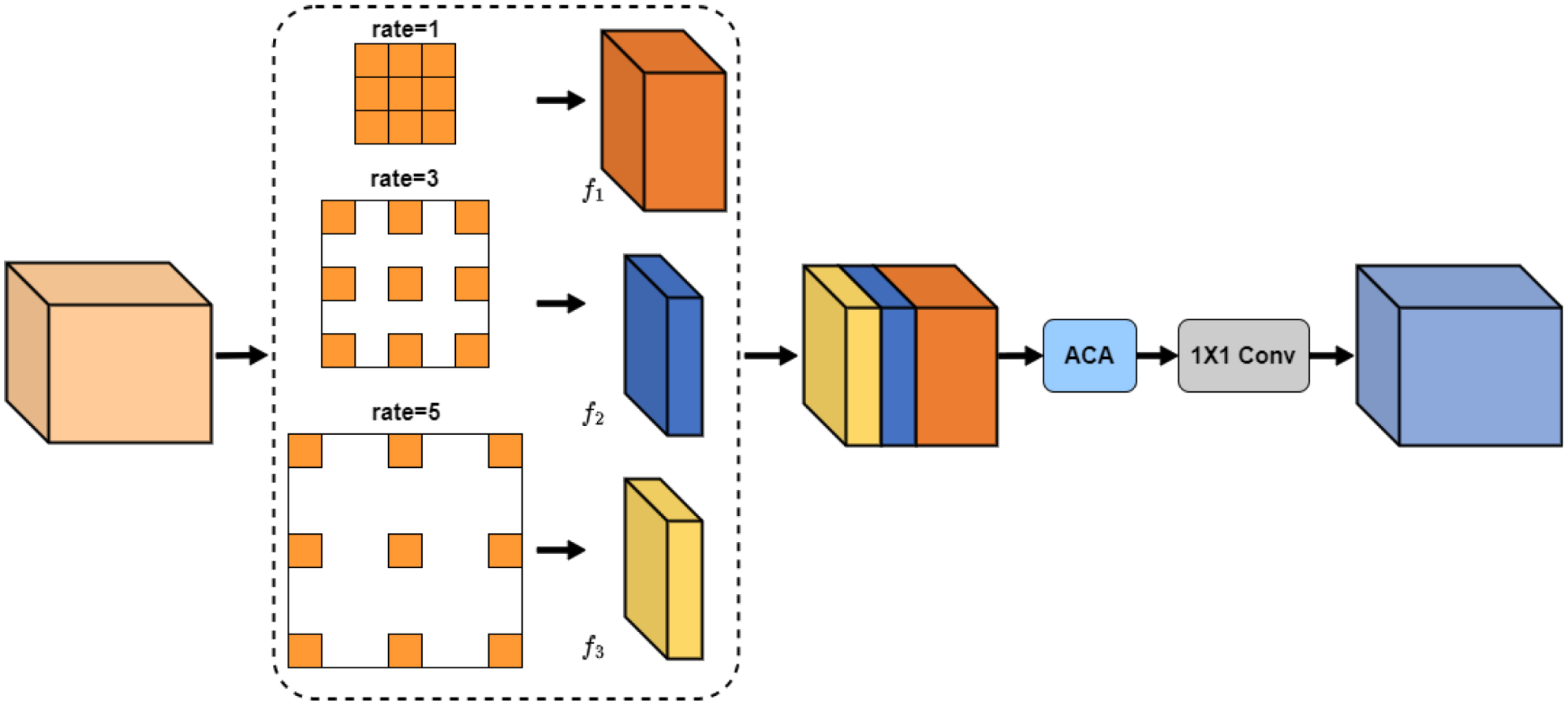


CMP-Unet

Gu Y., et al.

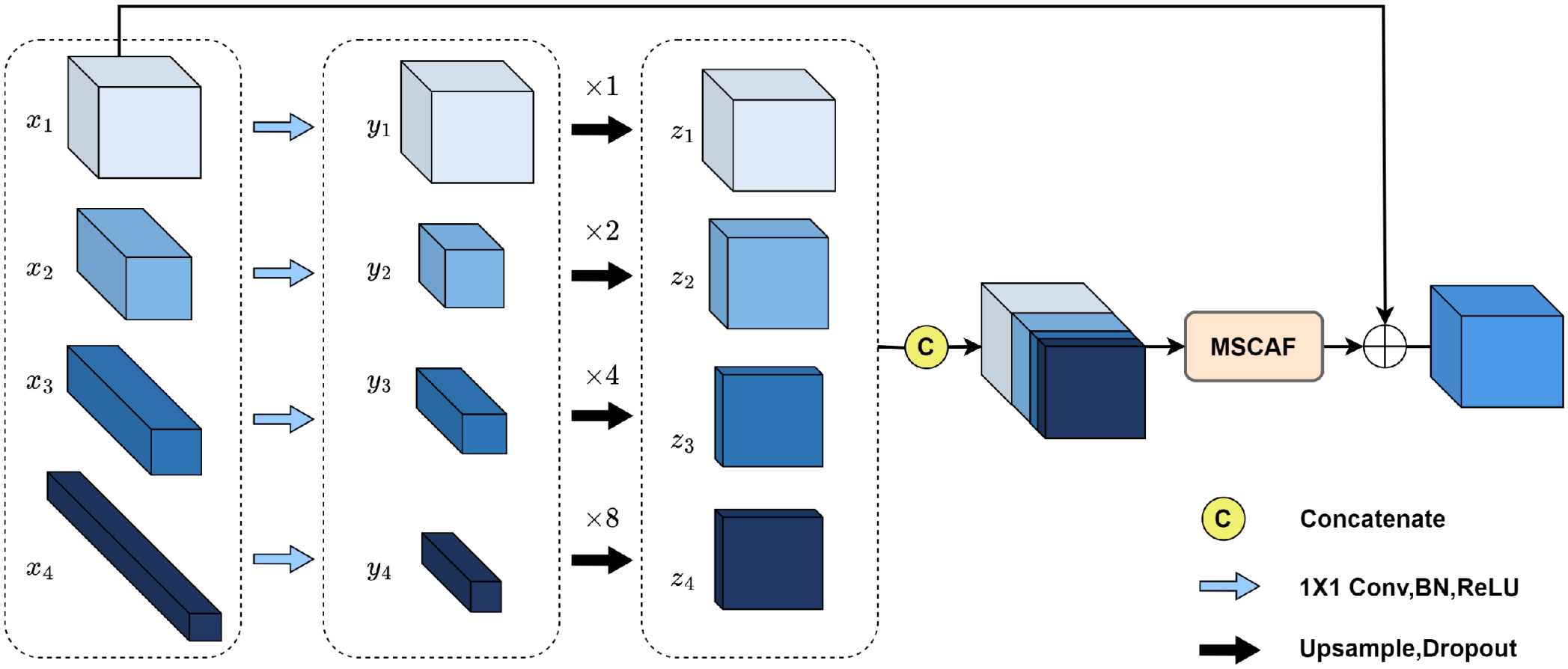


Multi-Scale Channel Adaptive Fusion



```python
def MSCAF(input, out_channels, dropout):
    x1 = Conv2D(filters=out_channels//2, kernel_size=3, padding='same', use_bias=False)(input)
    x1 = BatchNormalization()(x1)
    x1 = Activation("relu")(x1)

    x2 = Conv2D(filters=out_channels//4, kernel_size=3, dilation_rate=3, padding='same', use_bias=False)(input)
    x2 = BatchNormalization()(x2)
    x2 = Activation("relu")(x2)

    x3 = Conv2D(filters=out_channels//4, kernel_size=3, dilation_rate=5, padding='same', use_bias=False)(input)
    x3 = BatchNormalization()(x3)
    x3 = Activation("relu")(x3)

    out = Concatenate()([x1, x2, x3])
    out = channel_attention(out)
    out = Conv2D(out_channels, 1, use_bias = False)(out)
    out = BatchNormalization()(out)
    out = Activation("relu")(out)
    out = Dropout(dropout)(out)
    return(out)
```

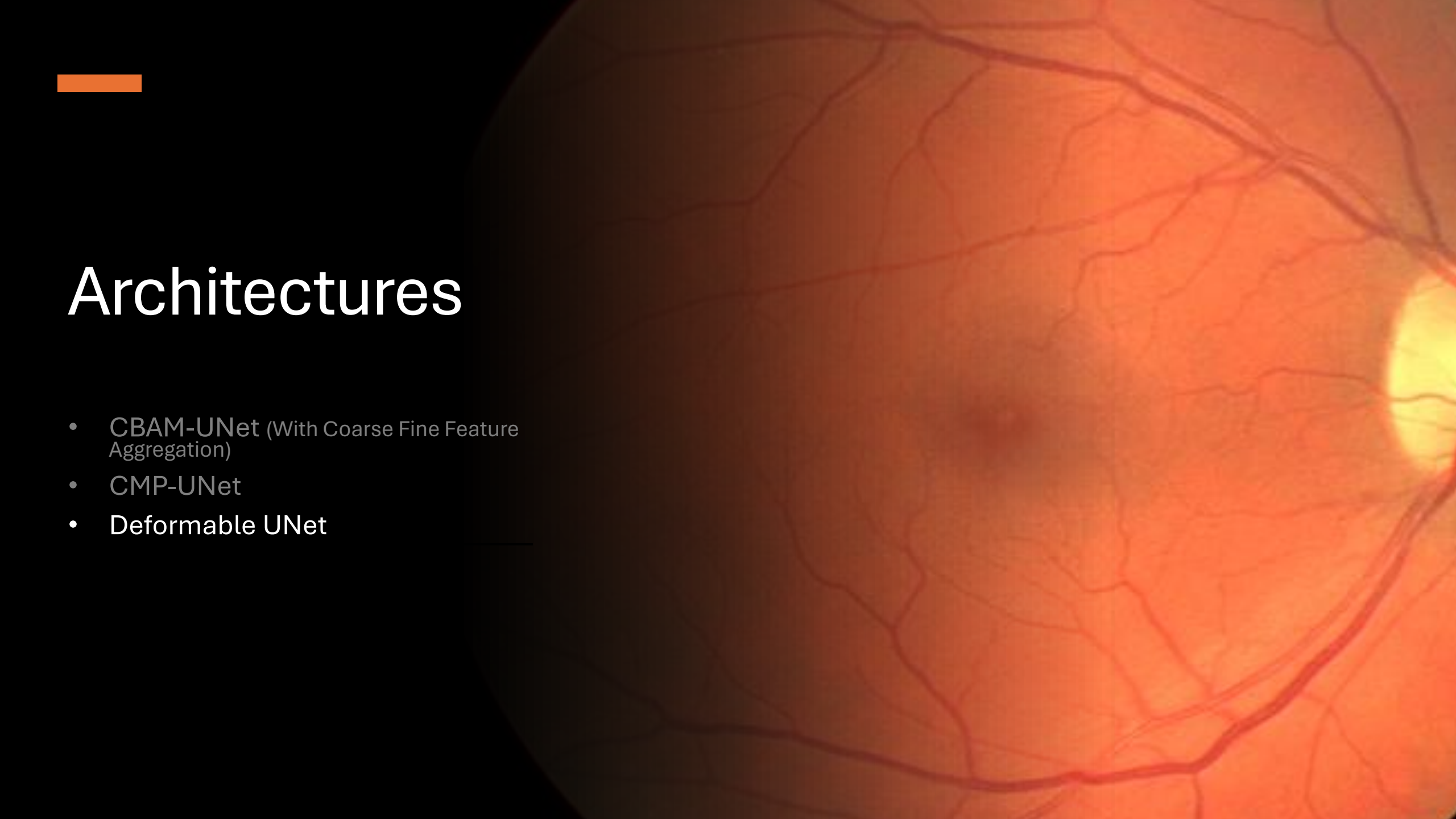
Pyramid Feature Fusion



```python
def PFF(input1, input2, input3, input4, num_filters):
    x1 = Conv2D(num_filters // 8, 1, use_bias = False)(input1)
    x1 = BatchNormalization()(x1)
    x1 = Activation("relu")(x1)
    x1 = UpSampling2D(8, interpolation='bilinear')(x1)

    x2 = Conv2D(num_filters // 8, 1, use_bias = False)(input2)
    x2 = BatchNormalization()(x2)
    x2 = Activation("relu")(x2)
    x2 = UpSampling2D(4, interpolation='bilinear')(x2)

    x3 = Conv2D(num_filters // 4, 1, use_bias = False)(input3)
    x3 = BatchNormalization()(x3)
    x3 = Activation("relu")(x3)
    x3 = UpSampling2D(2, interpolation='bilinear')(x3)

    x4 = Conv2D(num_filters // 2, 1, use_bias = False)(input4)
    x4 = BatchNormalization()(x4)
    x4 = Activation("relu")(x4)

    out = Concatenate()([x1, x2, x3, x4])
    print(out.shape)
    out = MSCAF(out, num_filters) + input4
    return out
```

Architectures

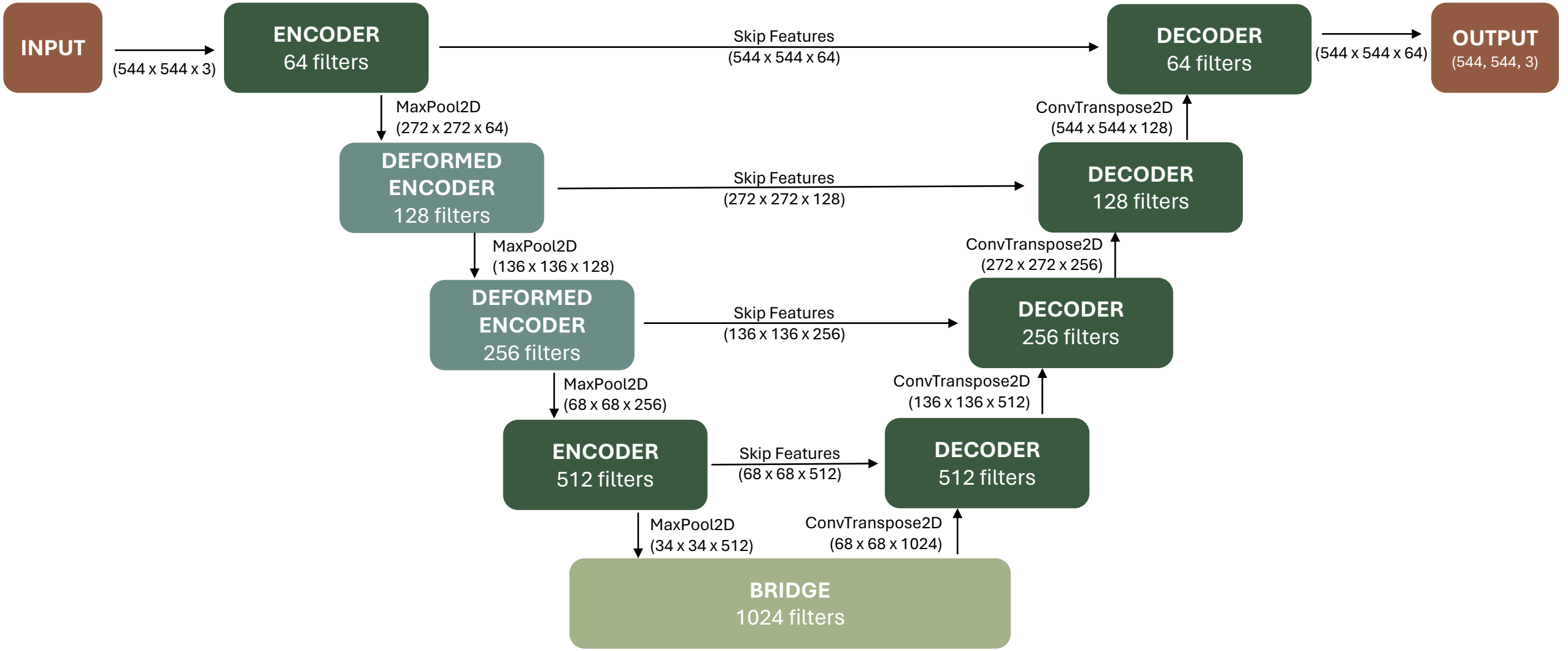
- CBAM-UNet (With Coarse Fine Feature Aggregation)
- CMP-UNet
- Deformable UNet



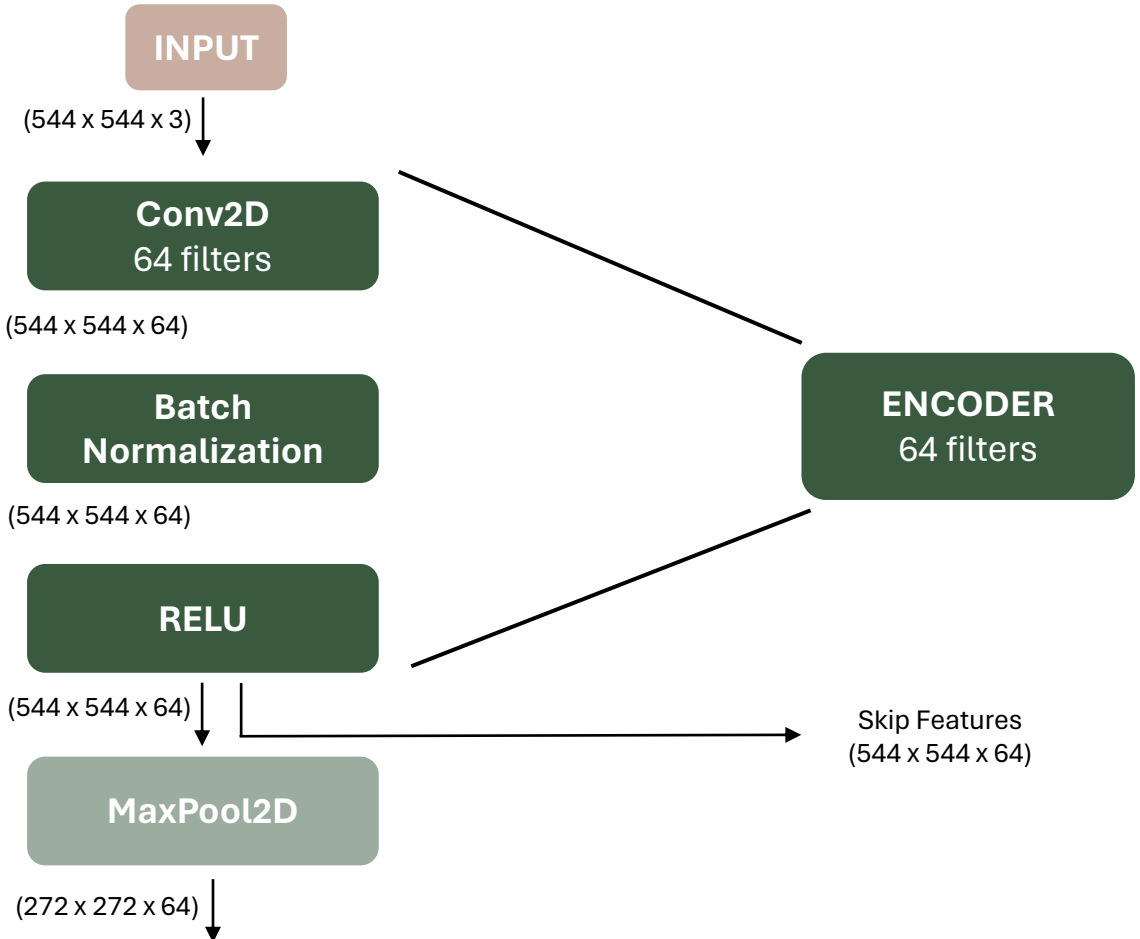
DU-Net

Jin, Q. et al.

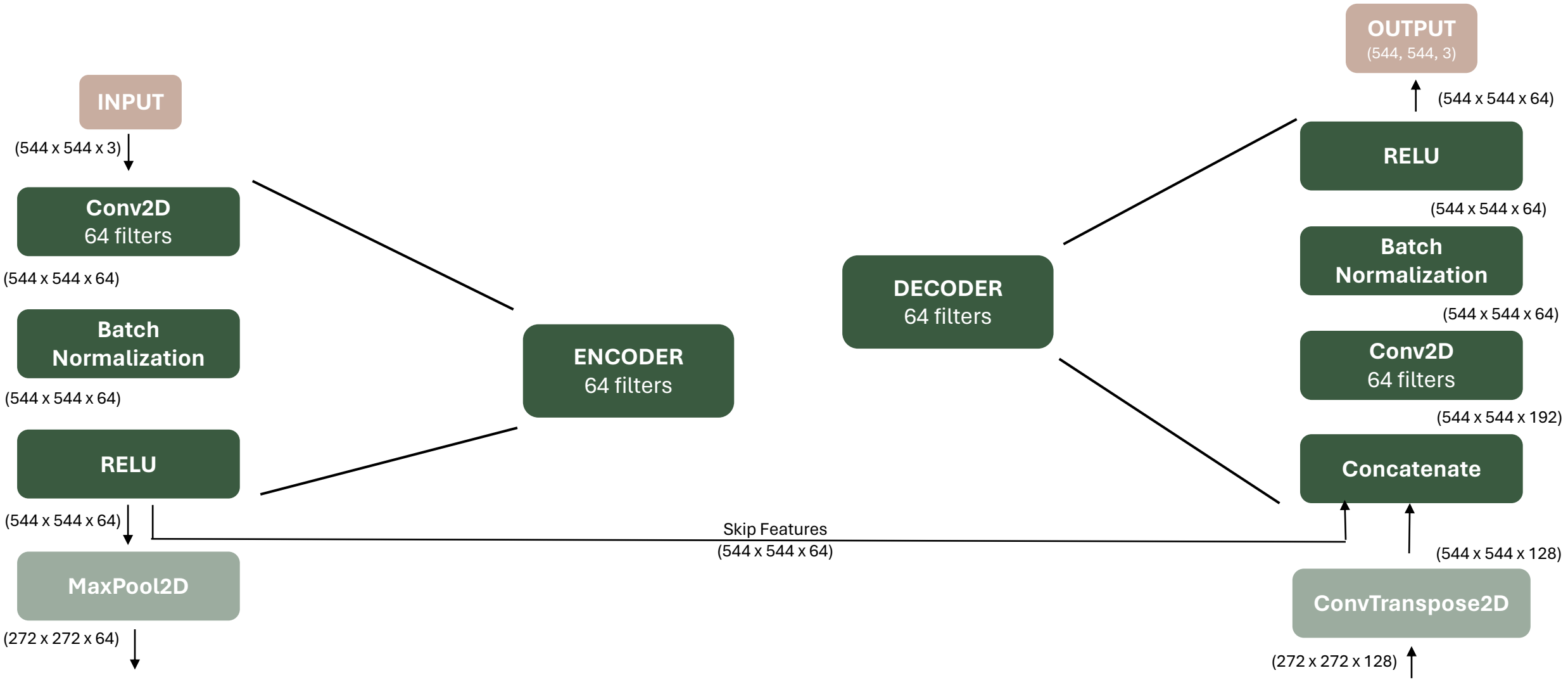
Architecture



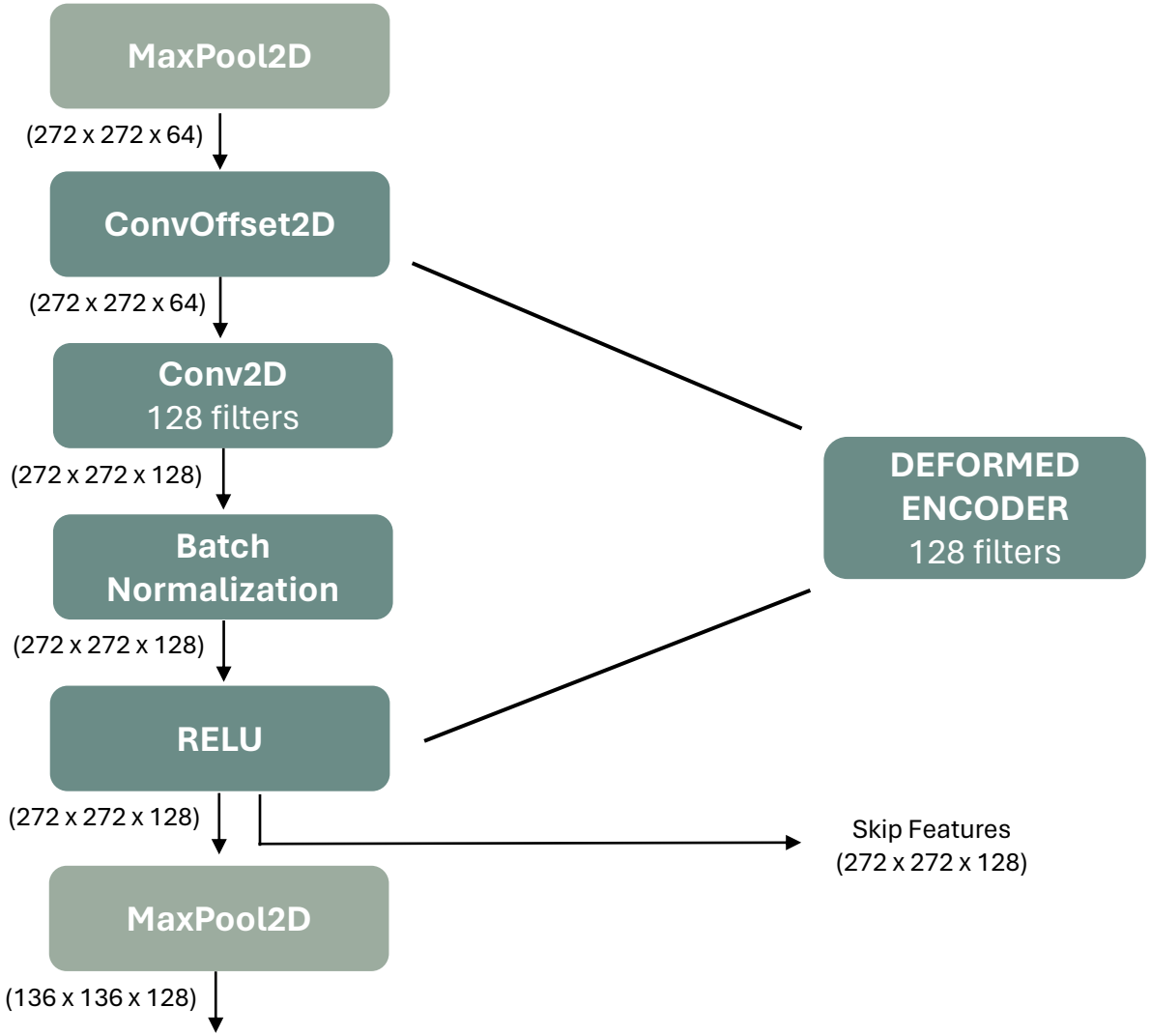
Architecture



Architecture



Architecture



Architecture

ConvOffset2D

Returns deformed feature map

- Calculates x and y offset for each ”pixel” in each feature map using Conv2D.

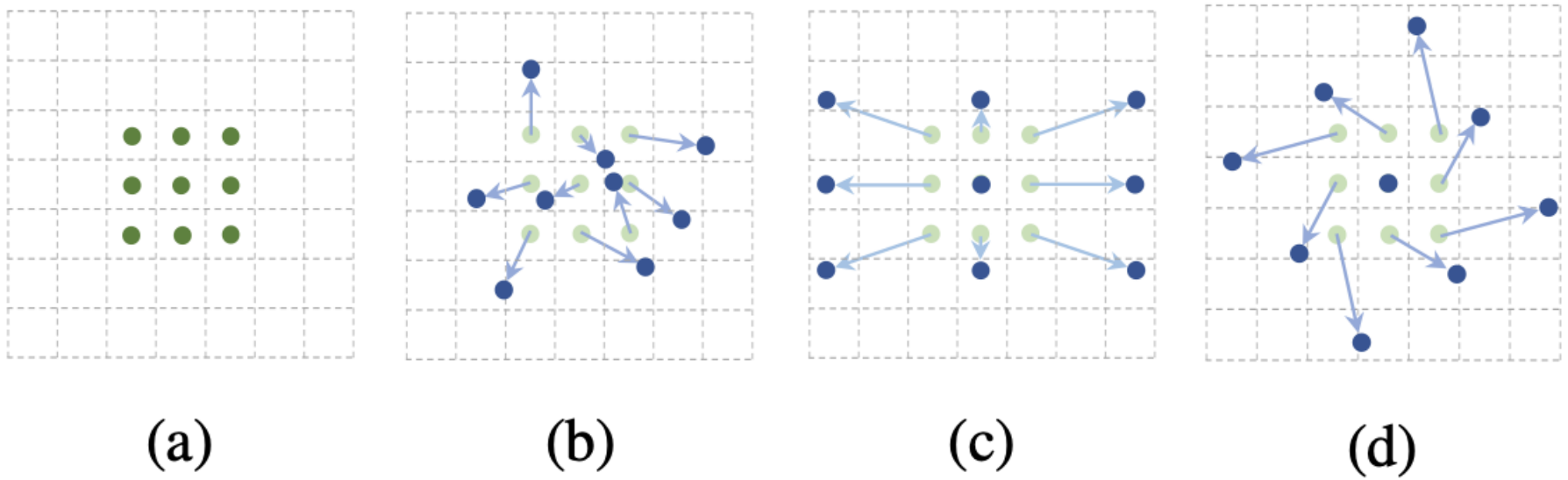


Image source:
Deformable Convolutional Networks – Dai, J. et al.

Architecture

ConvOffset2D

Returns deformed feature map

- Calculates x and y offset for each ”pixel” in each feature map

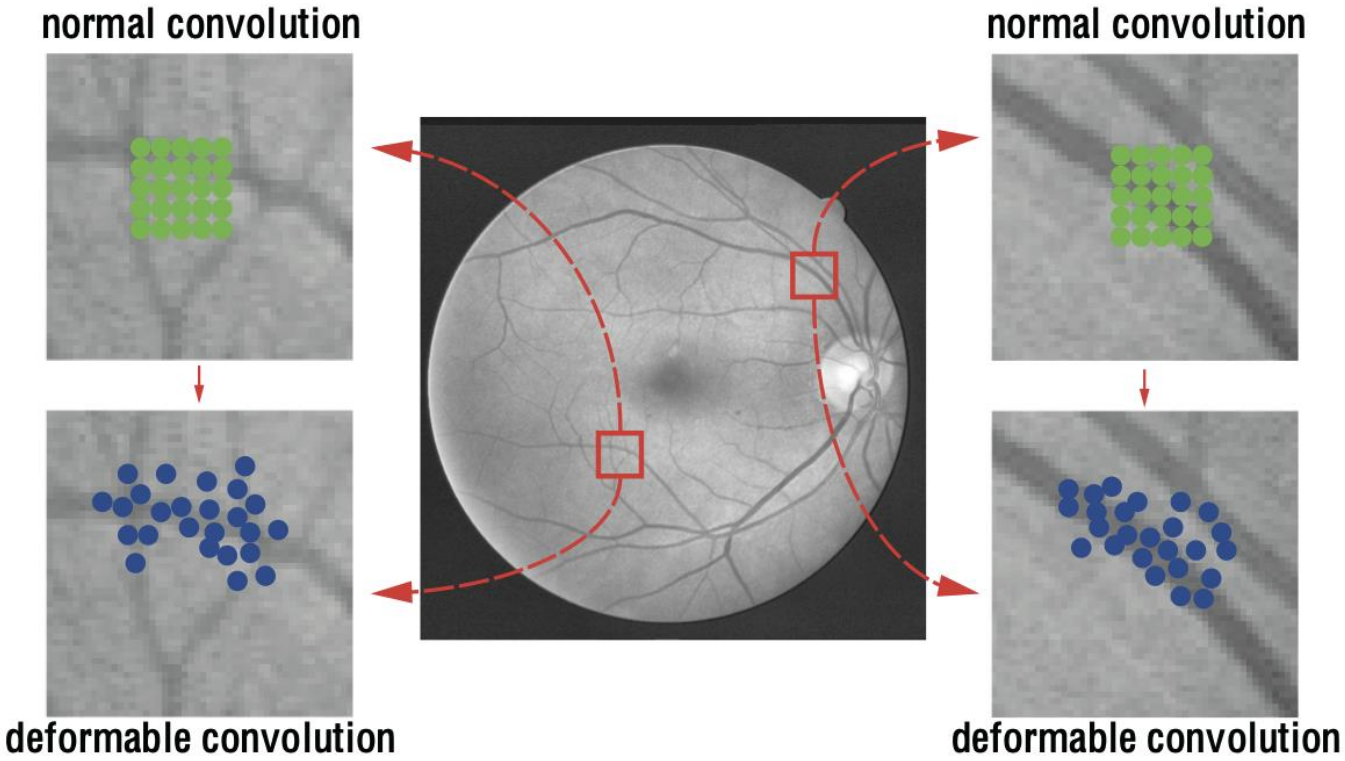


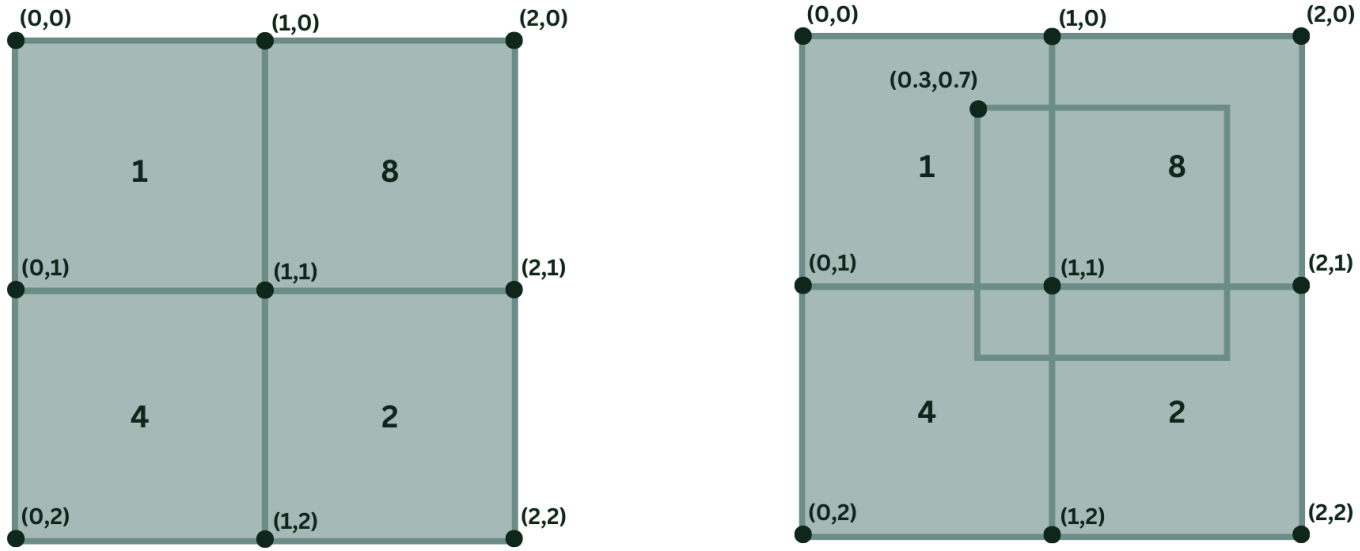
Image source:
DUNet: A deformable network for retinal vessel segmentation - Jin, Q. et al.

Architecture

ConvOffset2D

- Calculates x and y offset (Δp_n) for each ”pixel” in each feature map
- In the deformed feature map, value $x(p_0 + p_n)$ is replaced by $x(p_0 + p_n + \Delta p_n)$.

Bilinear Interpolation



$$x(p) = \sum_q \big(g(q_x + p_x) \cdot g(q_y + p_y) \cdot x(p)\big)$$

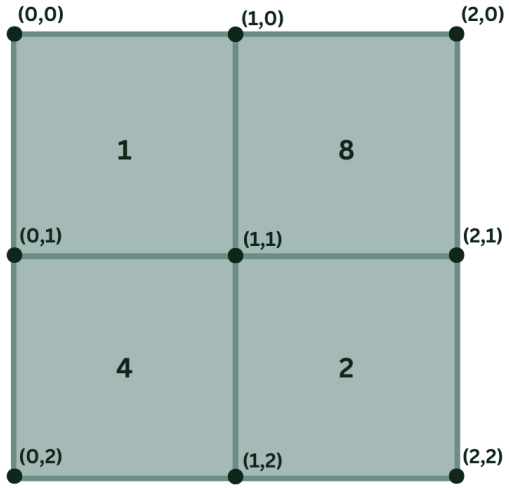
with $g(a, b) = \max(0, 1 - |a - b|)$

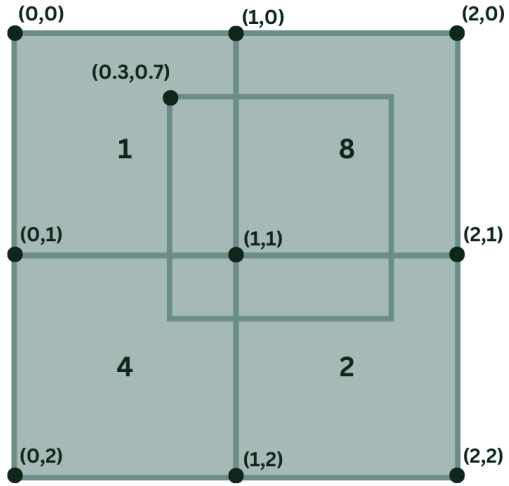
Architecture

ConvOffset2D

- Calculates x and y offset (Δp_n) for each ”pixel” in each feature map
- In the deformed feature map, value $x(p_0 + p_n)$ is replaced by $x(p_0 + p_n + \Delta p_n)$.

Bilinear Interpolation





$$x(p) = \sum_q \left(g(q_x, p_x) \cdot g(q_y, p_y) \cdot x(p)\right)$$

with $g(a, b) = \max(0, 1 - |a - b|)$

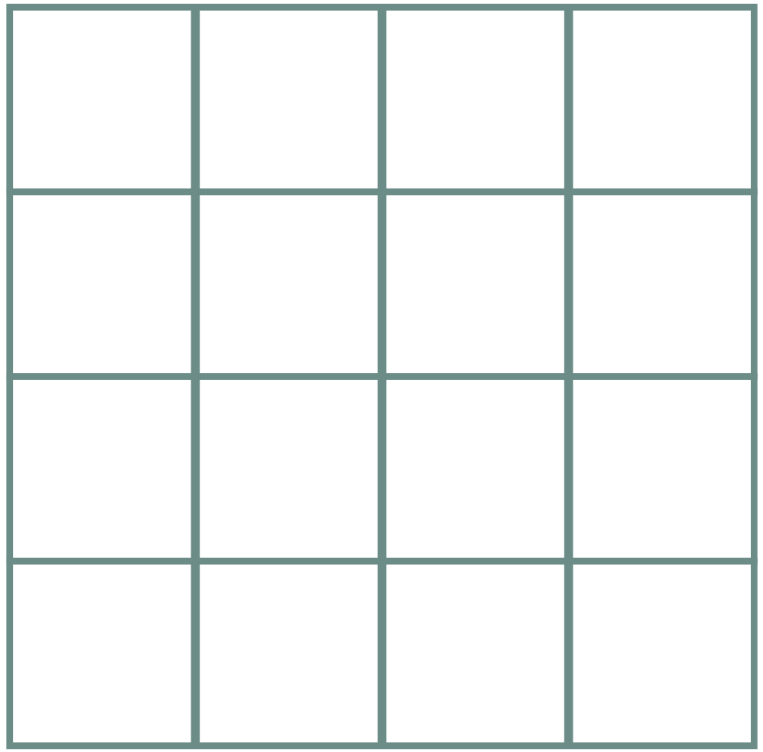
$$\begin{aligned} x(p) = {} & \max(0,1 - |0 - 0.7|) \cdot \max(0,1 - |0 - 0.3|) \cdot x(0,0) \\ & + \max(0,1 - |1 - 0.7|) \cdot \max(0,1 - |0 - 0.3|) \cdot x(1,0) \\ & + \max(0,1 - |0 - 0.7|) \cdot \max(0,1 - |1 - 0.3|) \cdot x(0,1) \\ & + \max(0,1 - |1 - 0.7|) \cdot \max(0,1 - |1 - 0.3|) \cdot x(1,1) \end{aligned}$$

$$\begin{aligned} x(p) = {} & 0.3 \cdot 0.7 \cdot 1 \\ & + 0.7 \cdot 0.7 \cdot 8 \\ & + 0.3 \cdot 0.3 \cdot 4 \\ & + 0.7 \cdot 0.3 \cdot 2 \end{aligned}$$

$$x(p) = 4.91 \approx 5$$

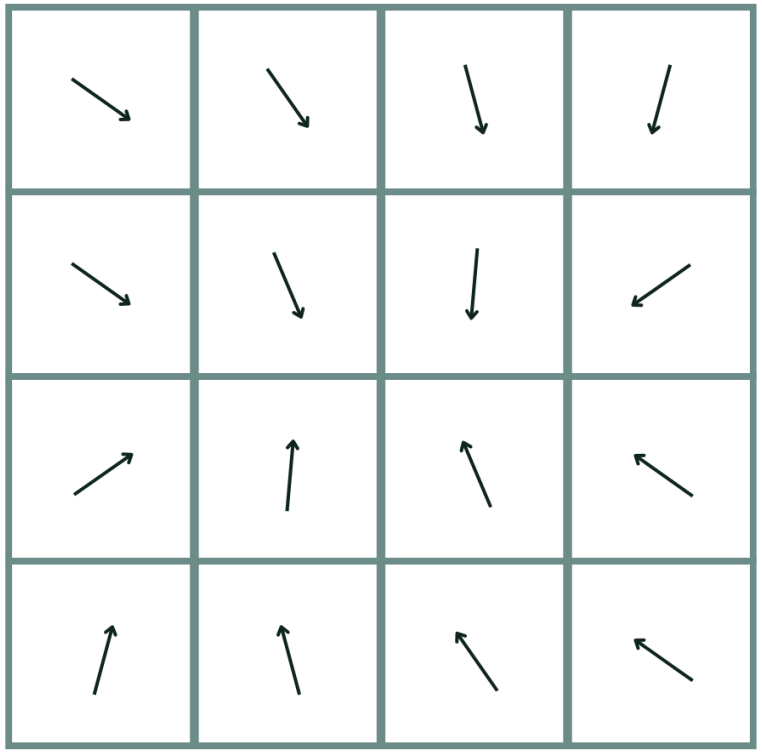
Architecture – Implementation Layer

ConvOffset2D



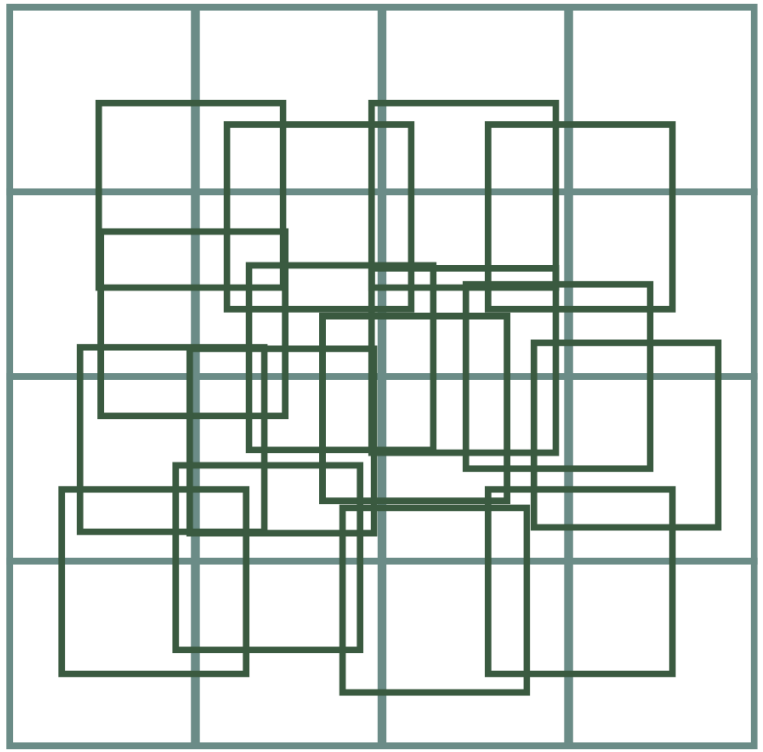
Architecture – Implementation Layer

ConvOffset2D



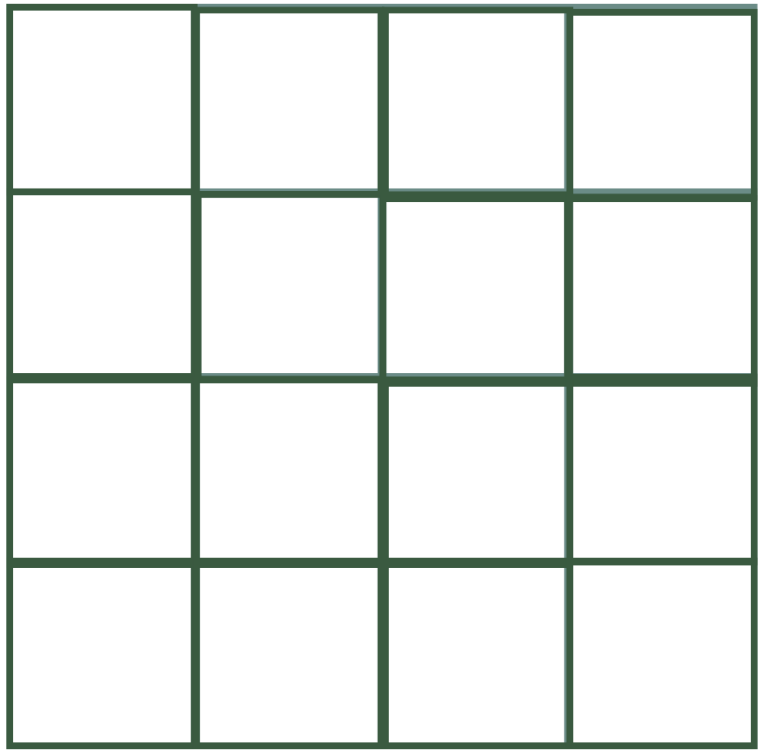
Architecture – Implementation Layer

ConvOffset2D



Architecture – Implementation Layer

ConvOffset2D



Architecture – Implementation Layer

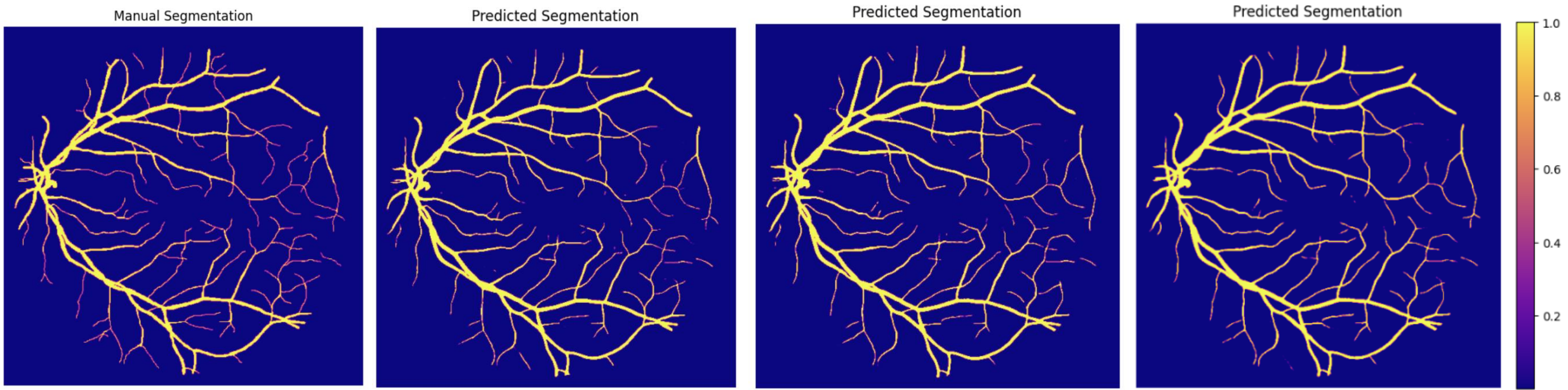
ConvOffset2D

```
10 usages
class ConvOffset2D(Conv2D):
    def __init__(self, filters, init_normal_stddev=0.01, **kwargs):
        self.filters = filters
        super(ConvOffset2D, self).__init__(
            self.filters * 2, (3, 3), padding='same', use_bias=False,
            kernel_initializer=RandomNormal(0, init_normal_stddev),
            **kwargs
        )
        self.x_offsets = None

    def call(self, x):
        x_shape = x.get_shape()
        offsets = super(ConvOffset2D, self).call(x)  # (b, h, w, 2c)

        # offsets: (b*c, h, w, 2)
        offsets = self._to_bc_h_w_2(offsets, x_shape)

        # x: (b*c, h, w)
        x = self._to_bc_h_w(x, x_shape)

        # X_offset: (b*c, h, w)
        x_offset = tf_batch_map_offsets(x, offsets)

        # x_offset: (b, h, w, c)
        x_offset = self._to_b_h_w_c(x_offset, x_shape)

        return x_offset
```

Based on:
https://github.com/kastnerkyle/deform-conv/tree/master

Performance

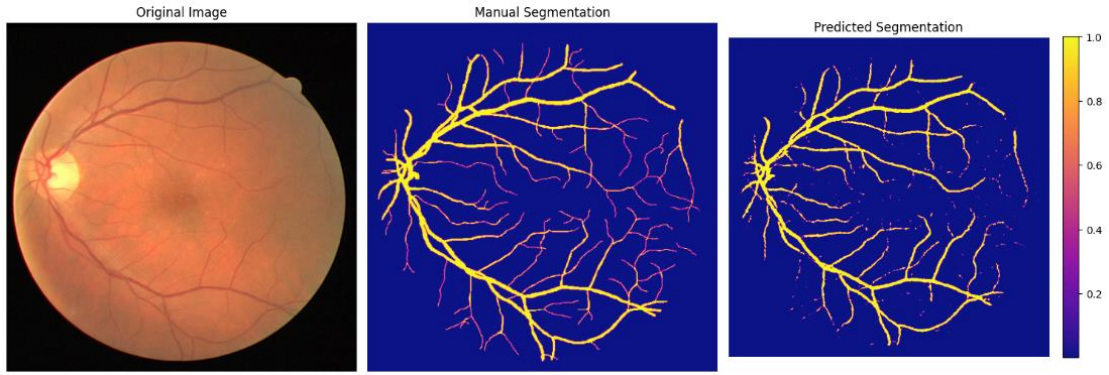
	U-Net	Deformable Encoder	DU-Net
Validation DICE	0.8246	0.8243	0.8233
Trainable Parameters	31 039 425	32 886 721	34 729 921

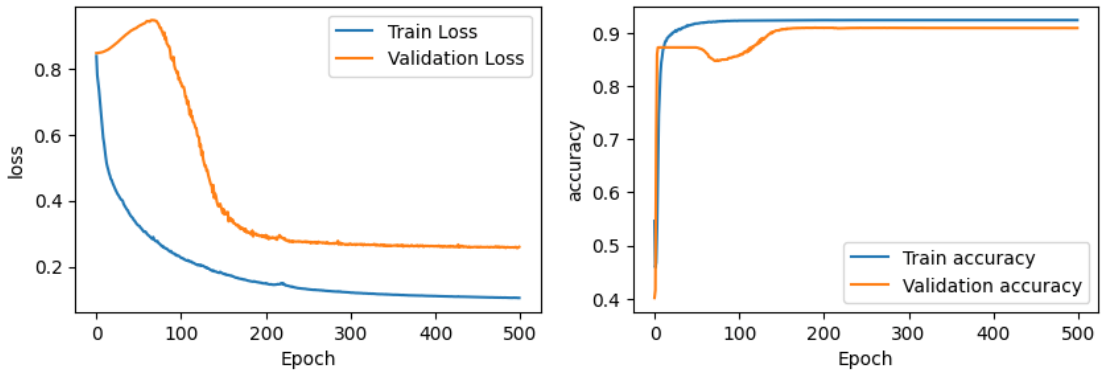


Performance

Deformable Encoder

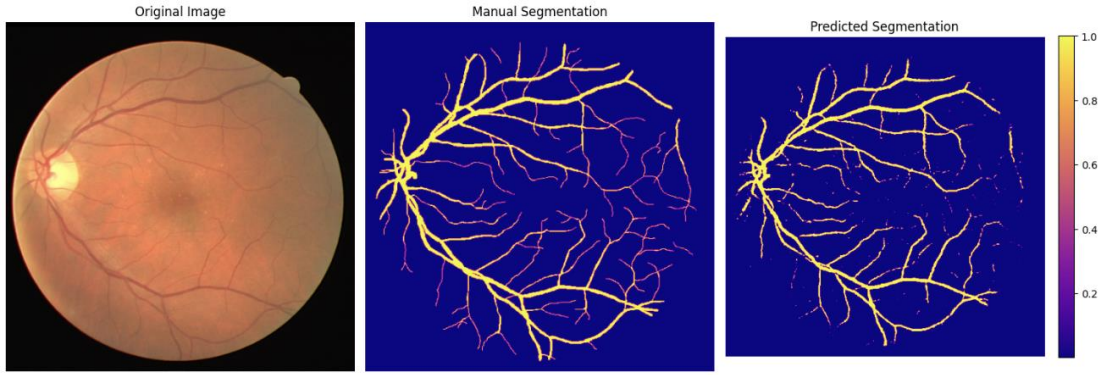
- Validation DICE: 0.7433
- Parameters: 32,898,497
  (trainable: 32,886,721)

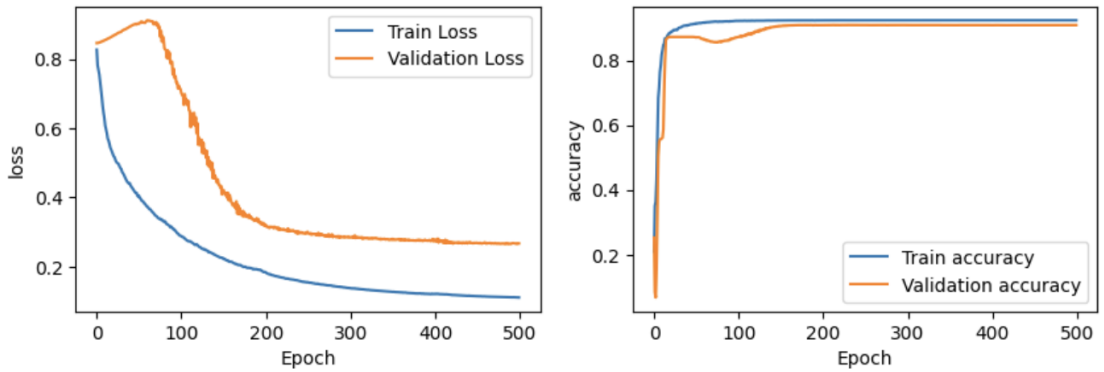




Full DU-Net

- Validation DICE: 0.7328
- Parameters: 34,741,697
  (trainable: 34,729,921)

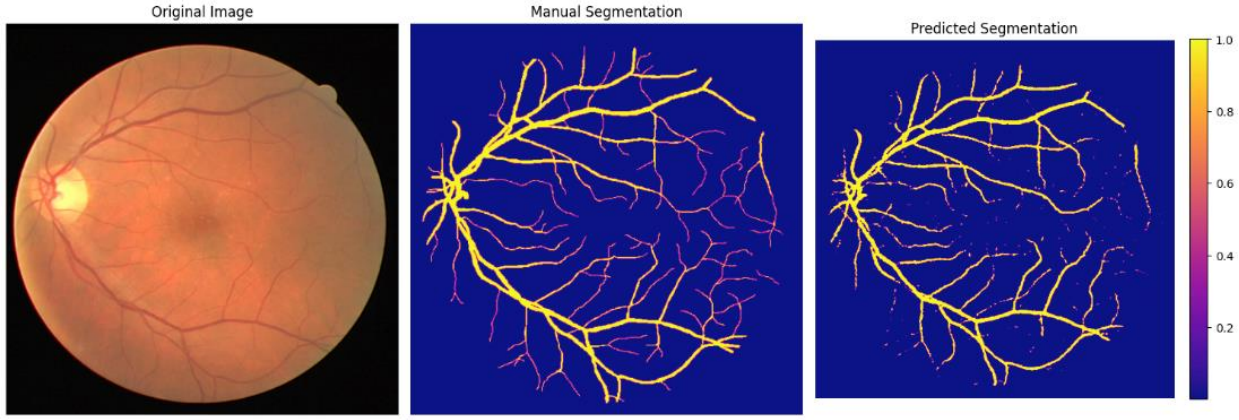


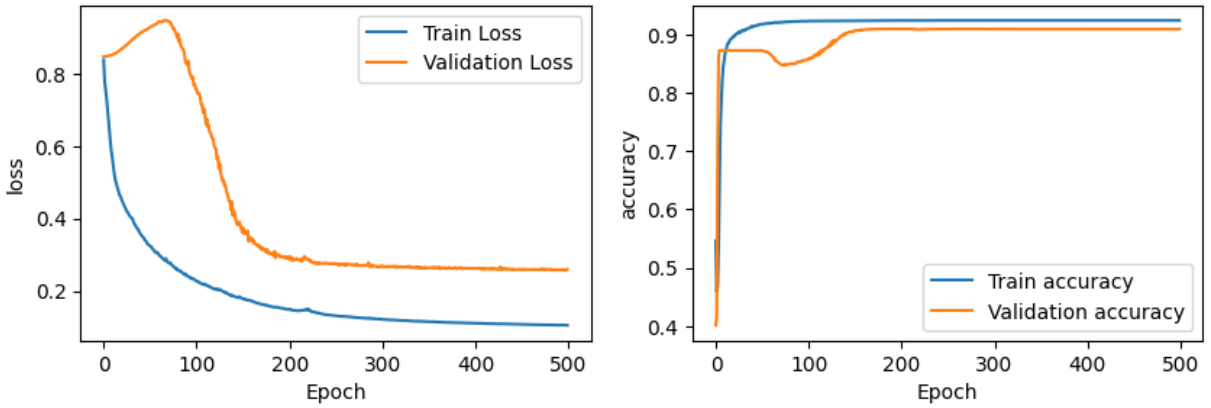


Performance

Deformable Encoder - Original Data

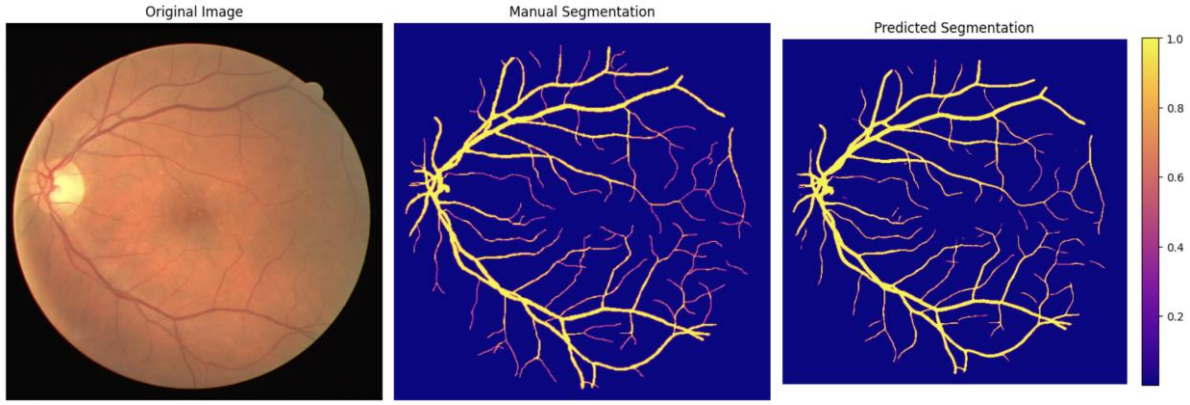
- Validation DICE: 0.7433

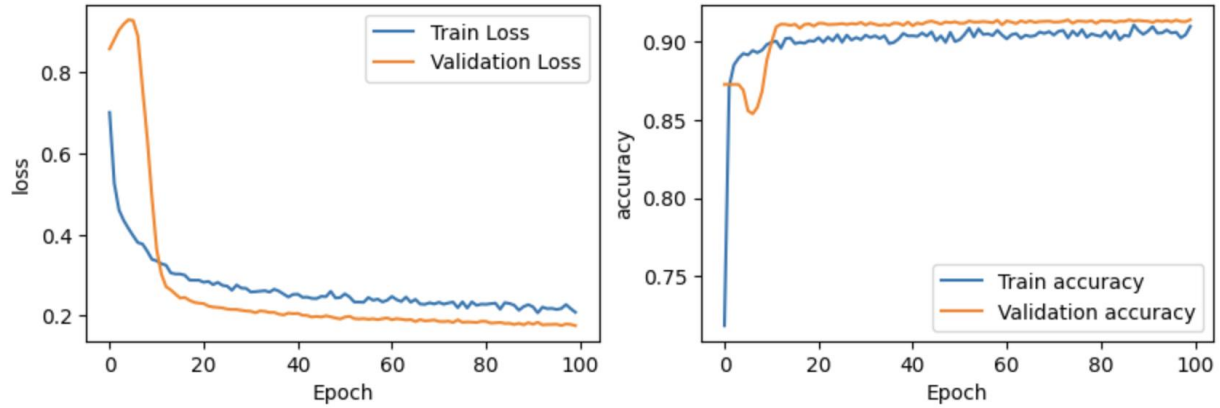




Deformable Encoder - Data Augmentation

- Validation DICE: 0.8243

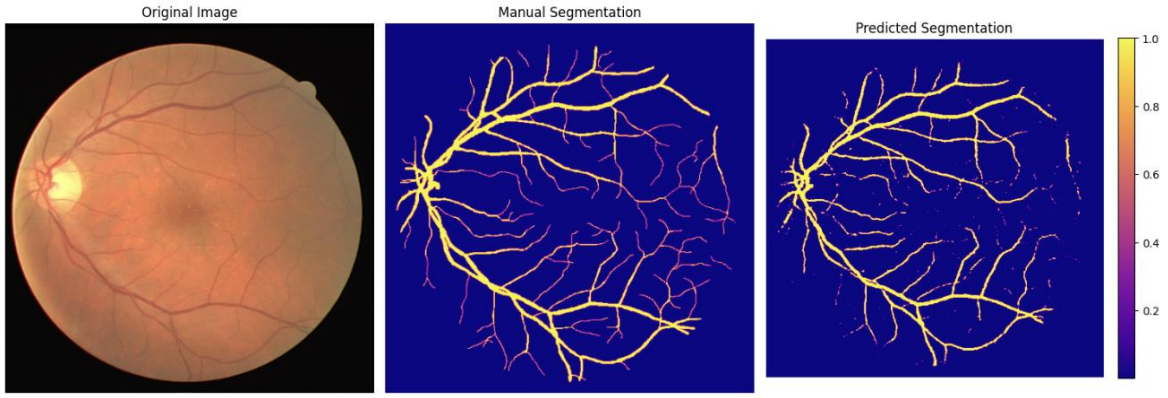


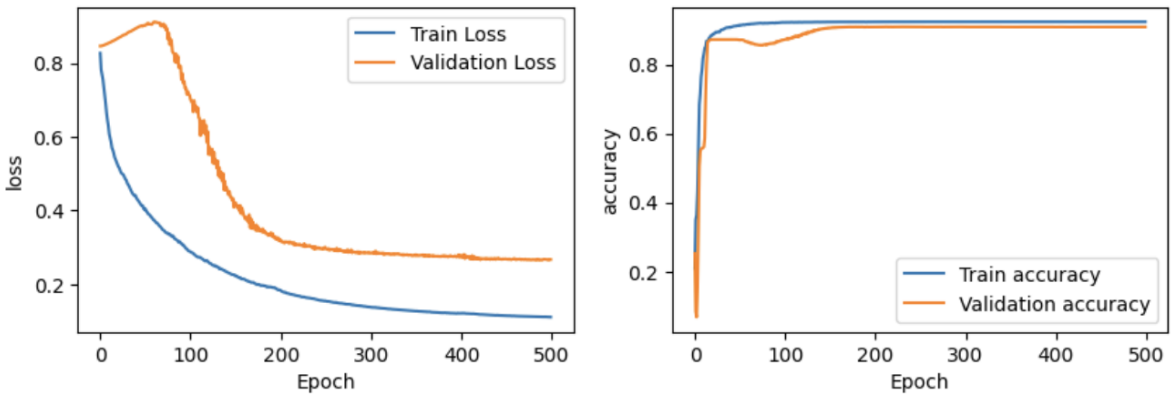


Performance

Full DUNet - Original Data

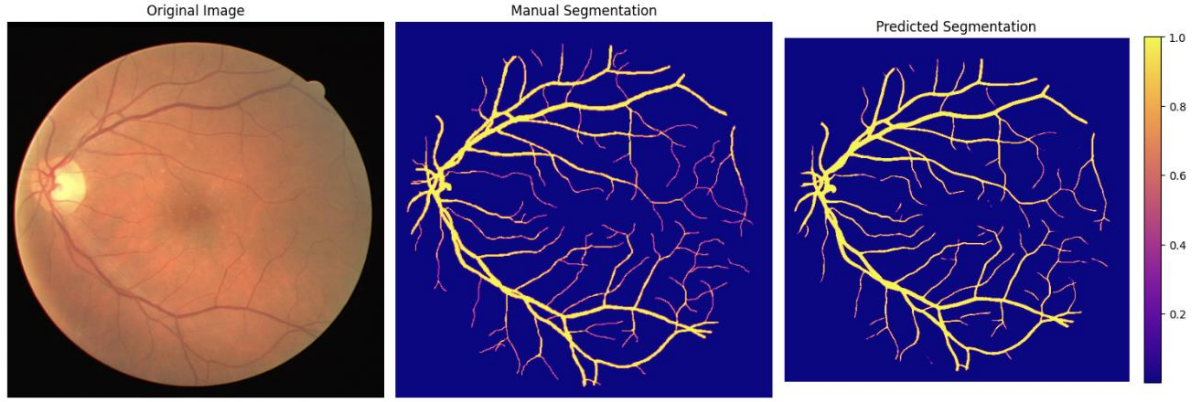
- Validation DICE: 0.7328

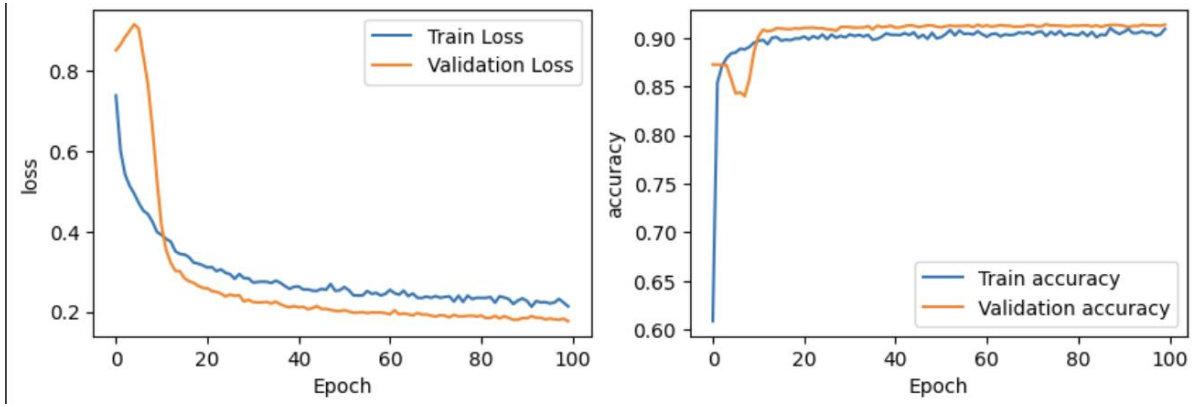




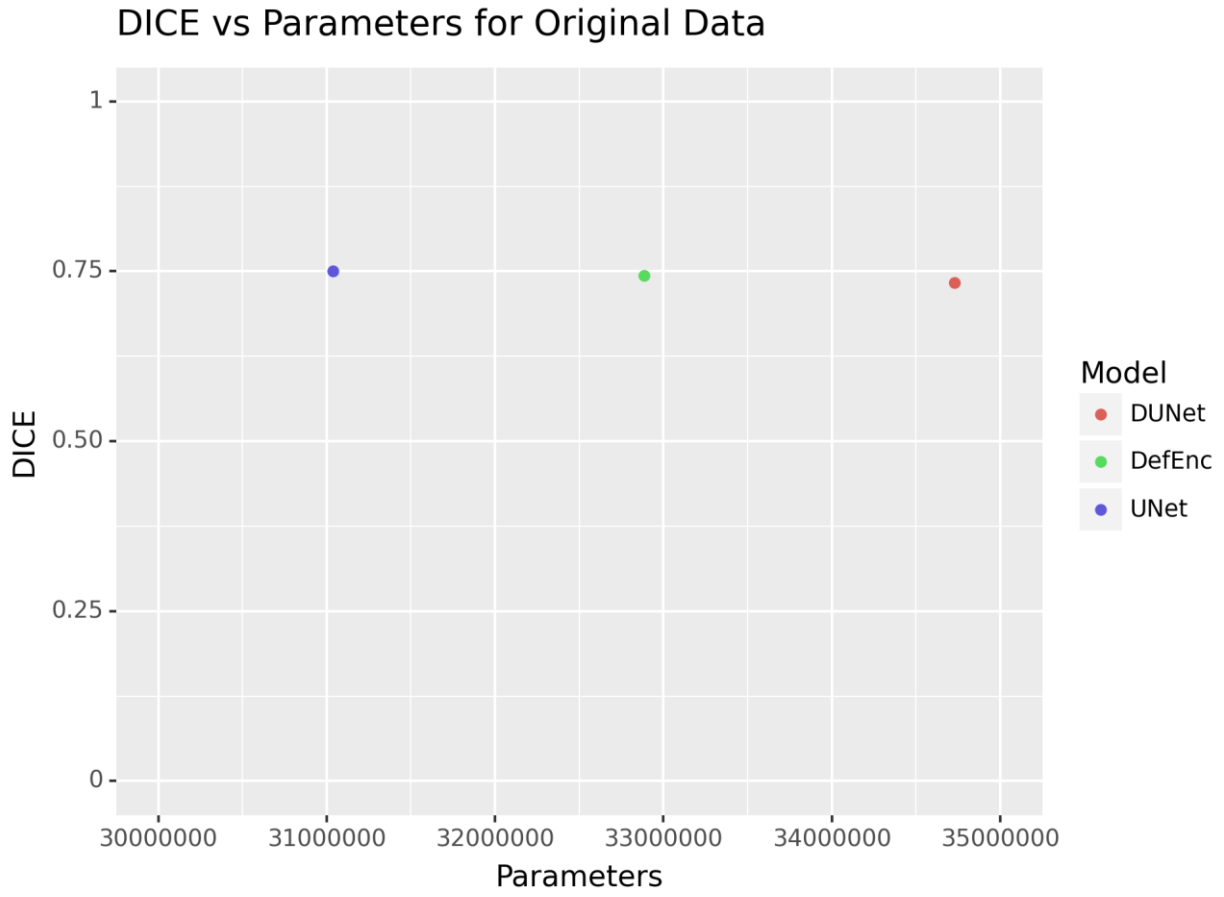
Full DUNet - Data Augmentation

- Validation DICE: 0.8233

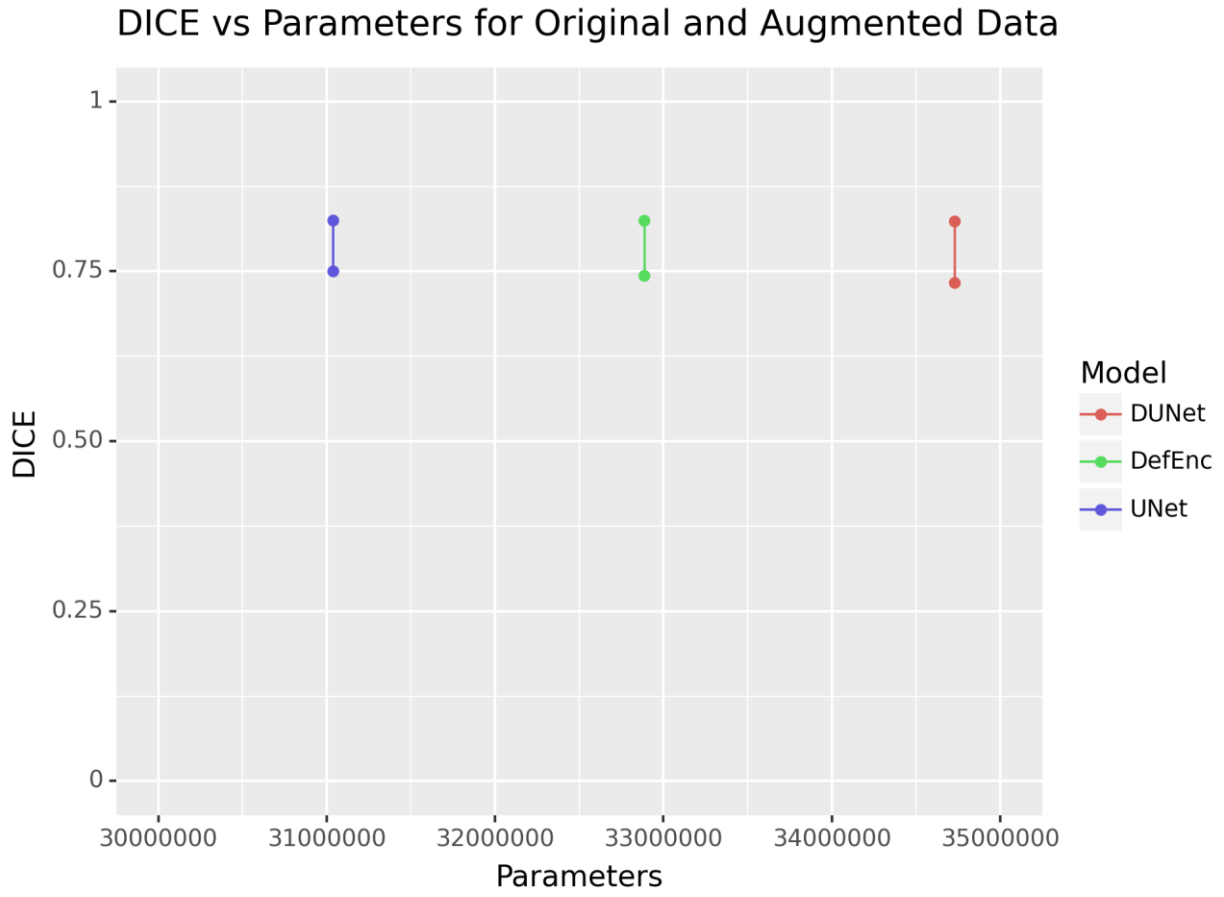




Effect of Data Augmentation

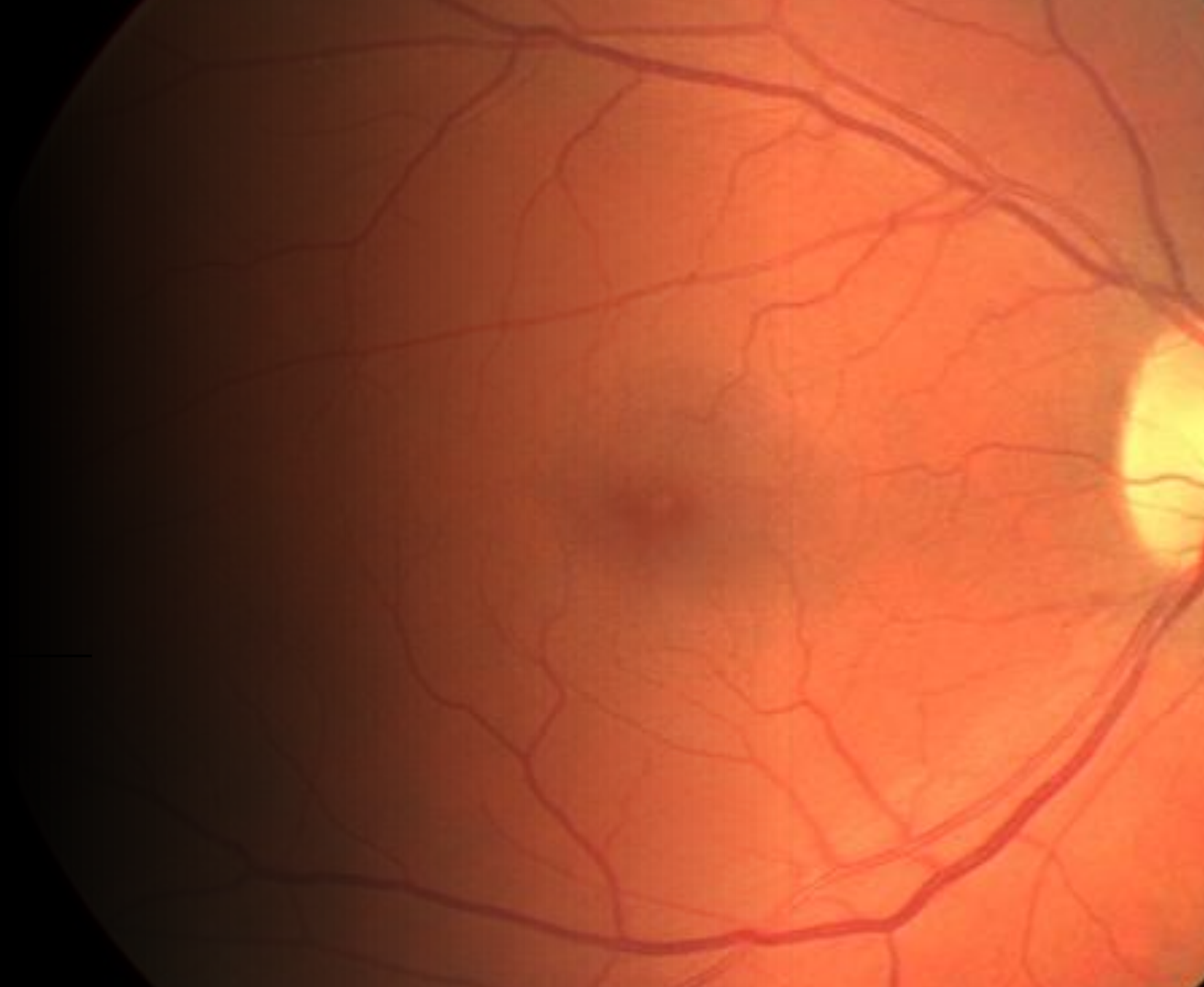


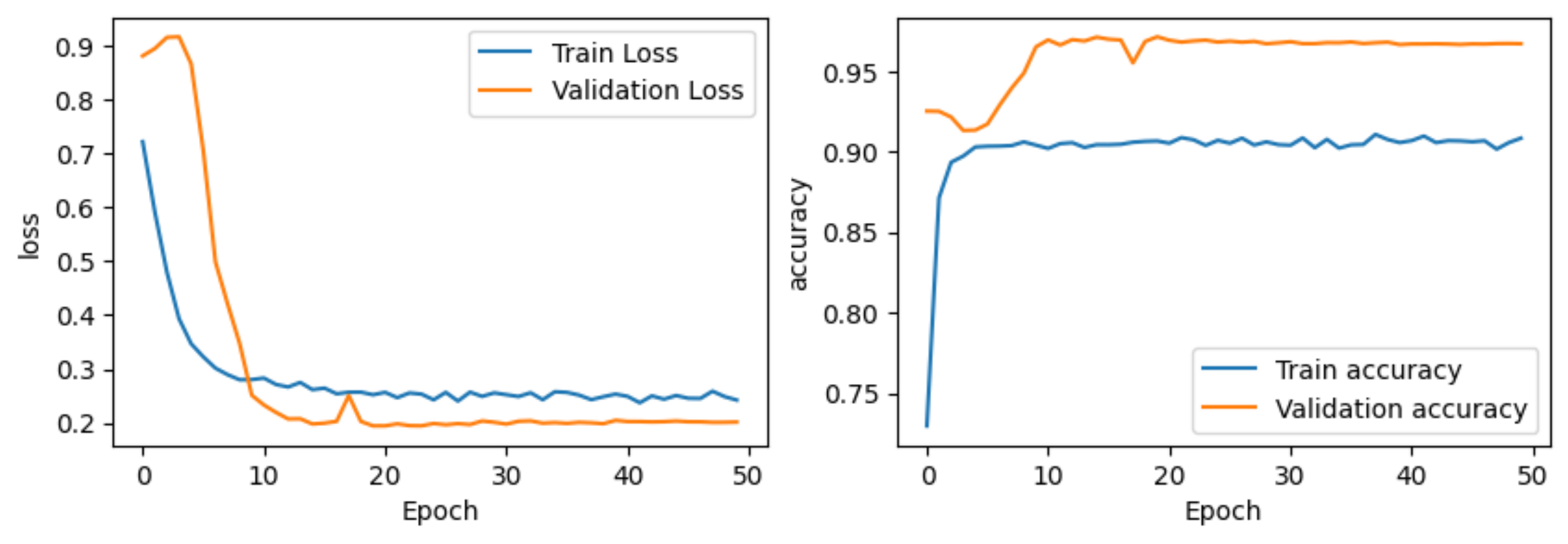
Effect of Data Augmentation



Results

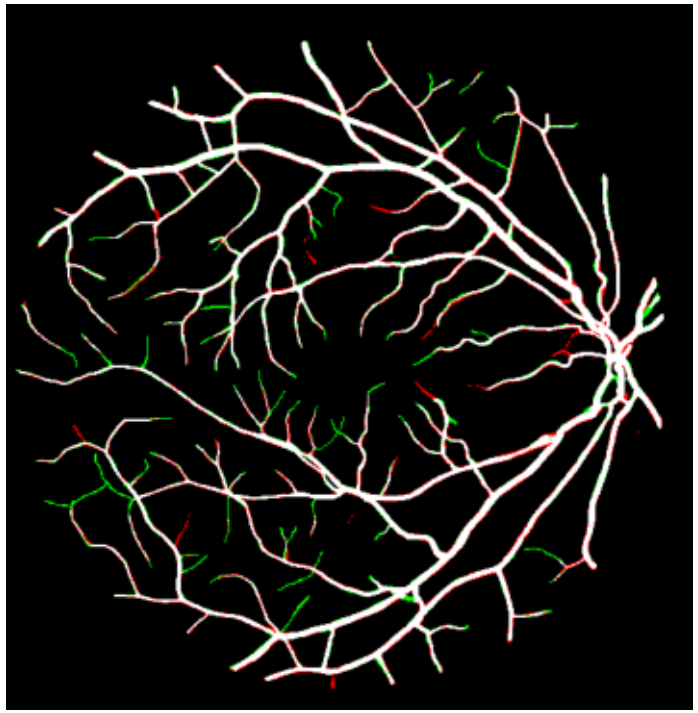
Visualizations




Train Loss
Validation Loss
loss
Epoch
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
10
20
30
40
50
Train accuracy
Validation accuracy
accuracy
Epoch
0.95
0.90
0.85
0.80
0.75

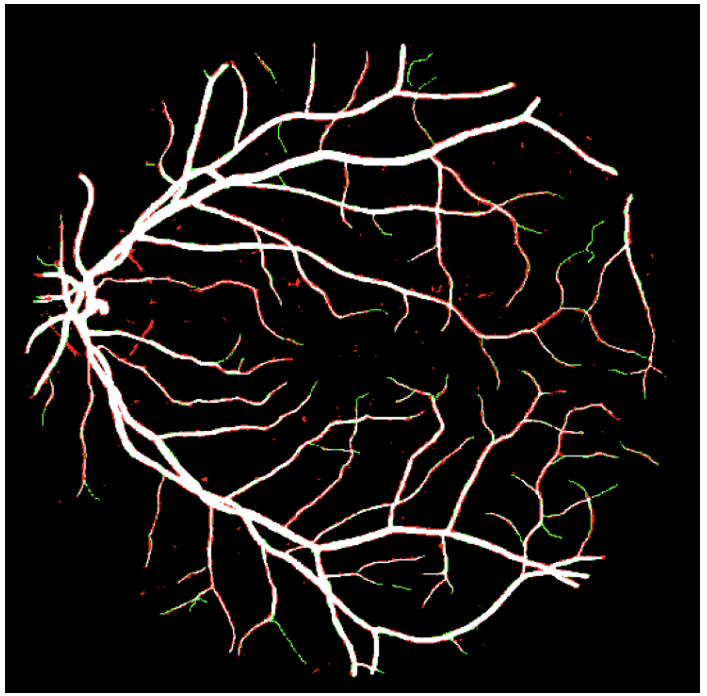
Results

CBAM-UNet (CFFA)
Test DICE: **0.8227**
Grand Challenge rank: 306



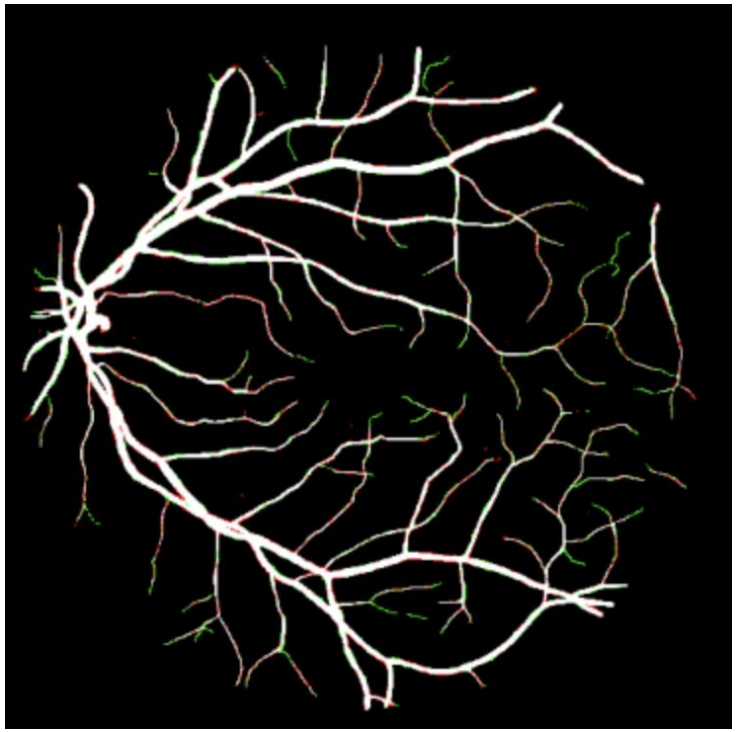
TP	
TN	
FP	
FN	

D-Unet
Validation DICE: **0.8233**



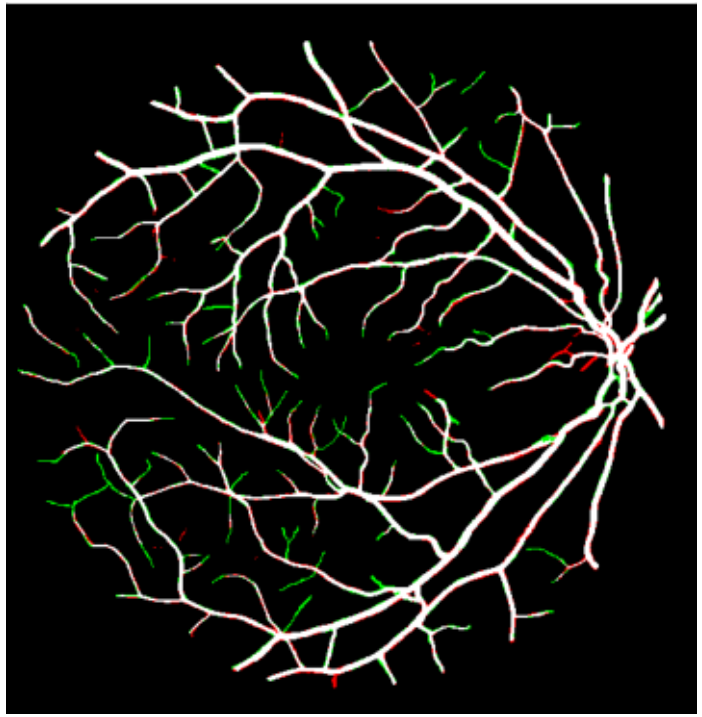
Results

UNet (3 encoders/decoders)
Test DICE: **0.8229**
Grand Challenge rank: 300



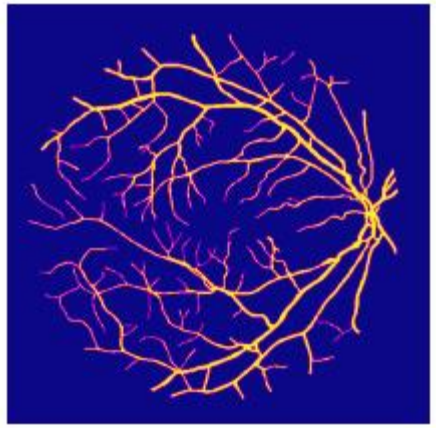
TP	
TN	
FP	
FN	

CMP-UNet
Test DICE: **0.8245**
Grand Challenge rank: 203

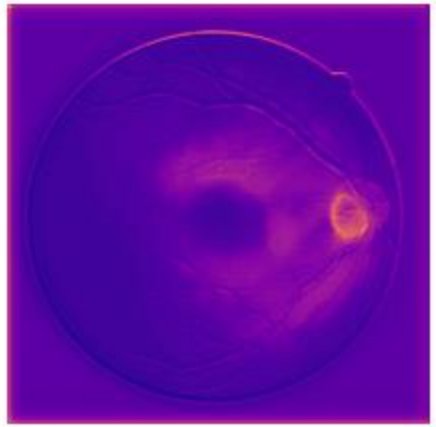


Epoch: 2

True

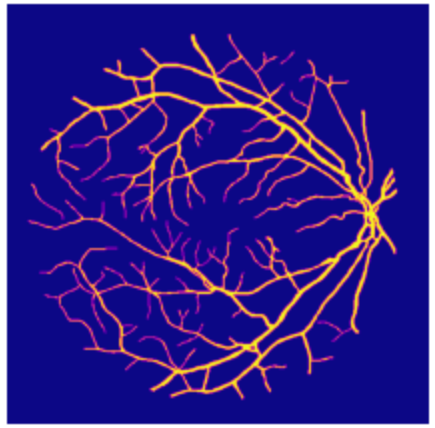


Pred

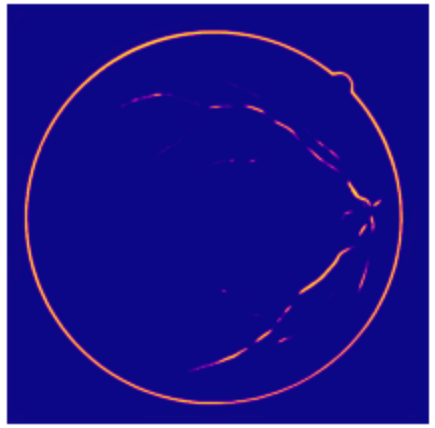


Epoch: 4

True

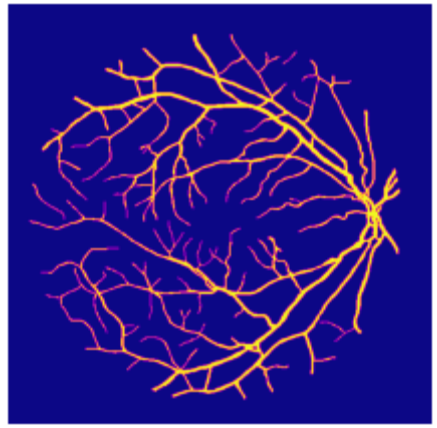


Pred

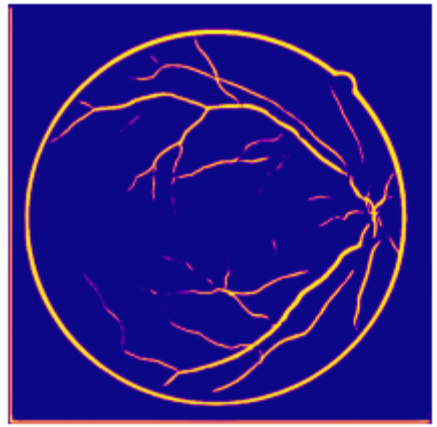


Epoch: 6

True

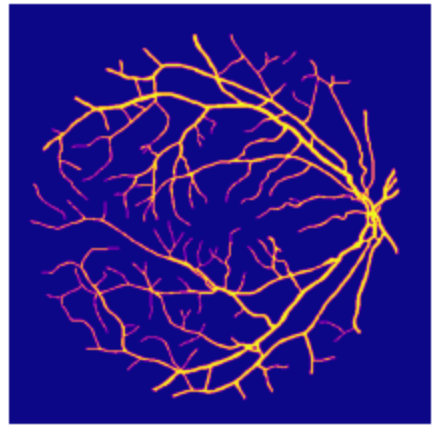


Pred

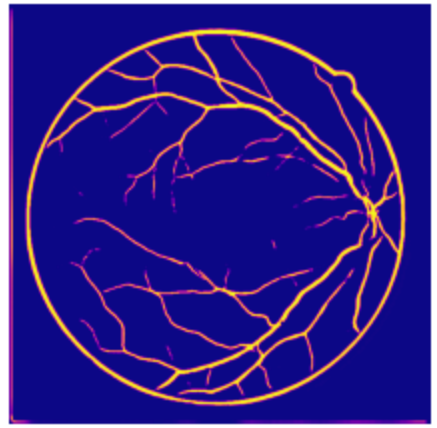


Epoch: 8

True

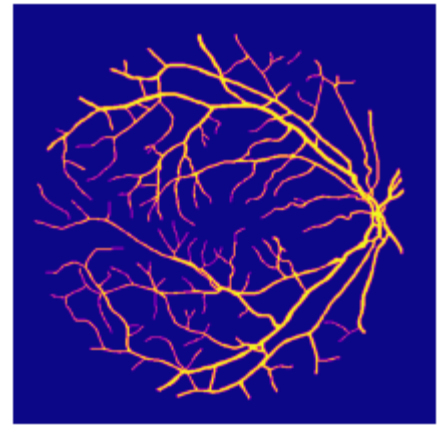


Pred

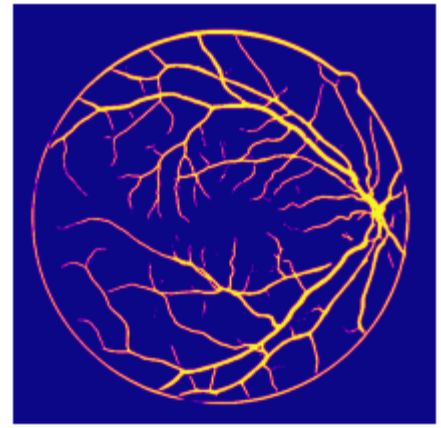


Epoch: 10

True



Pred



Bibliography

- Gu Y, Cao R, Wang D, Lu B. CMP-UNet: A Retinal Vessel Segmentation Network Based on Multi-Scale Feature Fusion. *Electronics*. 2023; 12(23):4743. https://doi.org/10.3390/electronics12234743
- Guo, C., Szemenyei, M., Yi, Y., Wang, W., Chen, B., & Fan, C. (2020). SA-UNet: Spatial Attention U-Net for Retinal Vessel Segmentation. ArXiv, abs/2004.03696.
- Woo, S., Park, J., Lee, J.-Y., & Kweon, I. S. (2018). CBAM: Convolutional Block Attention Module.

Timesheet

Name	Subject	Estimated Duration
Simon De Lange	Base UNet and helper functions	4h
	CMP-UNet	16h
	CBAM-UNet (and extra modules)	6h
	Slides CBAM-UNet + CMP-UNet	2h
Mariska Van de Sompele	DUNet (custom layer)	16h
	DUNet implementations	3h
	Slides and Illustrations	3h
Jente Nijs	Background research	3h
	Data augmentation slide	2h