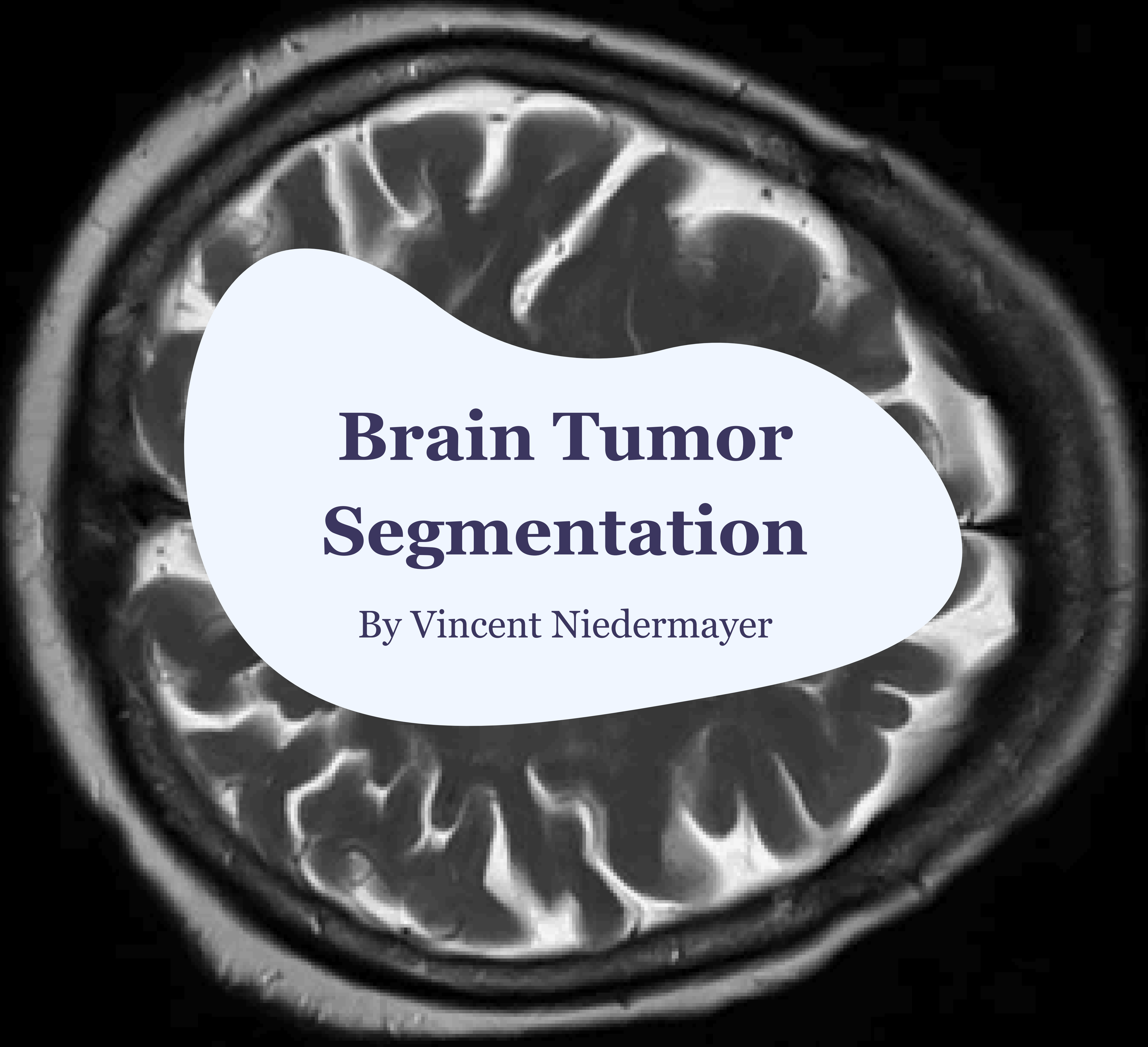
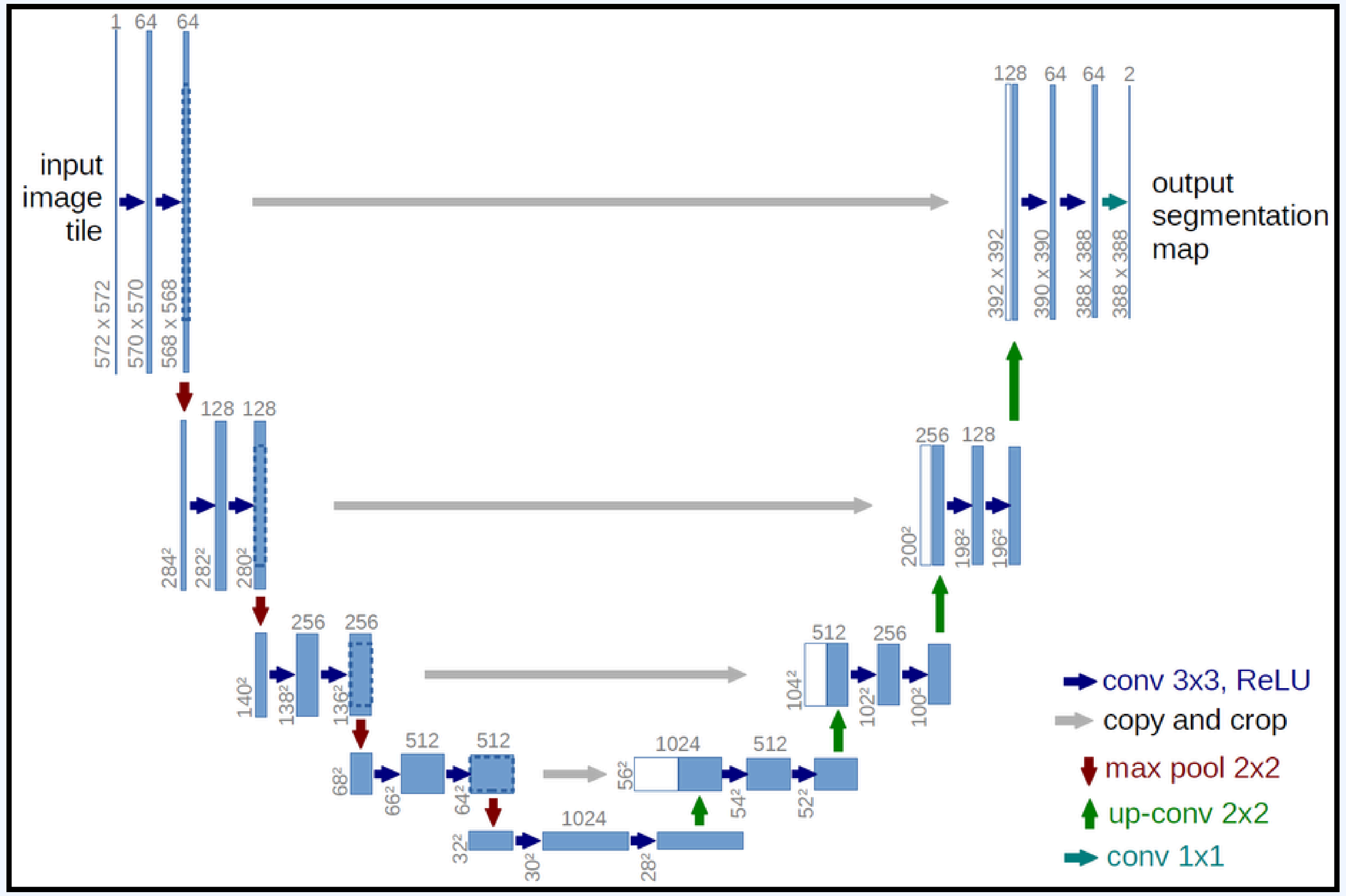


Brain Tumor Segmentation

By Vincent Niedermayer

Model Architecture



https://arxiv.org/pdf/1505.04597

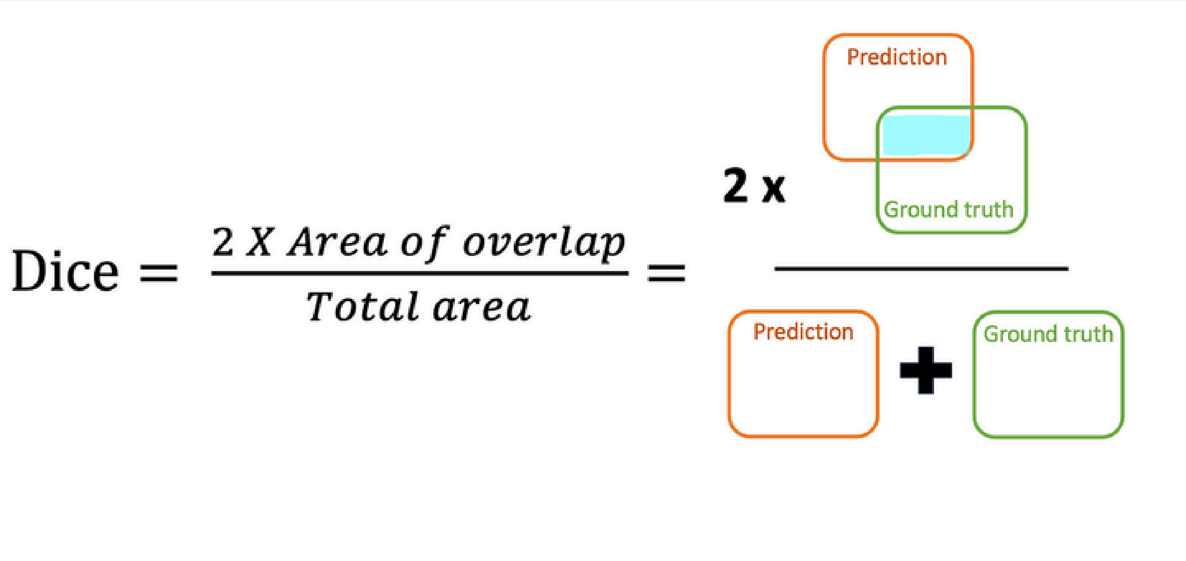
Key additions:

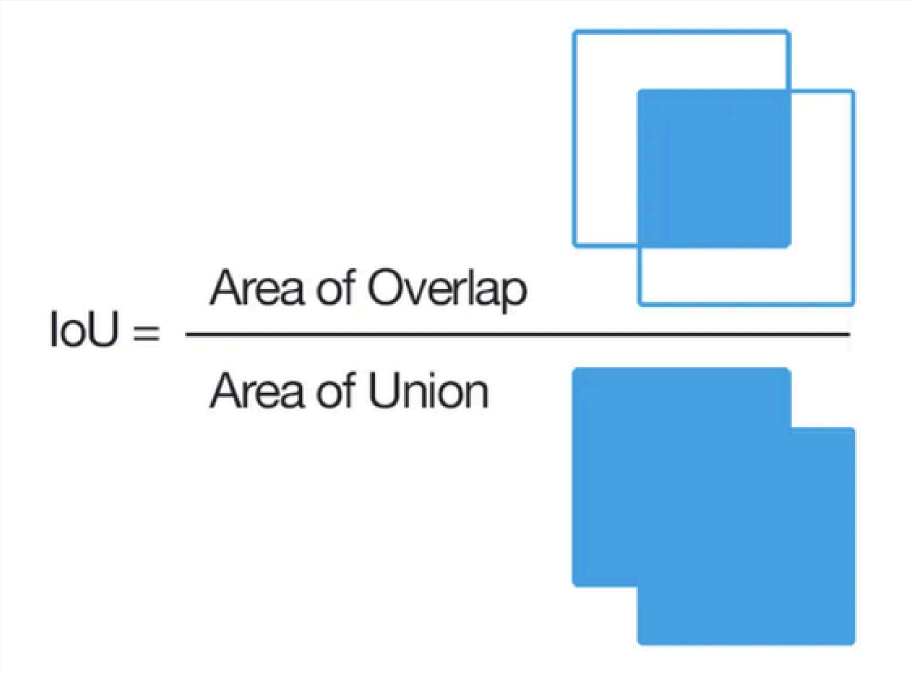
- Batch Normalization
- Dropout (.3)

```python
def build_model(self):
    inputs = Input(shape=(self.input_size, self.input_size, 3))  # Default input shape: (256, 256, 3)

    # Encoder - Downscaling
    c1 = Conv2D(self.base_filters, (3, 3), activation='relu', padding='same')(inputs)
    c1 = BatchNormalization()(c1) # Batch Normalization to allieviate overfitting
    c1 = Conv2D(self.base_filters, (3, 3), activation='relu', padding='same')(c1)
    c1 = BatchNormalization()(c1)
    p1 = MaxPooling2D((2, 2))(c1)

    c2 = Conv2D(self.base_filters * 2, (3, 3), activation='relu', padding='same')(p1)
    c2 = BatchNormalization()(c2)
    c2 = Conv2D(self.base_filters * 2, (3, 3), activation='relu', padding='same')(c2)
    c2 = BatchNormalization()(c2)
    p2 = MaxPooling2D((2, 2))(c2)

    c3 = Conv2D(self.base_filters * 4, (3, 3), activation='relu', padding='same')(p2)
    c3 = BatchNormalization()(c3)
    c3 = Conv2D(self.base_filters * 4, (3, 3), activation='relu', padding='same')(c3)
    c3 = BatchNormalization()(c3)
    c3 = Dropout(self.dropout_rate)(c3) # Dropout to allieviate overfitting
    p3 = MaxPooling2D((2, 2))(c3)

    c4 = Conv2D(self.base_filters * 8, (3, 3), activation='relu', padding='same')(p3)
    c4 = BatchNormalization()(c4)
    c4 = Conv2D(self.base_filters * 8, (3, 3), activation='relu', padding='same')(c4)
    c4 = BatchNormalization()(c4)
    c4 = Dropout(self.dropout_rate)(c4)
    p4 = MaxPooling2D((2, 2))(c4)

    # Bottleneck
    c5 = Conv2D(self.base_filters * 16, (3, 3), activation='relu', padding='same')(p4)
    c5 = BatchNormalization()(c5)
    c5 = Conv2D(self.base_filters * 16, (3, 3), activation='relu', padding='same')(c5)
    c5 = BatchNormalization()(c5)
    c5 = Dropout(self.dropout_rate)(c5)

    # Decoder - Upscaling
    u6 = Conv2DTranspose(self.base_filters * 8, (2, 2), strides=(2, 2), padding='same')(c5)
    u6 = Concatenate()([u6, c4]) # Concatenate for skip connection
    c6 = Conv2D(self.base_filters * 8, (3, 3), activation='relu', padding='same')(u6)
    c6 = BatchNormalization()(c6)
    c6 = Conv2D(self.base_filters * 8, (3, 3), activation='relu', padding='same')(c6)
    c6 = BatchNormalization()(c6)

    u7 = Conv2DTranspose(self.base_filters * 4, (2, 2), strides=(2, 2), padding='same')(c6)
    u7 = Concatenate()([u7, c3])
    c7 = Conv2D(self.base_filters * 4, (3, 3), activation='relu', padding='same')(u7)
    c7 = BatchNormalization()(c7)
    c7 = Conv2D(self.base_filters * 4, (3, 3), activation='relu', padding='same')(c7)
    c7 = BatchNormalization()(c7)

    u8 = Conv2DTranspose(self.base_filters * 2, (2, 2), strides=(2, 2), padding='same')(c7)
    u8 = Concatenate()([u8, c2])
    c8 = Conv2D(self.base_filters * 2, (3, 3), activation='relu', padding='same')(u8)
    c8 = BatchNormalization()(c8)
    c8 = Conv2D(self.base_filters * 2, (3, 3), activation='relu', padding='same')(c8)
    c8 = BatchNormalization()(c8)

    u9 = Conv2DTranspose(self.base_filters, (2, 2), strides=(2, 2), padding='same')(c8)
    u9 = Concatenate()([u9, c1])
    c9 = Conv2D(self.base_filters, (3, 3), activation='relu', padding='same')(u9)
    c9 = BatchNormalization()(c9)
    c9 = Conv2D(self.base_filters, (3, 3), activation='relu', padding='same')(c9)
    c9 = BatchNormalization()(c9)

    # Output
    outputs = Conv2D(1, (1, 1), activation='sigmoid')(c9)  # Sigmoid activation for prediction

    model = Model(inputs, outputs, name="UNet")
    return model
```

Loss Function

- Binary Cross Entropy
  - Initially I only used BCE
- Dice Loss
- IoU Loss
- Total loss
  - 0.4*bce + 0.4*dice + 0.2*iou





Training Loop

1. Training Function

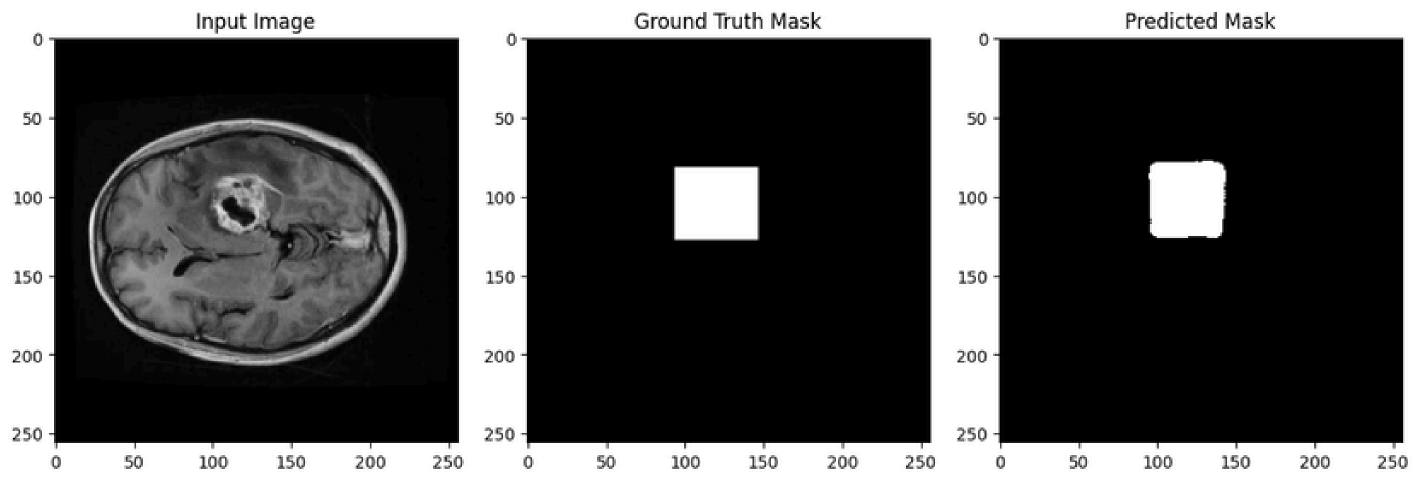
- Adam Optimization Algorithm
- Learning Rate scheduler
- Callbacks
    - Early Stopping
    - Model Checkpoint
        - saves model with best valid acc.

2. Hyperparameters

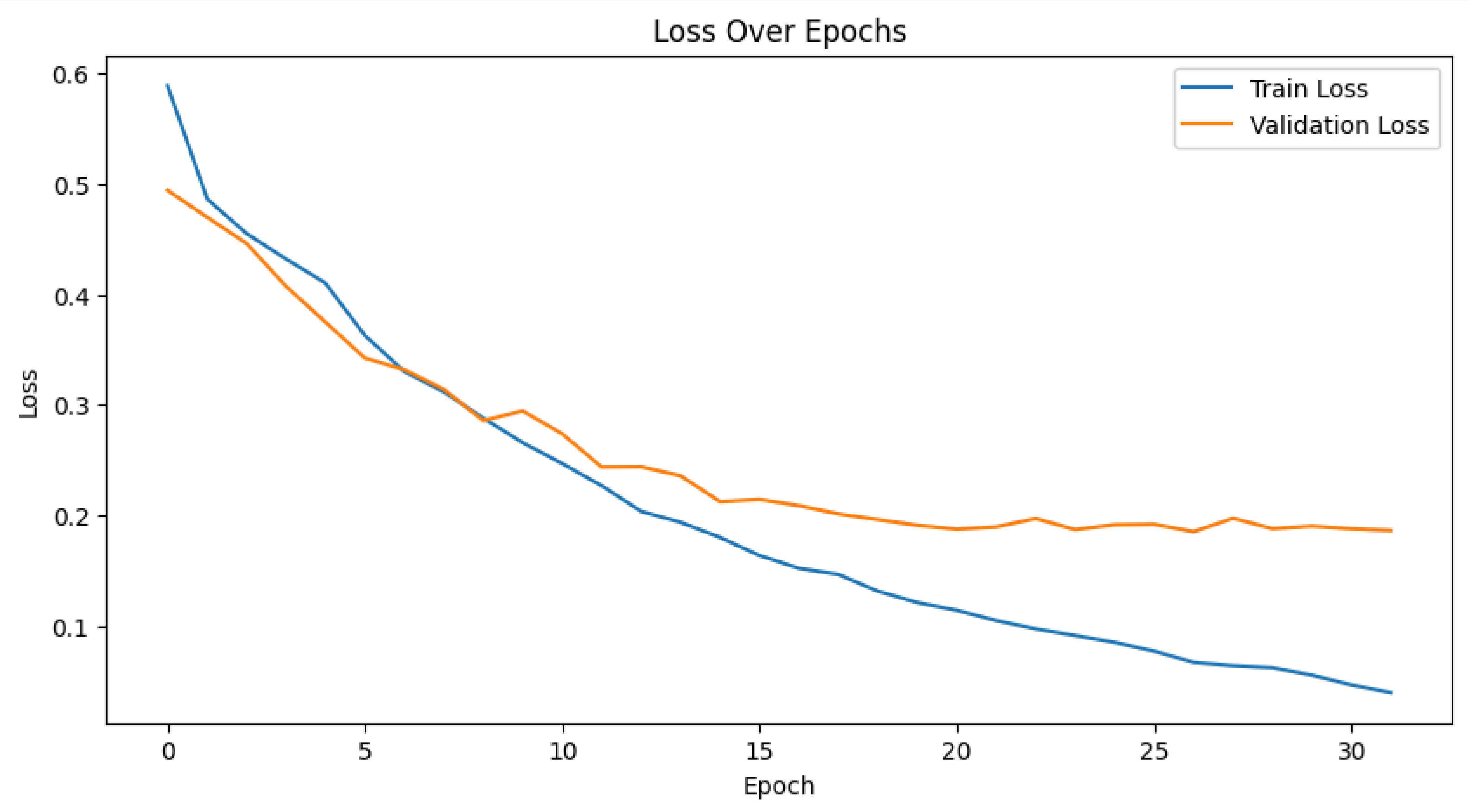
```
INPUT_SIZE = 256
BASE_FILTERS = 64
LEARNING_RATE = 1e-4
EPOCHS = 50
BATCH_SIZE = 16
DROPOUT_RATE = 0.3
DECAY_RATE = 0.9
DECAY_STEPS = 1000
```

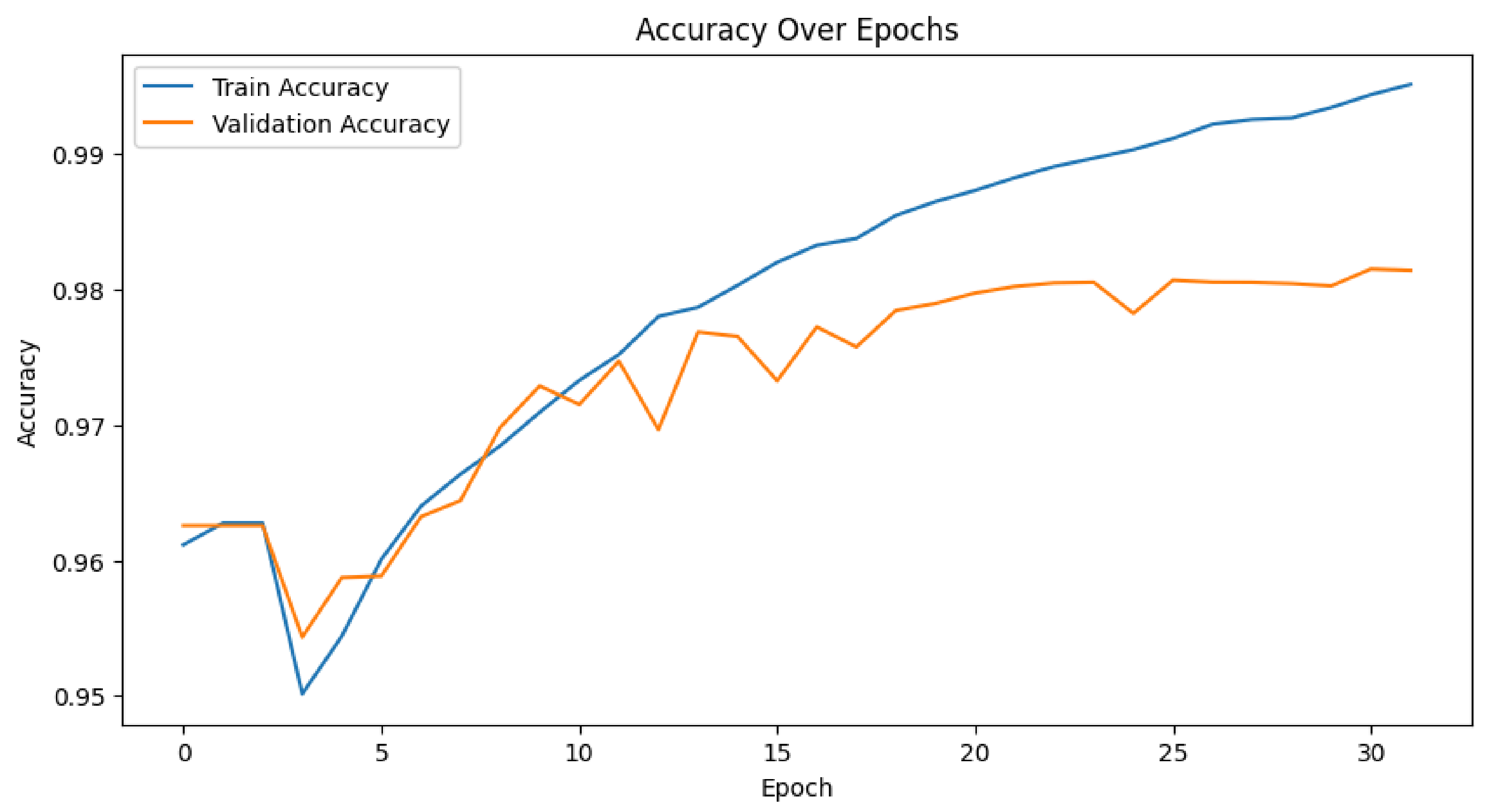
Evaluation and Results

```
Evaluation Metrics:
Loss: 0.2302
Accuracy: 0.9831
IoU: 0.6041
Dice Score: 0.7495
```



- Ultimately achieved a **test accuracy** of 98.31%
- However, Accuracy and Loss plots show telltale signs of overfitting, and segmentations appear "boxy"





Areas of Improvement

1. Data augmentation

- Shifting, rotating, & scaling
- Implement other brain tumor datasets

2. Hyperparameters fine tuning

- Input Filters
- Dropout rate
- Batch Size
- Decay Rate/Initial LR

3. Loss function

- Weigh scores differently

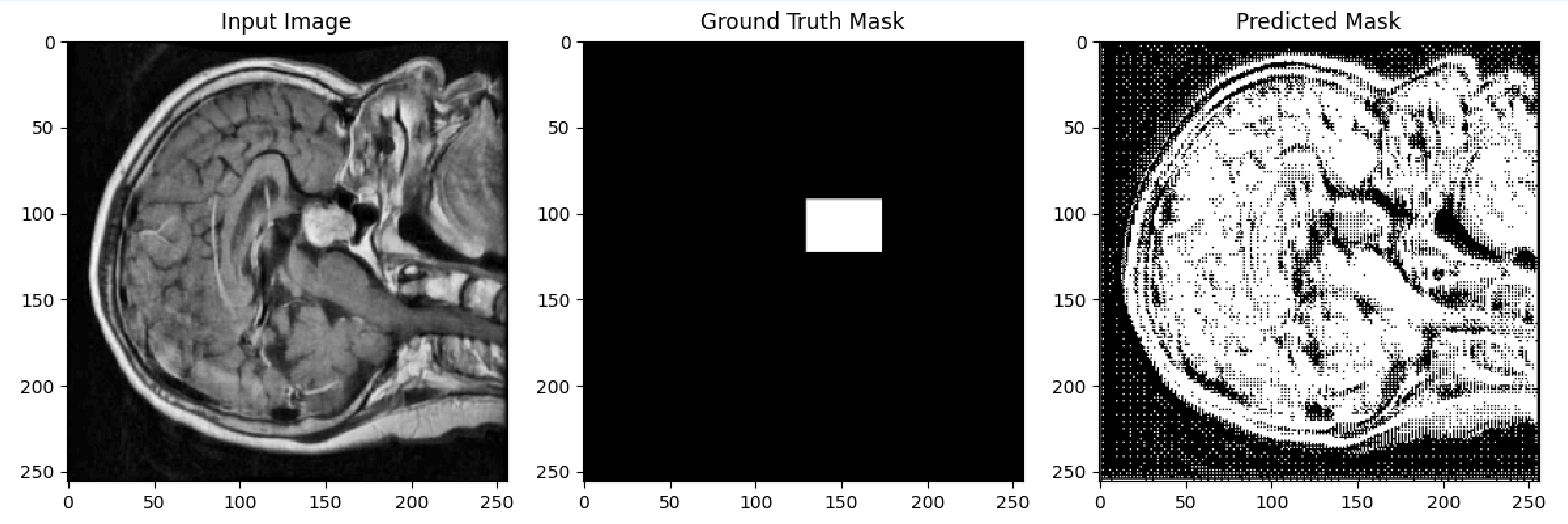
4. Activation functions

- LeakyReLU
- ELU

Thank you

Source code on GitHub:

https://github.com/Aramii0001/UNet_Project/tree/main/src



(It also works for image generation)