Skin Lesion Image Segmentation Using Attention U-Net and MHorUNet

1 Introduction

Skin lesion segmentation is a critical step for the automation of medical diagnosis, and this is especially true in detecting skin cancer. Accurate segmentations delineate the lesions' boundaries, thus enabling informed decisions by clinicians. This project is based on U-Net, Attention U-Net, and MHorUNet architectures. Segmentation performance was improved by incorporating attention mechanisms and novel feature extraction techniques.

2 Background

2.1 Role of U-Net in Medical Image Analysis

The convolutional neural network proposed by Ronneberger et al. in 2015 for biomedical image segmentation is called U-Net. Application of this network in medical image analysis is attributed to some architecture advantages that make feature extraction particularly effective in these complex data sets. Following are the key attributes of U-Net that establish it as important:

- **Encoder-Decoder Architecture:** U-Net possesses an overall symmetric encoder-decoder architecture. It compresses the input image through the encoder to derive high-level features, while it uses the decoder to reconstruct the image in such a way as to ensure meticulous localization of features crucial for segmentation.
- **Skip Connections:** Probably one of the most distinctive features defining the U-Net, skip connections between corresponding encoder and decoder layers are included, as the process of compressing features may discard some information critical for the final segmentation mask. Medical datasets come small and highly specialized. U-Net architecture

MODEL ARCHITECTURE CODE

```python
# Functions to build the encoder path
def conv_block(inp, filters, padding='same', activation='relu'):
    x = Conv2D(filters, (3, 3), padding=padding, activation=activation)(inp)
    x = Conv2D(filters, (3, 3), padding=padding)(x)
    x = BatchNormalization(axis=3)(x)
    x = Activation(activation)(x)
    return x

def encoder_block(inp, filters, padding='same', pool_stride=2, activation='relu'):
    # Encoder block of a UNet passes the result from the convolution block
    # above to a max pooling layer
    x = conv_block(inp, filters, padding, activation)
    p = MaxPooling2D(pool_size=(2, 2), strides=pool_stride)(x)
    return x, p


# Function to build decoder path
def decoder_block(inp,filters,concat_layer,padding='same'):
    # Upsample the feature maps
    x = Conv2DTranspose(filters, (2,2), strides=(2,2), padding=padding)(inp)
    x = concatenate([x, concat_layer]) # Concatenation/Skip conncetion with conjuagte encoder
    x = conv_block(x, filters) # Passed into the convolution block above
    return x

# Building the first block
def build_model(IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS=3):
    inputs = Input((IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS))
    d1, p1=encoder_block(inputs, 64)
    d2, p2=encoder_block(p1, 128)
    d3, p3=encoder_block(p2, 256)
    d4, p4=encoder_block(p3, 512)
    mid = conv_block(p4, 1024) # Midsection
    e2 = decoder_block(mid, 512, d4) # Conjugate of encoder 4
    e3 = decoder_block(e2, 256, d3) # Conjugate of encoder 3
    e4 = decoder_block(e3, 128, d2) # Conjugate of encoder 2
    # o1 = Conv2D(1, (1,1), activation=None)(e4) # Output from 2nd last decoder
    e5 = decoder_block(e4, 64, d1) # Conjugate of encoder 1
    outputs = Conv2D(1, (1, 1),activation='sigmoid')(e5) #Final Output
    model = tf.keras.Model(inputs=[inputs], outputs=[outputs], name='Unet')
    return model
```

Figure 1: U-Net Model Architecture Code

benefits from heavy augmentation and regularization-let's say overfitting in case of small datasets.

•Pixel-Level Predictions: UNet shines where the simple classification models fall behind because it is well suited for dense pixel predictions, providing segmentations in detail that are crucial in such tasks as tumor detection, delineation of organs, and lesion analytics.

The architecture and efficiency set a milestone in the analysis of medical images and placed U-Net as a necessary building block in many subsequent works. (refer to **figure 1** for U-Net Architecture code)

2.2 U-Net improvements: Attention U-Net

Given the success of U-Net, Attention U-Net further improves segmentation accuracy and robustness, embedding attention mechanisms within. The main novelties are attention gates that now promptly reinforce the feature maps by underlining the most relevant regions and suppress unnecessary and superfluous information. Among others, the main advantages of Attention U-Net are:

• Global Context Modeling: An attention gate combines contextual information, enhancing the global competence of the model in dealing with a variety of lesion shapes, textures, and sizes-abundant challenges in medical image segmentation.

• Stronger Metrics: AGs amplify the sensitivity (true positive rate) and specificity (true negative rate) of the network, crucial to ensure the number of false positives and negatives are few in medical diagnosis.

Attention U-Net significantly improves the generic version of U-Net with improvements in the shortcomings while extending the range of applications to more complicated and diverse datasets.

please refer to **Figure 2 and 3** below for the code of the model architecture.

MODEL ARCHITECTURE CODE

```python
# Functions to build the encoder path
def conv_block(inp, filters, padding='same', activation='relu'):
    x = Conv2D(filters, (3, 3), padding=padding, activation=activation)(inp)
    x = Conv2D(filters, (3, 3), padding=padding)(x)
    x = BatchNormalization(axis=3)(x)
    x = Activation(activation)(x)
    return x

def encoder_block(inp, filters, padding='same', pool_stride=2, activation='relu'):
    # Encoder block of a UNet passes the result from the convolution block
    # above to a max pooling layer
    x = conv_block(inp, filters, padding, activation)
    p = MaxPooling2D(pool_size=(2, 2), strides=pool_stride)(x)
    return x, p

def attention_gate(skip_connection, gating_signal, filters, padding='same'):
    """
    Compute attention weights for a skip connection in the U-Net.
    """
    # Downsample skip connection to match gating signal dimensions
    theta_x = Conv2D(filters, (1, 1), strides=(1, 1), padding=padding)(skip_connection)
    # Downsample gating signal
    phi_g = Conv2D(filters, (1, 1), strides=(1, 1), padding=padding)(gating_signal)

    # Upsample gating signal to match skip connection dimensions
    phi_g = tf.keras.layers.UpSampling2D(size=(theta_x.shape[1] // phi_g.shape[1],
                                               theta_x.shape[2] // phi_g.shape[2]))(phi_g)

    # Combine the skip connection and gating signal
    add = tf.keras.layers.Add()([theta_x, phi_g])
    # Apply activation
    act = Activation('relu')(add)
    # Compute attention weights
    psi = Conv2D(1, (1, 1), strides=(1, 1), padding=padding)(act)
    psi = Activation('sigmoid')(psi)
    # Upsample the attention weights to match skip connection dimensions
    psi = tf.keras.layers.UpSampling2D(size=(skip_connection.shape[1] // psi.shape[1],
                                             skip_connection.shape[2] // psi.shape[2]))(psi)
    # Apply attention weights
    out = Multiply()([skip_connection, psi])
```

Figure 2: Attention U-Net Model Architecture Code-Part1

```
    # Apply attention weights
    out = Multiply()([skip_connection, psi])
    return out



def decoder_block_with_attention(inp, filters, concat_layer, gating_signal, padding='same'):
    # Apply the attention gate to the encoder features
    attention_output = attention_gate(concat_layer, gating_signal, filters, padding)
    # Upsample the feature maps
    x = Conv2DTranspose(filters, (2, 2), strides=(2, 2), padding=padding)(inp)
    # Concatenate the attention-modulated encoder features
    x = concatenate([x, attention_output])
    # Pass through the convolution block
    x = conv_block(x, filters)
    return x

def build_model_with_attention(IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS=3):
    inputs = Input((IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS))
    # Encoder path
    d1, p1 = encoder_block(inputs, 64)
    d2, p2 = encoder_block(p1, 128)
    d3, p3 = encoder_block(p2, 256)
    d4, p4 = encoder_block(p3, 512)
    # Bottleneck
    mid = conv_block(p4, 1024)
    # Decoder path with attention gates
    e2 = decoder_block_with_attention(mid, 512, d4, mid)
    e3 = decoder_block_with_attention(e2, 256, d3, e2)
    e4 = decoder_block_with_attention(e3, 128, d2, e3)
    e5 = decoder_block_with_attention(e4, 64, d1, e4)
    # Final output
    outputs = Conv2D(1, (1, 1), activation='sigmoid')(e5)
    model = tf.keras.Model(inputs=[inputs], outputs=[outputs], name='Attention_Unet')
    return model
```

Figure 3: Attention U-Net Model Architecture Code-Part2

2.3 Understanding MHorUNet

Besides Attention U-Net, the performance of the MHorUNet is under focus; it is among those advanced segmentation frameworks developed for medical image analysis. The MHorUNet utilized state-of-the-art methodologies for the successful analysis of complex medical images. This is realized in the following novelties:

• Depthwise separable convolutional blocks: Unlike standard convolutional blocks, depthwise separable blocks help save lots of computational overhead while maintaining critical spatial information. This process is computationally very efficient.

• Attention bridges: Attention bridges between encoder-decoder paths dynamically weight the feature maps and emphasize clinically relevant information.

• Global-local filtering: Combining the two-a global frequency-domain with a local spatial-domain feature extraction-allows MHorUNet to capture the input data holistically and thus capture the variation in texture, shape, and intensity.

Such a fusion of traits renders MHorUNet an attractive architecture for the application of medical segmentations since it possesses remarkable capabilities in handling complex datasets that yield state-of-the-art results.

3 Methodology

3.1 Dataset

The ISIC 2018 dataset was used, comprising dermoscopic images with annotated lesion masks. The dataset was split into training, validation, and test sets.

- **Training Data**: 2075 images
- **Validation Data**: 519 images
- **Testing Data**: 1003 images. No ground truth provided; predictions were only inspected.

  Refer to **figure 4** below for a sample of the training images and its masks.

Figure 4: Training images and its ground truths

3.2 Preprocessing

- Images and labels were resized and normalized.
- Augmentation techniques (rotation, flipping, scaling) were applied using Albumentations to improve generalization. (refer **figure 5** for Data Augmentation Code)

3.3 Model Architectures

3.3.1 Attention U-Net

Attention U-Net was implemented with:

- **Encoder**: Sequential convolutions with ReLU activation and max pooling.
- **Attention Mechanisms**: Attention gates refine features before passing them to the decoder.
- **Decoder**: Transposed convolutions for upsampling with refined skip connections.
- **Batch Normalization**: Stabilizes and accelerates training.

```python
def augment(image, mask, size=(256, 256)):
    transforms = A.Compose([
        A.OneOf([
            A.Transpose(),
            A.VerticalFlip(),
            A.HorizontalFlip(),
            A.RandomRotate90(),
            A.NoOp()], p=0.75),

        A.ShiftScaleRotate(p=0.1),
        A.GridDistortion(p=0.1),
        A.ElasticTransform(p=0.1),

        A.OneOf([
            A.RandomBrightnessContrast(),
            A.RandomGamma()], p=0.2)

      ], additional_targets={'mask': 'mask'})

    data = {"image":image, "mask":mask}
    aug_data = transforms(**data)

    aug_img = aug_data["image"]
    aug_img = tf.cast(aug_img, tf.float32)

    aug_mask = aug_data["mask"]
    # Making sure labels are binary (background or label)
    aug_mask = np.where(aug_mask == 0, 0, 1)
    aug_mask = tf.cast(aug_mask, tf.int32)
    return aug_img, aug_mask
```

Figure 5: Data Augmentation Code

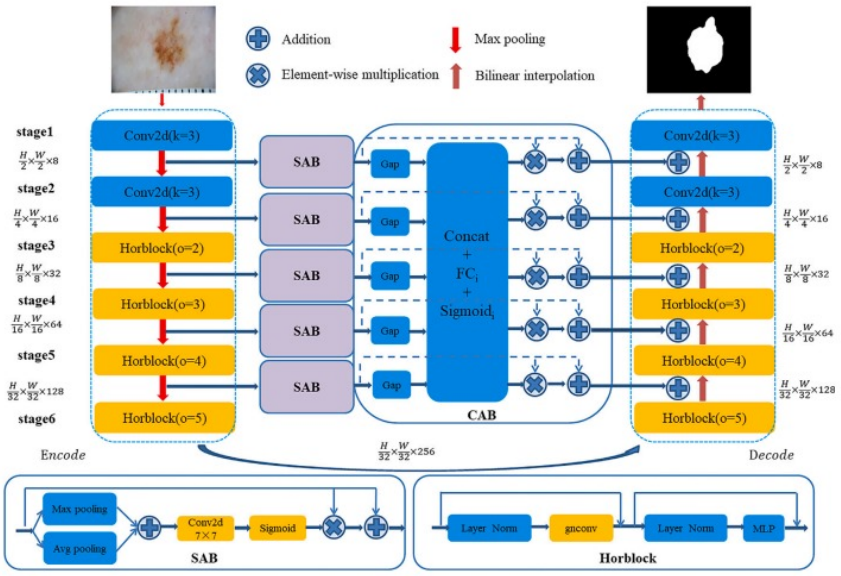

Fig. 2. The architecture overview of our MHorUNet. The overall structure is a UNet model framework, Horblock is a module with high-order interaction of gnconv, skip connection part using multi-stage and multi-scale fusion mechanism.

Figure 6: MHorUNet Architectural Design

3.3.2 MHorUNet

MHorUNet incorporates:

• Depthwise Separable Convolutions: Efficient spatial feature extraction with reduced computational cost.

• SC_Att_Bridge: Integrates channel and spatial attention for better localization.

• Global-Local Filtering: Combines spatial and frequency-domain features using Fourier Transforms.

• Residual HorBlocks: Adaptive feature extraction with enhanced depth and complexity.

Refer to **figure 6** above:

3.4 Loss Function

The loss function combined:

- **Binary Cross-Entropy Loss**: For pixel-wise classification.
- **Dice Loss**: To address class imbalance and improve overlap accuracy.

3.5 Evaluation Metrics

- **Dice Coefficient**: Measures overlap between predicted and true masks.
- **Intersection over Union (IoU)**: Evaluates segmentation accuracy.
- **Sensitivity and Specificity**: Captures the true positive rate and false positive reduction.

4 Experiments

4.1 Training Setup (U-Net and Attention U-Net)

- **Callbacks**: EarlyStopping and ModelCheckpoint ensured optimal training while preventing overfitting.
- **Optimizer**: Adam optimizer with a learning rate scheduler. (initial learning rate of **0.001**)
- **Epochs**: Both models were trained for 35 epochs with a batch size of 16.

4.2 MHorUNet: Challenges

Training MHorUNet presented unique challenges:

- **Initial Challenges**: Early training showed stagnation in performance metrics, likely due to the model's complexity and sensitivity to class imbalance.
- **Adjustments**: By reducing the learning rate and simplifying the SC_Att_Bridge module, the model began showing improvements. Eventually, it started overfitting once again.

  NOTE: It seems the model is too complex for the data or the training setup, leading to suboptimal performance. Simplifying the architecture, addressing class imbalance, and carefully tuning hyperparameters are likely to yield improvements.

4.3 Performance Evaluation(U-Net and Attention U-Net)

- U-Net achieved a Dice Coefficient of .83 and IoU of 0.73 on the validation set.

- Attention U-Net achieved a Dice Coefficient of .84 and IoU of 0.72 on the validation set.

Table 1: Comparison of Metrics Between U-Net and Attention U-Net

Metric	U-Net	Attention U-Net
Accuracy	92.83%	93.28%
Dice Coefficient	0.8349	0.8375
IoU	0.7283	0.7239
Specificity	93.5%	95.68%
Sensitivity	84.0%	84.19%

please see **figure 7 and 8** below for the visualizations of Evaluation metrics:

5 Challenges and Known Issues

- Attention U-Net had difficulty handling very small lesions.
- MHorUNet's complexity initially led to poor convergence, requiring extensive tuning.

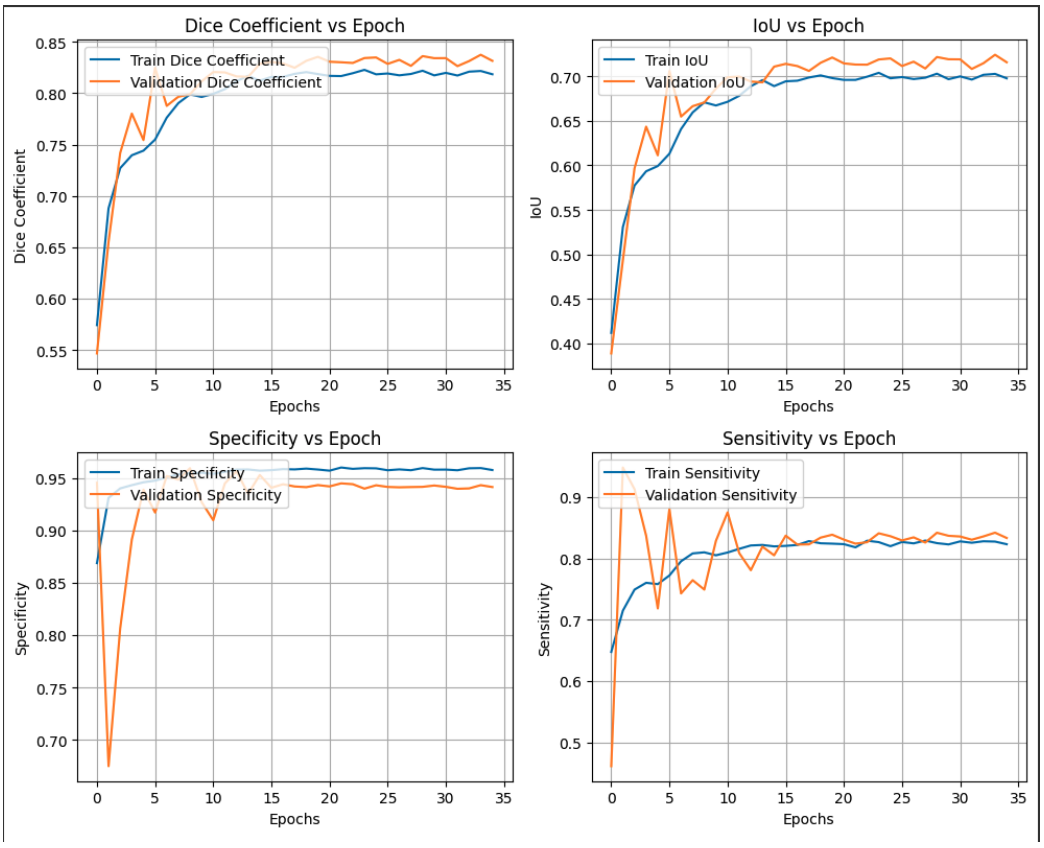

Figure 7: Visualization of Model Performance over epochs

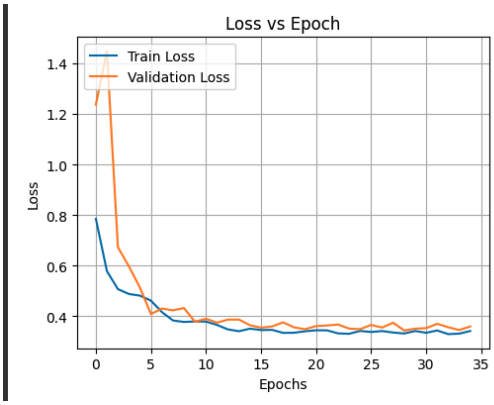

Figure 8: Visualization of Model Performance over epochs Contd..

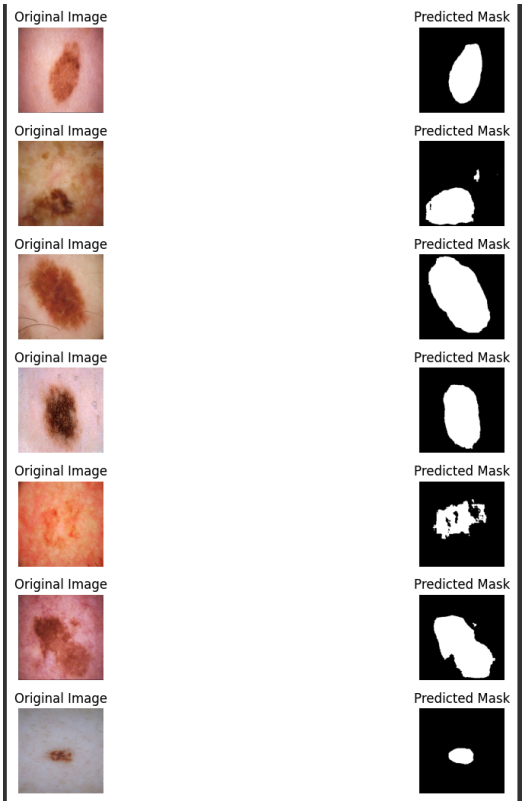

Figure 9: Predictions of the final model

6 Results

Figure 9 (above) displays some of the predicted masks:

7 Conclusion

Due to huge number of errors in re-implementing the complex architecture of the MHorUNet, we decided to drop the experiment and conclude with the Attention U-Net as our final model.

The Attention U-Net model successfully segments skin lesions, demonstrating the potential of attention mechanisms in medical image analysis.

Future work may involve addressing class imbalance and incorporating test ground truth for evaluation