Medical Image Segmenation Computer Vision

Matthew Yeseta

This project aims to implement an enhanced DeepLab v3+ deep learning network for medical image segmentation, utilizing a ResNet101 backbone and incorporating key architectural improvements for better performance. This comprehensive approach aims to improve segmentation accuracy for tounge, as higly difficult detection in medical images> Leveraging state-of-the-art techniques in deep learning DeepLabv3+ and computer vision.

DeepLab v3+ Model Construction:

Using ResNet101 as the backbone to provide strong feature extraction capabilities.

Efficient Channel Attention (ECA):

Integrating the ECA module into the residual blocks of ResNet101 to improve the network's ability to capture important features.

Feature Pyramid Networks (FPN):

Applied FPN to fuse features across different layers, enhancing multi-scale feature representation.

- **Squeeze-and-Excitation (SE)**: Incorporating SE blocks to apply adaptive channel attention, further refining the model's ability to focus on critical features.

Atrous Spatial Pyramid Pooling (ASPP):

Applied ASPP to capture multi-scale features through dilated convolutions, improving segmentation accuracy on objects of varying sizes.

Up-Sampling Strategy:

Replacing traditional up-sampling with two 2-fold up-sampling operations to improve pixel continuity and ensure smoother boundaries in segmentation results.

Dice Loss Function:

Implementing Dice Loss to optimize the model for medical image segmentation, where overlap between predicted and true regions is critical.

Stochastic Gradient Descent (SGD):

Training the model using the SGD optimizer to ensure efficient convergence and better generalization.

In [21]:
```
import os
import glob
import random
import tensorflow as tf
from tensorflow.keras import layers, models, optimizers
from tensorflow.keras.losses import Loss
from tensorflow.keras.applications import ResNet101
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
```

In [35]:
```
import tensorflow as tf
from tensorflow.keras import layers, models, optimizers
from tensorflow.keras.losses import Loss
from tensorflow.keras.applications import ResNet101

# Efficient Channel Attention (ECA) module
class ECALayer(layers.Layer):
    def __init__(self, gamma=2, b=1):
        super(ECALayer, self).__init__()
        self.gamma = gamma
        self.b = b

    def build(self, input_shape):
        t = int(abs((tf.math.log(tf.cast(input_shape[-1], tf.float32))) / tf.math.log(2.0) + self.b) / self.gamma)
        self.k = t if t % 2 else t + 1
        se;f.k = 20
        self.avg_pool = layers.GlobalAveragePooling2D()
        self.conv = layers.Conv1D(1, kernel_size=self.k, padding="same", use_bias=False)

    def call(self, x):
        y = self.avg_pool(x)
        y = tf.expand_dims(y, axis=-1)
        y = self.conv(y)
        y = tf.nn.sigmoid(y)
```

```
        y = tf.expand_dims(y, axis=-1)
        return x * y

# Atrous Spatial Pyramid Pooling (ASPP) for DeepLabV3+
def ASPP(x, filters=256):
    dims = tf.keras.backend.int_shape(x)

    pool1 = layers.GlobalAveragePooling2D()(x)
    pool1 = layers.Reshape((1, 1, dims[-1]))(pool1)
    pool1 = layers.Conv2D(filters, 1, padding="same", use_bias=False)(pool1)
    pool1 = layers.BatchNormalization()(pool1)
    pool1 = layers.ReLU()(pool1)
    pool1 = layers.UpSampling2D((dims[1], dims[2]), interpolation="bilinear")(pool1)

    conv1 = layers.Conv2D(filters, 1, padding="same", use_bias=False)(x)
    conv1 = layers.BatchNormalization()(conv1)
    conv1 = layers.ReLU()(conv1)

    conv2 = layers.Conv2D(filters, 3, padding="same", dilation_rate=6, use_bias=False)(x)
    conv2 = layers.BatchNormalization()(conv2)
    conv2 = layers.ReLU()(conv2)

    conv3 = layers.Conv2D(filters, 3, padding="same", dilation_rate=12, use_bias=False)(x)
    conv3 = layers.BatchNormalization()(conv3)
    conv3 = layers.ReLU()(conv3)

    conv4 = layers.Conv2D(filters, 3, padding="same", dilation_rate=18, use_bias=False)(x)
    conv4 = layers.BatchNormalization()(conv4)
    conv4 = layers.ReLU()(conv4)

    conv5 = layers.Conv2D(filters, 3, padding="same", dilation_rate=24, use_bias=False)(x)

    return layers.Concatenate()([pool1, conv1, conv2, conv3, conv4])

# Feature Pyramid Network (FPN) for feature fusion
def FPN(x, filters=256):
    # Start from the highest-level feature map (smallest spatial resolution)
    p5 = layers.Conv2D(filters, kernel_size=1)(x[3])  # Highest-level feature map
    p5_up = layers.UpSampling2D(size=(8, 8))(p5)  # Upsample to match the lowest level (64x64)

    p4 = layers.Conv2D(filters, kernel_size=1)(x[2])  # Second-highest level
    p4 = layers.Add()([p4, layers.UpSampling2D(size=(2, 2))(p5)])  # Upsample p5 to match p4 and add
    p4_up = layers.UpSampling2D(size=(4, 4))(p4)  # Upsample p4 to match the lowest level (64x64)

    p3 = layers.Conv2D(filters, kernel_size=1)(x[1])  # Third-highest level
```

```
    p3 = layers.Add()([p3, layers.UpSampling2D(size=(2, 2))(p4)])  # Upsample p4 to match p3 and add
    p3_up = layers.UpSampling2D(size=(2, 2))(p3)  # Upsample p3 to match the lowest level (64x64)

    p2 = layers.Conv2D(filters, kernel_size=1)(x[0])  # Lowest-level feature map
    p2 = layers.Add()([p2, p3_up])  # Add p3 directly to p2

    # Now concatenate the upsampled feature maps
    return layers.Concatenate()([p2, p3_up, p4_up, p5_up])

# DeepLabV3+ model with ResNet101 backbone and feature fusion
def DeepLabV3Plus(input_shape=(512, 512, 3), num_classes=21):
    inputs = layers.Input(shape=input_shape)

    # Load ResNet101 backbone with pre-trained weights
    base_model = ResNet101(include_top=False, weights="imagenet", input_tensor=inputs)

    # Extract specific layers for feature fusion
    layer_names = ["conv2_block3_out", "conv3_block4_out", "conv4_block23_out", "conv5_block3_out"]
    layers_out = [base_model.get_layer(name).output for name in layer_names]

    # Low-level feature (LF) from ResNet101 (conv2_block3_out)
    LF = layers_out[0]  # This corresponds to the lower-level feature map (conv2_block3_out)

    # Add FPN for feature fusion
    fpn_output = FPN(layers_out)

    # Add ASPP module for High-Level Feature (HF)
    aspp_output = ASPP(fpn_output)  # High-level feature output from ASPP

    # 1x1 convolution to reduce high-level feature (HF) dimensionality
    HF_conv = layers.Conv2D(256, kernel_size=(1, 1), padding="same", use_bias=False)(aspp_output)

    # First 2-fold up-sampling of HF
    HF_upsampled_1 = layers.UpSampling2D(size=(2, 2), interpolation="bilinear")(HF_conv)

    # Align spatial dimensions of fpn_output to match HF_upsampled_1
    fpn_output_upsampled = layers.UpSampling2D(size=(2, 2), interpolation="bilinear")(fpn_output)

    # Concatenate HF with Feature Fusion (FF)
    FF_concat = layers.Concatenate()([HF_upsampled_1, fpn_output_upsampled])

    # Second 2-fold up-sampling
    HF_upsampled_2 = layers.UpSampling2D(size=(2, 2), interpolation="bilinear")(FF_concat)

    # 1x1 convolution on Low-Level Feature (LF)
```

```
    LF_conv = layers.Conv2D(48, kernel_size=(1, 1), padding="same", use_bias=False)(LF)

    # Upsample LF to match the dimensions of HF_upsampled_2
    LF_upsampled = layers.UpSampling2D(size=(4, 4), interpolation="bilinear")(LF_conv)

    # Concatenate the 4x upsampled HF with LF
    LF_HF_concat = layers.Concatenate()([HF_upsampled_2, LF_upsampled])

    # Perform 3x3 convolution on the concatenated output
    x = layers.Conv2D(256, kernel_size=(3, 3), padding="same", use_bias=False)(LF_HF_concat)
    x = layers.BatchNormalization()(x)
    x = layers.ReLU()(x)

    # Final 4-fold up-sampling to match original input size
    x = layers.UpSampling2D(size=(4, 4), interpolation="bilinear")(x)

    # Output layer
    outputs = layers.Conv2D(num_classes, kernel_size=(1, 1), activation="softmax")(x)

    model = models.Model(inputs, outputs)
    return model

# DeepLabV3Plus model class with image display functionality
class DeepLabV3PlusModel:
    def __init__(self, input_shape=(512, 512, 3), num_classes=21):
        self.model = DeepLabV3Plus(input_shape=input_shape, num_classes=num_classes)
        self.model.compile(optimizer=optimizers.SGD(learning_rate=1e-3, momentum=0.9),
                           loss=DiceLoss(),
                           metrics=["accuracy"])

    def load_image(self, image_path, target_size=(512, 512)):
        image = Image.open(image_path).convert("RGB")
        image = image.resize(target_size)
        image_array = np.array(image) / 255.0
        return image_array

    def predict_and_display_images(self, image_dir, num_images=5):
        image_files = [f for f in os.listdir(image_dir) if f.endswith(".bmp")]
        selected_images = image_files[:num_images]

        fig, axs = plt.subplots(num_images, 2, figsize=(10, num_images * 5))

        for i, image_file in enumerate(selected_images):
            image_path = os.path.join(image_dir, image_file)
            img = self.load_image(image_path)
```

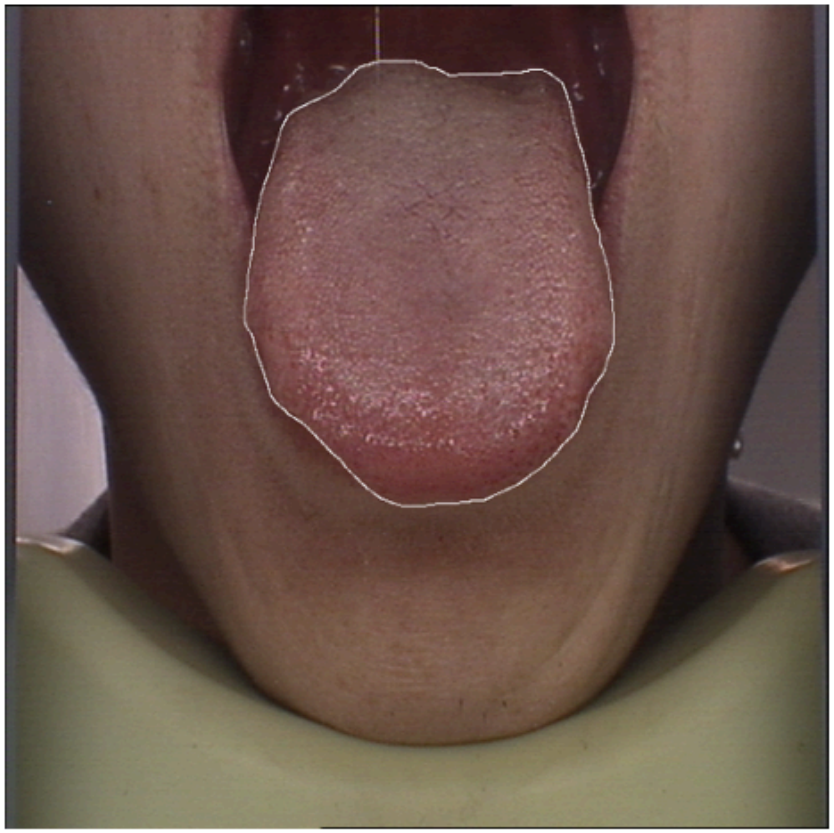
```
            # Predict the mask
            img_input = np.expand_dims(img, axis=0)
            pred_mask = self.model.predict(img_input)

            # Show the original image
            axs[i, 0].imshow(img)
            axs[i, 0].set_title(f"Original Image: {image_file}")
            axs[i, 0].axis("off")

            # Show the predicted mask
            axs[i, 1].imshow(np.argmax(pred_mask[0], axis=-1))
            axs[i, 1].set_title(f"Predicted Mask: {image_file}")
            axs[i, 1].axis("off")

        plt.tight_layout()
        plt.show()

# Dice loss function
class DiceLoss(Loss):
    def call(self, y_true, y_pred):
        numerator = 2 * tf.reduce_sum(y_true * y_pred)
        denominator = tf.reduce_sum(y_true + y_pred)
        return 1 - (numerator + 1) / (denominator + 1)

# Compile and train model
def train_deeplabv3plus(train_data, val_data):
    # Instantiate the DeepLabV3Plus model
    model = DeepLabV3Plus(input_shape=(512, 512, 3), num_classes=21)

    # Optimizer and compile model
    optimizer = tf.keras.optimizers.Adam(learning_rate=1e-4)
    model.compile(optimizer=optimizer,  # Replaced 'optimizers' with 'optimizer'
                  loss=DiceLoss(),
                  metrics=["accuracy"])

    # EarlyStopping callback to prevent overfitting
    early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

    # ReduceLROnPlateau callback to reduce learning rate when validation loss plateaus
    reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=5, min_lr=1e-6)

    # Train the model with early stopping and learning rate reduction
    history = model.fit(
        train_data,
```

```
        validation_data=val_data,
        epochs=120,
        callbacks=[early_stopping, reduce_lr],
        batch_size=16,
        verbose=1
    )

    return model
```

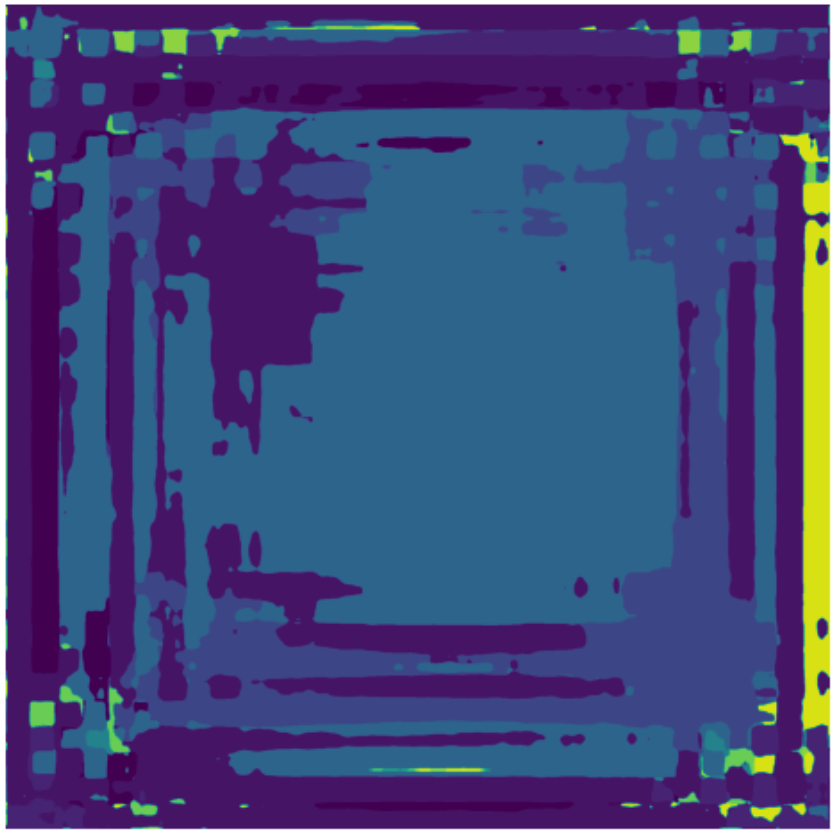
In [36]:
```
if __name__ == '__main__':
    working_directory = setPath()
    g_truth_path = os.path.join(working_directory, 'data-backup', 'groundtruth', 'image')
    g_truth_masked_mask = os.path.join(working_directory, 'data-backup', 'mask')
    image_files = glob.glob(os.path.join(g_truth_path, "*.bmp")) + glob.glob(os.path.join(g_truth_path, "*.BMP"))
    mask_files = glob.glob(os.path.join(g_truth_masked_mask, "*.bmp")) + glob.glob(os.path.join(g_truth_masked_mask, "*
    image_names = {os.path.basename(f).split('.')[0] for f in image_files}
    mask_names = {os.path.basename(f).split('.')[0] for f in mask_files}


    model = DeepLabV3PlusModel(input_shape=(512, 512, 3), num_classes=21)
    # Predict and display images
    model.predict_and_display_images(g_truth_path, num_images=5)
```

```
Current Working Directory: C:\Users\matth\Documents\master-degree\IOT
1/1 [==============================] - 74s 74s/step
1/1 [==============================] - 71s 71s/step
1/1 [==============================] - 74s 74s/step
1/1 [==============================] - 60s 60s/step
1/1 [==============================] - 61s 61s/step
```

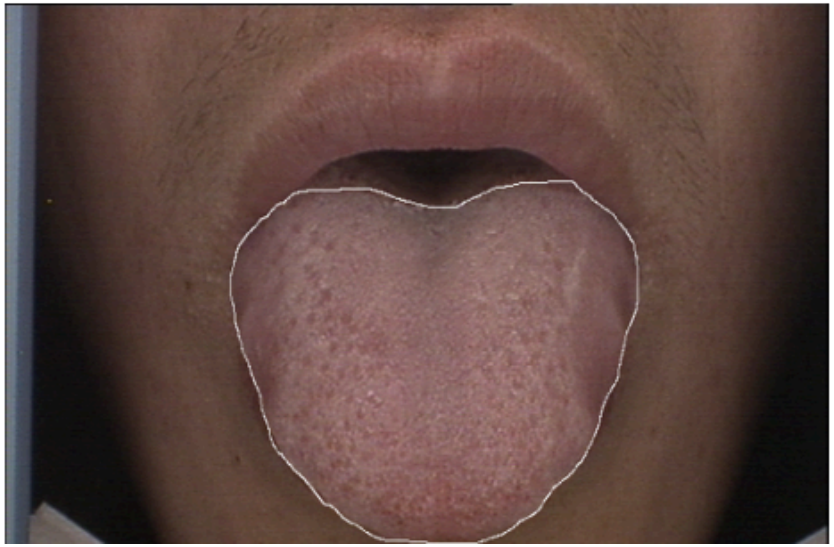
Original Image: 1.bmp



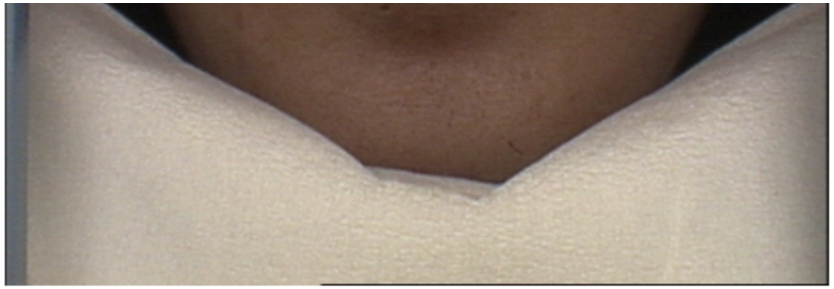
Predicted Mask: 1.bmp



Original Image: 10.bmp

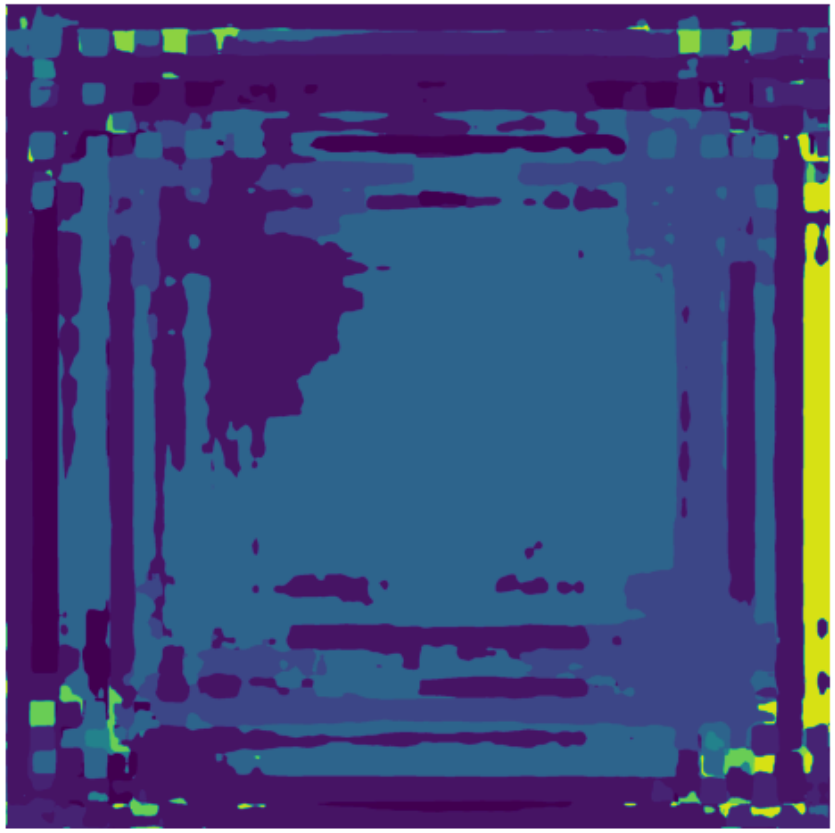


Predicted Mask: 10.bmp



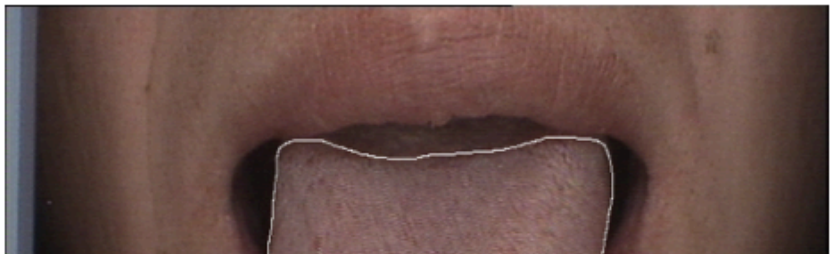
Original Image: 100.bmp

Predicted Mask: 100.bmp



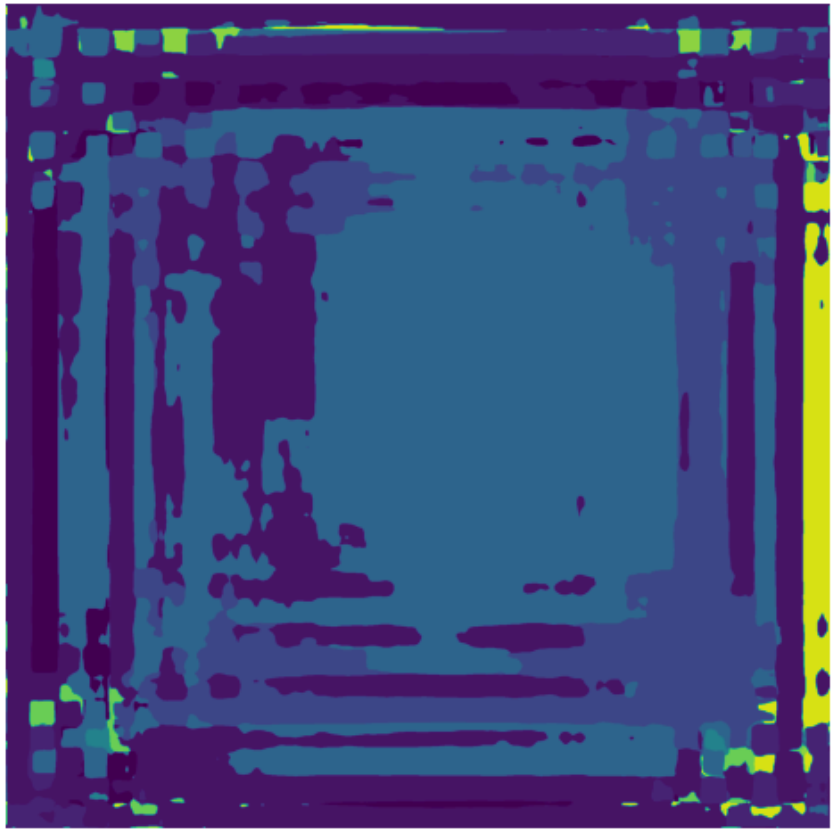
Original Image: 101.bmp

Predicted Mask: 101.bmp



Original Image: 102.bmp

Predicted Mask: 102.bmp



In [ ]:
```
model.summary()
```

In [ ]:

In [ ]: