Image Processing Project Report

Team Members:

Kheira Ab	2021170178
Marwa Ehab	2021170503
Salma Nasr-Eldin	2021170233
Salma Morad	2021170234
Smaher Nabil	2021170237
Zeinab Sakran	2021170219

Instructors: Dr Basant

16/12/2024

Introduction

For this image segmentation project, we implemented 2 main models: U-net and attention U-net, and attempted to implement the following models as well: PSPNet and DeepLabv3Plus.

This report will serve as a guide to the code, explaining the reason behind each part.

Importing, Preprocessing and Data Augmentation

First, we imported all required libraries. We made sure to set all possible random seeds to a chosen number because there was a huge difference in training accuracies between each run.

In the preprocessing steps, we created the `rgb_to_class_indices` function because it's necessary to convert RGB tuples into indices for using the U-Net architecture. Next, we created the `preprocess` function which resizes all input images to (128, 128, 3) and normalizes them to make them better fit for training. On our model, IMG_SIZE =128 worked best.

For data augmentation, we created the function `augment_image_and_label` that does the following transformations to augmented images: rotation, shearing, horizontal flipping and zooming. As a last step, it resizes the images to 128 to ensure consistency in the dataset. This helped double our training dataset size from 200 samples to 400. Note that data augmentation is only implemented for the training and not validation datasets to prevent data leakage.

All the previously mentioned functions are called inside one function called `load_data`, which is called twice, one for training data and once for validation.

Finally, we change our data type to tensorflow datasets, which we will use for model inputs. We also set batch_size to 8 and shuffle to 150, which were the best values found from trial and error.

U-Net Model

The U-Net model takes advantage of local feature extraction using traditional CNN architecture in the contracting path (`encoder_block`). It consists of the repeated application of two 3x3 convolutions where padding = same (`double_conv_block`), each followed by a rectified linear unit (ReLU) and a 2x2 max pooling operation with stride 2 for downsampling. Then there is an expansive path (`decoder_block`) which mirrors the contracting path, upsamples the features using transposed convolutions (deconvolutions) of size (2,2) and stride 2 and combines them with corresponding features from the contracting path via skip connections. These skip connections preserve spatial information and fine-grained details, leading to accurate pixel-level segmentation.

To prevent overfitting, we add L2 regularizer parameter to Conv2D (in encoder) and Conv2DTranspose (in decoder) and set it to 0.0001, and we also add a dropout layer before passing the output to the final softmax predicting layer.

Attention U-Net Model

The core idea behind Attention U-Net is to allow the network to focus on the most relevant parts of the input image during the segmentation process. This is achieved by using attention gates, which learn to highlight important features and suppress irrelevant ones.

The `attention_gate` function does this by taking the encoder's feature map (x) and the decoder's feature map (g) as inputs. It first reduces the channel dimensions of both input feature maps using 1x1 convolutional layers. The resulting feature maps are then added together and passed through a ReLU activation function. Then, a 1x1 convolutional layer is applied to generate an attention map, which is then normalized using a sigmoid activation function. Finally, the attention map is multiplied element-wise with the original feature map from the encoder.

The `decoder_blockwith_attention` function first upsamples the input features using a transposed convolution layer. It then passes the upsampled features and the corresponding skip connection features through an attention gate to generate an attention map. This attention map is used to modulate the upsampled features, emphasizing the most relevant information. Finally, the attention-modulated features are concatenated with the skip connection features and further refined by a double convolution block.

PSPNet Model

PSPNet, or Pyramid Scene Parsing Network, is a powerful semantic segmentation model that excels at capturing contextual information at multiple scales by employing a pyramid pooling module.

The Pyramid Pooling Module (PPM) is a powerful technique for capturing multi-scale contextual information in semantic segmentation. It works by applying average pooling with different kernel sizes to the input feature maps. This results in multiple feature maps, each capturing information at a different scale. These feature maps are then upsampled to the original size and concatenated with the original feature map. This fusion of multi-scale information allows the network to better understand the global context and make more accurate segmentation decisions.

The PSPNet architecture typically uses a state-of-the-art convolutional neural network as its backbone to extract high-level feature maps. These feature maps are then fed into the pyramid pooling module, which processes them at different resolutions to capture both fine-grained details and global context. The resulting feature maps are subsequently fused and upsampled to generate the final segmentation map.

To input data into the model, we wanted our input images to be the same size of the backbone model (ResNet50) that's used to train our PSPNet model. We tried changing the size of our dataset images from 128 (current size) to 224 using 2 ways: resizing and padding. For the model, we tried different values for pool_size and Conv2D filter number and sizes in each block.

DeepLabv3Plus Model

It introduces an encoder-decoder architecture to refine segmentation results, especially at object boundaries. The encoder utilizes dilated convolutions to capture multi-scale contextual information, while the decoder upsamples features and combines them with low-level features from the encoder to produce detailed segmentation maps.

Dilated convolutions are a type of convolution operation that expands the receptive field of a convolutional layer without increasing the number of parameters or the size of the filter. This is achieved by inserting spaces between the filter elements, allowing the filter to cover a larger area of the input image.

The provided code implements the DeepLabv3+ architecture using ResNet50 as the backbone. The model first extracts high-level feature maps from the input image using the ResNet50 backbone. An Atrous Spatial Pyramid Pooling (ASPP) module is then applied to these feature maps to capture multi-scale contextual information. The outputs of the ASPP module are concatenated and reduced to a single feature map. This feature map is then upsampled and combined with low-level features from the backbone through a decoder network. Finally, a 1x1 convolutional layer with softmax activation is used to generate the final segmentation map.

Grid Search Code

We implemented grid search to find the best hyperparameters (learning rate, decay steps and decay rate) using tf.keras.optimizers.schedules.ExponentialDecay. Each iteration of combination of these hyperparameters finds the validation accuracy after training the model for 50 epochs. This ofcourse led it to have a very large training time, which is why we ran this code once for each model and then used the best found hyperparameters for running the model thereafter.

Evaluation Metrics and Results

	Validation Accuracy	Dice Coefficient	Jaccard Index	Precision	Recall
U-Net					
Attention U-Net					

Conclusion

Training image segmentation models is a challenging task that requires careful consideration of various factors. While the underlying architecture, such as U-Net or DeepLabv3+, provides a strong foundation, the success of the model heavily relies on proper data preparation, hyperparameter tuning, and effective training strategies.

Data augmentation techniques, such as rotations, flips, and color jittering, can expand the dataset and improve generalization. However, excessive augmentation can introduce noise and negatively impact performance.

Hyperparameter tuning is another crucial aspect. The choice of optimizer, learning rate, batch size, and loss function significantly influences the training process. Experimentation and techniques like grid search help find the optimal hyperparameters.

The choice of backbone network, model depth, and the use of attention mechanisms also play important roles in model performance. A well-designed model architecture, combined with effective training strategies, can lead to state-of-the-art results.

Project GitHub Repo: https://github.com/sa10ma/Image-Segmentation-Project