Application of Medical Image Segmentation Techniques for Polyp Detection in Colonoscopy

Presented by 1705014,1705021,1705024

Problem Statement

Application of Medical Image Segmentation Techniques for Polyp Detection in Colonoscopy

- Developing accurate and reliable polyp detection in colonoscopy images is critical for improving patient outcomes.
- Medical image segmentation is a promising approach for identifying and isolating polyps from surrounding tissues.
- The project aims to develop an effective segmentation algorithm for polyp detection in colonoscopy images.

Dataset

CVC-ClinicDB :

- A dataset containing colonoscopy images and corresponding mask showing the position of polyp
- An open-access dataset
- 612 images
- Resolution: 384×288
- 31 colonoscopy sequences
- Used for medical image segmentation

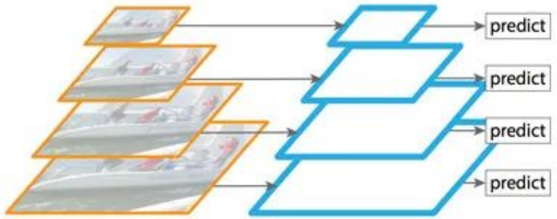
Proposed solution (architecture)

- Module: pytorch-lightning LightningModule
- Architecture: Feature Pyramid Networks (FPN)
- Encoder: ResNet34

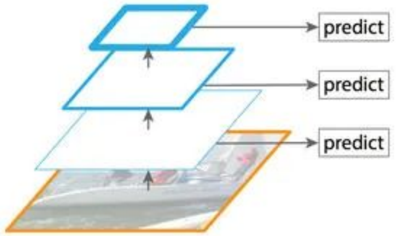
FPN Overview

A pyramid network can be composed of

- the **same image at different scales** to detect objects.
- or the **features** in the image to detect objects.



Pyramid of images



Pyramid of feature maps

FPN Overview

- FPN is a feature extractor designed for such pyramid concepts for object detection, semantic segmentation etc.

FPN Dataflow

- Composes a **bottom-up** and a **top-down** pathway.
- The bottom-up pathway is the usual convolutional network for feature extraction. As we go up, the spatial resolution increases.

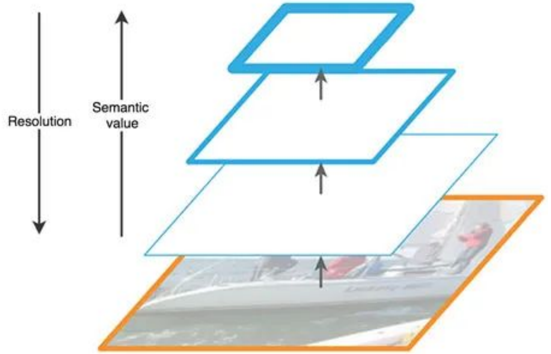


Fig: Feature Extraction in FPN

FPN Dataflow

- The top-down pathway is used to construct higher resolution layers from the semantic rich layer.

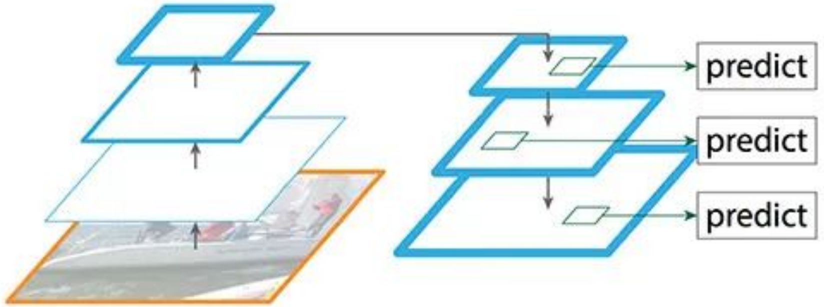
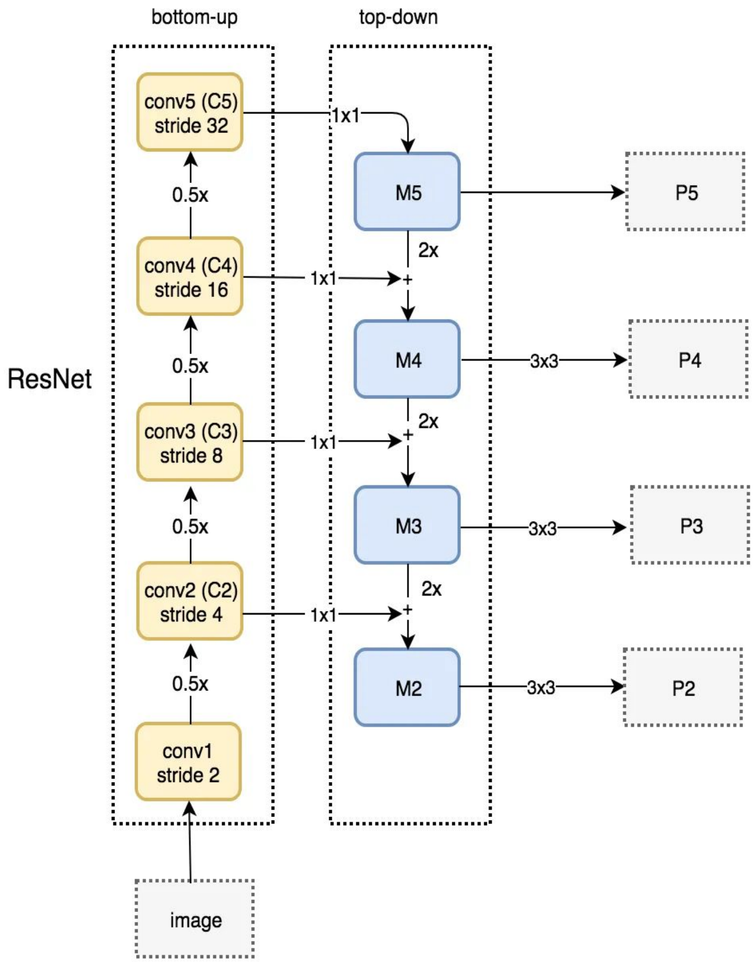


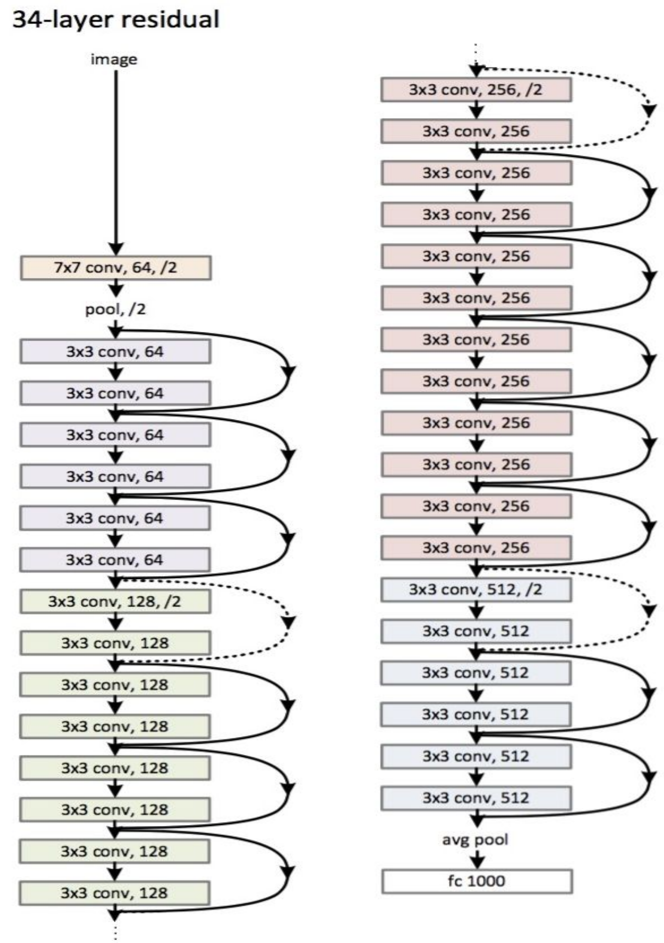
Fig: Reconstruct spatial resolution in the top-down pathway

FPN Building Blocks

- Bottom up (uses ResNet or VGG)
- Top down (uses convolution filters, lateral connections to upsample and merge features from lower level layers to features from higher level layers)



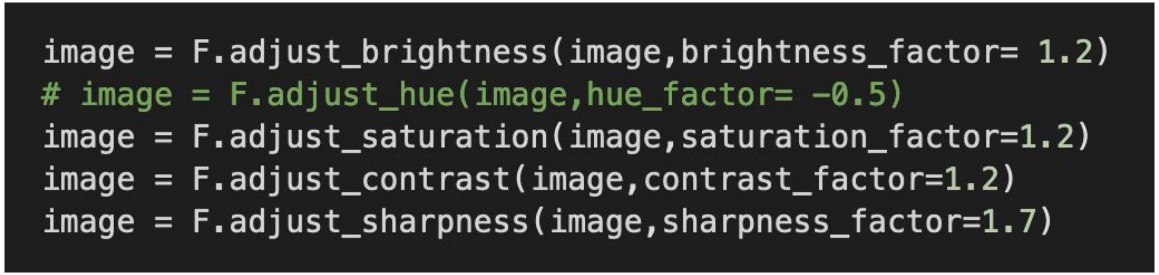
ResNet34 Building Blocks



Loss function

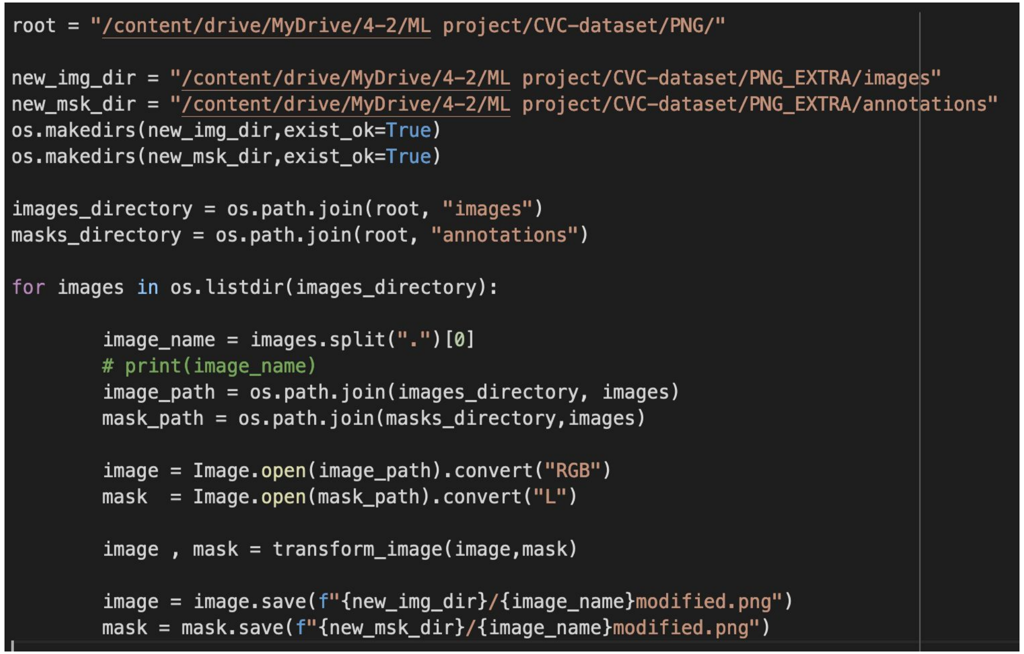
- Dice Loss
- Dice coefficient = 2 * Area Overlapped / Total Area
- Dice Loss = 1 - Dice coefficient
- 1 indicates a perfect match between the predicted and ground truth masks, while a lower loss indicates a lower overlap (i.e. lower quality of segmentation)

Adjustment



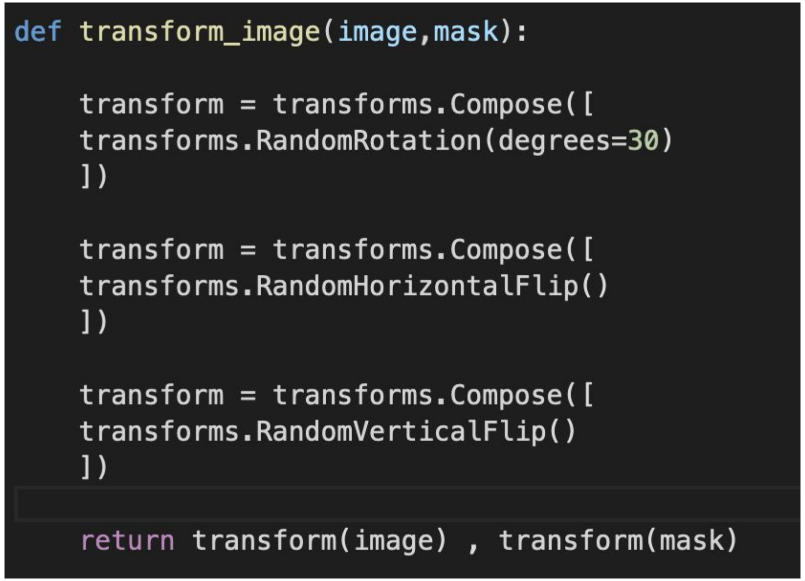
```
image = F.adjust_brightness(image,brightness_factor= 1.2)
# image = F.adjust_hue(image,hue_factor= -0.5)
image = F.adjust_saturation(image,saturation_factor=1.2)
image = F.adjust_contrast(image,contrast_factor=1.2)
image = F.adjust_sharpness(image,sharpness_factor=1.7)
```

Augmentation



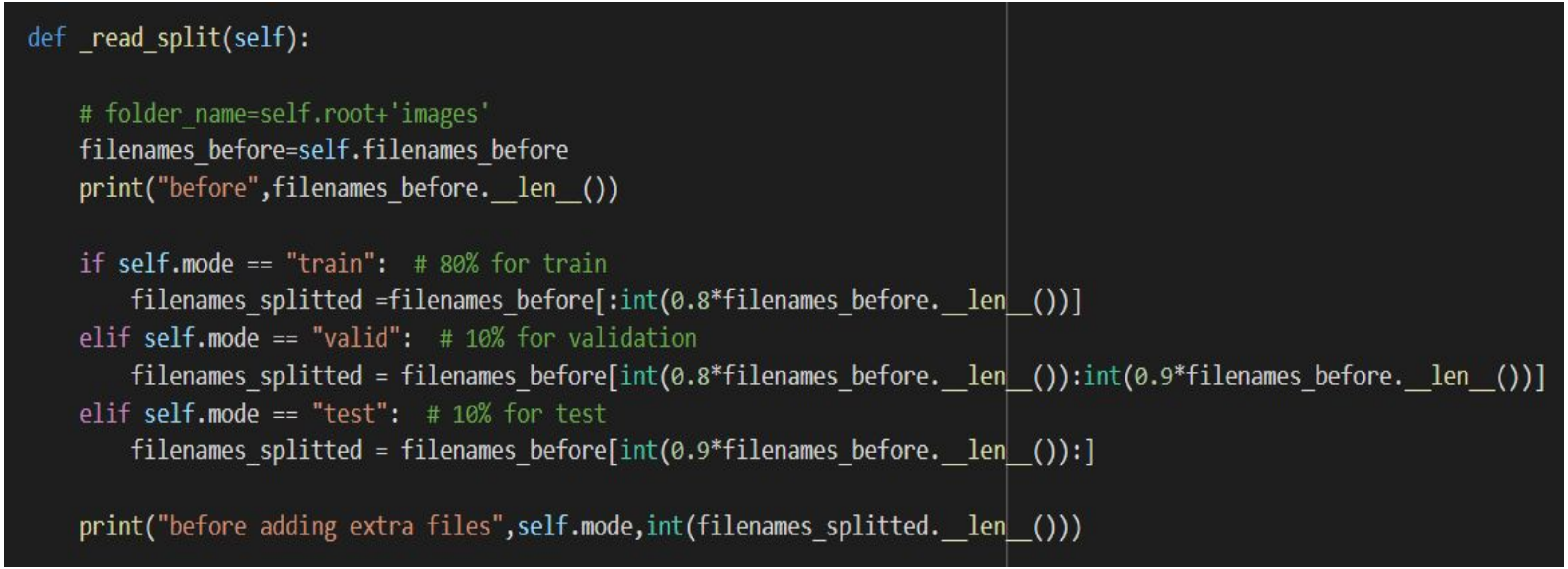
```
root = "/content/drive/MyDrive/4-2/ML project/CVC-dataset/PNG/"

new_img_dir = "/content/drive/MyDrive/4-2/ML project/CVC-dataset/PNG_EXTRA/images"
new_msk_dir = "/content/drive/MyDrive/4-2/ML project/CVC-dataset/PNG_EXTRA/annotations"
os.makedirs(new_img_dir,exist_ok=True)
os.makedirs(new_msk_dir,exist_ok=True)

images_directory = os.path.join(root, "images")
masks_directory = os.path.join(root, "annotations")

for images in os.listdir(images_directory):

        image_name = images.split(".")[0]
        # print(image_name)
        image_path = os.path.join(images_directory, images)
        mask_path = os.path.join(masks_directory,images)

        image = Image.open(image_path).convert("RGB")
        mask  = Image.open(mask_path).convert("L")

        image , mask = transform_image(image,mask)

        image = image.save(f"{new_img_dir}/{image_name}modified.png")
        mask = mask.save(f"{new_msk_dir}/{image_name}modified.png")
```

Augmentation



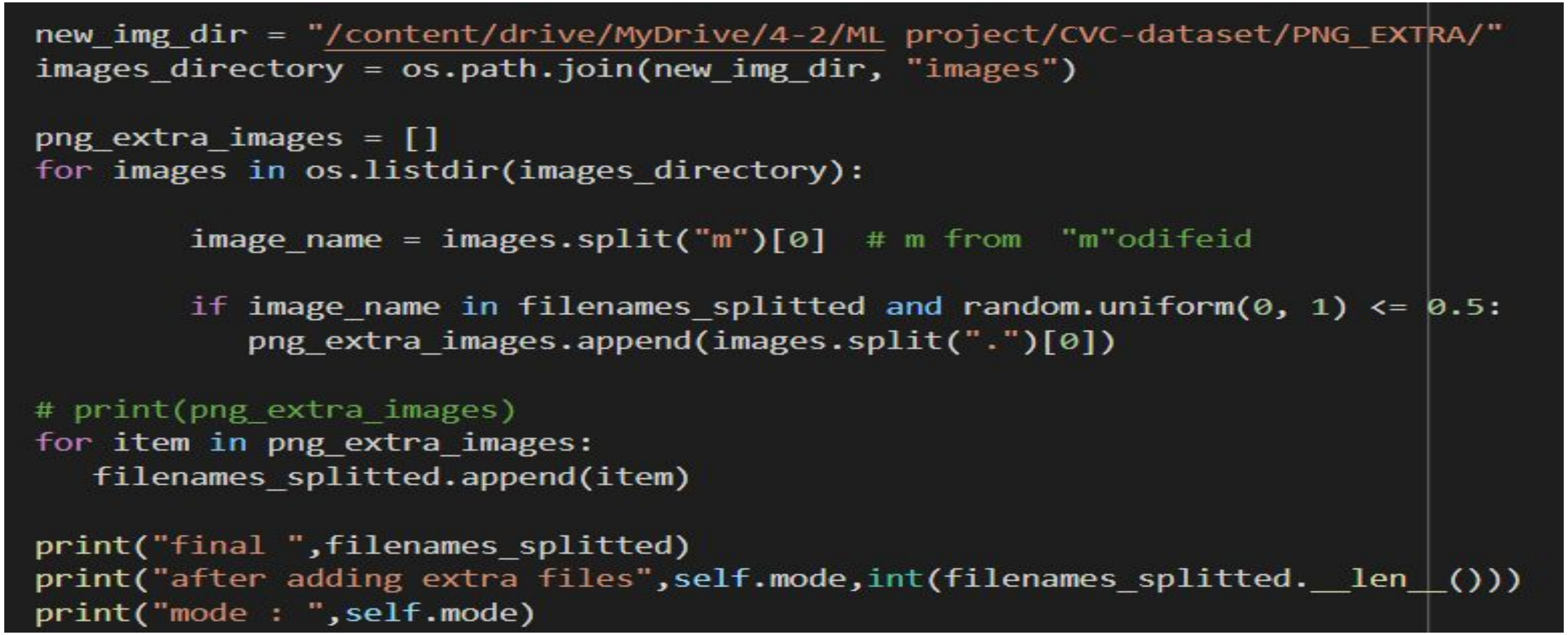
```
def transform_image(image,mask):

    transform = transforms.Compose([
    transforms.RandomRotation(degrees=30)
    ])

    transform = transforms.Compose([
    transforms.RandomHorizontalFlip()
    ])

    transform = transforms.Compose([
    transforms.RandomVerticalFlip()
    ])

    return transform(image) , transform(mask)
```

Data Splitting



```python
def _read_split(self):

    # folder_name=self.root+'images'
    filenames_before=self.filenames_before
    print("before",filenames_before.__len__())

    if self.mode == "train":  # 80% for train
        filenames_splitted =filenames_before[:int(0.8*filenames_before.__len__())]
    elif self.mode == "valid":  # 10% for validation
        filenames_splitted = filenames_before[int(0.8*filenames_before.__len__()):int(0.9*filenames_before.__len__())]
    elif self.mode == "test":  # 10% for test
        filenames_splitted = filenames_before[int(0.9*filenames_before.__len__()):]

    print("before adding extra files",self.mode,int(filenames_splitted.__len__()))
```

Pairing modified images with actual ones



```python
new_img_dir = "/content/drive/MyDrive/4-2/ML project/CVC-dataset/PNG_EXTRA/"
images_directory = os.path.join(new_img_dir, "images")

png_extra_images = []
for images in os.listdir(images_directory):

        image_name = images.split("m")[0]  # m from  "m"odifeid

        if image_name in filenames_splitted and random.uniform(0, 1) <= 0.5:
           png_extra_images.append(images.split(".")[0])

# print(png_extra_images)
for item in png_extra_images:
   filenames_splitted.append(item)

print("final ",filenames_splitted)
print("after adding extra files",self.mode,int(filenames_splitted.__len__()))
print("mode : ",self.mode)
```

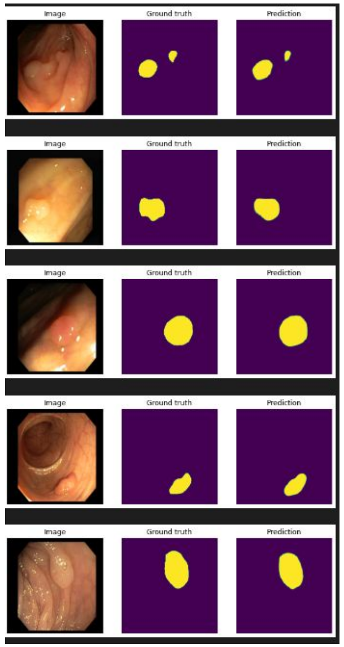
Performance (on the raw dataset)

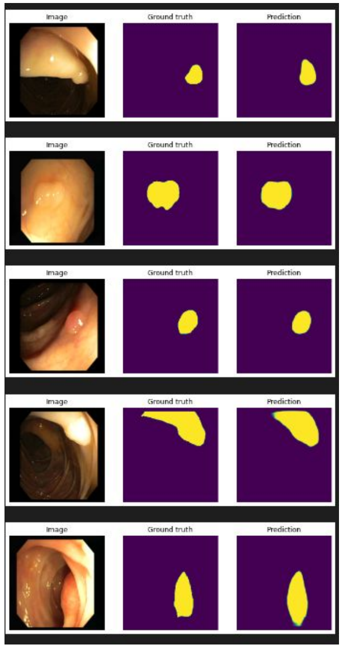
Dataset Size : 612 images

Train-Validation-Test: 80%-10%-10%

- **Validation IoU on Dataset:** 0.8739
- **Validation IoU per Image:** 0.8670
- **Test IoU on Dataset:** 0.8813
- **Test IoU per Image:** 0.8631

Performance (on the raw dataset)





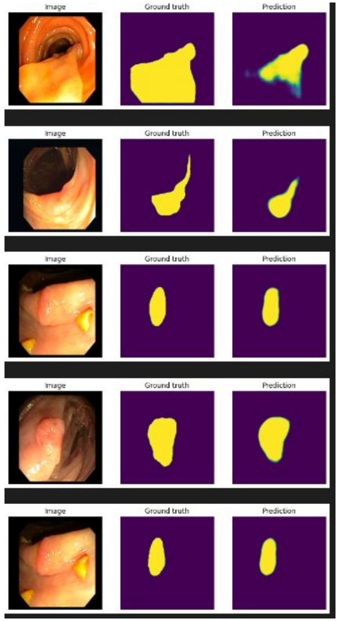
Performance (after augmentation)

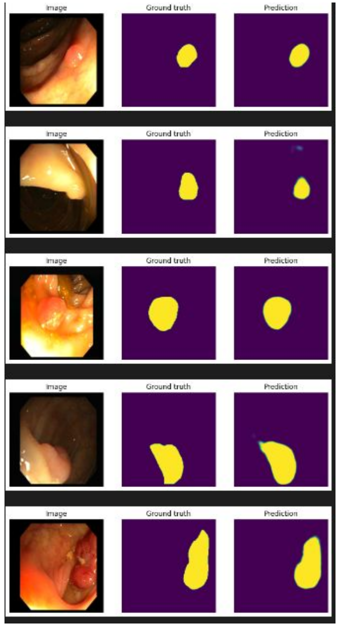
Dataset Size : 915 images

Train-Validation-Test: 80%-10%-10%

- **Validation IoU on Dataset:** 0.698
- **Validation IoU per Image:** 0.653
- **Test IoU on Dataset:** 0.615
- **Test IoU per Image:** 0.606

Performance (after augmentation)





Comparison with state-of-the-art methods

FCB-SwinV2 Transformer for Polyp Detection

mIou (SwinV2): 0.9189

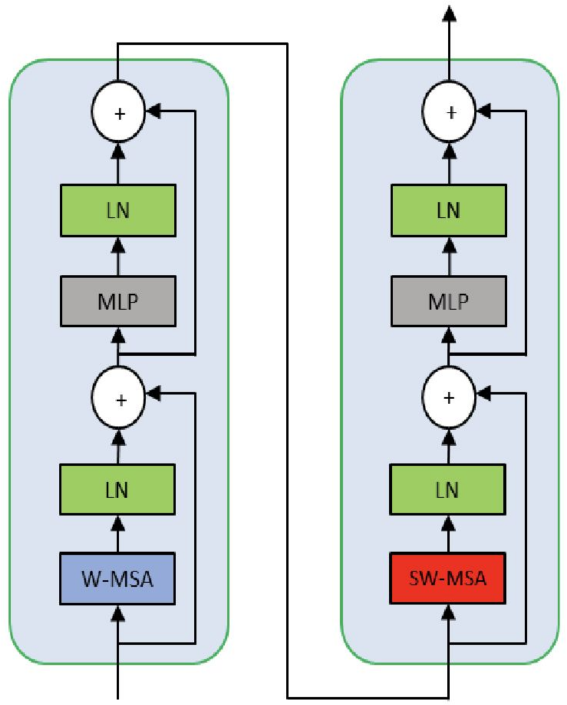


Figure 1: Two successive SwinV2 Transformer Blocks [12]. The residual post normalization configuration ensures layer normalization is conducted after attention mechanism layers and MLP layers.

Comparison with state-of-the-art methods

FCB-SwinV2 Transformer for Polyp Detection

mIou (SwinV2): 0.9189

mIou (FPN): 0.8813 (without augmentation)

mIou (FPN): 0.606 (with augmentation)

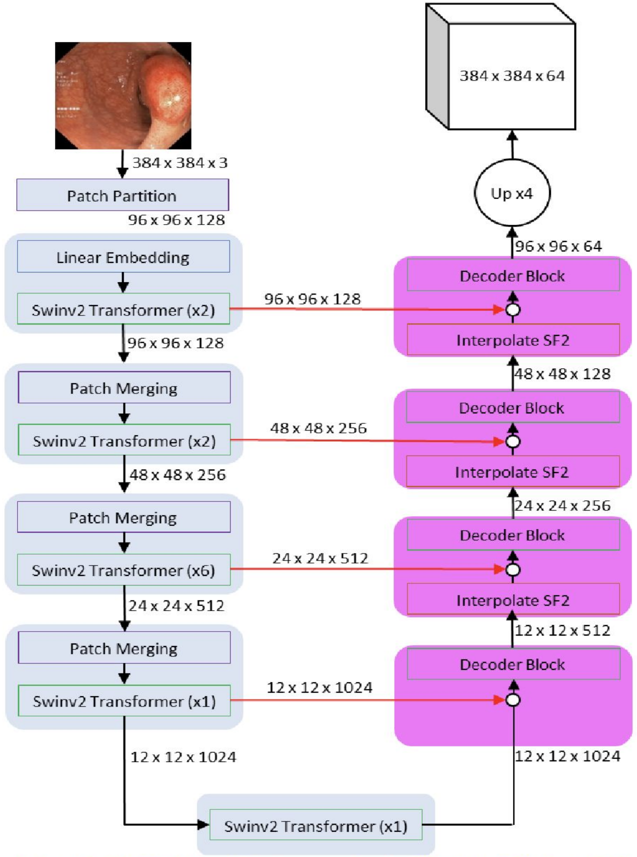


Figure 3: SwinV2-UNET [16] architecture used as the TB of the FCB-SwinV2 Transformer. The encoder stages reduce the spatial dimensions of feature maps while increasing the number of channel dimensions. Skip connections are used to pass feature maps generated by each stage of the encoder to decoder stages. The encoder is pre-trained using ImageNet22K [17].

Challenges

- Original model trained for The Oxford-IIIT Pet Dataset dataset
- Our dataset: Medical dataset consisting of Colonoscopy images
- New dataset not completely compatible with the original codebase
- Needed to do significant amount of modifications to the codebase
- Original codebase incorporated “trimap” which needed to be bypassed
- Image augmentation to increase the the original dataset