Computer Vision Course: Mini Project #5

Image segmentation

Seyed Mohammad Taha Tabatabaei

Deadline: Jan 20, 2024

Report date: Jan 20, 2024

Professor: Dr. Pourreza

Abstract

This report introduces a method for image segmentation using WideResNet and K-means clustering. Utilizing a pre-trained WideResNet_50_2 model, we extract features from the 6th layer and employ K-means to cluster pixels. Results on various images highlight success in segments with unique patterns, but challenges emerge with Gaussian distributed textures and patterns. The report concludes with visual representations of the segmentation outcomes.

Methods

As the problem definition indicates, we want to segment a set of images using WideResNet as a feature extractor and K-means clustering to segment pixels based on provided features. To do so, I used a pre-trained model from the PyTorch library. From PyTorch official documents, there are simple instructions on how to use a pre-trained model, getting data from a specific layer of the model, and the required pre-processing for the desired model. I followed those steps to initialize and infer the model. After initializing the model, we must pre-process images to make them suitable for the model. You can see the input pre-processing function in appendix [1]. After pre-processing, we pass the image to the model and get the output tensor of the 6th layer.

I used the method *extract_features* in the appendix [2] to do this. In this function, we slice the first 6 layers of the *WideResNet_50_2* model, and then we get the inference results. Before using our features for clustering, we must turn them into a vector usable for the sci-kit learn library k-means algorithm. You can see the pipeline in appendix [3]. Now that we have a vector of shape (512, 784), we use k-means to cluster these 784 samples into the desired number of classes. See appendix [4] for the clustering function. The output of the clustering method is a vector of shape (784, 1). With respect to the provided details in the problem definition, each of these 784 vectors represents a set of 8x8 pixels from the input image, so we reshape this vector to a (28, 28) matrix. We can imagine each element of this matrix as a representation for a window of 8x8 pixels in the input image. At the end of our pipeline, we show the results. See step 4 in appendix [3].

Discussion

You can see the outputs in images [1,2,3,4]. The results for images 501 and 504 were fairly better, especially for image 504, indicating that our proposed method works better with images in which each segment has a relatively unique pattern, like image 504, in which the zebra pattern is quite different from the background, and it does not work well in images consists of patterns that have a similar texture, like image 502, 503 and parts of 501. Our method obviously cannot segment Gaussian noise patterns, like image 503. See the input images in images [5,6,7,8].

Results

Images:

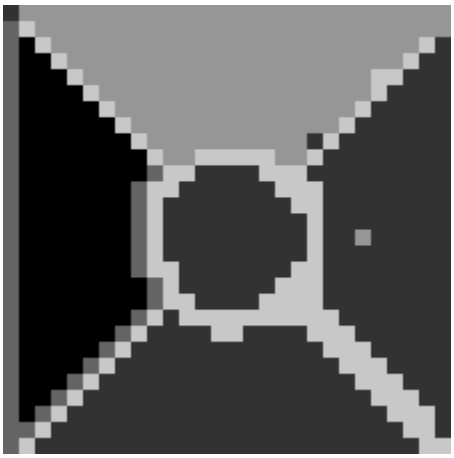

Image 1: Segmented image 501

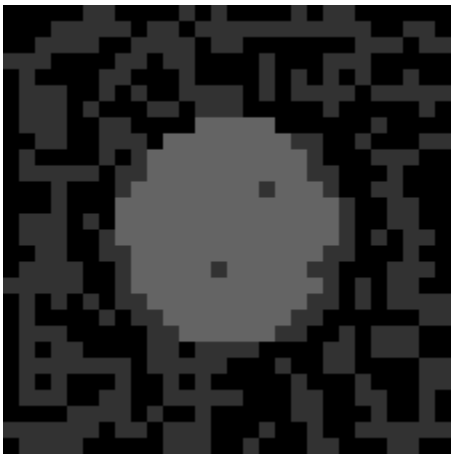

Image 2: Segmented image 502

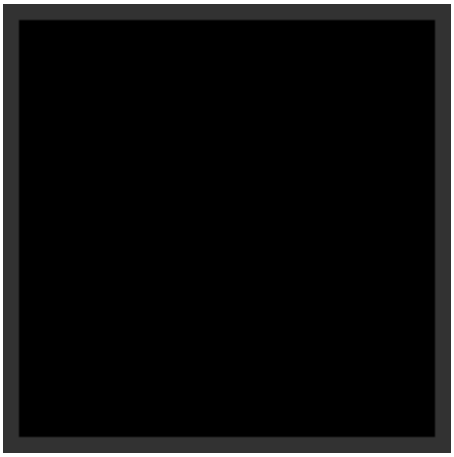

Image 3: Segmented image 503

Image 4: Segmented image 504

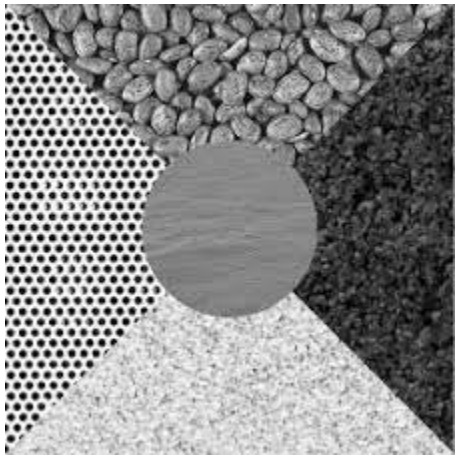

Image 5: Image 501

Image 6: Image 502

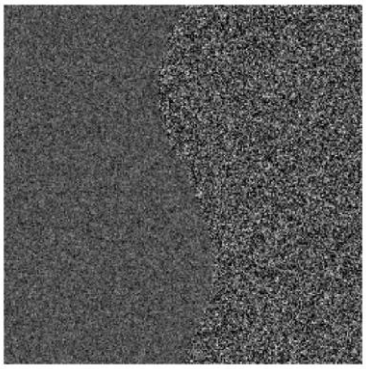

Image 7: Image 503

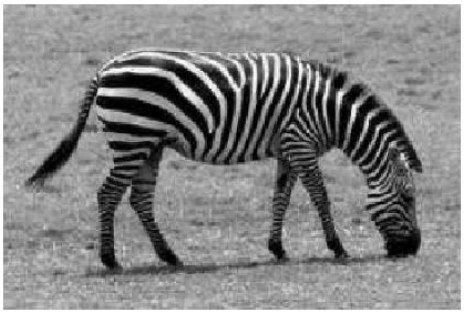

Image 8: Image 504

Appendix

Appendix [1]: the preprocessing function

```
def preprocess(filename):
    input_image = Image.open(filename)
    weights = Wide_ResNet50_2_Weights.IMAGENET1K_V2
    preprocess = weights.transforms()
    input_tensor = preprocess(input_image)
    print(f"input tensor shape: {input_tensor.shape}")
    input_batch = input_tensor.unsqueeze(0)

    return input_batch
```

Appendix [2]: feature extracting function

```
def extract_features(model, input, num_layers=6):

    # Select the 6th layer
    model = nn.Sequential(*list(model.children())[:num_layers])
    # Set the model to evaluation mode
    model.eval()

    with torch.no_grad():
        features = model(input)

    return features.squeeze()
```

Appendix [3]: the pipeline

```
from torchvision.models import wide_resnet50_2, Wide_ResNet50_2_Weights
import cv2
from functions import *

model = wide_resnet50_2(weights=Wide_ResNet50_2_Weights.IMAGENET1K_V2)
model.eval()

# Step 1: preprocessing
image_path = "50\Im504.jpg"
input_batch = preprocess(image_path)
```

```
# Step 2: Extract features
features = extract_features(model, input_batch, num_layers=6)

print(f"features shape: {features.shape}")

vectors = np.asarray(features).reshape(512, -1).T

print(f"vectors shape: {vectors.shape}")

# Step 3: Perform k-means clustering
clustered_labels = k_means_clustering(vectors, num_clusters=2)

cls = clustered_labels.reshape(28,28)

print(f"cls: {cls}")

# Step 4: Display the results
image_r = cv2.resize(cls.astype('uint8') , (224, 224), interpolation=3)
result = image_r*50
print(f"image_r shape: {result.shape}")

cv2.imshow('Original Image', cv2.imread(image_path))
cv2.imshow('Segmented Image', result)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Appendix [4]: clustering function

```
def k_means_clustering(features, num_clusters):
    kmeans = KMeans(n_clusters=num_clusters, init='k-means++')
    labels = kmeans.fit_predict(features)
    return labels
```