BBM418 Introduction to Computer Vision Lab. ASSignment4 # - Brain Tumor Detection using Faster R-CNN

name surname
student id
Department of Computer Engineering
Hacettepe University
Ankara, Turkey
email

Overview

In this assignment, you will implement Faster R-CNN to find out tumor on brain mr images. You will use the Brain Tumor MRI dataset [1] given in Figure 1 for localizing tumors by using classification and region proposal network.

1 Faster R-CNN

In this part, you are expected to explain your work for the first part in detail. You can include figures, tables, or formulas.

Step 1: Load or create a pre-trained model model = models.resnet18(pretrained=True)

Step 2: Add the classification part or RPN component using the properties of the model num$_features$ $= model.fc.in_features print(num_features) num_classes = 3 classnumber model.fc = nn.Linear(num_features, num_classes)$

Step 3: Use Softmax loss for classification part, L2 loss for RPN component softmax$_loss$ $= nn.CrossEntropyLoss() l2_loss = nn.MSELoss()$

Step4 : RGN import torch

def compute$_loss(reg, cls, train_labels)$: $Initialize loss variables total_loss = 0.0$

Iterate over each tensor in the train$_labels list for labels in train_labels$: $Reshape the label stensor labels = labels.view(-1, 5)$

Get the actual bounding box coordinates box$_coordinates = labels[:, : 3]$

Calculate classification loss (softmax loss) classification$_loss$ $= torch.nn.functional.cross_entropy(cls, labels[:, 3].long())$

Calculate the L2 loss for the RPN component regression$_loss$ $= torch.nn.functional.mse_loss(reg, box_coordinates)$

Calculate total loss (sum of softmax loss and L2 loss) total$_loss+$ $= classification_loss + regression_loss$

return total$_loss$

Step 5 : Choose an optimization algorithm and use the back propagation algorithm to train the model learning$_rate = 0.001optimizer = optim.SGD(model.parameters(), lr = learning_rate)$

Step6 :

import torch num$_epochs = 4$

for epoch in range(num$_epochs) : i = 0 for images, labels in train_loader :$

Get feature maps using model features = model(images) print(features.shape) features = features.unsqueeze(2).unsqueeze(3) features = features.repeat(1, 1, 224, 224)

reg, cls = rpn(features) print(labels) desired$_batch_size = 32 labels = [label.repeat(desired_batch_size//len(labels)) for label in labels] labels = torch.cat(labels)[: desired_batch_size]$

print(len(resized$_train_labels[i]))i = i + 1 Losscalculation resized_train_labels_tensor = torch.tensor(resized_train_labels[i]) labels_tensor = torch.unsqueeze(torch.tensor(resized_train_labels[i]), 0) print(labe compute_loss(reg, cls, labels_tensor)$

Backpropagation and gradient update optimizer.zero$_grad() loss.backward() optimizer.step()$