FIRE IN THE AMAZON

Understanding the Amazon with Pytorch

Sutter Schneider, Guillermina

December, 2019 - George Washington University

Background

The first idea for this final project was to work on a dataset on satellite images and poverty I found on GitHub. However, since the images did not have labels it was not suitable for this project.

After doing some research on Kaggle, we found a dataset on the Amazon forest. This dataset was meaningful to us. It not only covers a topic of interest for both Daniela (my teammate) and I: Latin America, but it is of newsly interest due to the recent fires in the Amazon. Working with this dataset would eventually allow us to build a model that would help explain how the fires in the Amazon develop over time, among other deforestation phenomena.

Individual work

Mainly, my role in this project was to research, implement, and evaluate alternative models to the CNN used at the very beginning of the project. I also provided assessment for the pretraining process: image preprocessing and data preparation. I also did exploratory data analysis on the dataset and debugged the code when necessary.

A considerable portion of the code we developed was recycled from the one submitted in Exam II. Since we were also dealing with a multi-label classification problem, we decided to put our efforts on building a good network or find a better pretrained model for the chosen dataset.

Daniela's CNN structure got a better score in Exam II, so we decided to move forward with her architecture instead of mine.

Individual work: details

I started by exploring the dataset. I built some lines of code that iterates over the images and outputs a bar chart with the labels count. Moreover, I developed more lines of code to

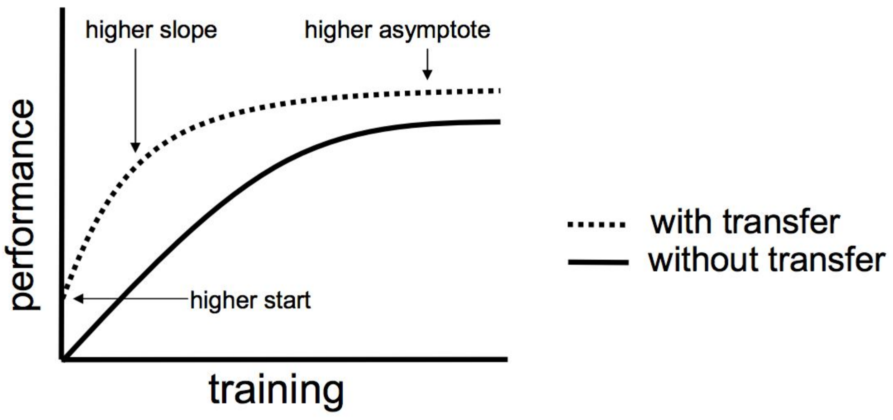
I contributed with customizing the transformation portion for the images in the data preprocessing. I read about best practices in image processing in pytorch. I read this, this, this, this, this, and this article to do the appropriate assessment.

I then customized the class responsible for iterating over the dataset. The core of this class came from the one built for Exam II. I had to customize it in order to make it suitable for the selected dataset. I changed the code in order for the class to be able to read the csv file containing the labels and images' paths and turn the images into tensors suitable for pytorch. For this part of the code I followed this and this article.

After we decided we needed to look into pretrained models in order to continue with the project, I started researching the extent and applications for ResNet50.

ResNet50 is a convolutional neural network that is trained on more than a million images. The network is 50 layers deep and can classify images into 1000 object categories. As a result, the network has learned rich feature representations for a wide range of images. The network has an image input size of at least 224x224.

I started by running a ResNet50 with 3 epochs, a learning rate of 0.03, and a batch of size 90. It threw a F-beta score of 0.73. While the model was running, I read this article which suggested to do fine tunning in ResNet50 by freezing some layers. This is called ResNet50 with Transfer Learning. Based on this article, transfer learning is an optimization that allows rapid progress or improved performance when modeling the second task.



For the next model to run, I froze the first two layers and switched the last layer of the network with one that suited our requirements (17 labels) as shown in the following code snippet. Since I was introducing this new change to the network, I kept the same parameters as in the previous run. This model threw a F-beta score of 0.781.

```
resnet50 = models.resnet50(pretrained=True)


ct = 0

for child in resnet50.children():

   ct += 1

   if ct < 3:

       for param in child.parameters():
            param.requires_grad = False



# Change the last layer
num_ftrs = resnet50.fc.in_features
resnet50.fc = nn.Linear(num_ftrs, 17)
```

The last model run, saw a slightly decline in the F-beta score. It decreased to 0.766. The only changes introduced in this last run was an image resize of 224 x 224. I also normalized the images as suggested by this pytorch documentation.

Every time a model ran, I kept track of the model, parameters and accuracy scores in the results table shown in the Results section.

In order to test our models, Daniela and I worked on the predict file. We decided to use the F-beta score since that metric was the one being used in the Kaggle competition. If we used the same metric as in the competition, we would be able to compare the performance of our models to those in Kaggle. We had to return the predicted labels into ones and zeros, since the F-beta score receives an array of integers. We turned the predicted probabilities into one if x >= 0.5, else x = 0.

Results

After we decided that building another CNN was not worth it because we were not seeing almost any improvement in the accuracy scores, we decided to start looking into pretrained models. I started testing out the ResNet50. The model structure looks as follows:

```
ResNet(
 (conv1): Conv2d(3, 64, kernel_size=(7, 7),stride=(2,2),padding=(3,3),bias=False)
 (bn1):BatchNorm2d(64,eps=1e-05,momentum=0.1,affine=True,track_running_stats=True)
 (relu): ReLU(inplace=True)
 (maxpool): MaxPool2d(kernel_size=3,stride=2,padding=1,dilation=1,ceil_mode=False)
 (layer1): Sequential(
   (0): Bottleneck(
     (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
     (bn1):BatchNorm2d(64,eps=1e-05,momentum=0.1,affine=True, track_running_stats=True)
     (conv2):Conv2d(64,64,kernel_size=(3,3),stride=(1,1),padding=(1,1),bias=False)
     (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
     (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
```

```
      (bn3):BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (1):BatchNorm2d(256,eps=1e-05,momentum=0.1, affine=True, track_running_stats=True)))
    (1): Bottleneck(
      (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2):Conv2d(64,64,kernel_size=(3,3),stride=(1,1),padding=(1,1),bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256,eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True))
    (2): Bottleneck(
      (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2):Conv2d(64,64,kernel_size=(3,3),stride=(1,1),padding=(1,1),bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256,eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)))
  (layer2): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05,momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(128, 128, kernel_size=(3,3), stride=(2,2), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05,momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(512, eps=1e-05,momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512,eps=1e-05,momentum=0.1,affine=True, track_running_stats=True)))
    (1): Bottleneck(
      (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(128,eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2):Conv2d(128,128,kernel_size=(3,3),stride=(1,1),padding=(1,1), bias=False)
      (bn2): BatchNorm2d(128,eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(512,eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True))
    (2): Bottleneck(
      (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(128,eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(128,128,kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(128,eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(512,eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True))
  (3): Bottleneck(
    (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(128,eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(128, 128, kernel_size=(3, 3),stride=(1,1),padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(128,eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(512,eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)))
(layer3): Sequential(
  (0): Bottleneck(
    (conv1): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(256,eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2):Conv2d(256,256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(256,eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(1024,eps=1e-05,momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (downsample): Sequential(
      (0): Conv2d(512, 1024, kernel_size=(1, 1), stride=(2, 2), bias=False)
      (1): BatchNorm2d(1024,eps=1e-05,momentum=0.1,affine=True,track_running_stats=True)))
  (1): Bottleneck(
    (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(256,eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(256, 256,kernel_size=(3,3),stride=(1, 1), padding=(1, 1), bias=False)
    (bn2):BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3):BatchNorm2d(1024,eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True))
  (2): Bottleneck(
    (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(256,eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(256, 256,kernel_size=(3,3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(256,eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(1024,eps=1e-05,momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True))
  (3): Bottleneck(
    (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(256,eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
```

```
      (conv2): Conv2d(256, 256, kernel_size=(3,3), stride=(1,1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256,eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024,eps=1e-05,momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True))
    (4): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256,eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(256,256, kernel_size=(3,3), stride=(1,1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256,eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024,eps=1e-05,momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True))
    (5): Bottleneck(
      (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05,momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3,3), stride=(1,1), padding=(1,1), bias=False)
      (bn2): BatchNorm2d(256,eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024,eps=1e-05,momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)))
  (layer4): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(512,eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(512,512, kernel_size=(3,3), stride=(2,2), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512,eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(2048,eps=1e-05,momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(1024, 2048, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(2048,eps=1e-05,momentum=0.1,affine=True,track_running_stats=True)))
    (1): Bottleneck(
      (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(512,eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(512,512, kernel_size=(3,3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512,eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(2048,eps=1e-05,momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True))
    (2): Bottleneck(
      (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(512,eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(512, 512, kernel_size=(3,3), stride=(1,1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512,eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(2048,eps=1e-05,momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)))
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=2048, out_features=17, bias=True))
```

For the first model I ran I chose a learning rate of 0.03, a batch of size 90, and 3 epochs. It threw a F-beta score of 0.73. After I shut down half of the layers and kept the other parameters constant, the model seemed to improve. The F-beta score score increased to 0.78.

For the last run, we both implemented batch normalization and resized the images to 224x224 based on this article we read. More specifically, we added the following line to the transformations section:

```
transforms.Resize(256)
transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
```

After we applied that change to the transformations portion, I ran the same model as before with only 2 epochs. The F-beta score decreased to 0.76.

I would have liked to run more models, but my instance kept crashing every time I tried to do so. It did not matter how small my batch size was or how many epochs I chose, I kept getting the same error on Pycharm:

```
RuntimeError: CUDA error: out of memory
```

While Daniela ran her models, I kept reading about best practices to increase the accuracy of our models through data processing or by testing out other pretrained models.

Conclusion

During this project I faced two types of restrictions: one associated with time and another one with the computational power. The computational power restriction was the most critical one for me. Daniela's models took around two complete hours to run (and sometimes even more) and my instance kept getting saturated. So the chances of us training a considerable amount of models to compare with each other was very limited.

Pretrained models were key to this project and, in general, a good alternative to CNNs that are built from scratch. I would have liked to be able to test more pretrained models and compared them with each other to turn this project into an evaluation one. In my opinion, that is what other people looking for more information on pytorch's pretrained models need: a more comprehensive and exhaustive implementation of different pretrained models on the same dataset. Had we had this sort of information, I guess we would have been able to find a more robust model with a higher accuracy score.

I edited/wrote 57% from the 157 lines of code in the code file. The rest was taken from sources such as Pytorch Forums, Pytorch documentation or Prof. Jafari's GitHub repo. Most of the lines taken from these sources were absolutely necessary for the code to run and not editable.