Machine Learning Engineer Nanodegree Capstone Report

Trevor LaViale

7/5/2020

Definition

Project Overview:

Transfer learning and computer vision tasks have become increasingly more important in machine learning research and applications, especially when combined. Computer vision is a highly researched topic, as it has applications ranging from self-driving cars to medical imaging. However, training large convolutional neural networks for computer vision is very computationally expensive, making transfer learning a feasible option for quicker training tasks.

In this project, I implemented two different neural networks. One utilized transfer learning on a pretrained ResNet[1] model from PyTorch, while the other benchmark model is a convolutional neural network implemented from scratch. Both networks were trained on an image dataset available on Kaggle.

Problem Statement:

The purpose of this project is to try to attain a higher accuracy of classifying images with the pretrained PyTorch model over the benchmark convolutional network. The following steps will be necessary for implementation:

1) Download dataset and split into train/test folders that can be used by PyTorch's ImageFolder
2) Load data using PyTorch's DataLoader
3) Transform images to size of 224x224 which is what is expected by the ResNet model
4) Define both ResNet model and benchmark convolutional network model. The ResNet model will be the ResNet-50 model from PyTorch with the last fully connected layer changed to match our classes. The convolutional network will consist of three convolutional layers, two linear layers, a max pooling layer, and a dropout layer, with ReLU activation.
5) Specify loss function and optimizer. Loss function will be cross entropy and optimizer will be stochastic gradient descent.
6) Train benchmark model for 20 epochs, and train ResNet model for 10 epochs.
7) Test both models on test dataset and compare accuracy.

Metrics:

For both models, we will look at class accuracy, as well as overall accuracy.

$$Accuracy\ of\ Class_i = \frac{Number\ of\ images\ in\ class_i\ correctly\ identified\ in\ test\ dataset}{Total\ number\ of\ images\ of\ class_i\ in\ test\ dataset}$$

$$Overall\ Accuracy = \frac{Total\ number\ of\ images\ correctly\ identified\ in\ test\ dataset}{Total\ number\ of\ images\ in\ test\ dataset}$$

All classes have relatively the same number of images, which is why accuracy was chosen as a metric over metrics such as log loss or AUC. The specific number of images in each class are as follows: airplane – 727, car – 968, cat – 885, dog – 702, flower – 843, fruit – 1000, motorbike – 788, person – 986.

---

[1] He, Kaiming, et al. *Deep Residual Learning for Image Recognition*. Microsoft Research, 10 Dec. 2015, arxiv.org/pdf/1512.03385.pdf.

Analysis

Data Exploration:

The dataset that is used is from Kaggle and contains 727 images of airplanes, 968 images of cars, 885 images of cats, 702 images of dogs, 843 images of flowers, 1000 images of fruit, 788 images of motorbikes, and 986 images of people. In percentages, airplanes make up 10.5% of the total images. Cars – 14%, cats – 12.8%, dogs – 10.2%, flowers – 12.2%, fruit – 14.5%, motorbikes – 11.4%, people – 14.3%. As one can see, the number of images in each class is relatively balanced.

Exploratory Visualization:

An example of the images after transformations to 224x224 sized images can be seen in figure 1 below.

Figure 1

As one can see, the features of each image seem to be distinct. However, cats and dogs do share similar qualities which is the only expected pair of similar features in the dataset.

Algorithms and Techniques:

Two classifiers will be used and compared against each other. The first classifier is a ResNet model from PyTorch. A representation of the ResNet architecture can be seen on the right in figure 2 below. The figure is taken from the ResNet paper cited in the Project Overview section above.

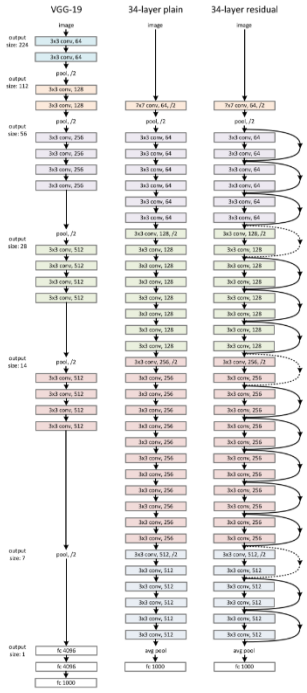

Figure 2

The ResNet model will be pretrained on the ImageNet dataset which contains millions of images. The last fully connected layer is the only layer of the model that will be changed. Instead of outputting 1000 classes, it will only output 8, matching our number of classes. Hyperparameters that were set before training the models include:

- Train/test split size
- Batch size
- Number of epochs
- Loss function
- Optimizer
- Learning rate

The other classifier is the benchmark convolutional neural network which will be explained in the section below.

Benchmark:

The benchmark that will be used is the accuracy of a convolutional neural network implemented from scratch and trained entirely on the train dataset. The convolutional neural net will consist of three convolutional layers, two fully connected layers, a max pooling layer, a dropout layer, and ReLU activation. The benchmark model had several hyperparameters as well, including but not limited to padding size, kernel size, and number of features. This model will be trained exclusively using the training data and then tested on the test dataset. The accuracy of each class, as well as the overall accuracy will be calculated and compared to the accuracy of the ResNet model.

Methodology

Data Preprocessing:

The only two preprocessing steps that was necessary for the datasets was to split the data into a train and test set and then to convert each image in the datasets to a 224x224 sized image. The test set was created by taking 20% of the images in each class. The train set was composed of the remaining 80% of the images. After the train and test sets were created, the images were resized to fit the expected input of the ResNet architecture. The data was loaded and resized using modules from PyTorch including but not limited to *torchvision.datasets*, *torchvision.models*, and *torchvision.transforms*.

Implementation:

First, the pretrained ResNet model was loaded from PyTorch and the last fully connected layer was changed to a linear layer that takes in the same number of inputs but outputs only 8 features instead of 1000. Then, all the layers in the ResNet model were frozen from training except for the last fully connected layer that was just implemented.

Next, the benchmark convolutional model was created. The model's layers are described above. Figure 3 below shows the model's full implementation.

```
Net(
  (conv1): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (fc1): Linear(in_features=50176, out_features=1000, bias=True)
  (fc2): Linear(in_features=1000, out_features=8, bias=True)
  (dropout): Dropout(p=0.2, inplace=False)
)
```

Figure 3

Next, both the loss function and optimizers were specified. Both models used the same cross entropy loss function. The optimizer that was used for the benchmark model was stochastic gradient descent, with all parameters allowed to update. The optimizer for the ResNet model was also stochastic gradient descent, but only the parameters for the last fully connected layer were to be updated. The learning rate for the benchmark optimizer was 0.01 and the learning rate for the ResNet optimizer was 0.001.

Next, each model was trained and tested. The ResNet model was trained for 10 epochs, calculating the loss at each epoch. Then, the ResNet model was tested using the test dataset and the accuracy of each class and overall accuracy was computed. After the testing of the ResNet was completed, the benchmark model was trained for 20 epochs, again calculating the loss at each epoch. Following training, the benchmark model was tested on the test dataset and the accuracy of each class and overall accuracy was computed.

Refinements:

The learning rate of both models were tweaked until both seemed to have a relatively small training loss. Additionally, the number of epochs were changed, but had to be considered carefully due to computational limitations.

Results

Model Evaluation and Validation:

The final ResNet model achieved a remarkably high accuracy on the test set, as well as a relatively low training loss on the training set. Figures 4 and 5 show the training loss and test set accuracy, respectively.

```
Epoch: 1, Loss: 0.23978356630154668
Epoch: 2, Loss: 0.17437866845095548
Epoch: 3, Loss: 0.136774968352524
Epoch: 4, Loss: 0.11844108150323261
Epoch: 5, Loss: 0.10552421487599574
Epoch: 6, Loss: 0.0913208026498939
Epoch: 7, Loss: 0.08729196772159968
Epoch: 8, Loss: 0.07861540841974314
Epoch: 9, Loss: 0.07090034538056679
Epoch: 10, Loss: 0.06736872076950476
```

Figure 4

```
Test accuracy of airplane: 100% (146/146)
Test accuracy of   car: 100% (194/194)
Test accuracy of   cat: 99% (177/178)
Test accuracy of   dog: 98% (139/141)
Test accuracy of flower: 100% (168/168)
Test accuracy of fruit: 100% (201/201)
Test accuracy of motorbike: 100% (158/158)
Test accuracy of person: 100% (186/186)

Overall test accuracy: 99% (1369/1372)
```

Figure 5

As one can see, the model achieved an extremely high accuracy on the test set, almost classifying every single image correctly. The only three images that were missed were of the cat and dog classes, which is expected as cats and dogs share similar features. The flower images in our dataset are from ImageNet, so it is expected that they should all be classified correctly due to ResNet being pretrained on the ImageNet dataset. However, none of the other images come from ImageNet, so it is still remarkably intriguing that almost all images were classified correctly when applying transfer learning on a different set of similar images.

Justification:

While our benchmark model did perform well, it was seriously lacking in accuracy compared to the ResNet model. The benchmark model was trained exclusively on our training set and tested against the same pictures in the test set. The benchmark model's accuracy can be seen in Figure 6 below.

```
Test accuracy of airplane: 86% (126/146)
Test accuracy of   car: 96% (187/194)
Test accuracy of   cat: 86% (154/178)
Test accuracy of   dog: 51% (73/141)
Test accuracy of flower: 77% (131/168)
Test accuracy of fruit: 100% (201/201)
Test accuracy of motorbike: 98% (155/158)
Test accuracy of person: 99% (185/186)

Overall test accuracy: 88% (1212/1372)
```

Figure 6

It appears our benchmark model had a very hard time classifying dog images correctly, which brought down the overall accuracy by a significant amount. However, most of the other classes had a relatively high accuracy, which is encouraging for our benchmark model. When comparing the two models, the benchmark model achieved an overall accuracy of 88% while the ResNet pretrained model achieved an overall accuracy of 99%. This was expected because neural networks learn as more data is shown to them. The ResNet pretrained model was trained on millions of images similar to the images in our dataset, while the benchmark model was only trained on around 5,000 images. It would be interesting to see if our benchmark model could perform much better if it was trained on as many images. Unfortunately, most common computers lack the computational power required to perform such a task.

Conclusion

Free Form Visualization:

Below in figure 7 is an example of the images that our benchmark model classified incorrectly and correctly. The real classes are in parentheses while the predicted class is not.

Figure 7

As one can see, the benchmark model had the hardest time classifying flowers, cats, and dogs. Meanwhile, the misclassifications in the ResNet model were of just cats and dogs, and there were far fewer.

Reflection:

To begin the project, I first needed to find a suitable pretrained network to use and learn about the implementation. I chose ResNet for its performance in classifying images in the ImageNet database. I thought the most difficult aspects of the project were just learning about how ResNet works and how it differs from regular convolutional neural nets. I also found this aspect to be one of the most interesting. There are so many ways to tweak neural networks to come up with a completely new way of implementation.

After learning about the model framework for residual networks, I then had to learn about the implementation in PyTorch. I needed to learn what type and size of images the model expects, and all of the layers in the model. Finally, after all of this learning, I was able to utilize a transfer learning approach and implement this ResNet model on my own. Due to ResNet's architecture, it was also a bit difficult to figure out how to freeze all the parameters except for the very last layer. This project has shown me how important it is to read the documentation when using code written by other people, and to make sure that I document my own code well, in case others do the same.

The final ResNet model performed how I was expecting it to relative to the benchmark model. The images in our dataset are similar to the images in ImageNet and since the ResNet model was pretrained on millions of images, it made sense that it had a very high performance on the images in our test set.

Improvement:

One improvement or suggestion I would make going forward is to time the training of both models. I didn't time them, but it seemed as though the pretrained network with only the fully connected layer being trained took much less time to train than the entire convolutional network. However, this may also be because the number of epochs for the pretrained model was much less.

Another improvement I would make going forward is to reduce the number of epochs for training the benchmark convolutional model, as the training loss started to increase after 19 epochs. Additionally, I would add more layers to the convolutional network to see if that improved accuracy.

The last improvement I would make is to use images that are more closely related to each other. The images in our dataset have very different features to each other. Going forward, I would choose to instead get more granular with one of the classes and classify cars by model for example.

The final ResNet model achieved a very high accuracy, and only misclassified three images out of our entire test set. It could potentially achieve a higher accuracy if it were trained for more than 10 epochs; however, we already achieved an accuracy close to 100%.