Assignment # 04: Modern Convolutional Neural Networks

Problem Statement: Train a CNN classifier using the Kaggle data for '225 Bird Species'.

Approach & Findings:

Dataset Used: The Kaggle data has a train + test + validation split of 31316+1125+1125 images of birds of 225 different species. The images are of size 224*224 with 3 channels. I pulled this data directly from Kaggle using python code and Kaggle api in Colab notebook. The matplotlib library was used to display 6 random images and a frequency plot was generated to confirm that 'there is almost no class imbalance present, not enough to make our task challenging'.

Model Training & Insights: To train the model I used PyTorch libraries and Colab Environment with ~16GB of Nvidia GPU for all the models trained. The training/validation and the model architectures were all coded from scratch and all the trained models were saved as '.pth' files which can be loaded to do prediction or can even be trained further.

[Note: For all the models, image data was converted to tensor with shape 3*224*224 and an augmentation of **horizontal flip** was done with a probability of 0.5 to get good predictions. The optimizer used was **SGD with momentum** and loss function used was **cross-entropy**. For GoogLeNet, ResNet and DenseNet, **Input data normalization** was also done]

Below are the CNN Architectures that were used (in the same order):

- **AlexNet**
  - Hyperparameters: batch size=128 (256 resulted in 100% GPU usage); epochs=25; lr=0.01; momentum=0.9
  - Performance (Accuracy): Training=87%; Validation=78%; Testing=**80%**
  - Useful Insights: After 12 epochs it was at 10% accuracy so I tried to increase the lr by 10x but that resulted in 'inf' loss which means the model parameters shifted in the right direction but must have shot off too far from the minima resulting in increase of loss to infinity. Increasing batch size was not an option and further training might not have resulted in better test accuracy.
- **VGG11** and **VGG16** (without BN)
  - Hyperparameters: batch size=64; epochs=25; lr=0.01/0.1/1.0/10; momentum=0.0/0.9
  - Performance (Accuracy): Training=0%; Validation=0%; Testing=**0%**
  - Useful Insights: Clearly a bigger model so I could only use batch size of 64. No Batch Normalization was used. Above hyperparameters and their combinations were tried and yet the loss didn't decrease and accuracy stayed at 0%. At this point I tried to normalize the input data assuming this might be a case of gradient saturation. Still no improvement.
- **GoogLeNet** (without BN)
  - Hyperparameters: batch size=256; epochs=10; lr=0.01/0.1; momentum=0.0/0.9
  - Performance (Accuracy): Training=0%; Validation=0%; Testing=**0%**
  - Useful Insights: A better architecture was not expected to have 0% accuracy. It was time to try Batch Normalization.
- **GoogLeNet** (with **BN**)
  - Hyperparameters: batch size=256; epochs=10+15; lr=0.1; momentum=0.9
  - Performance (Accuracy): Training=93%; Validation=87%; Testing=**88%**
  - Useful Insights: Finally, Batch Normalization has succeeded in avoiding the gradient saturation. A training of 10 epochs resulted in 71% accuracy which further increased on loading the model from .pth file and training it for 15 more epochs.
- **ResNet18**
  - Hyperparameters: batch size=256; epochs=10+10; lr=0.1/0.05; momentum=0.9
  - Performance (Accuracy): Training=99%; Validation=92%; Testing=**91%**
  - Useful Insights: Batch Normalization right from the beginning. Much faster training as compared to previous architectures, perhaps due to residual connections. Accuracy increased to 90% in just 10 epochs after which it seems to be overfitting.
- **DenseNet**
  - Hyperparameters: batch size=256; epochs=15; lr=0.1; momentum=0.9
  - Performance (Accuracy): Training=93%; Validation=88%; Testing=**90%**
  - Useful Insights: As seen in ResNet once again it's an architecture that went up to 90% accuracy very fast in just 12 epochs. This time however I didn't train it further as validation accuracy was clearly improving slower than train accuracy.

Inferences: From a failed model training (VGG), I got to know why batch normalization and normalized inputs are important as they can help in avoiding gradient saturation. From the successful trainings, I found out how batch normalization, residual connections and better architectures can lead not only to faster training but better accuracy as well. I also learned how to change hyperparameters (learning rate/epochs/batch size/momentum) to speed up training based on how well the training epochs are performing.

GitHub Link to Codes and Model Files: Below link is for the assignment 04 folder of GitHub repo with trained models and Colab notebook

https://github.com/quickSilverShanks/Dive-into-Deep-Learning/tree/master/Assignment%2004

Submitted by:
Name: Shashank Prakash e-Mail: shashank708.2606@gmail.com Date of Submission: 30th November 2020