Project 2

Deep Learning by PyTorch

Part 1: Improving BaseNet on CIFAR100

Model Architecture

BaseNet

Layer No.	Layer Type	Kernel size (For Conv layers)	Input | Output dimension	Input | Output channels (For Conv layers)
1	Conv2D	3	32 | 32	3 | 64
2	BatchNorm	-	32 | 32	-
3	ReLU	-	32 | 32	-
4	ResBlock * 3	3	32 | 32	64 | 64
5	ResBlock * 4	3	32 | 32	64 | 128
6	ResBlock * 6	3	32 | 32	128 | 256
7	ResBlock * 3	3	32 | 32	256 | 512
8	MaxPool2D	-	32 | 16	-
9	Linear	-	512 * 16 * 16| 50	-
10	ReLU	-	50 | 50	-
11	Linear	-	50 | 100	-
12	BatchNorm	-	100 | 100	-

ResBlock / feature_extraction_block

Layer No.	Layer Type	Kernel size (For Conv layers)	Input | Output dimension	Input | Output Channels (For Conv layers)
1	Conv2D	3	32 | 32	In | Out
2	BatchNorm	-	32 | 32	-
3	ReLU	-	32 | 32	-
4	Conv2D	3	32 | 32	Out | Out
5	BatchNorm	-	32 | 32	-
6	ReLU	-	32 | 32	-
7	Add [Input of 1 with Output of 6]	-	32 | 32	-

Explanation

In this model, there is an input convolution, kernel 3, which maps the channels from 3 to 64. After every convolution is a BatchNorm in the architecture, and this feature map gets sent to the models after it, I used 16 ResBlocks, where 3 of the ResBlocks have no expansion and the dimensions stay the same, i.e., 64, after that there are 4 ResBlocks which take the spectral resolution from 64 to 128. This feature map gets sent to the next set (6) of Residual Blocks, which take the channels from 128 to 256, which gets sent to the last set (3) of ResBlocks which expand the dimension to 512. These feature maps ultimately get sent to a Max Pooling layer which decreases the size of feature maps to 16 (from 32). Which then gets passed into the Fully Connected Network, where the actual classification happens.

In the ResBlock / feature_extraction_block there are two subblocks of Conv2d-BatchNorm-ReLU (which is a very common residual block) and the input to the block gets added to the eventual feature map from these aforementioned subblocks. For Example, in the second set of ResBlock the dimension increases from 64 to 128, so the first convolution inputs 64 channels and outputs 128 channels and after passing through the BatchNorm Layer and the ReLU layer the 128-channel feature map gets sent to the other Conv-BatchNorm-ReLU block.

Improvements

The techniques I employed to increase the networks validation accuracy were Normalization, Augmentation, making the model deeper, Batch Normalization, Residual Learning (all of these are explained in depth in the next section). I learnt from these experiments that Batch Normalization Layers work very well as they normalize the layer's feature maps, so adding ReLU after it does not make sense as the feature map is normalized, and it deteriorates the performance. Secondly, adding more and more layers just isn't wise because of issues like Gradient Vanishing, which can be resolved by Residual Learning.

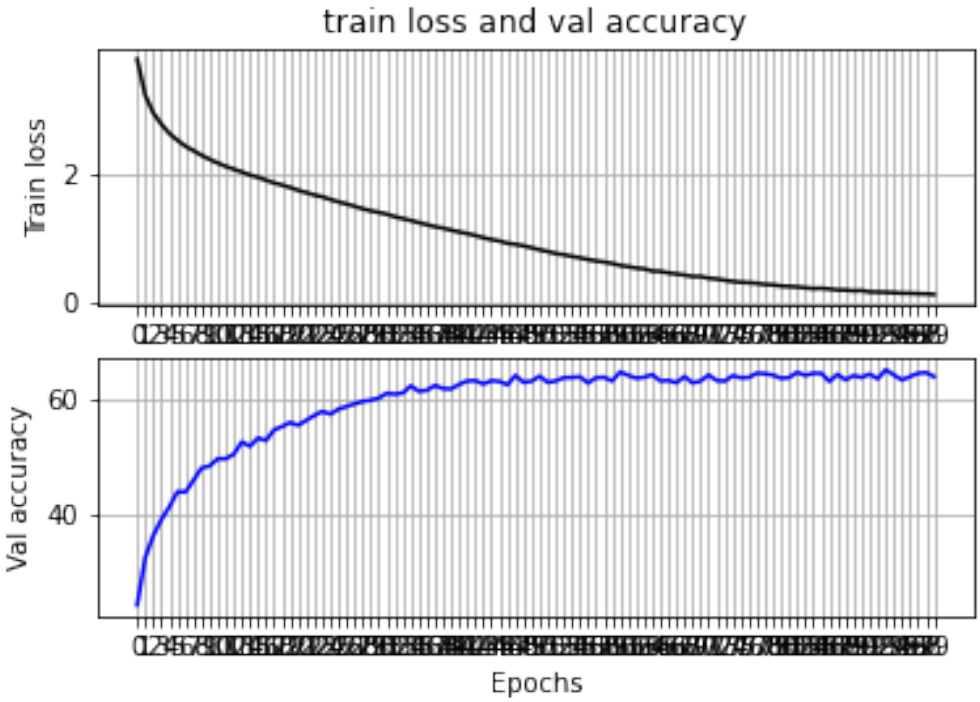
I also tried implementing group convolutions but running the model in its original form would require a lot of V-RAM and decreasing the number of parameters from the network just makes it not worthwhile.

Ablation Study

1. In the initial code I added a BatchNorm layer after the Conv layer and this improved the validation accuracy from 23% to 31%. I did try to insert it after the ReLU, but that approach was not highly effective.
2. After this I increased the number of these Conv Batch Norm subblocks to 5 and had max pooling after the 2nd and the 4th block and the accuracy increased to 38%.
3. Adding another subblock gave another 4% improvement.
4. I moved Conv, BatchNorm and ReLU to a separate class to make its objects and call it many times, after that I added the Conv-BatchNorm-ReLU subblock twice in the same class so to get twice as much learning and this proved the need for residual learning (very deep networks are prone to problems like vanishing gradient). Adding it twice decreased the validation accuracy to 17%.
5. After adding a residual connection (adding the input to the output), the validation accuracy gradually went up. Residual Connection means to take both feature maps of the learnt block and its original input, which relieves issues like vanishing gradients.
6. Multiplying the feature extraction part of the feature_extraction_block with a number like **0.1** gave a massive boost to the accuracy (x = 0.1*x + residual). This took the model back to 48% validation accuracy.
7. I added BatchNorm1D after the first Linear layer, but it decreased the validation accuracy to 42%.
8. Normalizing the dataset gave a 2% increase to the model. I took the whole dataset and calculated the standard deviation and mean values of the images in each channel. I used them as is.
9. I added some more transforms (like Horizontal Flip and Random Crop) to the dataset, which increased the accuracy to 59%, as it made the model more robust to the different transformations.
10. After this I increased the number of epochs to 100, and it increased the validation accuracy to 65%.
11. I tried gradually increasing the size of input convolution output, but this deteriorated the results, so ended up going with just the one convolution.

Results

I submitted the result on Kaggle which got the testing accuracy of **0.62600**.



Part 2: Transfer Learning

Changes to the Network

After getting the built-in model, I employed the following techniques:

- I changed the Number of Epochs to **30** because 10 epochs are not enough for the original ResNet to learn all the features.
- I changed the Learning rate to **0.0005** because the original network was not converging at all.
- Batch Size was set to **32**, usual rule of thumb for me is if the validation accuracy is oscillating, I increase the batch size, and 32 batch size seems to work with this network.
- For any pre-trained network, we must change the last **fully connected** (FC) layer to match our application's needs so for this I added a FC layer with **100** output classes and let the "whole" model to train (and not just the last fully connected layer).

For the Transformations I employed these transformations to the dataset:

- Random Resized Crop: which randomly resizes and scales/crops some images from the dataset to make the model more robust to scales of objects in images. I used some standard parameters for this transformation.
- Random Horizontal Flip: this transformation randomly samples the input dataset and flips the image in the x – axis (Horizontal) direction. This makes the model robust as the learning is enforced by providing more examples of the images with different perspectives.
- Resize: Resizes the images to a particular shape. This was already provided in the code.
- Center Crop: crops the image from the center. This was already provided in the code.
- To Tensor: Converts the images to tensors which can then be used by PyTorch. This was already provided.
- Normalization: as in the previous part, I took the whole dataset and calculated the standard deviation and mean values of the images in each channel. I used them as is.

Results

The original code without any changes gives ~15.5% training accuracy, but after the changes suggested above the model achieved final accuracy 68.67% training accuracy (training loss: 0.0824) and the testing accuracy being 36.04% (testing loss: 0.1022) after 30 epochs (when only Fully Connected layer is trained). When the whole Network is trained for 30 epoch the accuracy goes to 94.57% and the loss went down to 0.0263. The model was robust enough to get 56.31 % testing accuracy (test loss: 0.0603).

TRAINING

TRAINING Epoch 01/30 Loss 0.1687 Accuracy 0.0067
TRAINING Epoch 02/30 Loss 0.1579 Accuracy 0.0323
TRAINING Epoch 03/30 Loss 0.1480 Accuracy 0.1130
TRAINING Epoch 04/30 Loss 0.1382 Accuracy 0.2143
TRAINING Epoch 05/30 Loss 0.1283 Accuracy 0.3230
TRAINING Epoch 06/30 Loss 0.1189 Accuracy 0.4113
TRAINING Epoch 07/30 Loss 0.1104 Accuracy 0.4743
TRAINING Epoch 08/30 Loss 0.1026 Accuracy 0.5310
TRAINING Epoch 09/30 Loss 0.0956 Accuracy 0.5797
TRAINING Epoch 10/30 Loss 0.0891 Accuracy 0.6233
TRAINING Epoch 11/30 Loss 0.0834 Accuracy 0.6753
TRAINING Epoch 12/30 Loss 0.0778 Accuracy 0.7060
TRAINING Epoch 13/30 Loss 0.0726 Accuracy 0.7293
TRAINING Epoch 14/30 Loss 0.0688 Accuracy 0.7420
TRAINING Epoch 15/30 Loss 0.0641 Accuracy 0.7730
TRAINING Epoch 16/30 Loss 0.0597 Accuracy 0.7990
TRAINING Epoch 17/30 Loss 0.0564 Accuracy 0.8067
TRAINING Epoch 18/30 Loss 0.0531 Accuracy 0.8327
TRAINING Epoch 19/30 Loss 0.0497 Accuracy 0.8427
TRAINING Epoch 20/30 Loss 0.0466 Accuracy 0.8560
TRAINING Epoch 21/30 Loss 0.0440 Accuracy 0.8693
TRAINING Epoch 22/30 Loss 0.0413 Accuracy 0.8850
TRAINING Epoch 23/30 Loss 0.0395 Accuracy 0.8833
TRAINING Epoch 24/30 Loss 0.0373 Accuracy 0.8890
TRAINING Epoch 25/30 Loss 0.0348 Accuracy 0.9070
TRAINING Epoch 26/30 Loss 0.0327 Accuracy 0.9150
TRAINING Epoch 27/30 Loss 0.0309 Accuracy 0.9260
TRAINING Epoch 28/30 Loss 0.0291 Accuracy 0.9290
TRAINING Epoch 29/30 Loss 0.0278 Accuracy 0.9343
TRAINING Epoch 30/30 Loss 0.0263 Accuracy 0.9457

TESTING

Test Loss: 0.0603 Test Accuracy 0.5631