Deep Learning Mini-Project

Sri Vikas Prathanapu(sp6904), Kaushik Tummalapalli(kt2651), Sai Narasimha Vayilati(sv2448)

GitHub Repo: https://github.com/srivikas777/CustomResNet.git

Abstract:

We are implementing a custom ResNet model that is trained on the CIFAR10 dataset with less than 5 Million Parameters. The model uses convolutional layers with skip connections to learn the feature representations of the images in the dataset. The CustomResNet class is defined by inheriting from the PyTorch nn.Module class and overriding its constructor and forward methods to build the ResNet model. The model has 4 convolutional blocks, each containing 1 or 4 convolutional layers, and each block is followed by a skip connection that adds the output of the block to the output of the input. The last layer of the model is a fully connected layer that outputs the class probabilities.

Overview and Summary:

The model is trained for 300 epochs with a batch size of 128, using the Ada delta optimizer and the cross-entropy loss function. The training data is augmented with random cropping and horizontal flipping. A validation set is created by randomly selecting 20 of the training data. The training and validation accuracies are monitored after each epoch, and the best model based on the validation accuracy is saved.

After 300 epochs, the final training accuracy is 99.5% and the final validation accuracy is 89.75%. The model summary shows that the **model has 4,729,610 trainable parameters**, which makes it a relatively large model, but the skip connections help to reduce the vanishing gradient problem and improve the model's performance.

In summary, this PyTorch implementation of a custom ResNet model shows how convolutional layers with skip connections can be used to learn the feature representations of images in the CIFAR-10 dataset. The model achieves a high accuracy after 300 epochs of training, and the skip connections help to reduce the vanishing gradient problem. The model summary shows that it has a large number of trainable parameters, but this is necessary to achieve high performance on the task.

Methodology:

The goal of this Project is to achieve high accuracy in classifying images from the CIFAR-10 dataset by creating custom modules. We use **GPU's** for the training process to improve the training time.

Data Preprocessing and DataLoaders:

The data Pre-Processing is done using the torchvision.transforms module. The training data (Data Augmentation) is randomly cropped with padding of 4 pixels and flipped horizontally. The data is then normalized with the mean and standard deviation of the dataset. The training and validation data are loaded using the torchvision.datasets module. The training data is further split into a training and testing set, with a 80-20 split. **The training, validation, and test sets are loaded into PyTorch's DataLoader class.**

ResNet Architecture (Custom):

We define the **custom ResNet Architecture with the help of PyTorch's nn.Module class**. The custom ResNet model has 4 layers with 1, 1, 1, and 4 blocks in each layer, respectively. The number of blocks in each layer is stored in the number blocks array. Each block contains two convolutional layers, each followed by batch normalization and ReLU activation. Each layer has a stride of 1 except for the first layer, which has a stride of 2. The input to the ResNet model is a 32x32x3 image, which is passed through the first convolutional layer with a kernel size of (3, 1). The output of each layer is passed through an average pooling layer with a kernel size of 4 times the multiplier, where the multiplier is 4 for the second layer, 2 for the third layer, and 1 for the fourth layer. Finally, the output of the last layer is passed through a fully connected linear layer to obtain the final classification scores.

Optimizers:

The model is then trained using the **Adadelta optimizer with a learning rate of 0.1 and weight decay of 0.0001**. The loss function used is Cross Entropy Loss. The **model is trained for a total of 300 epochs**, with early stopping used to prevent overfitting. The best model is selected based on the highest validation accuracy.

Train and Eval Methods:

The train and eval() methods are used to train out custom neural network, where in training process we iterate through the dataloader and get the predictions and we perform the back-propogation and update the model parameters using the optiomizer. We use the criterion to predict the loss for the outputs and labels. We don't perform the back-prop step in the eval() and everything is mostly the same where while iterating through the validation data lodaer, we calculate the validation accuracy and loss.

Pros and Cons:

The following are the **Pros and Cons of different architectural choices**:

- The use of convolutional layers followed by batch normalization and ReLU activation in each block of the ResNet architecture is a standard choice that has been shown to work well in practice.
- The use of average pooling after each layer with a kernel size of 4 times the multiplier is a good choice as it helps to reduce the spatial dimension of the feature maps while maintaining the number of channels.
- The use of an initial convolutional layer with a kernel size of $(3, 1)$ is an interesting choice as it allows the model to capture features along the time dimension in addition to the spatial dimensions.
- The use of early stopping is a good choice as it helps to prevent overfitting and improve generalization performance.
- The use of an Ada delta optimizer with a learning rate of 0.1 and weight decay of 0.0001 is a reasonable choice, but other optimizers such as SGD with momentum or Adam may also work well.

During the design process, the following lessons were learned:

- The choice of the kernel size and number of blocks in each layer of the ResNet architecture has a significant impact on the performance of the model. It is important to experiment with different configurations to find the best one.
- Data augmentation can significantly improve the performance of the model, especially when there is a limited amount of training data.
- Preprocessing the data by normalizing it with the mean and standard deviation of the dataset is important as it helps to stabilize the training process and improve the performance of the model.

Results:

The model was trained for 300 epochs and achieved a **training accuracy of 99% and a validation accuracy of 90%**.

Model Architecture:

A custom implementation of ResNet with 4 layers, each with 1, 1, 1, and 4 blocks, respectively.

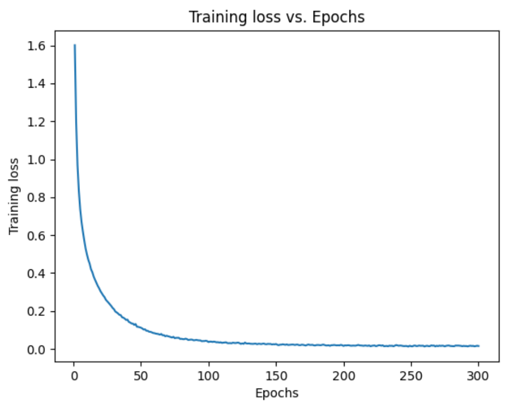

Figure 1: Training Loss VS Number of Epochs

.

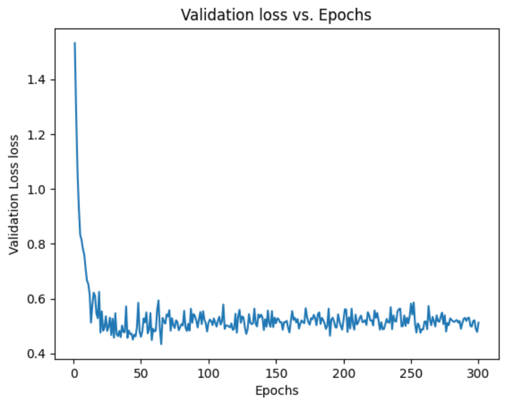

Figure 2: Validation Loss VS Number of Epochs

.

Number of Parameters: 4,729,610

The code defines a CustomResNet class that inherits from nn.Module. The constructor of the class initializes the attributes of the model such as the number of layers, the number of blocks per layer, the size of the convolutional kernel, etc. The make-layer method creates the residual blocks that are used in each layer, and the forward method defines the forward pass of the model.

The ResNet has 4 layers, each with a different number of blocks, and each block consists of two convolutional layers with batch normalization and ReLU activation. The output of the last layer is passed through a global average pooling layer and a fully connected layer with 10 output units (one for each class in the CIFAR-10 dataset).

The total number of trainable parameters in the model is 4,729,610, which is printed using the summary function from torchsummary. The function also shows the input size, the forward/backward pass size, and the size of the parameters in memory.

Citations:

References

[1] PyTorch Tutorial. (n.d.). ResNet on CIFAR10. *PyTorch Tutorial*. Retrieved April 14, 2023, from https://pytorch-tutorial.readthedocs.io/en/latest/tutorial/chapter03_intermediate/3_2_2_cnn_resnet_cifar10/

[2] Gulrajani, I. (2017). ResNets for CIFAR10. *Towards Data Science*. Retrieved April 14, 2023, from https://towardsdatascience.com/resnets-for-cifar-10-e63e900524e0

[3] Krizhevsky, A., Hinton, G. (n.d.). Learning multiple layers of features from tiny images. *University of Toronto*. Retrieved April 14, 2023, from https://www.cs.toronto.edu/~kriz/cifar.html