Resnet Report

What is Resnet?

Residual Network (ResNet) is a deep learning model used for computer vision applications. It is a Convolutional Neural Network (CNN) architecture designed to support hundreds or thousands of convolutional layers. Previous CNN architectures were not able to scale to a large number of layers, which resulted in limited performance. However, when adding more layers, researchers faced the "vanishing gradient" problem.

Neural networks are trained through a backpropagation process that relies on gradient descent, shifting down the loss function and finding the weights that minimize it. If there are too many layers, repeated multiplications will eventually reduce the gradient until it "disappears", and performance saturates or deteriorates with each layer added.

ResNet provides an innovative solution to the vanishing gradient problem, known as "skip connections". ResNet stacks multiple identity mappings (convolutional layers that do nothing at first), skips those layers, and reuses the activations of the previous layer. Skipping speeds up initial training by compressing the network into fewer layers.

Then, when the network is retrained, all layers are expanded and the remaining parts of the network—known as the residual parts—are allowed to explore more of the feature space of the input image.

Most ResNet models skip two or three layers at a time with nonlinearity and batch normalization in between. More advanced ResNet architectures, known as Highway Nets, can learn "skip weights", which dynamically determine the number of layers to skip.

DenseNet is another popular ResNet variation, which attempts to resolve the issue of vanishing gradients by creating more connections. The authors of DenseNet ensured the maximum flow of information between the network layers by connecting each layer directly to all the others. This model preserves the feed-forward capabilities by allowing every layer to obtain additional inputs from its preceding layers and pass on the feature map to subsequent layers.

Code Summary

The `sport_dataset` class is a custom dataset class for a sport classification task. It takes as input a root directory for the images, a DataFrame with the image names and labels, and optional `transform` and `target_transform` arguments. The `__len__` method returns the number of images in the dataset, and the `__getitem__` method returns a tuple of an image and its label, both transformed if the corresponding optional arguments are provided. The `sport_dataset_test` class is similar, but it only takes a root directory and an optional `transform` argument as input, and returns a tuple of an image and its file name. The `num_classes` and `num_epochs` variables are hyperparameters for the model, and `BATCH_SIZE` and `learning_rate` are hyperparameters for the training process. The `train_test_split` function from `sklearn` is used to split the data into a training set and a validation set, with a specified split size and shuffle option. The `value_counts` method is used to count the number of instances of each class in the training set and the `max` method is used to get the maximum count of instances among the classes. The `vct` DataFrame is created to store the difference between the maximum count of instances and the count of instances for each class, and a loop is used to apply transformations to the images in the training set to balance the class distribution. The code imports the `make_grid` function from `torchvision.utils` and the `ResidualBlock` and `ResNet` classes from `torch.nn`. The `for` loop iterates through the images in `train_loader` and displays them using `matplotlib`.

The `ResidualBlock` class is a subclass of `nn.Module` and it defines a residual block for a residual neural network (ResNet). It takes as input the number of input channels, the number of output channels, a stride, and an optional downsample layer. It has two convolutional layers with batch normalization and SiLU activation, and an optional downsample layer. The `forward` method computes the forward pass of the residual block.

The `ResNet` class is a subclass of `nn.Module` and it defines a ResNet model for image classification. It takes as input a block class, a list of integers representing the number of blocks in each layer, and the number of classes. It has a convolutional layer, a max pooling layer, and four residual blocks. It also has an average pooling layer and a fully connected layer. The `_make_layer` method creates a layer of residual blocks, with an optional downsample layer if the stride is greater than 1 or the number of input channels is different from the number of output channels. The `forward` method computes the forward pass of the ResNet model.

This code trains a ResNet model on the sport classification dataset and displays the training and validation loss and accuracy plots. The `ResNet` model is initialized with the `ResidualBlock` class and a list of 4 layers, and moved to the specified device. A cross entropy loss is defined and an Adam optimizer is used to minimize the loss. The model is trained in a loop over the number of epochs, with an inner loop over the training data in `train_loader`. The training loss and accuracy are calculated and appended to the corresponding lists after each epoch. The model's performance on the validation set is also evaluated after each epoch and the validation loss and accuracy are appended to the corresponding lists. The best model, based on validation accuracy, is saved. At the end of training, the training and validation loss and accuracy plots are displayed and the maximum validation accuracy is printed. Finally, a confusion matrix is constructed using the predictions on the validation set. It looks like you are training a ResNet model for a 6-class classification problem using the SiLU activation function. You are using the Adam optimizer and Cross Entropy Loss to train the model. You are also using the confusion matrix and ConfusionMatrixDisplay to visualize the performance of the model on the validation set.

The code seems to be correctly implementing the training loop, where in each epoch, the model is first put in training mode and the training dataset is used to update the model's parameters. You are also keeping track of the training loss and accuracy in each epoch. After the training loop, the model is put in evaluation mode and the validation set is used to evaluate the model's performance. You are keeping track of the validation loss and accuracy and also saving the model with the best validation accuracy. Finally, you are using the validation set to generate the confusion matrix to visualize the model's performance.

Thanks