CIFAR-10 Classification with MLP & ResNet-20

Jiyeong Oh, Dpt.DataScience

Hanyang University

Introduction

The difference between a fully connected neural network (MLP) and a convolutional neural network (CNN) lies in the presence (or absence) of 'key point extraction'. MLP produces a one-dimensional array of images and puts them in a neural network as input to calculate the weight, while CNN has less information loss to extract 'features' from the two-dimensional array of images, just like it is seen by humans. In addition, the computation of CNN is less than MLP because only the weights for the key points extracted by pooling are calculated.

This report covers the process of comparing two models: MLP and ResNet-20. By implementing each model and using it to classify CIFAR-10 data, the goal is to understand how each model works and analyze which model is advantageous for classifying image data.

Load CIFAR-10 Data – Data Split

Database used in this report is CIFAR-10, which includes images with the size of 32*32*3 (where 32 is its height and width, and 3 is the number of channels) and 10 labels. To apply the training, validation testing and testing process with the given data, function "val_split" which splits the train data into train/validation set with the same class proportion, was created.

```
def val_split(attr, tar): # attr: features, tar: target label
```

The function performs split taking two arguments: attribute and target(label).

First, with the target data, a dictionary with answer label as key and its indices as values, are being created. Also, in order to prevent data from not being mixed in the process of training/test split, the values under each key are shuffled here.

```
y_dict = dict(zip([i for i in np.unique(tar)], [list(np.where(tar==i))[0] for i in np.unique(tar)]))
y_dict = dict(zip([i for i in np.unique(tar)], [list(np.random.choice(y_dict[i], len(y_dict[i]), replace = False)) for i in y_dict.keys()])) # Shuffling
```

With the shuffled dictionary, dataset indices are trimmed at a consistent rate for each key: 500 images for validation set, and left-overs for train set.

```
val_tmp = [y_dict[i][:500] for i in sorted(y_dict.keys())] # Trimming validation set (500 for each label)
train_tmp = [y_dict[i][500:] for i in sorted(y_dict.keys())] #Ttrimming training set (leftovers)
```

With indices for train/validation saved in different list, value indexing was applied to split the train dataset into two. Finally, converting the fetched data into Pytorch Tensor type was applied here also.

```
X_train, y_train = torch.tensor(attr[train_idx], device=cuda), torch.tensor(tar.loc[train_idx].to_numpy(), device=cuda)
X_val, y_val = torch.tensor(attr[val_idx], device=cuda), torch.tensor(tar.loc[val_idx].to_numpy(), device=cuda)

return X_train, y_train, X_val, y_val
```

MLP for CIFAR-10

(a) Classifier Building & Train/Val/Test Code Description

First, model class “MLP_multi” was initialized. Two hidden layers with 256 units were built.

```
self.module1 = nn.Linear(input_dim, 256, bias=True) #input layer -> hidden layer1
self.module2 = nn.Linear(256, 256, bias=True) #hidden layer1 -> hidden layer2
self.module3 = nn.Linear(256, output_dim, bias=True) #hidden layer 2 -> output layer.
```

The dimension of output layer should be the number of classes: 10.

As a next step, a function ‘multiMLP’ in which the class defined above is used, was created. Unlike CNN, this MLP function includes an additional process to make the image matrix into a one-dimensional array.

```
#####################################
## reshaping Xs, only for MLP
#####################################
X_train = X_train.reshape(X_train.shape[0], 32*32*3)
X_val = X_val.reshape(X_val.shape[0], 32*32*3)
X_test = X_test.reshape(X_test.shape[0], 32*32*3)
data_size = len(X_train)
```

Loss function was set to cross entropy to make the model follow the distribution of correct answer, and softmax was used as activation function. As cross entropy in Pytorch applies softmax instead, the softmax code was not used in model class defining process.

```
criterion = nn.CrossEntropyLoss() # loss function
```

```
model = MLP_multi(X_train.shape[1], 10).cuda()
```

(b) Training for 5 epochs with batch size 64

Training and checking validation accuracy along 5 epochs with batch size of 64, the validation accuracy started from 0.29, kept increased and ended with 0.42.

```
Epoch Train Loss 0:  1771.42 , Validation Accuracy:  0.29
Epoch Train Loss 1:  1321.96 , Validation Accuracy:  0.30
Epoch Train Loss 2:  1245.99 , Validation Accuracy:  0.33
Epoch Train Loss 3:  1189.19 , Validation Accuracy:  0.41
Epoch Train Loss 4:  1164.84 , Validation Accuracy:  0.42
```

(c) Final test Accuracy

With the hyperparameters set below, the final test accuracy ended up with 0.42.

```
epochs_num = 5
batch_size = 64
optimizer_choose = 'Adam'
learning_rate = 0.0001
```

```
Final Test Accuracy:  0.42
```

Despite training the model during 5 epochs, it was confirmed that the performance of the model was not excellent. It was inferred that the reason for the poor performance score was that the ‘location information’ of the image was lost in the process of flattening the image in one dimension.

ResNet-20 for CIFAR-10

(a) Classifier Building & Train/Val/Test Code Description

Referring to the ResNet paper, the ResNet-20 model was implemented. First, the building block, which is responsible for residual learning that constitutes each layer of the ResNet model, was defined as class 'Building_Block'. The basic form of building block is the sequence of two convolution structures, batch normalization following each convolution, and skip connection that proceeds identity mapping to the result of ReLU between the two convolutions.

```
class Building_Block(nn.Module):
  def __init__(self, input_dim, output_dim, downsample=None, stride=1):
    super(Building_Block, self).__init__()
    self.conv1 = nn.Conv2d(input_dim, output_dim, kernel_size=3, stride=stride, padding=1, bias=False)
    self.batchNorm1 = nn.BatchNorm2d(output_dim)
    self.relu = nn.ReLU(inplace=True)
    self.conv2 = nn.Conv2d(output_dim, output_dim, kernel_size=3, stride=1, padding=1, bias=False)
    self.batchNorm2 = nn.BatchNorm2d(output_dim)
    self.downsample = downsample
    self.stride = stride

  def forward(self, x):
    identity_mapping = x

    out = self.conv1(x)
    out = self.batchNorm1(out)
    out = self.relu(out)
    out = self.conv2(out)
    out = self.batchNorm2(out)

    # skip connection
    if self.downsample != None:
      identity_mapping = self.downsample(x)
    out += identity_mapping
    out = self.relu(out)

    return out
```

Subsequently, class ‘ResNet-20’ was defined in a manner of stacking the generated basic building block class.

```
def forward(self, x):
  #first layer
  out = self.conv1(x)
  out = self.batchNorm1(out)
  out = self.relu(out)

  out = self.conv2_x(out)
  out = self.conv4_x(out)
  out = self.conv6_x(out)

  out = self.avgpool(out)
  out = torch.reshape(out, (-1,batch_size))
  y_pred = self.fc(out)

  return y_pred
```

Starting with the first layer of 3*3, 3 sets of layers with 3 building blocks were stacked, the network ends with fully connected layer followed by global average pooling; which is, total 20 layers are created.

In the second and third building blocks of the layer set, where the output map size does not decrease,

the stride is set to 1. However, in the ‘first’ convolution of the ‘first’ building block of each layer set, where the output map size has to be reduced, the stride was set to 2.

```
def make_layer(self,in_channel, out_channel):
  if in_channel != out_channel:
    downsample = nn.Sequential(
        nn.Conv2d(in_channel, out_channel, kernel_size=3, stride=2, padding=1, bias=False),
        nn.BatchNorm2d(out_channel)
    )
    layers_list = nn.ModuleList([Building_Block(in_channel, out_channel, downsample=downsample, stride=2)])
  else:
    layers_list = nn.ModuleList([Building_Block(in_channel, out_channel, downsample=None, stride=1)])

  for i in range(2): # 3-1 = 2
    layers_list.append(Building_Block(out_channel, out_channel, downsample=None, stride=1))
  return nn.Sequential(*layers_list)
```

This difference had been coded under the function ‘make_layer’. Setting stride to 2 to reduce output map size, and applying down-sampling in skip connection to match computational dimensions, were coded to match whether the number of output channels received by the argument was different from the number of input channels.

Just like MLP, loss function was set to cross entropy to make the model follow the distribution of correct answer, and softmax was used as activation function. As cross entropy in Pytorch applies softmax instead, the softmax code was not used in model class defining process. Validation accuracy along epoch, and the final test accuracy calculation with test set, is being applied, as usual.

```
model = ResNet_20().cuda()
```

However, unlike MLP, in ResNet, the dataset is not transformed in one dimension, but the image matrix itself is used as it is.

(b) Training for 5 epochs with batch size 64

Tuning to find model with good performance was conducted by changing the learning rate, which is a typical hyperparameter. As shown in the instruction in the paper, the initial learning rate was set to 0.1 and different values of learning rate (dividing it by 10) were attempted while monitoring the performance changes. The learning rate with the best validation accuracy was confirmed to be 0.0001, and the validation accuracy started from 0.49, kept increased and ended with 0.63 while training and checking validation accuracy along 5 epochs with batch size of 64.

```
Epoch Train Loss 0:  1110.69 , Validation Accuracy:  0.49
Epoch Train Loss 1:  938.98 , Validation Accuracy:  0.54
Epoch Train Loss 2:  855.43 , Validation Accuracy:  0.56
Epoch Train Loss 3:  798.90 , Validation Accuracy:  0.59
Epoch Train Loss 4:  757.93 , Validation Accuracy:  0.63
```

(c) Final test Accuracy

With the hyperparameters set below, the final test accuracy ended up with 0.62.

```
epochs_num = 5
batch_size = 64
optimizer_choose = 'Adam'
learning_rate = 0.0001
```

```
Final Test Accuracy:  0.62
```

Compared to MLP, it was confirmed that the performance score of ResNet-20 was significantly improved. This improvement in performance points to the advantages of the Convolutional Neural Network (CNN), which is the base of ResNet, as an image classification model; learning while maintaining spatial information of the image by not planarizing the photo data, and using the same filter to extract the same features.