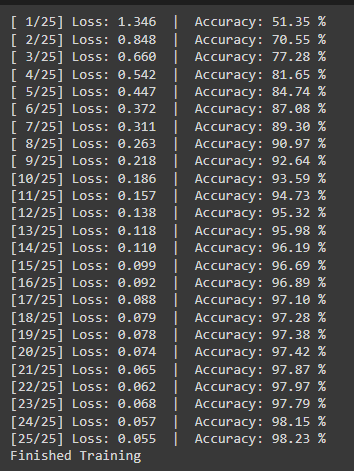
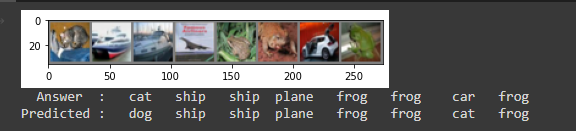
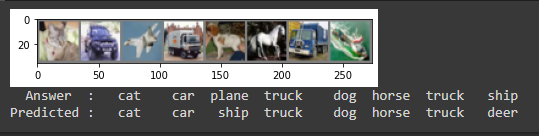
程式執行畫面：

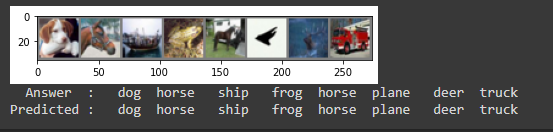
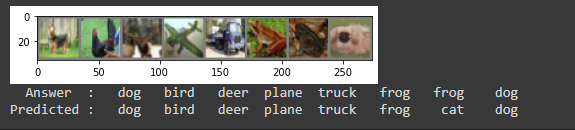
訓練（25輪）：



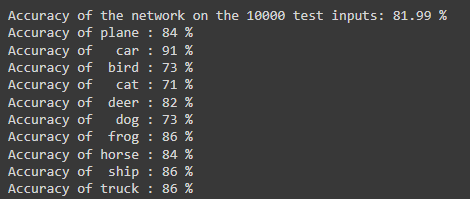
測試（大概執行了幾次）：

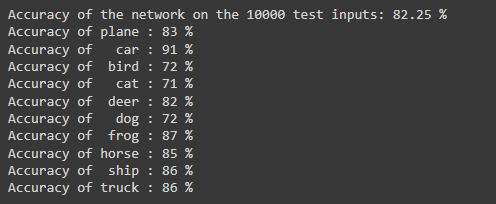


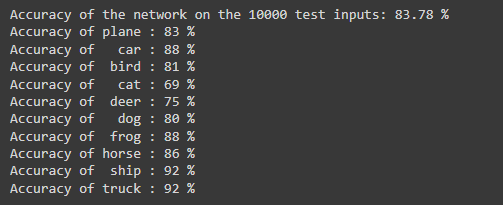










程式碼（以下程式碼皆可以複製）：

模型：

|  |
| --- |
| class Net(nn.Module):      def \_\_init\_\_(self):          super(Net, self).\_\_init\_\_()          self.features = nn.Sequential(              nn.Conv2d(3, 64, kernel\_size=3, padding=2),              nn.BatchNorm2d(64),              nn.ReLU(inplace=True),              nn.MaxPool2d(kernel\_size=3, stride=2),              nn.Conv2d(64, 192, kernel\_size=3, padding=2),              nn.BatchNorm2d(192),              nn.ReLU(inplace=True),              nn.MaxPool2d(kernel\_size=3, stride=2),              nn.Conv2d(192, 384, kernel\_size=3, padding=1),              nn.BatchNorm2d(384),              nn.ReLU(inplace=True),              nn.Conv2d(384, 256, kernel\_size=3, padding=1),              nn.BatchNorm2d(256),              nn.ReLU(inplace=True),              nn.Conv2d(256, 256, kernel\_size=3, padding=1),              nn.BatchNorm2d(256),              nn.ReLU(inplace=True),              nn.MaxPool2d(kernel\_size=3, stride=2),          )          self.avgpool = nn.AdaptiveAvgPool2d((7, 7))          self.classifier = nn.Sequential(              nn.Flatten(),              nn.Linear(256\*7\*7, 4096),              nn.ReLU(inplace=True),              nn.Dropout(p = 0.5),              nn.Linear(4096, 1024),              nn.ReLU(inplace=True),              nn.Dropout(p = 0.5),              nn.Linear(1024, 10)          )        def forward(self, x):          x = self.features(x)          x = self.avgpool(x)          x = x.view(x.size(0), -1)          x = self.classifier(x)          x = F.log\_softmax(x, dim=1)          return x |

模型說明：

我是參考AlexNet的架構所改出來的，因此可以看到**共8層**（1~5層為卷積層，6~8為全連接層）。

features內有**五層卷積層**（Convolutional Layers），其中在第一、二、五個卷積層後使用大小3x3、stride=2的Maxpooling。

classifier內有**三層全連接層**（Fully Connected Layers），其中第一、二全連接層後使用Dropout，隨機關閉神經元，藉此讓模型不過度依賴某些特徵，增強模型的泛化能力。

並且在輸出前，先用log型的Softmax函數（不過這個影響沒有很大）。

我已經盡力改了，但因為第一次碰，所以有點不太熟，原本想try看看其他模型（如ResNet、VGG之類的），但有些很複雜我看不懂QQ。改了很多時間才終於把準確率提升到80%以上，上不去了…。

訓練前的設定（參數、優化器等）：

|  |
| --- |
| net = Net().cuda()  # Parameters  criterion = nn.CrossEntropyLoss()  criterion.cuda()  lr = 0.0002  epochs = 25  optimizer = optim.Adam(net.parameters(), lr=lr, betas=(0.9, 0.999), eps=1e-08, weight\_decay=0.0001, amsgrad=False) |

參數說明：

損失函數使用CrossEntropy。

訓練25輪，其實到10多輪左右就有到80%以上了，抓25是因為較穩定，且也上不太去了。

原本優化器（optimizer）是使用SGD，而我嘗試換成Adam，有因此提升準確率。

訓練：

|  |
| --- |
| # Train  for epoch in range(epochs):      net.train()      running\_loss = 0.0      accuracy = 0      count = 0      for data in trainLoader:          inputs, labels = data          inputs, labels = inputs.cuda(), labels.cuda() # GPU          # Zero the parameter gradients          optimizer.zero\_grad()          # forward + backward + optimize          outputs = net(inputs)          loss = criterion(outputs, labels)          \_, predicted = torch.max(outputs.data, 1)          accuracy += (predicted == labels).sum().item()          count += len(inputs)          running\_loss += loss.item() \* len(labels)          loss.backward()          optimizer.step()      print('[%2d/%2d] Loss: %.3f  |  Accuracy: %.2f %%' % (epoch+1, epochs, running\_loss / count, 100 \* accuracy / count))  print('Finished Training') |

訓練說明：

基本上跟原本差不多，只是我改成訓練完一輪再輸出，並輸出該輪損失率及準確率。

測試：

|  |
| --- |
| # show image function  import matplotlib.pyplot as plt  def imshow(img):    img = img / 2 + 0.5    npimg = img.numpy()    plt.imshow(np.transpose(npimg, (1,2,0)))    plt.show()  # show test images dataset  dataiter = iter(testLoader)  images, labels = dataiter.next()  imshow(torchvision.utils.make\_grid(images))  print("  Answer  :", "  ".join("%5s" % classes[labels[i]] for i in range(8)))  outputs = net(images.cuda())  \_, predicted = torch.max(outputs, 1)  print("Predicted :", "  ".join("%5s" % classes[predicted[i]] for i in range(8))) |

測試說明：

這邊會輸出圖片（可以去Colab那邊看）以及其文字與預測的文字，便於比較，圖片數量依上方Cifar-10資料指定的batchSize決定。

然後下方也有原本的測試程式碼，用以看10000筆資料及各項類別的準確率。

|  |
| --- |
| # Test  net.eval()  correct = 0  total = 0  with torch.no\_grad():      for data in testLoader:          inputs, labels = data          inputs, labels = inputs.cuda(), labels.cuda()          outputs = net(inputs)          \_, predicted = torch.max(outputs.data, 1)          total += len(inputs)          correct += (predicted == labels).sum().item()  print('Accuracy of the network on the 10000 test inputs: %.2f %%' % (100 \* correct / total))  class\_correct = list(0. for i in range(10))  class\_total = list(0. for i in range(10))  with torch.no\_grad():      for data in testLoader:          inputs, labels = data          inputs, labels = inputs.cuda(), labels.cuda()          outputs = net(inputs)          \_, predicted = torch.max(outputs, 1)          c = (predicted == labels).squeeze()          for i in range(8):              label = labels[i]              class\_correct[label] += c[i].item()              class\_total[label] += 1  for i in range(10):      print('Accuracy of %5s : %2d %%' % (classes[i], 100 \* class\_correct[i] / class\_total[i])) |