

Homework #3, 繳交期限 2024/11/08
Classification & Convolution Neural Network

系所：工海所

姓名：劉樺

學號：R13525030

cifar10 資料集內含 10 個類別的圖片，分別是飛機、汽車、鳥、貓、鹿、狗、青蛙、馬、船、卡車，與 mnist 的主要不同之處在於維度，cifar10 是彩色圖片的資料集，有三個 channel，mnist 的黑白圖片僅有一個 channel。cifar10 共有 60000 張 32x32 大小的彩色圖片，被分成 50000 張的訓練集與 10000 張的測試集，每個類別有 6000 張圖片。請以作業中之 sample_code_3.ipynb 來下載 cifar10 資料集、畫出訓練集前 9 張圖形以及將圖片放入 DataLoader (PyTorch) 中。根據所提供的程式，建立全連接層之神經網路以訓練可辨識 cifar10 測試集之模型，畫出測試集前 25 張圖與其對應之辨識類別，並計算辨識正確率。



Homework #3, 繳交期限 2024/11/08

Classification & Convolution Neural Network

系所：工海所

姓名：劉樺

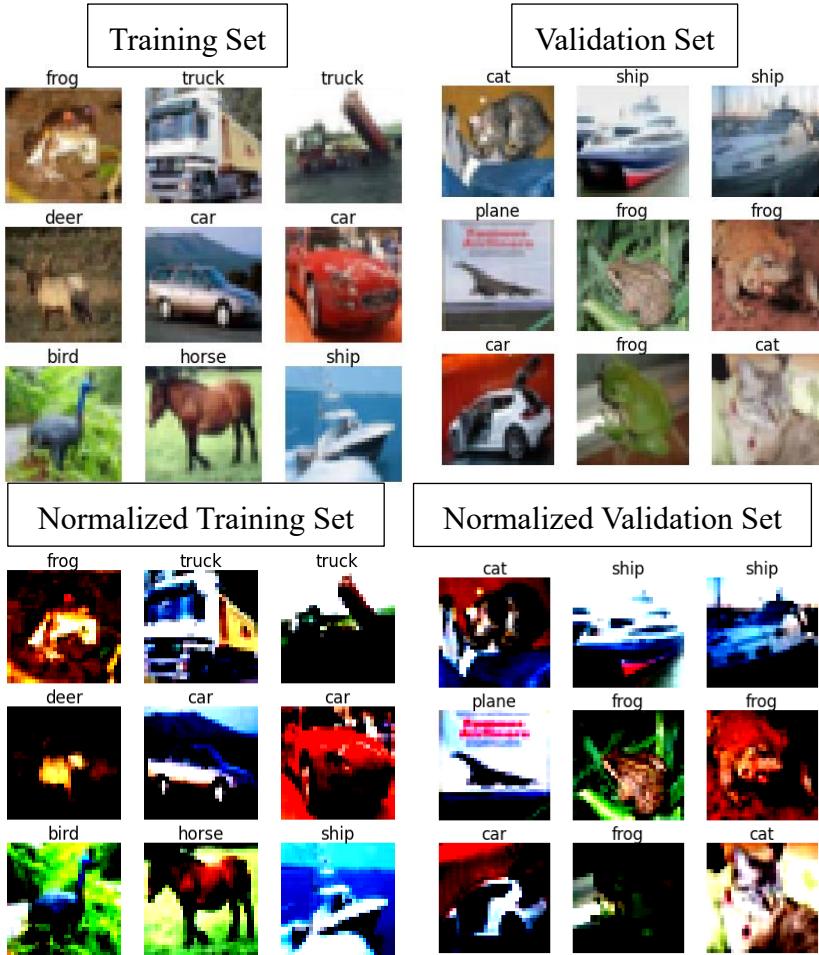
學號：R13525030

Basic

將 Configuration block 中之 XX 填入自己設定之 hyperparameters，同時將 Model block 中之 XX 填入通道與神經元數量，跑出結果圖與準確率即可達到 basic score。

```
● ● ●
1 #Configuration
2 ''' Homework!!! You have to fill the number in this block to achieve basic score '''
3 input_dim=3072
4 lr=1e-3 # learning rate, suggest from 1e-3 to 1e-5
5 batch_size=10 # batch size, suggest 10~250
6 n_epoch=1000 # number of epoch, suggest from 1000 to 5000
7 best_loss=100000 # initial best loss, suggest large enough like 100000
8 device=torch.device("cuda" if torch.cuda.is_available() else "cpu")
9
```

```
● ● ●
1 #Model(Convolution)
2 ''' Homework!!! You have to fill the number in this block to achieve basic score '''
3 class Convnet(nn.Module):
4     def __init__(self):
5         super().__init__()
6         self.conv1=nn.Conv2d(3, 32, kernel_size=3, padding=1)
7         self.act1=nn.Tanh()
8         self.pool1=nn.MaxPool2d(2)
9         '''Homework!!! you can add or change the code below to achieve intermediate level...
10        #self.dropout1=nn.Dropout2d(p=xx)
11        self.conv2=nn.Conv2d(32,8, kernel_size=3, padding=1)
12        self.act2=nn.Tanh()
13        self.pool2=nn.MaxPool2d(2)
14        #self.dropout2=nn.Dropout2d(p=xx)
15        self.linear1=nn.Linear(8*8*8,128)
16        self.act3=nn.Tanh()
17        self.linear2=nn.Linear(128,10)
```



```

Epoch: 50, Loss: 0.9203516244888306
Epoch: 51, Loss: 1.1668117046356201
Epoch: 52, Loss: 0.9157412648200989
Epoch: 53, Loss: 0.9086316823959351
Epoch: 54, Loss: 0.8461686372756958
Epoch: 55, Loss: 1.0263599157333374
Epoch: 56, Loss: 0.4591291546821594
Epoch: 57, Loss: 0.9789746999740601
Epoch: 58, Loss: 0.7611640095710754
Epoch: 59, Loss: 0.5228555798530579
Saving best model with best loss: 0.06850697100162506
Epoch: 60, Loss: 0.6215969920158386
Epoch: 61, Loss: 0.6016831398010254
Epoch: 62, Loss: 0.4951845109462738
Epoch: 63, Loss: 0.8691619634628296
Epoch: 64, Loss: 0.5538606643676758
Epoch: 65, Loss: 0.7075468897819519
Epoch: 66, Loss: 0.8665778040885925
Epoch: 67, Loss: 0.7312382459640503
Epoch: 68, Loss: 1.0528271198272705
Epoch: 69, Loss: 1.3647081851959229
Epoch: 70, Loss: 0.23839251697063446
Epoch: 71, Loss: 0.6889392137527466
Epoch: 72, Loss: 0.41265735030174255
Epoch: 73, Loss: 0.6470400094985962
Epoch: 74, Loss: 0.6149163246154785
Epoch: 75, Loss: 0.6285303831100464
Epoch: 76, Loss: 1.0645819902420044
Epoch: 77, Loss: 0.2758195102214813
Epoch: 78, Loss: 0.949715735281372
Epoch: 79, Loss: 0.10110101848840714
Epoch: 80, Loss: 0.7475836873054504
Saving best model with best loss: 0.06676698476076126
Epoch: 81, Loss: 0.6071261167526245

```

```

Epoch: 965, Loss: 0.007453274913132191
Epoch: 966, Loss: 0.007489409297704697
Epoch: 967, Loss: 0.013977378606796265
Epoch: 968, Loss: 0.0037452217657119036
Epoch: 969, Loss: 0.0033754161559045315
Epoch: 970, Loss: 0.009321331977844238
Epoch: 971, Loss: 0.005408561322838068
Epoch: 972, Loss: 0.006944934371858835
Epoch: 973, Loss: 0.014832854270935059
Epoch: 974, Loss: 0.008800619281828403
Epoch: 975, Loss: 0.010859649628400803
Epoch: 976, Loss: 0.007055880967527628
Epoch: 977, Loss: 0.007526332978159189
Epoch: 978, Loss: 0.0009498968720436096
Epoch: 979, Loss: 0.0038067144341766834
Epoch: 980, Loss: 0.007968791760504246
Epoch: 981, Loss: 0.004997144918888807
Epoch: 982, Loss: 0.00902383582544518
Epoch: 983, Loss: 0.004283061716705561
Epoch: 984, Loss: 0.009103351272642612
Epoch: 985, Loss: 0.008603313006460667
Epoch: 986, Loss: 0.006301791872829199
Epoch: 987, Loss: 0.007796525023877621
Epoch: 988, Loss: 0.010192962363362312
Epoch: 989, Loss: 0.007001877762377262
Epoch: 990, Loss: 0.009093978442251682
Epoch: 991, Loss: 0.0028148475103080273
Epoch: 992, Loss: 0.0039023221470415592
Epoch: 993, Loss: 0.0038134255446493626
Epoch: 994, Loss: 0.002082269173115492
Epoch: 995, Loss: 0.00902992021292448
Epoch: 996, Loss: 0.008577311411499977
Epoch: 997, Loss: 0.002414670074358525
Epoch: 998, Loss: 0.011229267343878746
Epoch: 999, Loss: 0.008806637488305569

```

Correction: 4.0%
Correction: 8.0%
Correction: 12.0%
Correction: 12.0%
Correction: 12.0%
Correction: 16.0%
Correction: 16.0%
Correction: 20.0%
Correction: 24.0%
Correction: 28.000000000000004%
Correction: 32.0%
Correction: 36.0%
Correction: 40.0%
Correction: 44.0%
Correction: 44.0%
Correction: 48.0%
Correction: 52.0%
Correction: 56.00000000000001%
Correction: 56.00000000000001%
Correction: 60.0%
Correction: 64.0%
Correction: 64.0%

Homework #3, 繳交期限 2024/11/08

Classification & Convolution Neural Network

系所：工海所

姓名：劉樺

學號：R13525030

Intermediate

為了克服 CNN overfitting(過擬合)之問題，因此可在每層 CNN 之間加入 Dropout 以機率方式增加訓練之隨機性，進而克服 overfitting 之問題。

```
1 #Model(Convolution)
2 class Convnet(nn.Module):
3     def __init__(self):
4         super().__init__()
5         self.conv1 = nn.Conv2d(3, 32, kernel_size=3, padding=1)
6         self.act1 = nn.Tanh()
7         self.pool1 = nn.MaxPool2d(2)
8         self.dropout1 = nn.Dropout2d(p=0.25) # 添加 dropout
9
10        self.conv2 = nn.Conv2d(32, 8, kernel_size=3, padding=1)
11        self.act2 = nn.Tanh()
12        self.pool2 = nn.MaxPool2d(2)
13        self.dropout2 = nn.Dropout2d(p=0.25) # 添加 dropout
14
15        self.linear1 = nn.Linear(8*8*8, 128)
16        self.act3 = nn.Tanh()
17        self.dropout3 = nn.Dropout(p=0.5)      # 添加 dropout
18        self.linear2 = nn.Linear(128, 10)
```

1. 添加了三個 Dropout 層：

- dropout1: 在第一個卷積層後，dropout 率為 0.25
- dropout2: 在第二個卷積層後，dropout 率為 0.25
- dropout3: 在全連接層後，dropout 率為 0.5

2. 在 forward 函數中相應地加入了 dropout 操作



```

Epoch: 38, Loss: 1.1035724878311157
Epoch: 39, Loss: 1.2980093955993652
Epoch: 40, Loss: 0.7891412377357483
Epoch: 41, Loss: 1.2902050018310547
Epoch: 42, Loss: 1.3095167875289917
Epoch: 43, Loss: 1.09000381803512573
Epoch: 44, Loss: 0.6847273707389832
Epoch: 45, Loss: 0.9351237416267395
Epoch: 46, Loss: 0.676507830619812
Epoch: 47, Loss: 1.3347176313400269
Epoch: 48, Loss: 1.3014150857925415
Epoch: 49, Loss: 0.9849372506141663
Epoch: 50, Loss: 1.2373034954071045
Epoch: 51, Loss: 1.2821416854858398
Saving best model with best loss: 0.5856910347938538
Epoch: 52, Loss: 0.8869602084159851
Epoch: 53, Loss: 1.125016212463379
Epoch: 54, Loss: 0.834112286567688
Epoch: 55, Loss: 1.3942548036575317
Saving best model with best loss: 0.5309020280838013
Epoch: 56, Loss: 1.1238149404525757
Epoch: 57, Loss: 1.3645730018615723
Epoch: 58, Loss: 0.9325029253959656
Epoch: 59, Loss: 1.1578153371810913
Epoch: 60, Loss: 1.1449464559555054
Epoch: 61, Loss: 1.4775980710983276
Epoch: 62, Loss: 0.8595580458641052
Epoch: 63, Loss: 0.788140058517456
Epoch: 64, Loss: 0.8127356767654419
Epoch: 65, Loss: 1.1243802309036255
Epoch: 66, Loss: 0.6031419634819031
Epoch: 67, Loss: 0.92477947473526
Epoch: 68, Loss: 0.8618582487106323
Epoch: 69, Loss: 0.8227213025093079
Epoch: 70, Loss: 0.9445861577987671
Epoch: 71, Loss: 0.6601676940917969
Epoch: 72, Loss: 1.4964605569839478
Epoch: 73, Loss: 0.8822178244590759
Saving best model with best loss: 0.5244764089584351

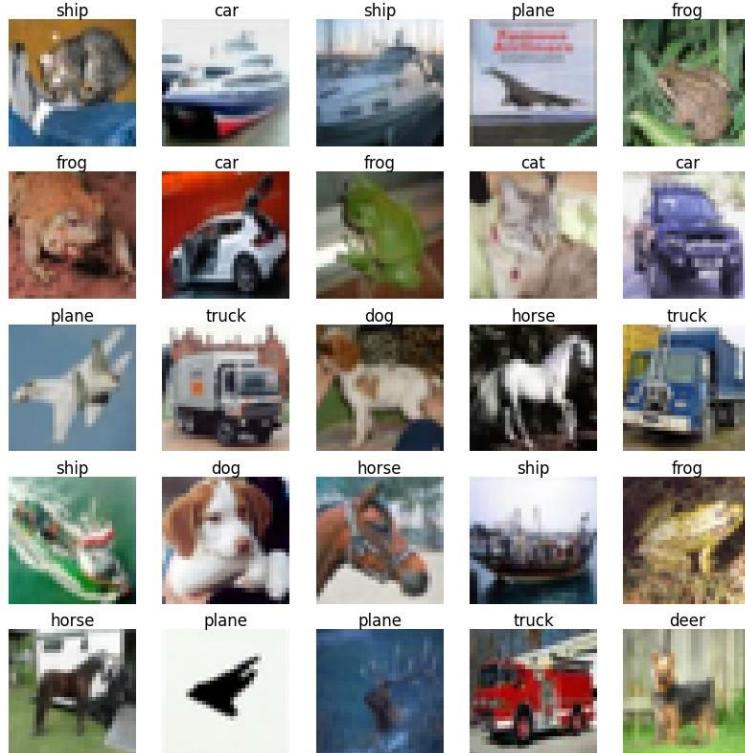
```

```

Epoch: 961, Loss: 0.03977096825838089
Epoch: 962, Loss: 0.050828680396080002
Epoch: 963, Loss: 0.11375655233860016
Epoch: 964, Loss: 0.10192723572254181
Epoch: 965, Loss: 0.1356799155473709
Epoch: 966, Loss: 0.08606917411088943
Epoch: 967, Loss: 0.06464323401451111
Epoch: 968, Loss: 0.04968060180544853
Epoch: 969, Loss: 0.06839265674352646
Epoch: 970, Loss: 0.0451239943504335
Epoch: 971, Loss: 0.069516621530056
Epoch: 972, Loss: 0.1086975634098053
Epoch: 973, Loss: 0.06547818332910538
Epoch: 974, Loss: 0.06943277269601822
Epoch: 975, Loss: 0.09090308099985123
Epoch: 976, Loss: 0.02240993268787861
Epoch: 977, Loss: 0.04689973592758179
Epoch: 978, Loss: 0.02995654568076137
Epoch: 979, Loss: 0.02219798043370247
Epoch: 980, Loss: 0.06665419787168503
Epoch: 981, Loss: 0.08131923526525497
Epoch: 982, Loss: 0.1359425038099289
Epoch: 983, Loss: 0.05393002927303314
Epoch: 984, Loss: 0.0873061791062355
Epoch: 985, Loss: 0.05005749315023422
Epoch: 986, Loss: 0.05626797676086426
Epoch: 987, Loss: 0.08323075622320175
Epoch: 988, Loss: 0.07047758251428604
Epoch: 989, Loss: 0.0889035314321518
Epoch: 990, Loss: 0.05049773305654526
Epoch: 991, Loss: 0.05776713043451309
Epoch: 992, Loss: 0.0641380175948143
Epoch: 993, Loss: 0.0617222934961319
Epoch: 994, Loss: 0.08828379213809967
Epoch: 995, Loss: 0.0645817294716835
Epoch: 996, Loss: 0.08191368728876114
Epoch: 997, Loss: 0.09174858033657074
Epoch: 998, Loss: 0.0552963949739933
Epoch: 999, Loss: 0.06580603122711182

```

Correction: 4.0%
Correction: 8.0%
Correction: 12.0%
Correction: 16.0%
Correction: 16.0%
Correction: 20.0%
Correction: 24.0%
Correction: 28.000000000000004%
Correction: 28.000000000000004%
Correction: 32.0%
Correction: 36.0%
Correction: 40.0%
Correction: 44.0%
Correction: 48.0%
Correction: 52.0%
Correction: 56.00000000000001%
Correction: 60.0%
Correction: 64.0%
Correction: 68.0%
Correction: 72.0%
Correction: 76.0%
Correction: 76.0%
Correction: 80.0%
Correction: 84.0%
Correction: 84.0%



Homework #3, 繳交期限 2024/11/08

Classification & Convolution Neural Network

系所：工海所

姓名：劉樺

學號：R13525030

Advance

- (1) 分別印出捲積神經網路與 sample code 內之全連接神經網路之參數總量。
 - (2) Batch normalization 為提高影像辨識率之重要方法，當模型訓練到最後辨識率無法提升時，該方法為提升辨識率之重要手段。嘗試了解並實作 Batch normalization。並印出其準確率(該實作之準確率需大於 basic score 之準確率)。
- 計算並印出兩個模型的參數總量

```
● ● ●  
1 def count_parameters(model):  
2     """Count the total number of trainable parameters in a model"""  
3     return sum(p.numel() for p in model.parameters() if p.requires_grad)  
4  
5 # Datasets (Cifar10)  
6 data_path='./cifar10/datasets/'  
7 cifar10_train=datasets.CIFAR10(data_path, train=True, download=True, transform=transforms.ToTensor())  
8 cifar10_val=datasets.CIFAR10(data_path, train=False, download=True, transform=transforms.ToTensor())  
9 classes = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck')  
10
```

替換原始的 Convnet 類別為新的帶有 BatchNorm 的版本

```
● ● ●  
1 class ConvNetWithBN(nn.Module):  
2     def __init__(self):  
3         super().__init__()  
4         # First convolutional block  
5         self.conv1 = nn.Conv2d(3, 32, kernel_size=3, padding=1)  
6         self.bn1 = nn.BatchNorm2d(32)  
7         self.act1 = nn.ReLU()  
8         self.pool1 = nn.MaxPool2d(2)  
9  
10        # Second convolutional block  
11        self.conv2 = nn.Conv2d(32, 8, kernel_size=3, padding=1)  
12        self.bn2 = nn.BatchNorm2d(8)  
13        self.act2 = nn.ReLU()  
14        self.pool2 = nn.MaxPool2d(2)  
15  
16        # Fully connected layers  
17        self.linear1 = nn.Linear(8*8*8, 128)  
18        self.bn3 = nn.BatchNorm1d(128)  
19        self.act3 = nn.ReLU()  
20        self.linear2 = nn.Linear(128, 10)  
21  
22    def forward(self, x):  
23        # first block  
24        out = self.conv1(x)  
25        out = self.bn1(out)  
26        out = self.act1(out)  
27        out = self.pool1(out)  
28  
29        # Second block  
30        out = self.conv2(out)  
31        out = self.bn2(out)  
32        out = self.act2(out)  
33        out = self.pool2(out)  
34  
35        # Fully connected layers  
36        out = out.view(-1, 8*8*8)  
37        out = self.linear1(out)  
38        out = self.bn3(out)  
39        out = self.act3(out)  
40        out = self.linear2(out)  
41  
42    return out
```

```
● ● ●  
1 #Training!  
2 training(model=conv_model,  
3           n_epoch=n_epoch,  
4           loss_fn=loss_fn,  
5           optimizer=optimizer,  
6           scheduler=scheduler, # 新增scheduler參數  
7           train_loader=train_loader,  
8           val_loader=val_loader,  
9           best_loss=best_loss)  
10
```

Basic

Correction: 4.0%
Correction: 4.0%
Correction: 4.0%
Correction: 4.0%
Correction: 8.0%
Correction: 12.0%
Correction: 12.0%
Correction: 12.0%
Correction: 16.0%
Correction: 16.0%
Correction: 20.0%
Correction: 24.0%
Correction: 28.00000000000004%
Correction: 32.0%
Correction: 36.0%
Correction: 40.0%
Correction: 44.0%
Correction: 44.0%
Correction: 48.0%
Correction: 52.0%
Correction: 56.000000000001%
Correction: 56.000000000001%
Correction: 60.0%
Correction: 64.0%
Correction: 64.0%

Advanced

Epoch: 962, Train Loss: 0.0002, Val Loss: 0.8209, Accuracy: 72.76%
Epoch: 963, Train Loss: 0.0002, Val Loss: 0.8186, Accuracy: 73.13%
Epoch: 964, Train Loss: 0.0002, Val Loss: 0.8126, Accuracy: 73.11%
Epoch: 965, Train Loss: 0.0002, Val Loss: 0.8171, Accuracy: 73.21%
Epoch: 966, Train Loss: 0.0002, Val Loss: 0.8123, Accuracy: 73.18%
Epoch: 967, Train Loss: 0.0002, Val Loss: 0.8183, Accuracy: 72.98%
Epoch: 968, Train Loss: 0.0002, Val Loss: 0.8170, Accuracy: 73.14%
Epoch: 969, Train Loss: 0.0002, Val Loss: 0.8190, Accuracy: 72.96%
Epoch: 970, Train Loss: 0.0002, Val Loss: 0.8143, Accuracy: 72.95%
Epoch: 971, Train Loss: 0.0002, Val Loss: 0.8152, Accuracy: 72.94%
Epoch: 972, Train Loss: 0.0002, Val Loss: 0.8221, Accuracy: 72.97%
Epoch: 973, Train Loss: 0.0002, Val Loss: 0.8189, Accuracy: 72.88%
Epoch: 974, Train Loss: 0.0002, Val Loss: 0.8168, Accuracy: 73.12%
Epoch: 975, Train Loss: 0.0002, Val Loss: 0.8157, Accuracy: 73.09%
Epoch: 976, Train Loss: 0.0002, Val Loss: 0.8199, Accuracy: 73.03%
Epoch: 977, Train Loss: 0.0002, Val Loss: 0.8201, Accuracy: 73.04%
Epoch: 978, Train Loss: 0.0002, Val Loss: 0.8165, Accuracy: 72.99%
Epoch: 979, Train Loss: 0.0002, Val Loss: 0.8158, Accuracy: 73.01%
Epoch: 980, Train Loss: 0.0002, Val Loss: 0.8163, Accuracy: 72.90%
Epoch: 981, Train Loss: 0.0002, Val Loss: 0.8187, Accuracy: 72.98%
Epoch: 982, Train Loss: 0.0002, Val Loss: 0.8196, Accuracy: 73.10%
Epoch: 983, Train Loss: 0.0002, Val Loss: 0.8203, Accuracy: 72.93%
Epoch: 984, Train Loss: 0.0002, Val Loss: 0.8201, Accuracy: 73.03%
Epoch: 985, Train Loss: 0.0002, Val Loss: 0.8173, Accuracy: 72.96%
Epoch: 986, Train Loss: 0.0002, Val Loss: 0.8202, Accuracy: 73.05%
Epoch: 987, Train Loss: 0.0002, Val Loss: 0.8151, Accuracy: 72.95%
Epoch: 988, Train Loss: 0.0002, Val Loss: 0.8258, Accuracy: 72.70%
Epoch: 989, Train Loss: 0.0002, Val Loss: 0.8159, Accuracy: 73.14%
Epoch: 990, Train Loss: 0.0002, Val Loss: 0.8166, Accuracy: 73.22%
Epoch: 991, Train Loss: 0.0002, Val Loss: 0.8131, Accuracy: 73.22%
Epoch: 992, Train Loss: 0.0002, Val Loss: 0.8166, Accuracy: 73.16%
Epoch: 993, Train Loss: 0.0002, Val Loss: 0.8227, Accuracy: 72.91%
Epoch: 994, Train Loss: 0.0002, Val Loss: 0.8155, Accuracy: 73.18%
Epoch: 995, Train Loss: 0.0002, Val Loss: 0.8127, Accuracy: 73.23%
Epoch: 996, Train Loss: 0.0002, Val Loss: 0.8184, Accuracy: 72.83%
Epoch: 997, Train Loss: 0.0002, Val Loss: 0.8185, Accuracy: 73.27%
Epoch: 998, Train Loss: 0.0002, Val Loss: 0.8174, Accuracy: 72.98%
Epoch: 999, Train Loss: 0.0002, Val Loss: 0.8229, Accuracy: 73.09%

