



به نام خدا



دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

یادگیری عمیق با کاربردها

تمرین شماره دو

امید نائیج نژاد	نام و نام خانوادگی
610301189	شماره دانشجویی
1404/1/10	تاریخ ارسال گزارش

فهرست مطالب

5	طبقه بندی تصاویر با استفاده از شبکه های عمیق پیچشی (CNN)
20.....	تشخیص چهره به کمک ساختار شبکه های سیامی
29.....	مراجع

فهرست شکل ها

5.....	شکل 1- تابع آماده سازی دیتاست cifar-10
6.....	شکل 2 - تصویر تصادفی بعد از اعمال افزونه ها و انجام پیش پردازش
7.....	شکل 3 - توزیع کلاس ها در داده اعتبار سنجی
7.....	شکل 4 - نتایج اعمال افزونه ها
8.....	شکل 5 - پیاده سازی BaseCNN
9.....	شکل 6 - بلوک A
9.....	شکل 7 - نمودار دقت و هزینه برای داده های آموزش و اعتبار سنجی بلوک A
10.....	شکل 8 - فضای ویژگی های بدست آمده با بلوک A
10.....	شکل 9 - بلوک B
11.....	شکل 10 - نمودار دقت و هزینه برای داده های آموزش و اعتبار سنجی بلوک B
11.....	شکل 11 - فضای ویژگی های بدست آمده با بلوک B
12.....	شکل 12 - ماتریس درهم ریختکی بهترین مدل
12.....	شکل 13 - بلوک C
13.....	شکل 14 - نمودار دقت و هزینه برای داده های آموزش و اعتبار سنجی بلوک C
13.....	شکل 15 - فضای ویژگی های بدست آمده با بلوک C
14.....	شکل 16 - بلوک D
14.....	شکل 17 - نمودار دقت و هزینه برای داده های آموزش و اعتبار سنجی بلوک D
15.....	شکل 18 - فضای ویژگی های بدست آمده با بلوک D
15.....	شکل 19 - نمودار دقت و هزینه شبکه برای داده های آموزش و اعتبار سنجی بلوک D (Depth Wise)
16.....	شکل 20 - فضای ویژگی های بدست آمده با بلوک D (Depth Wise)
17.....	شکل 21 - بلوک E
17.....	شکل 22 - نمودار دقت و هزینه شبکه برای داده های آموزش و اعتبار سنجی بلوک E
18.....	شکل 23 - فضای ویژگی های بدست آمده با بلوک E
18.....	شکل 24 - خروجی لایه های کانولوشنی
20.....	شکل 25 - 2 نمونه تصادفی از دیتاست تصاویر چهره
21.....	شکل 26 - کلاس دیتاست چهره
22.....	شکل 27 - ساختار شبکه سیامی توسعه داده شده
23.....	شکل 28 - پیاده سازی تابع هزینه مقایسه ای
24.....	شکل 29 - پیاده سازی جستجو تصادفی
25.....	شکل 30 - نمودار اتلاف برای داده های آموزش و اعتبار سنجی بهترین مدل

25.....	شکل 31 - نمودار جعبه ای فاصله بردارهای ویژگی تصاویر تفکیک شده براساس برچسب
26.....	شکل 32 - هیستوگرام فاصله بردارهای ویژگی تصاویر تفکیک شده براساس برچسب
26.....	شکل 33 - بررسی 10 همسایه نزدیک به 5 تصویر رندوم دیتابست
27.....	شکل 34 - نمودار اتلاف برای داده های آموزش و اعتبار سنجی با افزایش حاشیه به دو
28.....	شکل 35 - پیاده سازی تابع اتلاف سه گانه

فهرست جدول ها

طبقه بندی تصاویر با استفاده از شبکه های عمیق پیچشی (CNN)

آماده سازی داده

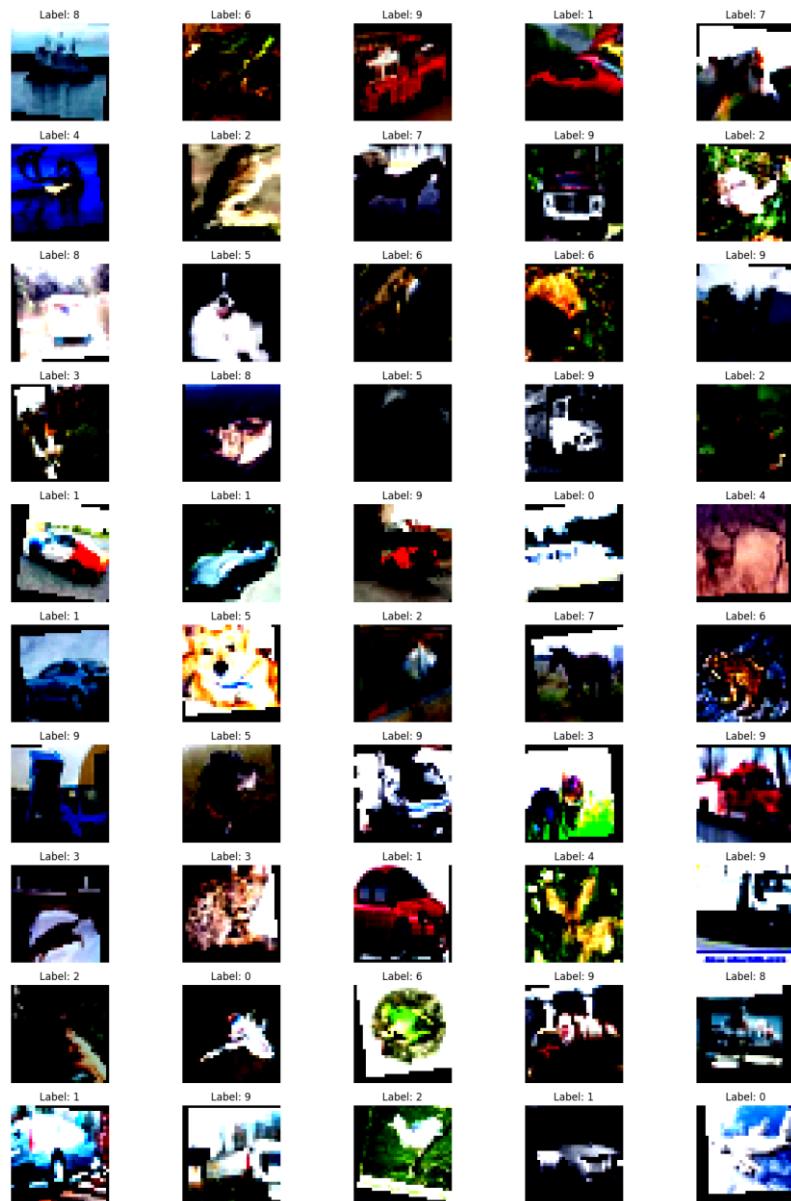
مطابق با ساختار گفته شده برای انجام تمرین ها، تمامی مراحل دانلود و آماده سازی دیتابست در فایل data_loader.py می شود. برای دانلود کردن دیتابست CIFAR-10 از <https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz> استفاده میکنیم. سپس آنها را extract میکنیم. این مراحل در تابع download_and_extract_cifar10 انجام میشود.

در ادامه افزونه ها و پیش پردازش های گفته شده در جدول ۱-۱ و ۱-۲ فایل توضیح پردازه روی تصاویر دانلود شده اعمال میشود. ۵۰ تصویر رندوم نمایش داده های موجود در مسیر train فایل دانلود شده، از هر کلاس ۱۰۰۰ نمونه را برای اعتبارسنجی جدا میکنیم و توزیع آنها را نمایش میدهیم تا از صحت کارهای انجام شده مطمئن شویم.

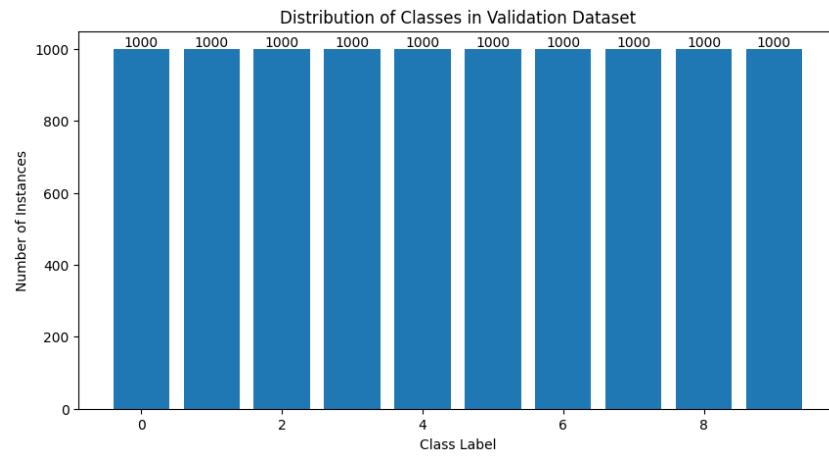


```
 1 def load_cifar10(data_dir='./data/cifar10', batch_size=64):
 2     transform = transforms.Compose([
 3         transforms.ColorJitter(brightness=0.25, contrast=0.25, saturation=0.25),
 4         transforms.RandomAffine(degrees=10, translate=(0.1, 0.1), scale=(0.9, 1.1)),
 5         transforms.RandomHorizontalFlip(p = 0.5),
 6         transforms.ToTensor(),
 7         transforms.Normalize(
 8             mean=[0.49139968, 0.48215827, 0.44653124],
 9             std=[0.24703233, 0.24348505, 0.26158768]
10         )
11     ])
12
13     download_and_extract_cifar10(root_dir=data_dir)
14
15     train_dataset = CIFAR10Dataset(root_dir=data_dir, train=True, transform=transform)
16     test_dataset = CIFAR10Dataset(root_dir=data_dir, train=False, transform=transform)
17
18     plot_random_images(train_dataset)
19
20     # Create a dictionary to store indices of each class
21     class_indices = defaultdict(list)
22     for i, (image, label) in enumerate(train_dataset):
23         class_indices[label].append(i)
24
25     # Create a list to store indices for validation set
26     val_indices = []
27     for label in range(10):
28         val_indices.extend(class_indices[label][:1000]) # Select 1000 instances from each class
29
30     # Create validation and training datasets using Subset
31     val_dataset = Subset(train_dataset, val_indices)
32     train_indices = [i for i in range(len(train_dataset)) if i not in val_indices]
33     train_dataset = Subset(train_dataset, train_indices)
34
35     print(f"Train dataset length: {len(train_dataset)}")
36     print(f"Validation dataset length: {len(val_dataset)}")
37     print(f"Test dataset length: {len(test_dataset)}")
38
39     train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
40     val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)
41     test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
42
43     valdata_target_distribution(val_dataset)
44
45     plot_augmentation_results(train_dataset)
46
47     return train_loader, val_loader, test_loader
```

شکل ۱- تابع آماده سازی دیتابست cifar-10

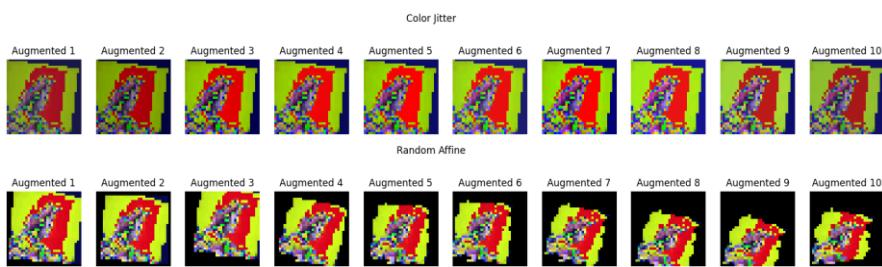


شکل 2 - 50 تصویر تصادفی بعد از اعمال افزونه ها و انجام پیش پردازش



شکل 3 - توزیع کلاس ها در داده اعتبارسنجی

همچنین یک تصویر رندوم از دیتاست را انتخاب کرده و 10 بار هر یک از افزونه های Jitter Color و Affine Random را روی آن اعمال کردم و نتایج در شکل زیر قابل مشاهده است:



شکل 4 - نتایج اعمال افزونه ها

پیاده سازی مدل پایه

ساختار اولیه مدل شامل لایه های ابتدایی و لایه انتها یی که فیکس هستند در کلاس BaseCNN مشخص شده است همچنین تعداد و اندازه کانال ورودی و خروجی بلاک های A تا E تعیین شده است تا در زمان تعریف کردن هر کدام از مدل ها صرفا نوع بلاک ها داده شود .

```

1  class BaseCNN(nn.Module):
2      def __init__(self, block_config, use_separable_conv=False):
3          super(BaseCNN, self).__init__()
4
5          block_map = {'A': BlockA, 'B': BlockB, 'C': BlockC, 'D': BlockD, 'E': BlockE}
6
7          # Initial layers based on the table
8          self.initial_layers = nn.Sequential(
9              nn.Conv2d(3, 32, kernel_size=3, padding=1, bias=False),
10             nn.BatchNorm2d(32),
11             nn.ReLU(),
12             nn.Conv2d(32, 64, kernel_size=3, padding=1, bias=False),
13             nn.BatchNorm2d(64),
14             nn.ReLU(),
15             nn.Conv2d(64, 128, kernel_size=3, padding=1, bias=False),
16             nn.BatchNorm2d(128),
17             nn.ReLU()
18         )
19
20         # Channel sizes between layers (according to table)
21         channels = [128, 128, 128, 256, 256, 256, 512, 512, 512]
22
23         # Create the 8 configurable blocks
24         self.blocks = nn.ModuleList()
25         for i, block in enumerate(block_config):
26             in_channels = channels[i]
27             out_channels = channels[i + 1] if i < len(channels)-1 else channels[i]
28
29             if block == 'D':
30                 self.blocks.append(block_map[block](in_channels, out_channels, use_separable_conv))
31             else:
32                 self.blocks.append(block_map[block](in_channels, out_channels))
33
34         # Add MaxPool at specific locations based on table
35         if i in [2, 4, 7]:
36             self.blocks.append(nn.MaxPool2d(kernel_size=2, stride=2))
37
38         # Final layers
39         self.global_avg_pool = nn.AdaptiveAvgPool2d((1, 1))
40         self.dropout = nn.Dropout(p=0.25)
41         self.fc = nn.Linear(512, 10)
42
43     def forward(self, x, test_mode=False):
44         x = self.initial_layers(x)
45
46         for block in self.blocks:
47             x = block(x) # Forward pass through each block
48
49         x = self.global_avg_pool(x)
50         x = torch.flatten(x, 1)
51         x = self.dropout(x)
52         x = self.fc(x)
53
54         return x

```

شکل 5 - پیاده سازی BaseCNN

آموزش مدل

فرایند آموزش مدل از طریق تابع `train_model` در فایل `scripts/train.py` در مورد نظر ذخیره میکند. همچنین مقدار دقیق و تابع اتلاف برای داده آموزش و اعتبارسنجی را حین فرایند یادگیری محاسبه و گزارش میکند و در نهایت با فراخوانی تابع `plot_loss_and_acc` نمودار دقیق و هزینه شبکه برای داده های آموزش و اعتبارسنجی در حین آموزش را رسم میکند.

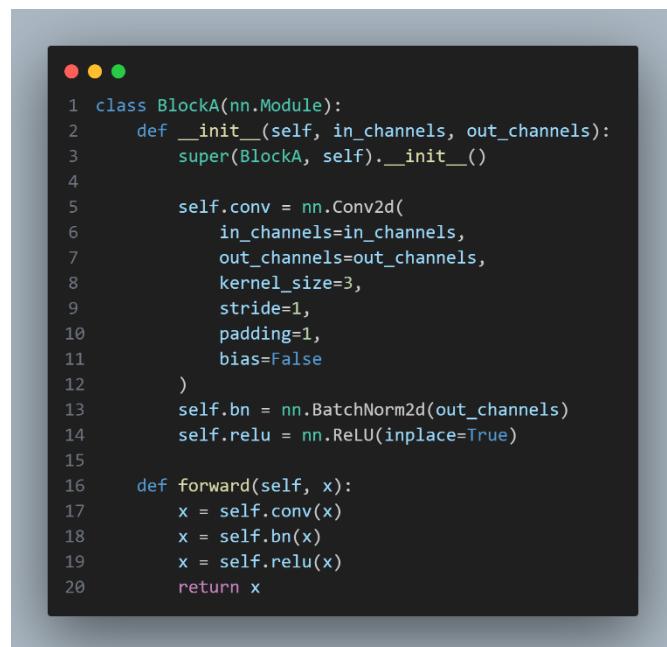
نحوه عملکرد زمان بند :ExponentialLR

همانطور که میدانیم اگر نرخ یادگیری در ابتدا به اندازه کافی زیاد باشد و به مرور زمان کاهش یابد، موجب افزایش سرعت همگرایی در ابتدای آموزش و همچنین کاهش ریسک واگرایی که بینه سازی خواهد شد. یکی از روش های پیاده سازی این ایده، استفاده از زمان بند ExponentialLR است که به صورت نمایی نرخ یادگیری را کاهش میدهد. مقدار نرخ یادگیری در ایپاک t طبق رابطه زیر بدست می آید:

$$\text{رابطه 1} \quad lr_t = lr_0 \times \gamma^t$$

در رابطه بالا γ یک مقدار بین صفر و یک است که سرعت کاهش نرخ یادگیری را تعیین میکند. lr_0 مقدار اولیه نرخ یادگیری است.

بلوک A



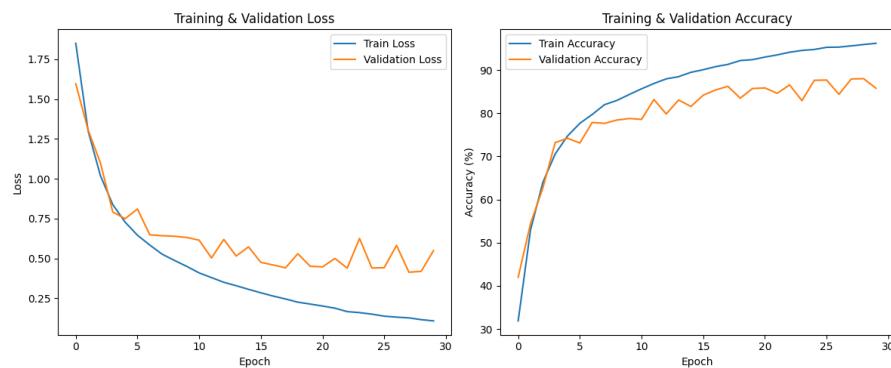
```

1 class BlockA(nn.Module):
2     def __init__(self, in_channels, out_channels):
3         super(BlockA, self).__init__()
4
5         self.conv = nn.Conv2d(
6             in_channels=in_channels,
7             out_channels=out_channels,
8             kernel_size=3,
9             stride=1,
10            padding=1,
11            bias=False
12        )
13        self.bn = nn.BatchNorm2d(out_channels)
14        self.relu = nn.ReLU(inplace=True)
15
16    def forward(self, x):
17        x = self.conv(x)
18        x = self.bn(x)
19        x = self.relu(x)
20
21        return x

```

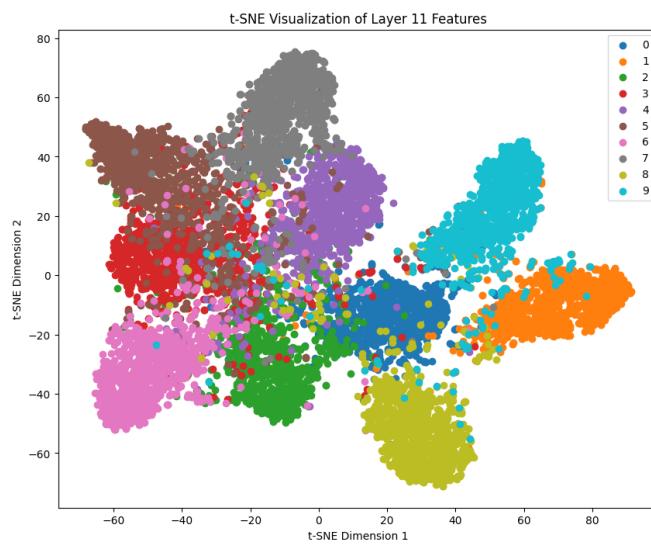
شکل 6 - بلوک A

- تعداد پارامترهای قابل آموزش مدل = 7771658
- مدت زمان آموزش مدل = 2485.55 seconds
- نمودار دقت و هزینه شبکه برای داده های آموزش و اعتبارسنجی در حین آموزش:



شکل 7 - نمودار دقت و هزینه برای داده های آموزش و اعتبارسنجی بلوک A

- فضای ویژگی های بدست آمده برای داده های تست با استفاده از t-SNE



شکل 8 - فضای ویژگی های بدست آمده با بلوک A

بلوک B

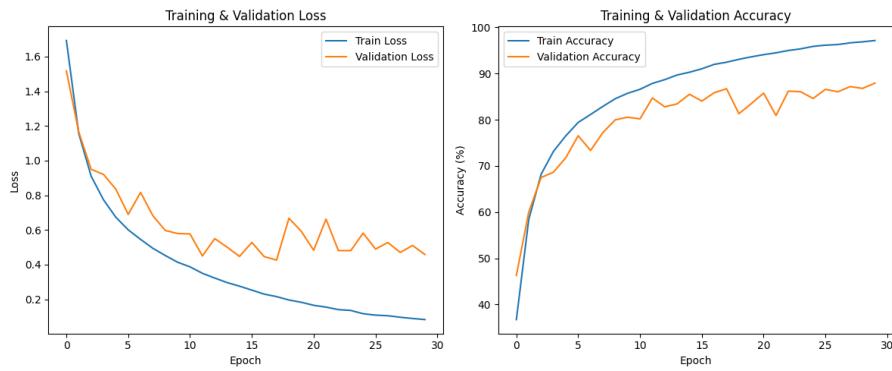
```

● ● ●
1  class BlockB(nn.Module):
2      def __init__(self, in_channels, out_channels):
3          super(BlockB, self).__init__()
4
5          self.block_a = BlockA(in_channels, out_channels)
6          self.block = nn.Sequential(
7              nn.Conv2d(out_channels, 1, kernel_size=1, bias=False),
8              nn.Sigmoid()
9          )
10
11     def forward(self, x):
12         x = self.block_a(x)
13         return self.block(x) * x
14

```

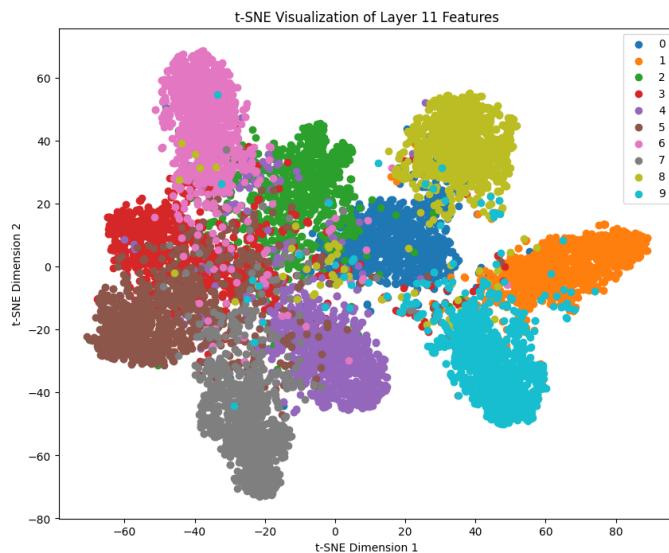
شکل 9 – بلوک B

- تعداد پارامترهای قابل آموزش مدل = 7774226
- مدت زمان آموزش مدل = 2579.59 seconds
- نمودار دقت و هزینه شبکه برای داده های آموزش و اعتبارسنجی در حین آموزش:



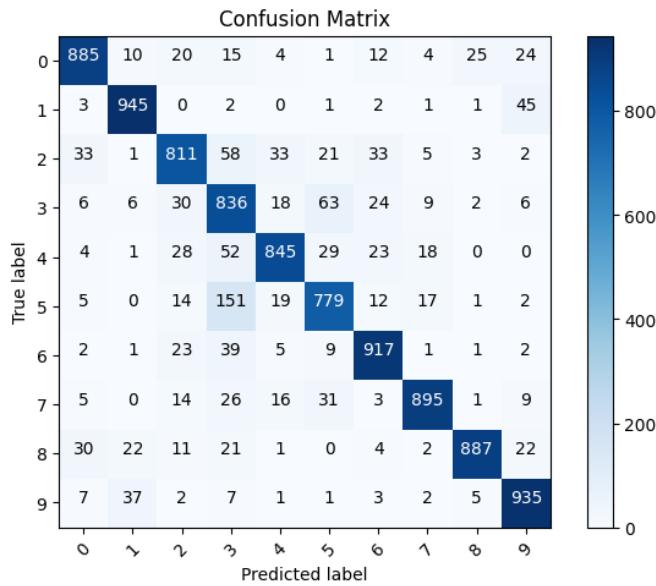
شکل 10 - نمودار دقت و هزینه برای داده های آموزش و اعتبارسنجی بلوک B

- فضای ویژگی های بدست آمده برای داده های تست با استفاده از t-SNE :



شکل 11 - فضای ویژگی های بدست آمده با بلوک B

- ماتریس درهم ریختگی بهترین مدل (با توجه به دقت برای داده های اعتبارسنجی که در این مدل بیشتر از بقیه است، این ساختار بهترین مدل میباشد):



شکل 12 - ماتریس درهم ریختگی بهترین مدل

بلوک C

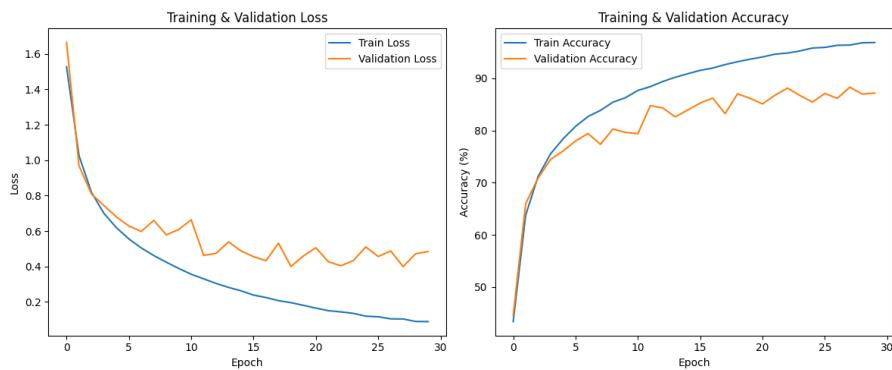
```

1  class BlockC(nn.Module):
2      def __init__(self, in_channels, out_channels, reduction=16):
3          super(BlockC, self).__init__()
4          self.block_a = BlockA(in_channels, out_channels) # Ensure channels match
5          self.global_pool = nn.AdaptiveAvgPool2d(1)
6          self.fc1 = nn.Linear(out_channels, out_channels // reduction)
7          self.relu = nn.ReLU(inplace=True)
8          self.fc2 = nn.Linear(out_channels // reduction, out_channels)
9          self.sigmoid = nn.Sigmoid()
10
11     def forward(self, x):
12         x = self.block_a(x) # Apply BlockA first
13         batch_size, channels, _, _ = x.size()
14         y = self.global_pool(x).view(batch_size, channels) # Global pooling
15         y = self.fc1(y)
16         y = self.relu(y)
17         y = self.fc2(y)
18         y = self.sigmoid(y).view(batch_size, channels, 1, 1)
19         return x * y # Channel-wise recalibration
20

```

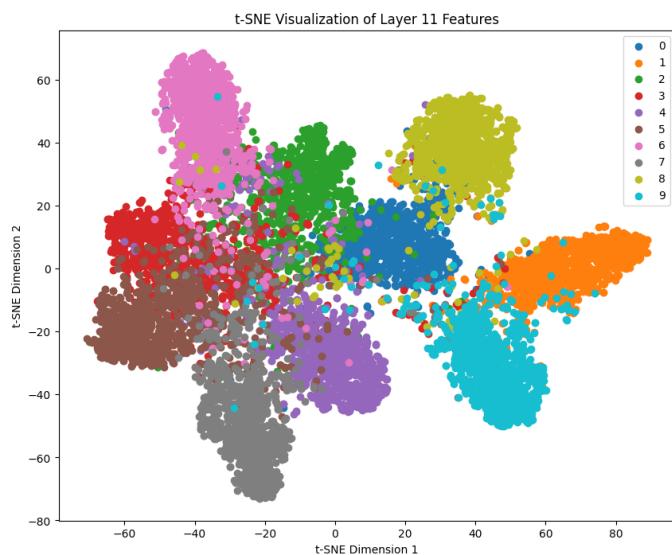
شکل 13 – بلوک C

- تعداد پارامترهای قابل آموزش مدل = 7901354
- مدت زمان آموزش مدل = 2616.70 seconds
- نمودار دقیق و هزینه شبکه برای داده های آموزش و اعتبارسنجی در حین آموزش:



شکل 14 - نمودار دقت و هزینه برای داده های آموزش و اعتبارسنجی بلوک C

فضای ویژگی های بدست آمده برای داده های تست با استفاده از :t-SNE •



شکل 15 - فضای ویژگی های بدست آمده با بلوک C

بلوک D

```

1  class DepthwiseSeparableConv(nn.Module):
2      """Implements Depthwise Separable Convolution."""
3      def __init__(self, in_channels, out_channels, kernel_size=3, stride=1, padding=1):
4          super(DepthwiseSeparableConv, self).__init__()
5          self.depthwise = nn.Conv2d(in_channels, in_channels, kernel_size=kernel_size,
6                                  stride=stride, padding=padding, groups=in_channels, bias=False)
7          self.pointwise = nn.Conv2d(in_channels, out_channels, kernel_size=1, bias=False)
8
9      def forward(self, x):
10         x = self.depthwise(x)
11         x = self.pointwise(x)
12         return x
13
14 class Block0(nn.Module):
15     def __init__(self, in_channels, out_channels, use_separable_conv=False):
16         super(Block0, self).__init__()
17         self.use_shortcut = in_channels == out_channels
18
19         if use_separable_conv:
20             self.conv = DepthwiseSeparableConv(in_channels, out_channels, kernel_size=3, padding=1)
21         else:
22             self.conv = nn.Conv2d(in_channels, out_channels, kernel_size=3, padding=1, bias=False)
23
24         self.batch_norm = nn.BatchNorm2d(out_channels)
25         self.relu = nn.ReLU(inplace=True)
26
27         # 1x1 Conv for channel matching when needed
28         self.shortcut = nn.Conv2d(in_channels, out_channels, kernel_size=1, bias=False) if not self.use_shortcut else nn.Identity()
29
30     def forward(self, x):
31         residual = x
32         x = self.conv(x)
33         x = self.batch_norm(x)
34
35         if not self.use_shortcut:
36             residual = self.shortcut(residual)
37
38         x += residual # Skip connection
39         x = self.relu(x)
40
41         return x

```

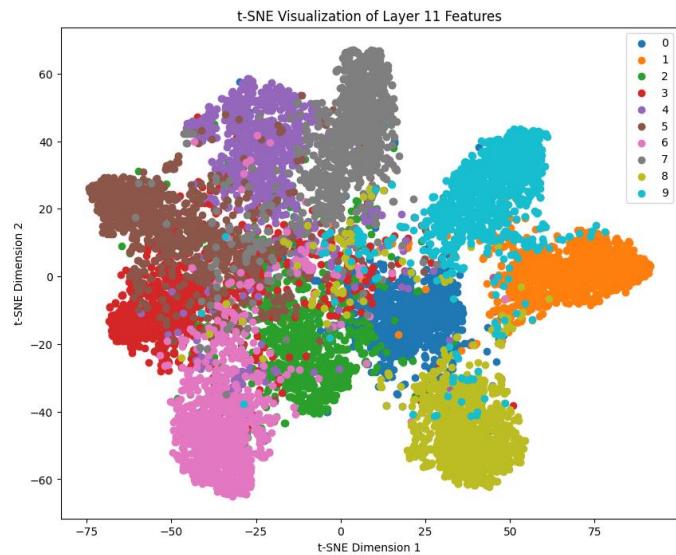
شکل 16 - بلوک D

- تعداد پارامترهای قابل آموزش مدل = 7935498
- مدت زمان آموزش مدل = 2661.10 ثانیه
- نمودار دقیق و هزینه شبکه برای داده های آموزش و اعتبارسنجی در حین آموزش:



شکل 17 - نمودار دقیق و هزینه برای داده های آموزش و اعتبارسنجی بلوک D

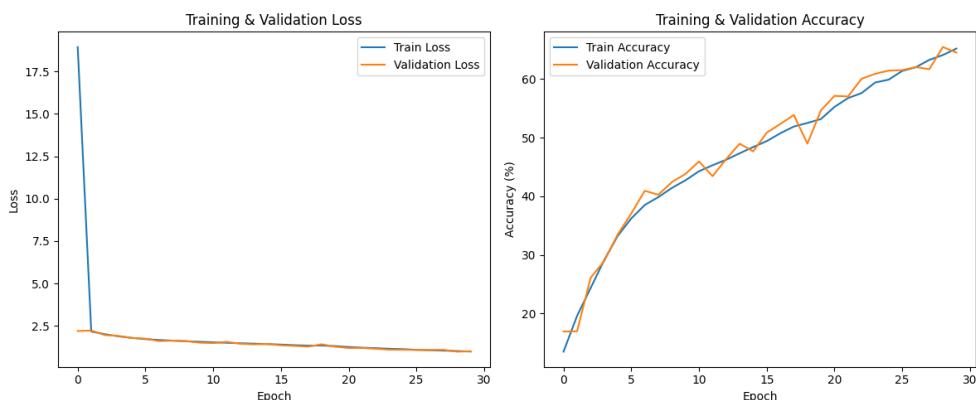
- فضای ویژگی های بدست آمده برای داده های تست با استفاده از :t-SNE



شکل 18- فضای ویژگی های بدست آمده با بلوک D

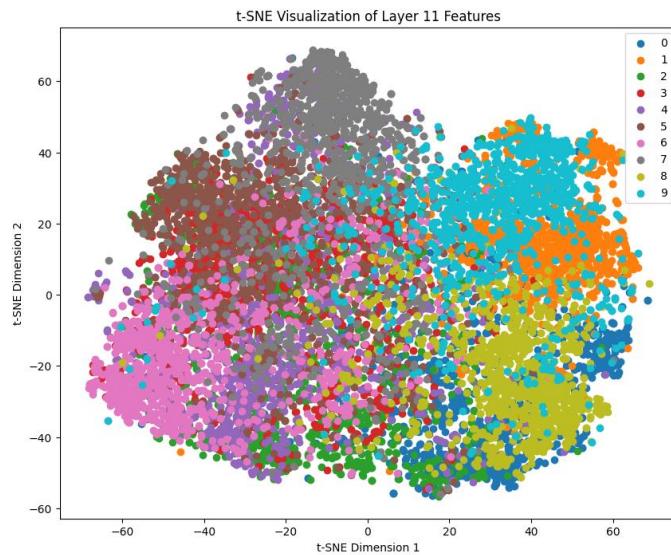
حالا کانولوشن 3×3 در ساختار بلوک D را به صورت Depth wise separable پیاده سازی میکنیم و شبکه را مجددآموزش میدهیم.

- تعداد پارامترهای قابل آموزش مدل = 1139338
- مدت زمان آموزش مدل = 2131.98 ثانیه
- نمودار دقت و هزینه شبکه برای داده های آموزش و اعتبارسنجی در حین آموزش:



شکل 19 - نمودار دقت و هزینه شبکه برای داده های آموزش و اعتبارسنجی بلوک D(Depth Wise)

- فضای ویژگی های بدست آمده برای داده های تست با استفاده از t-SNE:



شکل 20 - فضای ویژگی های بدست آمده با بلوک D(DepthWise)

مطابق انتظار تعداد پارامترهای شبکه نسبت به حالت قبل با کاهش قابل توجهی روبه رو شد (تقریباً $\frac{1}{7}$ بلوک D) و به طبع زمان آموزش شبکه هم کاهش یافت اما به دلیل کم شدن پیچیدگی، مدل به خوبی حالت قبل نتوانست با یادگیری ویژگی های کلاس های مختلف آنها را از هم تفکیک و تشخیص دهد. همانطور که در شکل 19 میبینید مدل به اینک های بیشتری برای یادگیری بهتر نیاز دارد. طبق فضای ویژگی بدست آمده از الگوریتم t-SNE که در شکل های 18 و 20 قابل مشاهده است، در حالت دوم (بیاده سازی بلوک D به صورت Depth wise separable) نسبت به حالت اول (کانولوشن 3×3) داده های کلاس ها در هم آمیخته هستند و شبکه عصبی نتوانسته به خوبی آنها را از هم تفکیک دهد و به همین دلیل عمکلرد مدل برای طبقه بندی داده های اعتبار سنجی با افت قابل توجه همراه شده است.

بلوک E

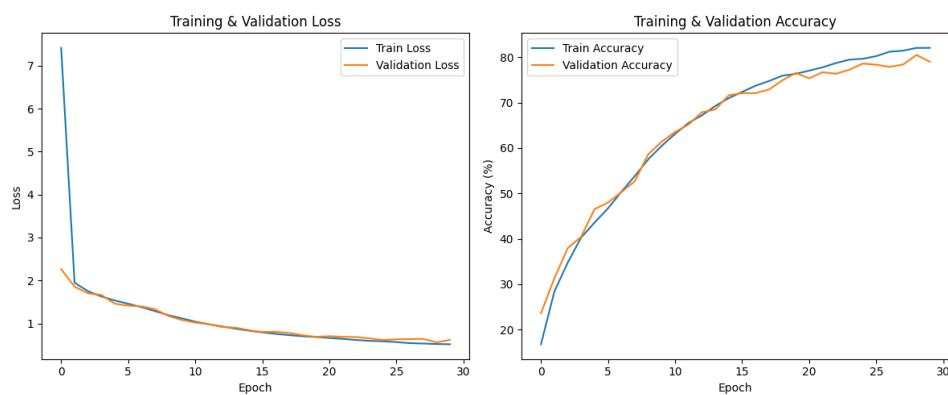
```

1  class BlockE(nn.Module):
2      def __init__(self, in_channels, out_channels, b=4, g=8):
3          super(BlockE, self).__init__()
4
5          b_channels = out_channels // b # Reduce dimensions for grouped conv
6
7          # First 1x1 convolution
8          self.conv1x1_1 = nn.Conv2d(in_channels, b_channels, kernel_size=1, bias=False)
9          self.bn1 = nn.BatchNorm2d(b_channels)
10         self.relu = nn.ReLU(inplace=True)
11
12         # Grouped 3x3 convolution
13         self.conv3x3 = nn.Conv2d(b_channels, b_channels, kernel_size=3, padding=1, groups=in_channels // g, bias=False)
14         self.bn2 = nn.BatchNorm2d(b_channels)
15
16         # Second 1x1 convolution
17         self.conv1x1_2 = nn.Conv2d(b_channels, out_channels, kernel_size=1, bias=False)
18         self.bn3 = nn.BatchNorm2d(out_channels)
19
20         # Branch 2: Projection if input channels ≠ output channels
21         self.projection = None
22         if in_channels != out_channels:
23             self.projection = nn.Conv2d(in_channels, out_channels, kernel_size=1, bias=False)
24
25     def forward(self, x):
26         identity = x # Keep original input for skip connection
27
28         # Main Path
29         out = self.conv1x1_1(x)
30         out = self.bn1(out)
31         out = self.relu(out)
32
33         out = self.conv3x3(out)
34         out = self.bn2(out)
35         out = self.relu(out)
36
37         out = self.conv1x1_2(out)
38         out = self.bn3(out)
39
40         # Branch 2: Apply projection if in_channels ≠ out_channels
41         if self.projection is not None:
42             identity = self.projection(identity)
43
44         # Branch 3: Direct skip connection if dimensions match
45         out += identity # Skip connection
46         out = self.relu(out)
47
48     return out
49

```

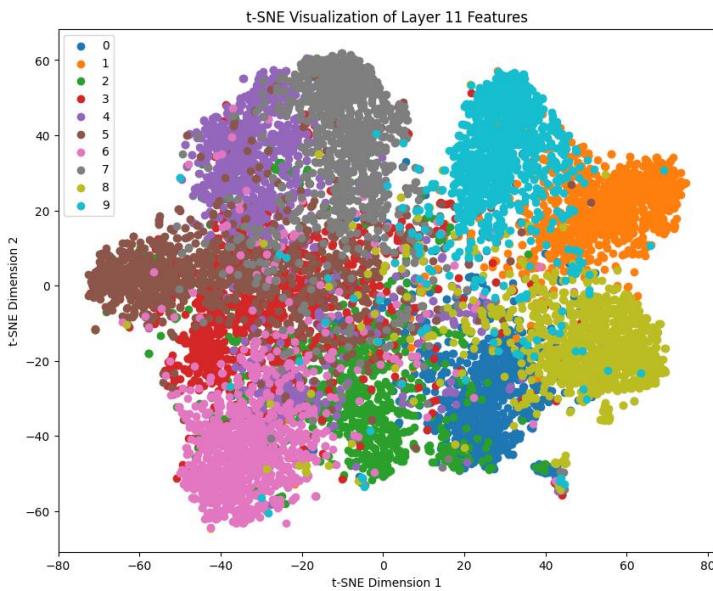
شکل 21 - بلوک E

- تعداد پارامترهای قابل آموزش مدل = 752266
- مدت زمان آموزش مدل = 2229.84 ثانیه
- نمودار دقت و هزینه شبکه برای داده های آموزش و اعتبارسنجی در حین آموزش:



شکل 22 - نمودار دقت و هزینه شبکه برای داده های آموزش و اعتبارسنجی بلوک E

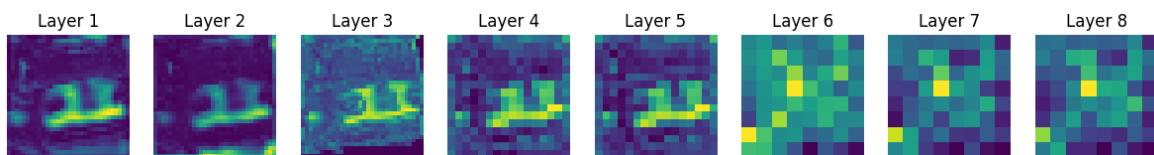
- فضای ویژگی های بدست آمده برای داده های تست با استفاده از t-SNE



شکل 23 - فضای ویژگی های بدست آمده با بلوک E

تصویرسازی خروجی لایه های شبکه

به این منظور از مدل با بلوک E که در بخش قبل آموزش داده شد استفاده کردم و با تابع `visualize_feature_maps` که در `visualization.py` قرار دارد به ازای یک تصویر دلخواه، خروجی لایه های مدل را به تصویر کشیدم که بیانگر توجه ویژه شبکه های کانولوشنی به لبه هاست. اما در لایه های اخر تصویر معناداری قابل تشخیص نیست چون در آن قسمت از مدل، تصویر داده شده به یک فضای دیگر که قابل فهم برای انسان نیست نگاشت شده است.



شکل 24 - خروجی لایه های کانولوشنی

تحلیل و نتیجه گیری

برای انتخاب بهترین مدل معیار های زیر مهم هستند:

- .I. مقدار دقت برای داده اعتبارسنجی (و تابع هزینه)
- .II. زمان یادگیری
- .III. تعداد پارامتر

در نتیجه نسخه بلوک D با Depth wise separable و بلوک E به دلیل کم بودن دقت (به ترتیب %64.53 و %78.98) قابل قبول نیستند. بلوک C نسبت به بلوک B دارای تعداد پارامتر بیشتر است اما دقت آن تقریباً یکسان است. بین بلوک A با

85.77٪ دقت و 7771658 پارامتر، بلوک B با 87.93٪ دقت و 7774226 پارامتر و بلوک D با 86.21٪ دقت و 7935498 پارامتر واضحاً بلوک B از همه بهتر است چرا که دقت بالاتر دارد و در زمان معقول یادگیری انجام شده است.

مدل در تفکیک کدام کلاس ها از هم با چالش بیشتری مواجه بوده است؟

با توجه به شکل 11 و 12 متوجه میشویم که مدل بیشترین مشکل را در تفکیک کردن داده های کلاس 3 و 5 داشته است؛ طبق شکل 12 که نشان دهنده ماتریس درهم ریختنگی است، مدل ما 151 داده آزمایش از کلاس 5 را به اشتباه کلاس 3 پیش بینی کرده و 63 داده آزمایش از کلاس 3 را به اشتباه کلاس 5 پیش بینی کرده است. البته این چالش، مشکل چندان حادی نیست چون کلاس 3 گربه و کلاس 5 سگ است و جدا کردن بعضی نزاده های این دو گونه برای انسان هم کار راحتی نیست!

آیا نحوه خوش بندی داده های کلاس های مختلف در این نمایش فضای ویژگی بدست آمده و فواصل آنها از یکدیگر معنادار است؟

بله، به عنوان مثال نزدیکی خوش بندی کلاس 1 (خودرو) با خوش بندی کلاس 9 (کامیون) که با رنگ های نارنجی و فیروزه ای در شکل 11 قابل مشاهده است، با توجه به نزدیک بودن فرم توبولوژیک و ساختاری این دو دسته و همچنین زمینه کاربرد آنها معنادار است. این اتفاق برای کلاس های 4 و 5 و 6 (آهو سگ قورباغه) حتی باشد بیشتری رخ داده. فاصله بیشتر خوش بندی کلاس 8 (کشتی) از سایرین هم طبق تفاوت بسیار زیاد این دسته با 9 کلاس دیگر قابل درک است.

آیا براساس این نمایش میتوان چالش های مدل در طبقه بندی را توجیه کرد؟

بله، همانطور که بالاتر توضیح داده شد، بیشترین چالش مدل در جدا کردن داده های دسته های مربوط به حیوانات از هم هست که میتوان با اضافه کردن ترم های جدید به تابع هزینه، اشتباه مدل در دسته بندی کردن این تصاویر را با جریمه بیشتری در نظر گرفت و حین فرایند بهینه سازی و کمینه کردن تابع هزینه به صورت خودکار شبکه توجه بیشتری رو این دسته از داده ها خواهد کرد چون بدون کم کردن خطای در طبقه بندی آنها، کمینه کردن تابع هزینه ممکن نخواهد بود.

تشخیص چهره به کمک ساختار شبکه های سیامی

آماده سازی داده

مطابق با ساختار گفته شده برای انجام تمرین ها، تمامی مراحل دانلود و آماده سازی دیتاست در فایل data_loader.py انجام می شود که برای این سوال تابع load_kaggle_dataset کارهای لازم را انجام میدهد. این تابع ابتدا دیتاست را از سایت Kaggle دانلود میکند و با کمک کلاس دیتاست توسعه داده شده برای انجام این ترسک خاص و برای داده های آموزش، اعتبارسنجی و آزمایش دیتالودر ساخته و انها را برمیگرداند.

در ادامه 2 نمونه تصادفی از دیتاست که با تابع visualization.py در فایل show_random_faces بدهست آمده را مشاهده میکنید:



شکل 25 - 2 نمونه تصادفی از دیتاست تصاویر چهره

برای آماده سازی ذیتاست دانلود شده برای استفاده در ساختار های مبتنی بر شبکه های سیامی که بر مبنای یادگیری شباهت هستند، کلاس FaceRecognitionDataset پیاده سازی شده است که این کلاس، تصاویر را به صورت دو تایی و با سیاست زیر سازمان دهی میکند:

- در صورتی که دو تصویر مربوط به یک نفر هستند، برچسب 1 و در غیر این صورت برچسب 0 قرار میدهد. اینکه تصاویر مربوط به یک نفر هستند یا خیر نیز به صورت تصادفی و با احتمال 50% تعیین میشود.

برای پیاده سازی ساختار خواسته شده، روی همه تصاویر پیمایش میکنیم و در هر مرحله با تولید یک عدد اعشاری رندوم بین صفر و یک مشخص میشود که باید دو تصویر یک نفر برگردانده شود یا خیر (با مقایسه عدد تولید شده با عدد 0.5 مطابق شکل زیر). همچنین مجموعه آموزش و اعتبارسنجی را مطابق با شرایط گفته شده در فایل تمرین تولید میکند.

```

1  class FaceRecognitionDataset(Dataset):
2      def __init__(self, root_dir, transform=None, split='train', val_split=0.001, seed=42):
3          """
4              Args:
5                  root_dir (str): Directory with subdirectories for each person's images.
6                  transform (callable, optional): Transformations to apply to the images.
7                  split (str): One of ['train', 'val', 'test'].
8                  val_split (float): Fraction of training data to use for validation.
9                  seed (int): Random seed for reproducibility.
10             """
11         self.root_dir = root_dir
12         self.transform = transform if transform else transforms.ToTensor()
13         self.split = split
14         random.seed(seed)
15
16         # Get list of people (subdirectories)
17         self.people = os.listdir(root_dir)
18
19         # Create a dictionary mapping person -> images
20         self.data = {}
21         for person in self.people:
22             person_path = os.path.join(root_dir, person)
23             if os.path.isdir(person_path):
24                 images = [os.path.join(person_path, img) for img in os.listdir(person_path)]
25                 if len(images) >= 2: # Ensure at least two images per person
26                     self.data[person] = images
27
28         # Split validation data from training
29         if split in ['train', 'val']:
30             self._split_validation(val_split, seed)
31
32         # Create list of pairs for sampling
33         self.pairs = self._generate_pairs()
34
35     def _split_validation(self, val_split, seed):
36         """Splits training data into train and validation sets, ensuring each person has at least two images per split."""
37         random.seed(seed)
38         self.train_data = {}
39         self.val_data = {}
40
41         for person, images in self.data.items():
42             random.shuffle(images)
43             split_idx = max(2, int(len(images) * (1 - val_split))) # Ensure at least two images per set
44             self.train_data[person] = images[:split_idx]
45             self.val_data[person] = images[split_idx:] if len(images[split_idx:]) >= 2 else images[:2]
46
47         self.data = self.train_data if self.split == 'train' else self.val_data
48         self.people = list(self.data.keys())
49
50     def _generate_pairs(self):
51         """Generate pairs of images dynamically based on a 50% chance rule."""
52         pairs = []
53         for person in self.people:
54             images = self.data[person]
55             num_images = len(images)
56
57             for img1 in images:
58                 if random.random() < 0.5: # 50% chance to pick same person
59                     img2 = random.choice([img for img in images if img != img1])
60                     label = 1
61                 else: # 50% chance to pick different person
62                     other_person = random.choice([p for p in self.people if p != person])
63                     img2 = random.choice(self.data[other_person])
64                     label = 0
65
66                 pairs.append((img1, img2, label))
67
68         return pairs
69
70     def __len__(self):
71         return len(self.pairs)
72
73     def __getitem__(self, idx):
74         img1_path, img2_path, label = self.pairs[idx]
75
76         img1 = Image.open(img1_path).convert('RGB')
77         img2 = Image.open(img2_path).convert('RGB')
78
79         if self.transform:
80             img1 = self.transform(img1)
81             img2 = self.transform(img2)
82
83         return img1, img2, torch.tensor(label, dtype=torch.float32)
84

```

شکل 26 - کلاس دیتابست چهره

پیاده سازی شبکه

در کلاس SiameseNetwork که در فایل model.py پیاده سازی شده است، شبکه از پیش آموزش دیده EfficientNetB0 لود میشود و با جایگزین کردن بخش طبقه بند آن با یک لایه اتصال کامل و یک لایه یکسوساز خطی که تعداد نرون های لایه خروجی که بیانگر تعداد ویژگی های معنادار استخراج شده از شبکه است، قابل تنظیم است؛ ساختار موردنظر مسئله بدست می آید.



```
1 class SiameseNetwork(nn.Module):
2     def __init__(self, embedding_dim=512):
3         super(SiameseNetwork, self).__init__()
4
5         self.efficientnet = efficientnet_b0(weights=EfficientNet_B0_Weights.DEFAULT)
6         self.feature_extractor = nn.Sequential(*list(self.efficientnet.children())[:-1])
7
8         self.fc = nn.Sequential(
9             nn.Linear(1280, embedding_dim),
10            nn.ReLU()
11        )
12
13     def forward(self, img1, img2):
14         emb1 = self.fc(self.feature_extractor(img1).view(img1.size(0), -1))
15         emb2 = self.fc(self.feature_extractor(img2).view(img2.size(0), -1))
16
17         return emb1, emb2
```

شکل 27 - ساختار شبکه سیامی توسعه داده شده

تابع اتلاف مقایسه ای

پرسش- از نظر شهودی این تابع اتلاف چه طور باعث یادگیری شباهت می شود؟ به بیان دیگر در مواجهه با داده های مشابه (با برجسب 1) و داده های غیرمشابه (با برجسب 0) این تابع چه رفتاری از خود نشان می دهد و چگونه عمل می کند؟

اگر داده ورودی مشابه باشند، $y^* = 1$ است پس تابع هزینه برای آن به فرم زیر خواهد بود: (رابطه 2)

$$\mathcal{L}(A, B) = \|f(A) - f(B)\|^2$$

پس در زمان مشابه بودن داده های ورودی داده شده، هر چه بردارهای ویژگی بدست آمده به هم نزدیکتر باشند تابع هزینه مقدار کمتری خواهد داشت.

و اگر داده ورودی مشابه نباشند، $y^* = 0$ است پس تابع هزینه برای آن به فرم زیر خواهد بود: (رابطه 3)

$$\mathcal{L}(A, B) = \max(0, m^2 - \|f(A) - f(B)\|^2)$$

پس تضمین میکند که فاصله بین $f(A)$ و $f(B)$ حداقل برابر m باشد به بیان دیگر اگر در حالت مشابه نبودن ورودی، فاصله بین $f(A)$ و $f(B)$ کمتر از m باشد، تابع هزینه مقدار مثبت میگیرد و با پس انتشار دادن گرادیان، وزن های شبکه به نحوی اپدیت میشود که فاصله بین بردار های ویژگی را بیشتر کند تا به حاشیه تضمین شده برسد.

در واقع کمینه شدن تابع هزینه مقایسه ای زمانی رخ میدهد که برای A و B مشابه $f(A)$ و $f(B)$ هم مشابه باشند و برای A و B نامشابه $f(A)$ و $f(B)$ حداقل فاصله ای برابر m داشته باشند.

پرسش- نقش مقدار حاشیه، در اتلاف مقایسه ای چیست؟ پیش بینی می کنید با افزایش مقدار حاشیه، عملکرد مدل در نگاشت تصاویر هم فرد و غیر هم فرد چه تغییری کند؟

حاشیه بالاتر، جداسازی مورد نیاز بین نقاط داده غیر مشابه را افزایش می دهد و مدل را نسبت به تفاوت ها حساس تر می کند.

اگر m خیلی بزرگ باشد، مدل ممکن است برای جداسازی موثر داده های غیر مشابه مشکل داشته باشد.

اگر m خیلی کوچک باشد، مدل ممکن است بین جفت های مشابه و غیر مشابه به درستی تمایز قائل نشود.

پس مقدار m اهمیت بسیار زیادی در نحوه عملکرد مدل دارد.

پیاده سازی این تابع هزینه در شکل زیر و در فایل `model.py` قابل مشاهده است:



```

1 class ContrastiveLoss(nn.Module):
2     def __init__(self, margin=1.0):
3         super(ContrastiveLoss, self).__init__()
4         self.margin = margin
5
6     def forward(self, emb1, emb2, label):
7         distance = torch.norm(emb1 - emb2, p=2, dim=1)
8         loss = label * distance**2 + (1 - label) * torch.max(torch.tensor(0, dtype=torch.float32, device=emb1.device), torch.tensor(self.margin**2, dtype=torch.float32, device=emb1.device) - distance**2)
9         return loss.mean()

```

شکل 28 - پیاده سازی تابع هزینه مقایسه ای

آموزش و تنظیم ابر پارامترها

دو رویکرد اصلی برای تنظیم ابر پارامترهای یک شبکه وجود دارد: جست و جوی شبکه ای و جست و جوی تصادفی. هریک از این دو روش را به صورت خلاصه توضیح دهید. مزیت اصلی روش جست و جوی تصادفی در چیست؟

جست و جوی شبکه ای (Grid Search) : در این روش، یک مجموعه از مقادیر ممکن برای هر ابر پارامتر تعریف شده و سپس تمامی ترکیبات ممکن از این مقادیر یک به یک بررسی می شوند. به همین دلیل این روش برای فضاهای جست و جوی کوچک مناسب است.

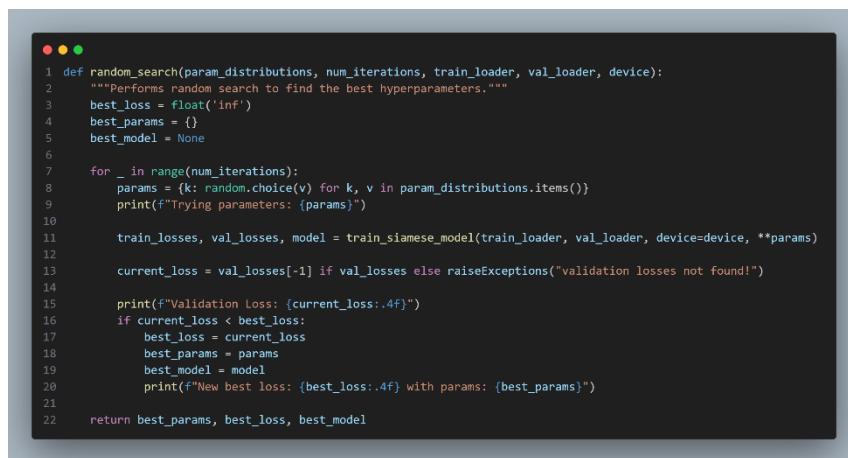
جست و جوی تصادفی (Random Search) : در این روش، مقادیر ابر پارامترها به طور تصادفی از یک توزیع مشخص انتخاب می شوند و مدل با این مقادیر ارزیابی می شود. برخلاف جست و جوی شبکه ای که تمامی ترکیبات را بررسی می کند، این روش

تنها بخش کوچکی از فضای جستجو را آزمایش می‌کند و به این شکل پیاده سازی آن در فضاهای جستجو بزرگ عملیاتی خواهد بود.

مزیت اصلی جستجوی تصادفی این است که در فضاهای جستجوی بزرگ، احتمال یافتن ترکیب‌های بهینه بیشتر است، زیرا به جای بررسی سیستماتیک، می‌تواند مقادیر مناسبی را بدون نیاز به ارزیابی تمامی ترکیبات پیدا کند. بهویژه زمانی که برخی از ابرپارامترها تأثیر بیشتری نسبت به سایرین دارند، این روش کارآمدتر خواهد بود.

پیاده سازی جستجو تصادفی

تابع `random_search` که در فایل `main.py` پیاده سازی شده و در شکل زیر قابل مشاهده است، توزیع پارامترها و تعداد سعی‌های مورد نیاز برای جستجو مدل بهتر را ورودی می‌گیرد و به صورت تصادفی یک کانفیگ را انتخاب می‌کند و مدل را با آن مقداردهی ابرپارامترها آموزش میدهد و اینکار را `num_iterations` بار تکرار می‌کند و در هر مرحله در صورت رسیدن به یک مدل بهتر (با تابع `هزینه کمتر روی داده‌های اعتبارسنجی`)، `best_params`, `best_loss`, `best_model` به روز رسانی می‌کند و در انتها آنها را بر می‌گرداند.



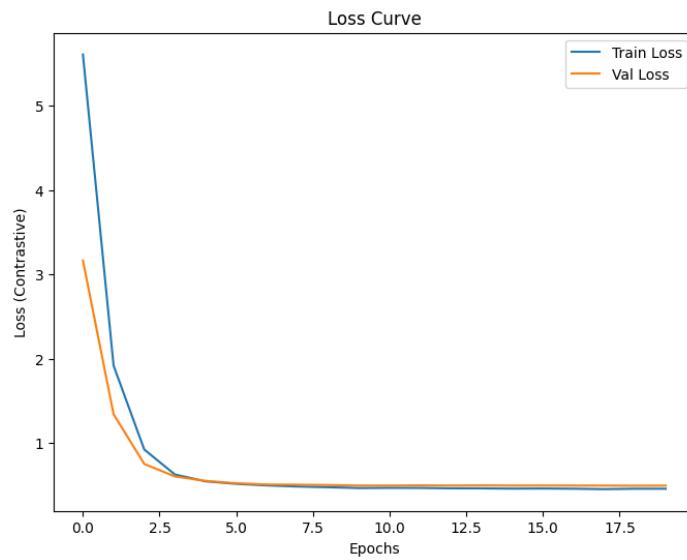
```
1 def random_search(param_distributions, num_iterations, train_loader, val_loader, device):
2     """Performs random search to find the best hyperparameters."""
3     best_loss = float('inf')
4     best_params = {}
5     best_model = None
6
7     for _ in range(num_iterations):
8         params = {k: random.choice(v) for k, v in param_distributions.items()}
9         print(f"Trying parameters: {params}")
10
11     train_losses, val_losses, model = train_siamese_model(train_loader, val_loader, device=device, **params)
12
13     current_loss = val_losses[-1] if val_losses else raiseExceptions("Validation losses not found!")
14
15     print(f"Validation Loss: {current_loss:.4f}")
16     if current_loss < best_loss:
17         best_loss = current_loss
18         best_params = params
19         best_model = model
20         print(f"New best loss: {best_loss:.4f} with params: {best_params}")
21
22     return best_params, best_loss, best_model
```

شکل 29 - پیاده سازی جستجو تصادفی

بهترین مدل پیاده شده توسط جستجو تصادفی، با مقادیر زیر برای ابرپارامترها آموزش داده شده بود:

Best hyperparameters found: {'lr': 0.001, 'loss_margin': 1.0, 'step_size': 10, 'gamma': 0.1, 'num_epochs': 20}

Best validation loss: 0.4965

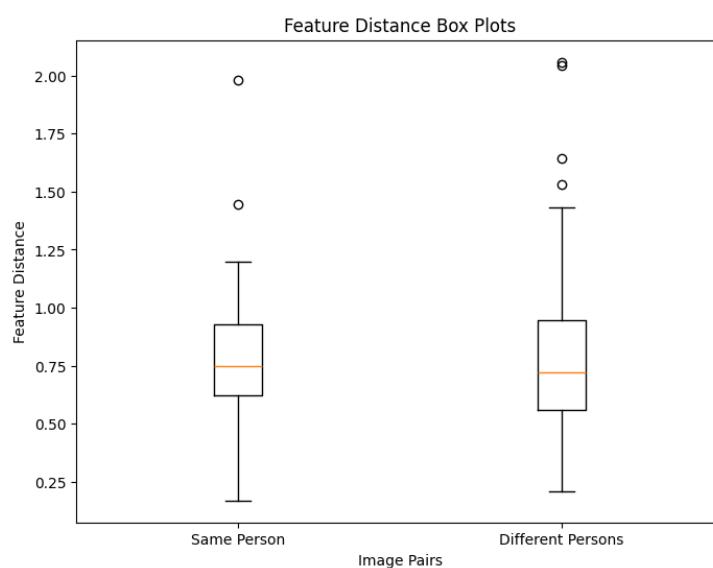


شکل 30 - نمودار اتلاف برای داده های آموزش و اعتبارسنجی بهترین مدل

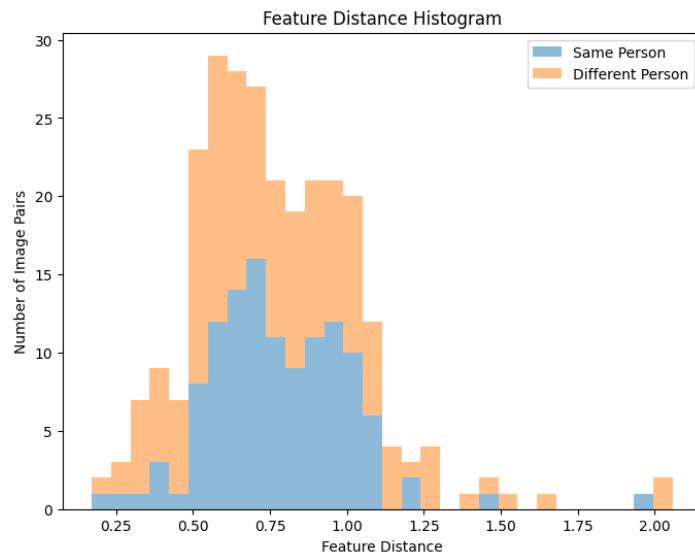
تحلیل رفتار مدل

بررسی فاصله بردارهای ویژگی

میانه فاصله بردارهای ویژگی تصاویر مربوط به یک فرد اندکی بالاتر از میانه تصاویر مربوط به افراد متفاوت است اما اصلاً قابل توجه نیست. همچنین به دلیل وجود همپوشانی قابل توجه بین برد میان چارکی دو دسته که در شکل 31 قابل مشاهده است، تغییر حاصله تابع اتلاف مقایسه ای تاثیر مثبتی در خروجی مدل نخواهد داشت و احتمالاً با تغییر تعداد نرون های خروجی شبکه سیامی و نگاشت تصاویر به یک فضای با بعد بالاتر فاصله معنادارتری بین دو دسته شکل خواهد گرفت.



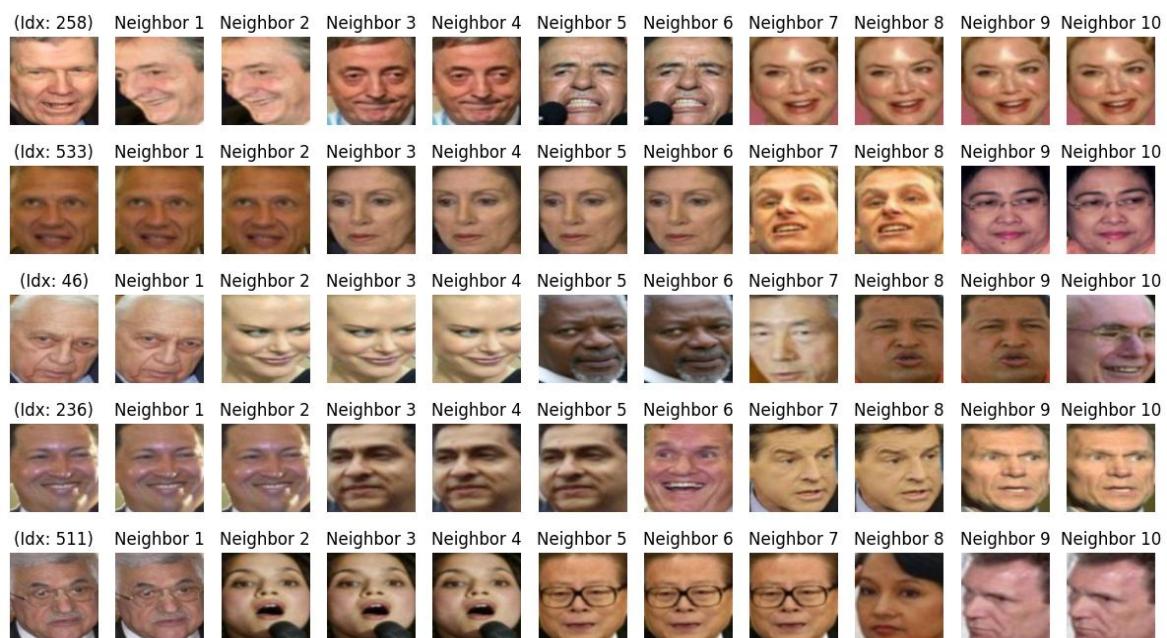
شکل 31 - نمودار جعبه ای فاصله بردارهای ویژگی تصاویر تفکیک شده براساس برچسب



شکل 32 - هیستوگرام فاصله بردارهای ویژگی تصاویر تفکیک شده براساس برچسب

بررسی همسایگی در فضای ویژگی

عملکرد مدل در این بخش بسیار مورد تایید قرار گرفت! البته با تکرار بررسی این بخش برای نمونه های بیشتر، متوجه گیر کردن مدل برای بازیابی تصاویر دیگر افراد عینکی شدم در واقع تصویر هر فرد عینکی دیگر را نیز بسیار مشابه تشخیص میداد و گاهای نزدیکترین همسایه را هم برای این دسته از نمونه ها غلط مشخص میکرد.

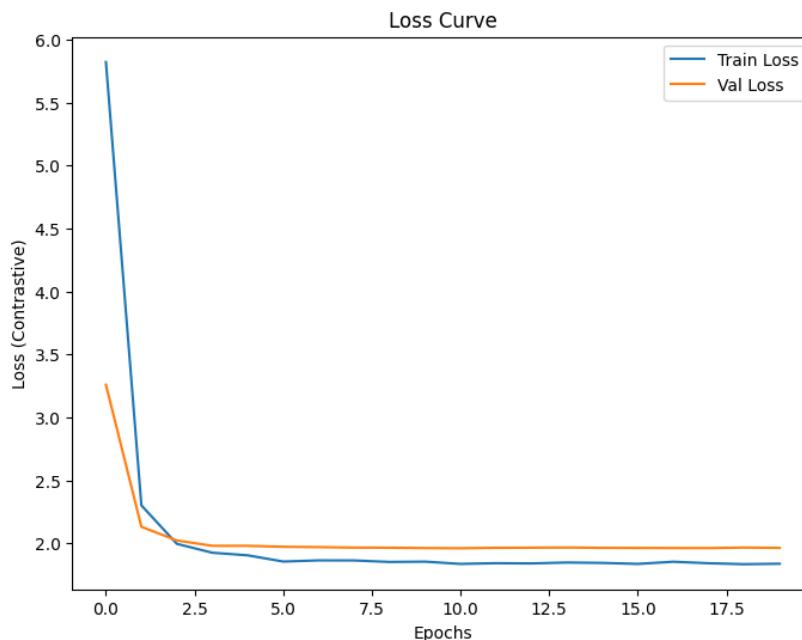


شکل 33 - بررسی 10 همسایه نزدیک به 5 تصویر رندوم دیتاست

سوالات امتیازی

بررسی نقش مقدار حاشیه در تابع اتلاف

همانطور که بالاتر گفته شد به دلیل نبود فاصله معنادار بین بردارهای ویژگی دو دسته، افزایش مقدار حاشیه نه تنها باعث بهبود عملکرد نشد بلکه مقدار تابع اتلاف برای داده های اعتبار سنجی را هم از 0.4965 به 1.9639 افزایش داد. بنابراین باید روش های دیگر را امتحان کرد.



شکل 34 - نمودار اتلاف برای داده های آموزش و اعتبار سنجی با افزایش حاشیه به دو

پیاده سازی تابع اتلاف سه گانه

اتلاف سه گانه برای یادگیری فاصله ای (metric learning) به کار می رود. این تابع اتلاف سه نمونه را در نظر می گیرد:

- مثبت: (Positive, x^+): نمونه ای که متعلق به کلاس نمونه لنگر (Anchor) است.
- لنگر: (Anchor, x^a): نمونه اصلی که قصد داریم فاصله آن را با نمونه های دیگر مقایسه کنیم.
- منفی: (Negative, x^-): نمونه ای که متعلق به کلاسی متفاوت از نمونه لنگر است.

رابطه ریاضی تابع اتلاف سه گانه به صورت زیر است :

$$L = \sum_{i=1}^N \max (d(x_i^a, x_i^+) - d(x_i^a, x_i^-) + \alpha, 0)$$

که در آن:

- α فاصله بین دو بردار است (میتوان از فاصله اقلیدسی یا کسینوسی استفاده کرد.)
- α حاشیه است، که مقدار حداقلی فاصلهای را که نمونه منفی باید نسبت به نمونه مثبت داشته باشد مشخص می‌کند. (مشابه مفهوم حاشیه در تابع اتلاف مقایسه ای)
- N تعداد نمونه‌های سه‌گانه در مجموعه داده است.

توضیح مفهوم تابع اتلاف سه‌گانه

این تابع تلاش می‌کند تا فاصله‌ی نمونه لنگر از نمونه مثبت را کمتر از فاصله‌ی آن از نمونه منفی نگه دارد.

- اگر فاصله بین نمونه لنگر و مثبت بهاندازه کافی کوچک و فاصله بین نمونه لنگر و منفی بهاندازه کافی بزرگ باشد (یعنی اختلاف آن‌ها بیشتر از α باشد)، مقدار تابع اتلاف برابر صفر خواهد شد.
- اگر فاصله نمونه لنگر از نمونه منفی کمتر یا نزدیک به فاصله‌ی آن از نمونه مثبت باشد، تابع اتلاف مقدار مثبتی خواهد داشت و مدل را مجبور می‌کند که امبدینگ‌ها را بهبود بیخشد.

هدف تابع اتلاف سه‌گانه، نزدیک کردن نمونه‌های یک کلاس به هم و دور کردن نمونه‌های کلاس‌های مختلف، از یکدیگر است.



```

1  class TripletLoss(nn.Module):
2      def __init__(self, margin=1.0):
3          super(TripletLoss, self).__init__()
4          self.margin = margin
5
6      def forward(self, anchor, positive, negative):
7          pos_dist = torch.norm(anchor - positive, p=2, dim=1)  # ||f(A) - f(P)||_
8          neg_dist = torch.norm(anchor - negative, p=2, dim=1)  # ||f(A) - f(N)||_
9          loss = torch.mean(torch.relu(pos_dist - neg_dist + self.margin))
10
11      return loss

```

شکل 35 - پیاده سازی تابع اتلاف سه گانه

مراجع

1. <https://www.cs.toronto.edu/~kriz/cifar.html>
2. <https://www.kaggle.com/datasets/sudhanshu2198/face-recognition-dataset-siamese-network>
3. <https://towardsdatascience.com/triplet-loss-advanced-intro-49a07b7d8905/>