به نام خدا

دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده برق و کامپیوتر

درس تحلیل و طراحی شبکه های عصبی عمیق

تمرین امتیازی اول

نام و نام خانوادگی : کیانا هوشانفر

شماره دانشجویی : ۸۱۰۱۰۱۳۶۱

آذر ماه ۱۴۰۲

فهرست

سوال اول : آشنایی با ساختار شبکه های عصبی

- کدهایی که در تمام بخش‌های سوال از آن استفاده شده است:

۱. dataset loader:

در این سوال از دیتاست cifar10 استفاده شده است که به شکل زیر آن را لود می‌کنیم:

همچنین توجه داشته باشیم که برای افزایش دقت شبکه و جلوگیری از overfitting از روش های تقویت داده استفاده باید کنی. در اینجا از تبدیل های data_augmentation_train برای مجموعه داده آموزشی طراحی شده اند و شامل چرخش های افقی و عمودی تصادفی برای افزایش تنوع، چرخش تصادفی ۱۵ درجه برای تنوع بیشتر، تبدیل تصویر به تنسور PyTorch با استفاده از transforms.ToTensor() و نرمال سازی مقادیر تنسور با استفاده از transforms.Normalize() با مقادیر میانگین و انحراف استاندارد روی (۰,۵، ۰,۵، ۰,۵). این تبدیلات توانایی مدل را برای تعمیم با قرار دادن آن در معرض نماهای متنوع از تصاویر آموزشی افزایش می‌دهند. از سوی دیگر، تبدیل‌های data_preprocessing_test برای مجموعه داده تست در نظر گرفته شده‌اند.

```
# Data augmentation transformations for training data
data_augmentation_train = transforms.Compose([
    transforms.RandomHorizontalFlip(),  # Randomly flip the image horizontally
    transforms.RandomRotation(15),  # Randomly rotate the image by 15 degrees
    transforms.RandomVerticalFlip(),  # Randomly flip the image vertically
    transforms.ToTensor(),  # Convert the image to a PyTorch tensor
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))  # Normalize the
tensor values
])

# Data preprocessing transformations for test/validation data
data_preprocessing_test = transforms.Compose([
    transforms.ToTensor(),  # Convert the image to a PyTorch tensor
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)) # Normalize the tensor
values
])
```

لود دیتاست به شکل زیر است که این روش به ما ۳ مجموعه داده متوازن می دهد. (۱۰ درصد از داده ها به عنوان val جدا کرده ایم.)

```
# Load CIFAR-10 dataset
# Training set with data augmentation
train_dataset = datasets.CIFAR10(root='./data', train=True, download=True,
transform=data_augmentation_train)
# Test set with data preprocessing
test_dataset = datasets.CIFAR10(root='./data', train=False, download=True,
transform=data_preprocessing_test)

# Splitting the training dataset into training and validation sets
train_size = int(0.9 * len(train_dataset))
val_size = len(train_dataset) - train_size
train_dataset, val_dataset = random_split(train_dataset, [train_size, val_size])

# Data loaders
batch_size = 128
# Training data loader
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
# Validation data loader
val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)
# Test data loader
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
```

۲. training loop:

هایپرپارامترها را طبق موارد گفته شده در صورت سوال به شکل زیر پیاده سازی می کنیم:

```
# Optimizer, Learning Rate Scheduler, and Loss Criterion
# Stochastic Gradient Descent (SGD) optimizer
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9)
# Learning rate scheduler with step decay
scheduler = StepLR(optimizer, step_size=50, gamma=0.1)
# CrossEntropyLoss criterion for classification tasks
criterion = nn.CrossEntropyLoss()
```

و حلقه آموزش هم بصورت زیر است:

حلقه در تعداد معینی از دوره‌ها تکرار می‌شود که هر دوره شامل یک مرحله آموزشی و یک مرحله اعتبار سنجی است. در حین آموزش، مدل روی حالت آموزش تنظیم می شود و برای هر دسته در بارگذار داده آموزشی، forward and backward passes انجام می دهد و پارامترهای مدل را بر اساس تلفات محاسبه شده و با استفاده از بهینه ساز مشخص شده به روز می کند. دقت آموزش با مقایسه پیش‌بینی‌های مدل با برچسب‌های حقیقی محاسبه می‌شود. پس از مرحله آموزش یک دوره، میانگین تلفات و دقت تمرین

چاپ می شود. سپس مدل به حالت ارزیابی برای مرحله اعتبار سنجی تغییر می کند، جایی که مجموعه داده اعتبار سنجی را بدون به روز رسانی پارامترها پردازش می کند. loss اعتبار و دقت محاسبه، چاپ می‌شود، و مقادیر آموزشی و اعتبارسنجی برای ترسیم بعدی اضافه می‌شوند. زمان‌بندی نرخ یادگیری برای تنظیم نرخ یادگیری قدم گذاشته است. و وزن های بهترین نسخه از شبکه آموزش داده شده را نیز ذخیره می کنیم.

```
train_losses = []
train_accuracies = []
val_losses = []
val_accuracies = []
best_val_accuracy = 0
best_model_weights = None

# Number of epochs
epochs = 100

# Training loop
for epoch in range(1, epochs + 1):
    # Training
    model.train()
    total_loss = 0
    correct = 0
    total_samples = 0
    for data, target in tqdm(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = criterion(output, target)
        loss.backward()
        optimizer.step()

        # Compute training accuracy
        pred = output.argmax(dim=1, keepdim=True)
        correct += pred.eq(target.view_as(pred)).sum().item()
        total_samples += data.size(0)

        total_loss += loss.item()

    # Calculate average training loss and accuracy
    average_loss = total_loss / len(train_loader.dataset)
    accuracy_train = correct / total_samples

    print(f'Train Epoch: {epoch}, Average Loss: {average_loss:.4f}, Accuracy:
{accuracy_train:.2}')
```

```
    # Validation
    model.eval()
    val_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in val_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            val_loss += criterion(output, target).item()
            pred = output.argmax(dim=1, keepdim=True)
            correct += pred.eq(target.view_as(pred)).sum().item()

    val_loss /= len(val_loader.dataset)
    accuracy_val = correct / len(val_loader.dataset)
    print(f'Validation set: Average loss: {val_loss:.4f}, Accuracy:
{accuracy_val:.2f}')

    # Append values for plotting
    train_losses.append(average_loss)
    train_accuracies.append(accuracy_train)

    # Append values for plotting
    val_losses.append(val_loss)
    val_accuracies.append(accuracy_val)

    if accuracy_val > best_val_accuracy:
       best_val_accuracy = accuracy_val
       best_model_weights = model.state_dict()

    scheduler.step()
```

کدهای دقت شبکه بر روی داده های تست به شکل زیر است:

توجه داشته باشید که در مرحله قبل وزن های بهترین نسخه از شبکه آموزش داده ذخیره شده است و در اینجا از آن استفاده می کنیم.

۳. محاسبه متریک های خواسته شده

در تمرین دوم کدها بهینه شده SI توضیح کامل داده شده است که در اینجا از آن ها استفاده می کنیم.

کدها در لینک زیر قرار گرفته اند.

https://github.com/K-Hooshanfar/data_complexity_measures

در این قسمت از ۱۰درصد از داده ها استفاده میکنیم که بصورت متوارن تقسیم شده اند. داده هارا ابتدا نرمالایز کرده و به تنسور تبدیل میکنیم.

```
from sklearn.model_selection import train_test_split

# Load CIFAR-10 dataset
train_dataset = datasets.CIFAR10(root='./data', train=True, download=True,
transform=transform_train)
test_dataset = datasets.CIFAR10(root='./data', train=False, download=True,
transform=transform_test)

# Splitting train dataset into train and validation sets
train_size = int(0.9 * len(train_dataset))
val_size = len(train_dataset) - train_size
train_dataset, val_dataset = random_split(train_dataset, [train_size, val_size])

# Convert test_dataset to a list or NumPy array
test_data_list = [(img, label) for img, label in test_dataset]

# Further split train_dataset into train_loader_dataset and 10% balanced subset
train_loader_dataset, _ = train_test_split(train_dataset.dataset,
train_size=int(0.1 * len(train_dataset)),
                                           test_size=None, shuffle=True,
stratify=train_dataset.dataset.targets)

# Similarly, for the test_loader
test_loader_dataset, _ = train_test_split(test_data_list, train_size=int(0.1 *
len(test_dataset)),
                                          test_size=None, shuffle=True,
stratify=test_dataset.targets)

# Data loaders
train_loader = DataLoader(train_loader_dataset, batch_size=128, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=128, shuffle=False)
test_loader = DataLoader(test_loader_dataset, batch_size=128, shuffle=False)
```

در ادامه برای اینکه featureهای بعد از هر لایه را بدست بیاریم به شکل زیر عمل می کنیم:

```
features_per_layer = {}
labels = []

# Get the total number of layers in the model
total_layers = len(list(model.children()))

# Exclude the last four layers
layers_to_be_deleted = set(list(model.children())[-4:])
```

```
# Attach hooks to each layer except the excluded ones
features_per_layer = {}
for name, layer in model.named_children():
    if layer not in layers_to_be_deleted:
        features_per_layer[name] = []
        def hook(module, input, output, name=name):
            features_per_layer[name].append(output.detach())
        layer.register_forward_hook(hook)
```

هدف این کد استخراج ویژگی‌های میانی از هر لایه یک مدل شبکه عصبی ، به استثنای چهار لایه آخر است. features_per_layer برای ذخیره ویژگی‌های هر لایه شروع می‌شود. تعداد کل لایه ها در مدل با استفاده از طول لیست model children تعیین می شود. در مرحله بعد، مجموعه ای به نام «layers_to_be_deleted» ایجاد می‌شود که شامل چهار لایه آخر مدل است. پس از آن، یک حلقه از طریق هر لایه model children با استفاده از روش "named_children" تکرار می شود. برای هر لایه ای که در مجموعه لایه هایی قرار نمی گیرد که باید حذف شوند، یک هوک رو به جلو برای لایه ثبت می شود. این قلاب که به عنوان تابعی به نام «hook» تعریف می‌شود، ویژگی‌های خروجی لایه را به ورودی مربوطه در پ «features_per_layer» اضافه می‌کند. با اتصال این قلاب‌ها به لایه‌های مورد نظر، کد مجموعه‌ای از ویژگی‌های میانی را در حین محاسبات forward pass امکان‌پذیر می‌سازد، و تحلیل بیشتر یا تجسم نمایش‌های داخلی مدل را تسهیل می‌کند.

کد زیر را برای محاسبه ویژگی و لیبل ها برای داده های ترین و تست استفاده می کنیم:

```
model.to('cuda:0' if torch.cuda.is_available() else 'cpu')
model.eval()
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

with torch.no_grad():
    for inputs, targets in tqdm(train_loader): ##test_loader
        inputs = inputs.to(device)
        model(inputs)
        labels.append(targets.cpu())

for layer_name, layer_features in features_per_layer.items():
    if layer_features:
        features_per_layer[layer_name] = torch.cat([f.view(f.size(0), -1) for f
in layer_features])

labels = torch.cat(labels)
```

این کد برای استخراج ویژگی های میانی از هر لایه از مدل شبکه عصبی در حین پردازش مجموعه داده های آموزشی طراحی شده است. مدل با استفاده از model.eval() روی حالت ارزیابی تنظیم می شود. یک حلقه از طریق مجموعه داده آموزشی/تست با استفاده از یک بارگذار داده تکرار می شود. برای هر دسته، ورودی ها به دستگاه مشخص شده منتقل می شوند و مدل ورودی ها را پردازش می کند. اهداف در لیست "برچسب ها" در CPU ذخیره می شوند. پس از حلقه، برچسب های ذخیره شده به هم متصل می شوند تا یک تنسور را تشکیل دهند. متعاقباً، ویژگی‌های میانی جمع‌آوری‌شده در طول گذر رو به جلو برای هر لایه، مسطح شده و در امتداد بعد دوم به هم متصل می‌شوند و یک تانسور دو بعدی برای هر لایه تشکیل می‌دهند. features_per_layer در آن هر کلید با نام لایه مطابقت دارد و مقدار مرتبط تنسوری است که حاوی ویژگی های مسطح از آن لایه در کل مجموعه داده است. این کد استخراج و سازماندهی ویژگی های میانی را از لایه های مختلف مدل برای تجزیه و تحلیل یا تجسم بیشتر تسهیل می کند.

کدهای محاسبه SI و CSI در فایل های ضمیمه شده قرار دارند.

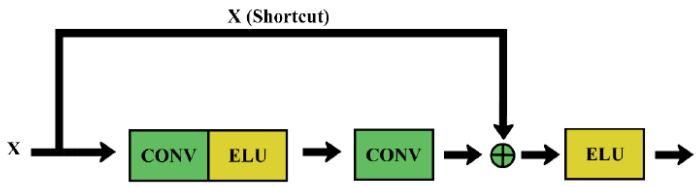
الف)

```
# Define a ResNet block
class MyResNetBlock(nn.Module):
    # Constructor with input and output channel parameters
    def __init__(self, in_channels, out_channels):
        super(MyResNetBlock, self).__init__()
        # First convolutional layer
        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3,
padding=1)
        # ELU activation function
        self.activation1 = nn.ELU()
        # Second convolutional layer
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3,
padding=1)

    # Forward method for the block
    def forward(self, x):
        shortcut = x  # Preserve the input for the shortcut connection
        x = self.conv1(x)
        x = self.activation1(x)
        x = self.conv2(x)
        x += shortcut  # Add the input to the output (residual connection)
        x = self.activation1(x)
        return x
```

```python
# Define the main ResNet model
class MyResNet(nn.Module):
    # Constructor
    def __init__(self):
        super(MyResNet, self).__init__()
        # Initial convolutional layers
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, padding=1)
        self.activation1 = nn.ELU()
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.activation2 = nn.ELU()
        self.maxpool1 = nn.MaxPool2d(kernel_size=2)

        # Residual blocks
        self.resblock1 = MyResNetBlock(64, 64)
        self.maxpool2 = nn.MaxPool2d(kernel_size=2)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, padding=1)
        self.activation3 = nn.ELU()
        self.maxpool3 = nn.MaxPool2d(kernel_size=2)
        self.conv4 = nn.Conv2d(128, 256, kernel_size=3, padding=1)
        self.activation4 = nn.ELU()
        self.resblock2 = MyResNetBlock(256, 256)

        # Average pooling and fully connected layers
        self.avgpool = nn.AvgPool2d(kernel_size=3, stride=2)
        self.flatten = nn.Flatten()
        self.dense1 = nn.Linear(256, 256)
        self.dense2 = nn.Linear(256, 10)
        self.softmax = nn.Softmax(dim=1)

    # Forward method for the model
    def forward(self, x):
        x = self.conv1(x)
        x = self.activation1(x)
        x = self.conv2(x)
        x = self.activation2(x)
        x = self.maxpool1(x)

        x = self.resblock1(x)
        x = self.conv3(x)
        x = self.activation3(x)
        x = self.maxpool2(x)

        x = self.conv4(x)
        x = self.activation4(x)
        x = self.maxpool3(x)
        x = self.resblock2(x)
        x = self.avgpool(x)
```

```
        x = self.flatten(x)
        x = self.dense1(x)
        x = self.dense2(x)
        x = self.softmax(x)
        return x
```

این کد یک نسخه ساده از معماری ResNet را برای طبقه بندی تصاویر تعریف می کند. برای سادگی ما بلوک ResNet ("MyResNetBlock") را جدا تعریف می شود که شامل دو لایه کانولوشن با عملکردهای فعال سازی ELU است. روش فوروارد بلوک یک اتصال residual را با افزودن ورودی به خروجی در بر می گیرد. این بلوک در صورت سوال به شکل زیر آمده است.



شکل ۱ - Residual Block

سپس مدل اصلی MyResNet ساخته می‌شود که دارای لایه‌های کانولوشن اولیه، یک سری بلوک‌های MyResNetBlock residual و average pooling و لایه‌های کاملاً متصل است. معماری با یک لایه کانولوشن و به دنبال آن فعال سازی ELU، یک لایه کانولوشن دیگری با فعال سازی ELU و یک لایه max-pooling شروع می شود. متعاقبا، یک بلوک باقیمانده MyResNetBlock اعمال می شود و به دنبال آن لایه های کانولوشنال و max-pooling اعمال می شود. یک بلوک residual دیگر قبل از لایه average pooling، flattern کردن و لایه های کاملاً متصل استفاده می شود. لایه خروجی از تابع softmax برای طبقه بندی چند کلاسه استفاده می کند و احتمالات کلاس را تولید می کند. این نوع ResNet نسبتاً کم عمق است اما دارای اتصالات باقیمانده برای کاهش مشکل گرادیان ناپدید شدن و سهولت آموزش شبکه های عمیق است. روش فوروارد مدل، ورودی را از طریق لایه‌های تعریف‌شده به صورت متوالی پردازش می‌کند، با استفاده از بلوک‌های کانولوشنال و residual، max-pooling و لایه‌های کاملاً متصل. فعال سازی ELU در سراسر شبکه استفاده می شود. لایه نهایی از softmax برای تولید احتمالات کلاس برای هر تصویر در دسته ورودی استفاده می کند. این ترتیب مطابق با ساختاری است که داده شده است. توجه داشته باشید که برای آخرین `avgpool` باید `stride` را ۲ قرار دهیم تا `output shape` مطابق چیزی باشد که در صورت سوال داده شده است.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 32, 32, 32]             896
              ELU-2          [-1, 32, 32, 32]               0
            Conv2d-3          [-1, 64, 32, 32]          18,496
              ELU-4          [-1, 64, 32, 32]               0
         MaxPool2d-5          [-1, 64, 16, 16]               0
            Conv2d-6          [-1, 64, 16, 16]          36,928
              ELU-7          [-1, 64, 16, 16]               0
            Conv2d-8          [-1, 64, 16, 16]          36,928
              ELU-9          [-1, 64, 16, 16]               0
    MyResNetBlock-10          [-1, 64, 16, 16]               0
           Conv2d-11         [-1, 128, 16, 16]          73,856
             ELU-12         [-1, 128, 16, 16]               0
        MaxPool2d-13          [-1, 128, 8, 8]               0
           Conv2d-14          [-1, 256, 8, 8]         295,168
             ELU-15          [-1, 256, 8, 8]               0
        MaxPool2d-16          [-1, 256, 4, 4]               0
           Conv2d-17          [-1, 256, 4, 4]         590,080
             ELU-18          [-1, 256, 4, 4]               0
           Conv2d-19          [-1, 256, 4, 4]         590,080
             ELU-20          [-1, 256, 4, 4]               0
    MyResNetBlock-21          [-1, 256, 4, 4]               0
        AvgPool2d-22          [-1, 256, 1, 1]               0
          Flatten-23                [-1, 256]               0
           Linear-24                [-1, 256]          65,792
           Linear-25                 [-1, 10]           2,570
          Softmax-26                 [-1, 10]               0
================================================================
Total params: 1,710,794
Trainable params: 1,710,794
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 3.26
Params size (MB): 6.53
Estimated Total Size (MB): 9.79
----------------------------------------------------------------
```

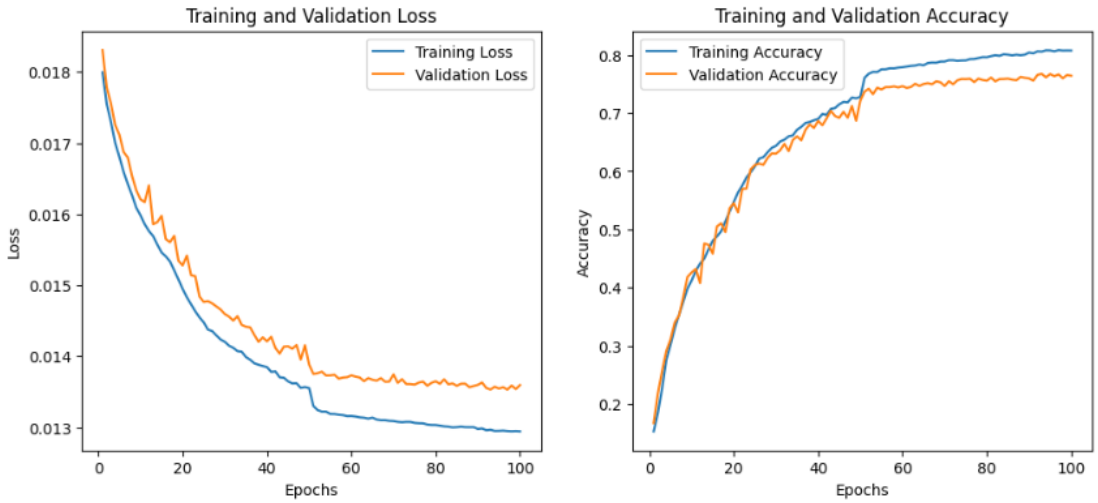
شکل ۲ – ساختار مدل

مدل را آموزش می دهیم و به نتایج زیر دست پیدا کردیم:

دقت نهایی بر روی test :

Test set: Average loss: 0.0133, Accuracy: 0.77

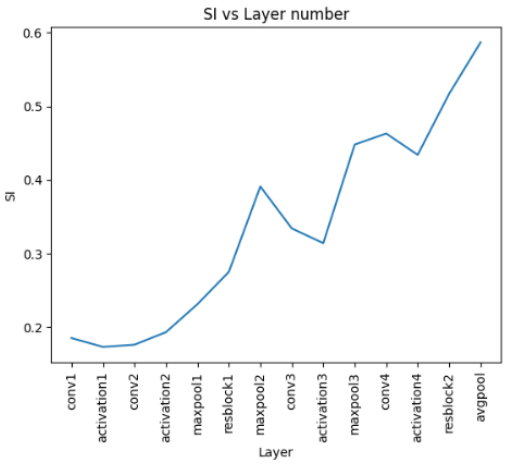
منحنی های دقت و loss:



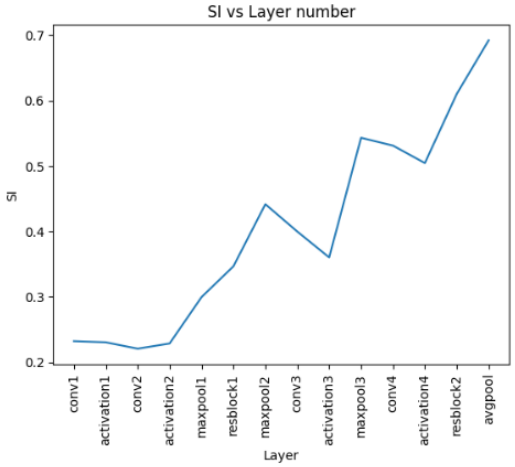
شکل ۳ - منحني هاي دقت و loss برای مدل اول

نتایج متریک ها:

- منحنی SI بر روی تمامی لایه ها:



شکل ٤ - منحنی SI بر روی تمامی لایه ها برای داده های test



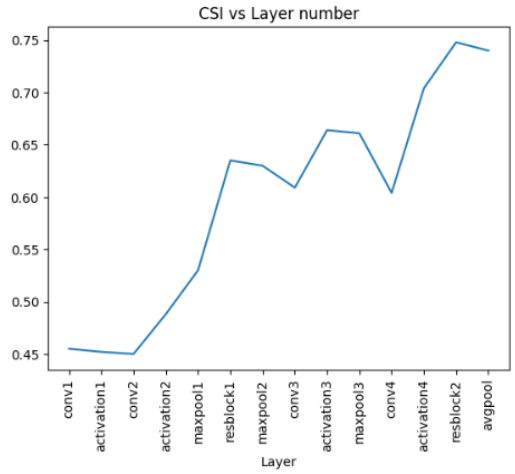
شکل ٥ -منحنی SI بر روی تمامی لایه ها برای داده های train

منحنی ها روند صعودی دارند. در conv3 مشاهده می کنیم که SI کاهش پیدا کرده است. که نشان میدهد میتوانیم این لایه راحذف کنیم.

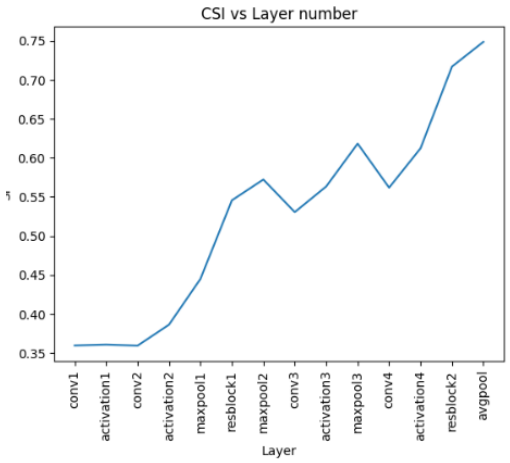
جدول ١ – مقادیر si برای هر لایه برای داده های ترین و تست

si_train	si_test
[('conv1', 0.21088889241218567), ('activation1', 0.21222221851348877), ('conv2', 0.20222222805023193), ('activation2', 0.19866666197776794), ('maxpool1', 0.28022223711013794), ('resblock1', 0.34066668152809143), ('maxpool2', 0.44022223353385925), ('conv3', 0.39399999380111694), ('activation3', 0.36622223258018494), ('maxpool3', 0.526888906955719), ('conv4', 0.5180000066757202), ('activation4', 0.48222222924232483), ('resblock2', 0.608222246170044), ('avgpool', 0.6777777671813965)]	[('conv1', 0.20200000703334808), ('activation1', 0.20000000298023224), ('conv2', 0.19700001180171967), ('activation2', 0.1940000057220459), ('maxpool1', 0.242000013589859), ('resblock1', 0.2800000011920929), ('maxpool2', 0.3750000298023224), ('conv3', 0.3290000259876251), ('activation3', 0.31700000166893005), ('maxpool3', 0.45100003480911255), ('conv4', 0.4610000252723694), ('activation4', 0.4140000343322754), ('resblock2', 0.5360000133514404), ('avgpool', 0.6030000448226929)]

- منحنی CSI بر روی تمامی لایه ها:



شکل ۷ - منحنی CSI بر روی تمامی لایه ها برای داده های test

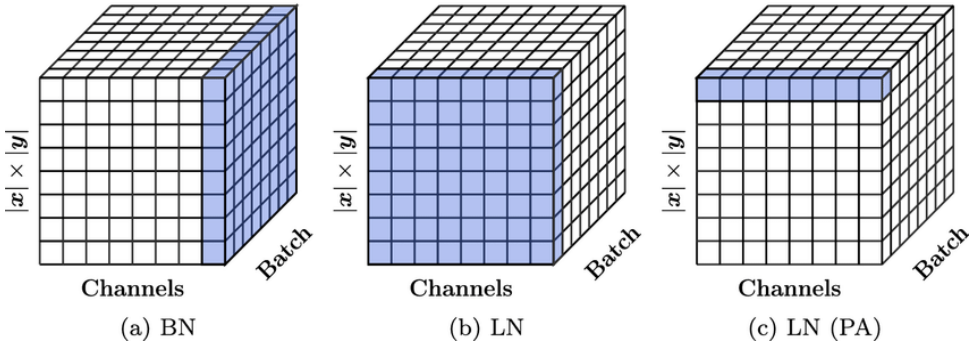


شکل ٦ -منحنی CSI بر روی تمامی لایه ها برای داده های train

جدول ۲ – مقادیر csi برای هر لایه برای داده های ترین و تست

csi_train	csi_test
[('conv1', 0.36133334040641785), ('activation1', 0.36533334851264954), ('conv2', 0.3633333444595337), ('activation2', 0.39044445753097534), ('maxpool1', 0.44244444370269775), ('resblock1', 0.5460000038146973), ('maxpool2', 0.5695555806159973), ('conv3', 0.5291110873222351), ('activation3', 0.5635555386543274), ('maxpool3', 0.6286666989326477), ('conv4', 0.565333366394043), ('activation4', 0.6217777729034424), ('resblock2', 0.7084444761276245), ('avgpool', 0.7468888759613037)]	[('conv1', 0.44200003147125244), ('activation1', 0.4390000104904175), ('conv2', 0.453000009059906), ('activation2', 0.48900002241134644), ('maxpool1', 0.5160000324249268), ('resblock1', 0.64000004529953), ('maxpool2', 0.6500000357627869), ('conv3', 0.6140000224113464), ('activation3', 0.6720000505447388), ('maxpool3', 0.6810000538825989), ('conv4', 0.6160000562667847), ('activation4', 0.6960000395774841), ('resblock2', 0.7190000414848328), ('avgpool', 0.706000030040741)]

ب)



شکل ۸ – مقایسه بین انواع لایه های نرمال سازی

لایه‌های نرمال‌سازی، مانند Normalization Batch (BN)، Normalization Group (GN) و Layer Normalization (LN)، نقش مهمی در بهبود آموزش و عملکرد شبکه‌های عصبی دارند. هدف اولیه آنها رسیدگی به مسائل مربوط به تغییر متغیر داخلی و کمک به تثبیت و تسریع روند آموزش است. لایه‌های نرمال‌سازی تسریع تمرین، بهبود همگرایی و ارائه نرمال‌سازی با حفظ توزیع‌های فعالسازی پایدار در طول تمرین است. انتخاب بین BN، GN و LN به عواملی مانند مجموعه داده، اندازه دسته و معماری شبکه بستگی دارد. BN اغلب انتخاب پیش‌فرض است، اما GN و LN می‌توانند در موقعیت‌های خاصی مناسب‌تر باشند

۱. نرمال سازی دسته ای BN : نرمال سازی دسته ای برای فعال سازی یک لایه در یک دسته از نمونه ها اعمال می شود. با کم کردن میانگین دسته ای و تقسیم بر انحراف استاندارد دسته ای و به دنبال آن مقیاس بندی و جابجایی، ورودی را نرمال می کند. BN به کاهش مشکل گرادیان ناپدید/انفجار کمک می‌کند، استفاده از نرخ‌های یادگیری بالاتر را امکان‌پذیر می‌سازد و به عنوان نوعی منظم‌سازی عمل می‌کند. معمولاً برای لایه های کاملاً متصل و کانولوشن اعمال می شود.

۲. نرمال سازی گروه GN : نرمال سازی گروه اصلاحی در نرمال سازی دسته ای است که به گونه ای طراحی شده است که حساسیت کمتری نسبت به اندازه های دسته ای داشته باشد و برای سناریوهایی که از دسته های کوچکتر استفاده می شود یا اندازه دسته متفاوت است مناسب است. به جای عادی سازی در کل دسته، GN کانال ها را به گروه ها تقسیم می کند و هر گروه را به طور مستقل عادی می کند. این می تواند در شرایطی که اندازه دسته محدود است یا در سناریوهایی که اندازه دسته سازگار نیست مفید باشد.

۳. Layer Normalization (LN): Layer Normalization فعال سازی یک لایه را در بین ویژگی ها به طور مستقل برای هر نمونه نرمال می کند. بر روی هر ویژگی به طور مستقل عمل می کند و میانگین و انحراف استاندارد را برای هر ویژگی در تمام مکان های فضایی در دسته محاسبه می کند. LN می تواند در سناریوهایی که اندازه دسته کوچک است یا هنگام کار با RNN که مفهوم "بچ" در مراحل زمانی به خوبی تعریف نشده است مفید باشد.

ج)

۱. پیاده سازی batch normalization:

```
# Define a ResNet block with batch normalization
class MyResNetBlock(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(MyResNetBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3,
padding=1)
        self.bn1 = nn.BatchNorm2d(out_channels)  # Batch normalization
        self.activation1 = nn.ELU()
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3,
padding=1)
        self.bn2 = nn.BatchNorm2d(out_channels)  # Batch normalization

    def forward(self, x):
        shortcut = x
        x = self.conv1(x)
        x = self.bn1(x)
        x = self.activation1(x)
        x = self.conv2(x)
        x = self.bn2(x)
        x += shortcut
        x = self.activation1(x)
        return x

# Define the main ResNet model with batch normalization
class MyResNet(nn.Module):
    def __init__(self):
        super(MyResNet, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, padding=1)
        self.bn1 = nn.BatchNorm2d(32)  # Batch normalization
        self.activation1 = nn.ELU()
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.bn2 = nn.BatchNorm2d(64)  # Batch normalization
        self.activation2 = nn.ELU()
        self.maxpool1 = nn.MaxPool2d(kernel_size=2)
```

```
        self.resblock1 = MyResNetBlock(64, 64)
        self.maxpool2 = nn.MaxPool2d(kernel_size=2)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, padding=1)
        self.bn3 = nn.BatchNorm2d(128)  # Batch normalization
        self.activation3 = nn.ELU()
        self.maxpool3 = nn.MaxPool2d(kernel_size=2)
        self.conv4 = nn.Conv2d(128, 256, kernel_size=3, padding=1)
        self.bn4 = nn.BatchNorm2d(256)  # Batch normalization
        self.activation4 = nn.ELU()
        self.resblock2 = MyResNetBlock(256, 256)

        self.avgpool = nn.AvgPool2d(kernel_size=3, stride=2)
        self.flatten = nn.Flatten()
        self.dense1 = nn.Linear(256, 256)
        self.dense2 = nn.Linear(256, 10)
        self.softmax = nn.Softmax(dim=1)

    def forward(self, x):
        x = self.conv1(x)
        x = self.bn1(x)
        x = self.activation1(x)
        x = self.conv2(x)
        x = self.bn2(x)
        x = self.activation2(x)
        x = self.maxpool1(x)

        x = self.resblock1(x)
        x = self.conv3(x)
        x = self.bn3(x)
        x = self.activation3(x)
        x = self.maxpool2(x)

        x = self.conv4(x)
        x = self.bn4(x)
        x = self.activation4(x)
        x = self.maxpool3(x)
        x = self.resblock2(x)
        x = self.avgpool(x)
        x = self.flatten(x)
        x = self.dense1(x)
        x = self.dense2(x)
        x = self.softmax(x)
        return x
```

کد یک معماری ResNet را برای طبقه بندی تصاویر تعریف می کند، با ادغام نرمال سازی دسته ای در هر دو بلوک ResNet منفرد MyResNetBlock و مدل کلی MyResNet. نرمال سازی دسته ای بعد از هر لایه کانولوشن برای نرمال سازی فعال سازی ها اعمال می شود که به تثبیت و تسریع روند تمرین کمک می کند. MyResNetBlock از دو لایه کانولوشن با توابع فعال سازی ELU تشکیل شده است که هر کدام با نرمال سازی دسته ای دنبال می شوند. روش فوروارد بلوک شامل یک اتصال باقیمانده با افزودن ورودی به خروجی است که آموزش شبکه‌های عمیق‌تر را ارتقا می‌دهد. مدل اصلی، MyResNet، با لایه‌های کانولوشن اولیه، نرمال‌سازی دسته‌ای، فعال‌سازی ELU و max-pooling شروع می‌شود. متعاقباً از دو نمونه از بلوک ResNet تعریف شده MyResNetBlock استفاده می کند که هر کدام با نرمال سازی دسته ای، کانولوشن، فعال سازی و لایه های max-pooling همراه هستند. این مدل با average pooling، flattern کردن و لایه‌های کاملاً متصل به پایان می‌رسد و در نهایت احتمالات کلاس را از طریق یک تابع فعال‌سازی softmax تولید می‌کند.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 32, 32, 32]             896
       BatchNorm2d-2          [-1, 32, 32, 32]              64
              ELU-3          [-1, 32, 32, 32]               0
            Conv2d-4          [-1, 64, 32, 32]          18,496
       BatchNorm2d-5          [-1, 64, 32, 32]             128
              ELU-6          [-1, 64, 32, 32]               0
         MaxPool2d-7          [-1, 64, 16, 16]               0
            Conv2d-8          [-1, 64, 16, 16]          36,928
       BatchNorm2d-9          [-1, 64, 16, 16]             128
             ELU-10          [-1, 64, 16, 16]               0
           Conv2d-11          [-1, 64, 16, 16]          36,928
      BatchNorm2d-12          [-1, 64, 16, 16]             128
             ELU-13          [-1, 64, 16, 16]               0
    MyResNetBlock-14          [-1, 64, 16, 16]               0
           Conv2d-15         [-1, 128, 16, 16]          73,856
      BatchNorm2d-16         [-1, 128, 16, 16]             256
             ELU-17         [-1, 128, 16, 16]               0
         MaxPool2d-18           [-1, 128, 8, 8]               0
           Conv2d-19           [-1, 256, 8, 8]         295,168
      BatchNorm2d-20           [-1, 256, 8, 8]             512
             ELU-21           [-1, 256, 8, 8]               0
         MaxPool2d-22           [-1, 256, 4, 4]               0
           Conv2d-23           [-1, 256, 4, 4]         590,080
      BatchNorm2d-24           [-1, 256, 4, 4]             512
             ELU-25           [-1, 256, 4, 4]               0
           Conv2d-26           [-1, 256, 4, 4]         590,080
      BatchNorm2d-27           [-1, 256, 4, 4]             512
             ELU-28           [-1, 256, 4, 4]               0
    MyResNetBlock-29           [-1, 256, 4, 4]               0
        AvgPool2d-30           [-1, 256, 1, 1]               0
          Flatten-31                 [-1, 256]               0
           Linear-32                 [-1, 256]          65,792
           Linear-33                  [-1, 10]           2,570
          Softmax-34                  [-1, 10]               0
================================================================
Total params: 1,713,034
Trainable params: 1,713,034
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 4.69
Params size (MB): 6.53
Estimated Total Size (MB): 11.24
----------------------------------------------------------------
```

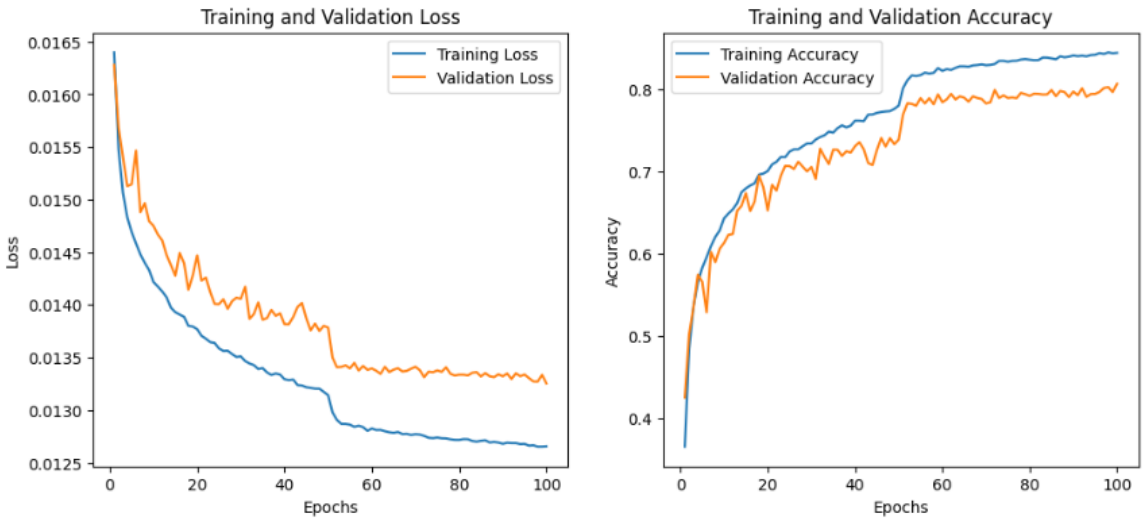
شکل ۹ – خلاصه مدل پیاده سازی شده

مدل را آموزش می دهیم و به نتایج زیر دست پیدا کردیم:

دقت نهایی بر روی test :

Test set: Average loss: 0.0130, Accuracy: 0.82

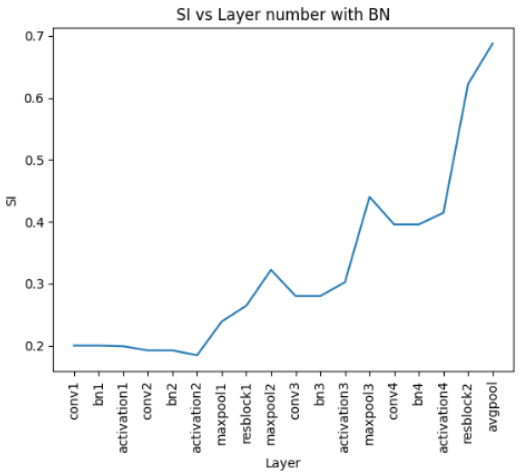
منحنی های دقت و loss:



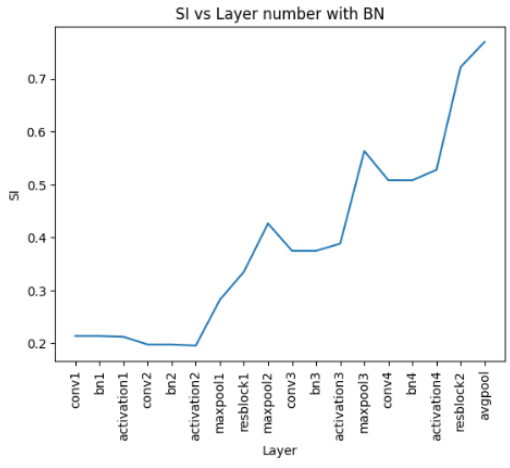
شکل ۱۰- منحني هاي دقت و loss برای مدل اول

نتایج متریک ها:

- منحنی SI بر روی تمامی لایه ها:



شکل ۱۱ - منحنی SI بر روی تمامی لایه ها برای داده های test



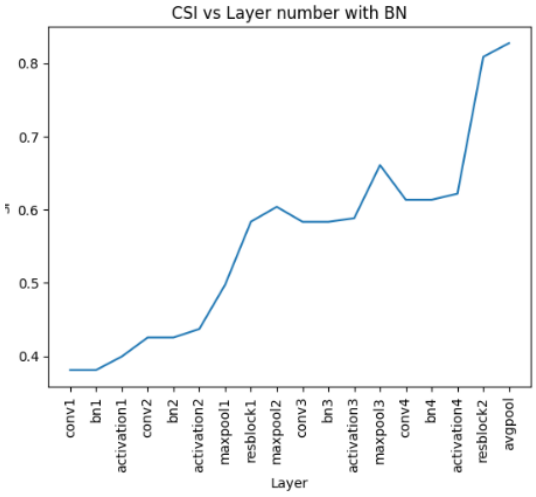
شکل ۱۲ -منحنی SI بر روی تمامی لایه ها برای داده های train

منحنی ها روند صعودی دارند. در conv3 باز مشاهده می کنیم که SI کاهش پیدا کرده است. ولی این کاهش کمتر از حالت قبل است.

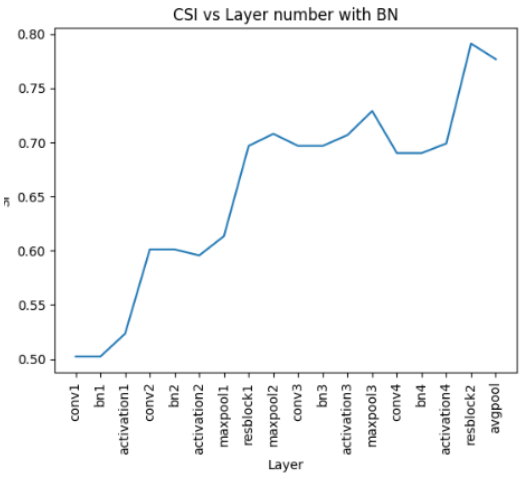
جدول ۳ - مقادیر si برای هر لایه برای داده های ترین و تست

si_train	si_test
[('conv1', 0.21382716298103333), ('bn1', 0.21382716298103333), ('activation1', 0.2123456746339798), ('conv2', 0.19777777791023254), ('bn2', 0.19777777791023254), ('activation2', 0.1958024650812149), ('maxpool1', 0.2822222113609314), ('resblock1', 0.33530864119529724), ('maxpool2', 0.42641976475715637), ('conv3', 0.37481480836868286), ('bn3', 0.37481480836868286), ('activation3', 0.3883950710296631), ('maxpool3', 0.5634567737579346), ('conv4', 0.5081481337547302), ('bn4', 0.5081481337547302), ('activation4', 0.5279012322425842), ('resblock2', 0.7219753265380859), ('avgpool', 0.7696296572685242)]	[('conv1', 0.20000000298023224), ('bn1', 0.20000000298023224), ('activation1', 0.198888897895813), ('conv2', 0.1922222226858139), ('bn2', 0.1922222226858139), ('activation2', 0.18444444239139557), ('maxpool1', 0.23888888955116272), ('resblock1', 0.2644444406032562), ('maxpool2', 0.3222222328186035), ('conv3', 0.2800000011920929), ('bn3', 0.2800000011920929), ('activation3', 0.30222222208976746), ('maxpool3', 0.4399999976158142), ('conv4', 0.3955555558204651), ('bn4', 0.3955555558204651), ('activation4', 0.4144444465637207), ('resblock2', 0.6222222447395325), ('avgpool', 0.6877778172492981)])]

- منحنی CSI بر روی تمامی لایه ها:



شکل ۱۳ -منحنی CSI بر روی تمامی لایه ها برای داده های train



شکل ۱۴ - منحنی CSI بر روی تمامی لایه ها برای داده های test

جدول ٤ – مقادیر csi برای هر لایه برای داده های ترین و تست

csi_train	csi_test
[('conv1', 0.38098764419555664), ('bn1', 0.38098764419555664), ('activation1', 0.39950618147850037), ('conv2', 0.42543208599090576), ('bn2', 0.42543208599090576), ('activation2', 0.43703705072402954), ('maxpool1', 0.49753087759017944), ('resblock1', 0.5837036967277527), ('maxpool2', 0.6039506196975708), ('conv3', 0.583456814289093), ('bn3', 0.583456814289093), ('activation3', 0.5883950591087341), ('maxpool3', 0.6609876751899719), ('conv4', 0.6135802268981934), ('bn4', 0.6135802268981934), ('activation4', 0.621975302696228), ('resblock2', 0.8088889122009277), ('avgpool', 0.8276543021202087)]	[('conv1', 0.5022222399711609), ('bn1', 0.5022222399711609), ('activation1', 0.5233333706855774), ('conv2', 0.601111114025116), ('bn2', 0.601111114025116), ('activation2', 0.5955555438995361), ('maxpool1', 0.6133333444595337), ('resblock1', 0.6966666579246521), ('maxpool2', 0.7077777981758118), ('conv3', 0.6966666579246521), ('bn3', 0.6966666579246521), ('activation3', 0.7066667079925537), ('maxpool3', 0.7288889288902283), ('conv4', 0.6899999976158142), ('bn4', 0.6899999976158142), ('activation4', 0.698888897895813), ('resblock2', 0.7911111116409302), ('avgpool', 0.7766667008399963)]

۲. پیاده سازی group normalization:

```
# Define a ResNet block with Group normalization
class MyResNetBlock(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(MyResNetBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3,
padding=1)
        self.bn1 = nn.GroupNorm(4, out_channels)  # Group normalization with 4
groups
        self.activation1 = nn.ELU()
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3,
padding=1)
        self.bn2 = nn.GroupNorm(4, out_channels)  # Group normalization with 4
groups

    def forward(self, x):
        shortcut = x
        x = self.conv1(x)
        x = self.bn1(x)
```

```
        x = self.activation1(x)
        x = self.conv2(x)
        x = self.bn2(x)
        x += shortcut
        x = self.activation1(x)
        return x

# Define the main ResNet model with Group normalization
class MyResNet(nn.Module):
    def __init__(self):
        super(MyResNet, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, padding=1)
        self.bn1 = nn.GroupNorm(32, 32)  # Group normalization with 4 groups
        self.activation1 = nn.ELU()
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.bn2 = nn.GroupNorm(32, 64)  # Group normalization with 4 groups
        self.activation2 = nn.ELU()
        self.maxpool1 = nn.MaxPool2d(kernel_size=2)

        self.resblock1 = MyResNetBlock(64, 64)
        self.maxpool2 = nn.MaxPool2d(kernel_size=2)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, padding=1)
        self.bn3 = nn.GroupNorm(32, 128)  # Group normalization with 4 groups
        self.activation3 = nn.ELU()
        self.maxpool3 = nn.MaxPool2d(kernel_size=2)
        self.conv4 = nn.Conv2d(128, 256, kernel_size=3, padding=1)
        self.bn4 = nn.GroupNorm(32, 256)  # Group normalization with 4 groups
        self.activation4 = nn.ELU()
        self.resblock2 = MyResNetBlock(256, 256)

        self.avgpool = nn.AvgPool2d(kernel_size=3, stride=2)
        self.flatten = nn.Flatten()
        self.dense1 = nn.Linear(256, 256)
        self.dense2 = nn.Linear(256, 10)
        self.softmax = nn.Softmax(dim=1)

    def forward(self, x):
        x = self.conv1(x)
        x = self.bn1(x)
        x = self.activation1(x)
        x = self.conv2(x)
        x = self.bn2(x)
        x = self.activation2(x)
        x = self.maxpool1(x)

        x = self.resblock1(x)
        x = self.conv3(x)
        x = self.bn3(x)
```

```
        x = self.activation3(x)
        x = self.maxpool2(x)

        x = self.conv4(x)
        x = self.bn4(x)
        x = self.activation4(x)
        x = self.maxpool3(x)
        x = self.resblock2(x)
        x = self.avgpool(x)
        x = self.flatten(x)
        x = self.dense1(x)
        x = self.dense2(x)
        x = self.softmax(x)
        return x
```

این کد نوعی از معماری ResNet را برای طبقه‌بندی تصاویر را تعریف می‌کند و به جای Normalization دسته‌ای، از نرمال سازی گروهی GN استفاده می‌کند. این معماری از اصول ResNet پیروی می کند و از بلوک های باقیمانده MyResNetBlock و لایه های کانولوشنال استفاده می کند. GN بعد از هر لایه کانولوشن و در داخل بلوک های باقیمانده اعمال می شود. تعداد گروه ها در GN برای هر نمونه روی ۴ تنظیم شده است، به این معنی که کانال ها برای نرمال سازی به ۴ گروه تقسیم می شوند. مدل اصلی MyResNet با لایه های کانولوشن اولیه، GN، فعال سازی ELU و max-pooling شروع می شود. سپس از دو نمونه از بلوک ResNet تعریف شده MyResNetBlock استفاده می کند که هر کدام با لایه های GN، convolution، activation و max-pooling دنبال می شوند. این مدل با average pooling، flattern کردن و لایه‌های کاملاً متصل به پایان می‌رسد و احتمالات کلاس را از طریق یک تابع فعال‌سازی softmax تولید می‌کند. نرمال سازی گروهی به عنوان جایگزینی برای نرمال سازی دسته ای معرفی شده است، که ثبات را در طول آموزش ارائه می دهد، به ویژه در هنگام سر و کار با اندازه های دسته کوچکتر قابل اجرا است.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 32, 32, 32]             896
       GroupNorm-2          [-1, 32, 32, 32]              64
             ELU-3          [-1, 32, 32, 32]               0
            Conv2d-4          [-1, 64, 32, 32]          18,496
       GroupNorm-5          [-1, 64, 32, 32]             128
             ELU-6          [-1, 64, 32, 32]               0
         MaxPool2d-7          [-1, 64, 16, 16]               0
            Conv2d-8          [-1, 64, 16, 16]          36,928
       GroupNorm-9          [-1, 64, 16, 16]             128
            ELU-10          [-1, 64, 16, 16]               0
           Conv2d-11          [-1, 64, 16, 16]          36,928
      GroupNorm-12          [-1, 64, 16, 16]             128
            ELU-13          [-1, 64, 16, 16]               0
  MyResNetBlock-14          [-1, 64, 16, 16]               0
           Conv2d-15         [-1, 128, 16, 16]          73,856
      GroupNorm-16         [-1, 128, 16, 16]             256
            ELU-17         [-1, 128, 16, 16]               0
        MaxPool2d-18           [-1, 128, 8, 8]               0
           Conv2d-19           [-1, 256, 8, 8]         295,168
      GroupNorm-20           [-1, 256, 8, 8]             512
            ELU-21           [-1, 256, 8, 8]               0
        MaxPool2d-22           [-1, 256, 4, 4]               0
           Conv2d-23           [-1, 256, 4, 4]         590,080
      GroupNorm-24           [-1, 256, 4, 4]             512
            ELU-25           [-1, 256, 4, 4]               0
           Conv2d-26           [-1, 256, 4, 4]         590,080
      GroupNorm-27           [-1, 256, 4, 4]             512
            ELU-28           [-1, 256, 4, 4]               0
  MyResNetBlock-29           [-1, 256, 4, 4]               0
        AvgPool2d-30           [-1, 256, 1, 1]               0
         Flatten-31                 [-1, 256]               0
          Linear-32                 [-1, 256]          65,792
          Linear-33                  [-1, 10]           2,570
         Softmax-34                  [-1, 10]               0
================================================================
Total params: 1,713,034
Trainable params: 1,713,034
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 4.69
Params size (MB): 6.53
Estimated Total Size (MB): 11.24
----------------------------------------------------------------
```

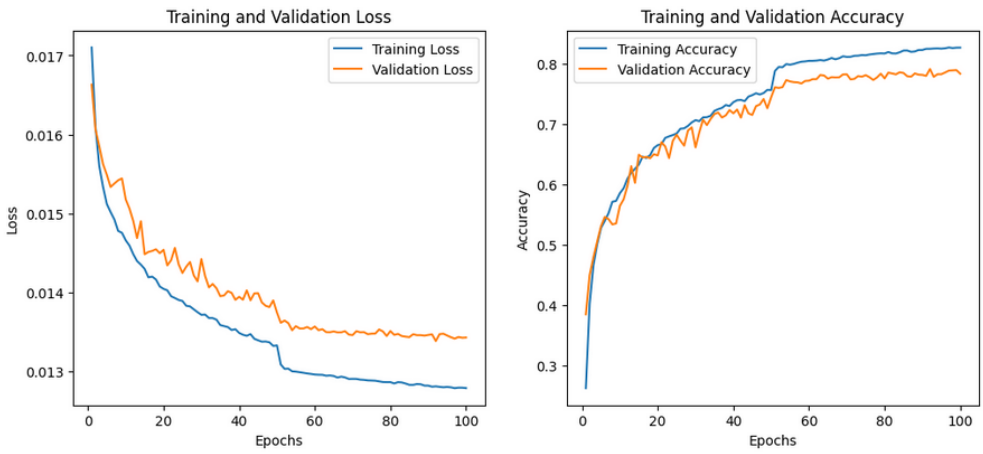
شکل ۱۵ – خلاصه مدل پیاده سازی شده در این بخش

مدل را آموزش می دهیم و به نتایج زیر دست پیدا کردیم:

دقت نهایی بر روی test :

Test set: Average loss: 0.0132, Accuracy: 0.80

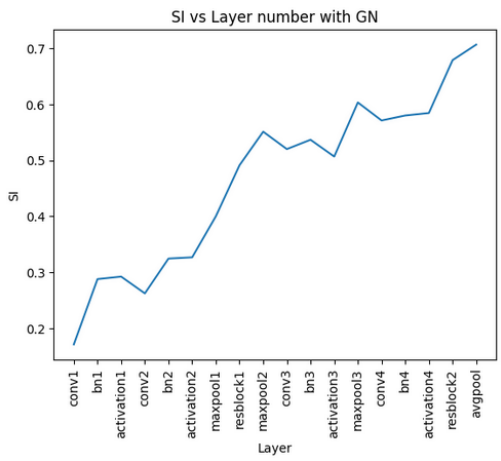
منحنی های دقت و loss:



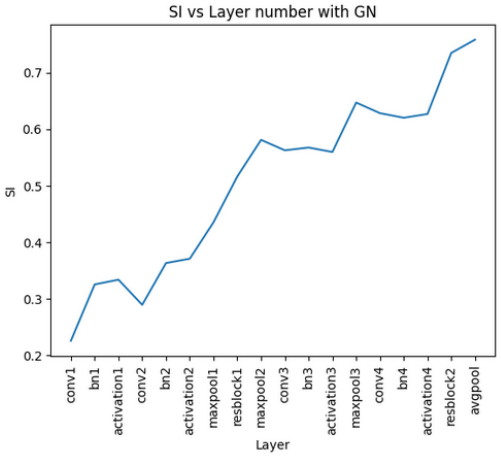
شکل ۱۶- منحني هاي دقت و loss برای مدل اول

نتایج متریک ها:

- منحنی SI بر روی تمامی لایه ها:



شکل ۱۷ - منحنی SI بر روی تمامی لایه ها برای داده های test



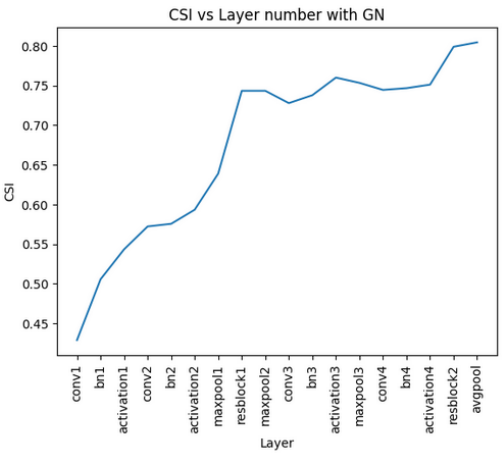
شکل ۱۸ -منحنی SI بر روی تمامی لایه ها برای داده های train

منحنی ها روند صعودی دارند. در conv3 باز مشاهده می کنیم که SI کاهش پیدا کرده است. ولی این کاهش کمتر از حالت قبل است.

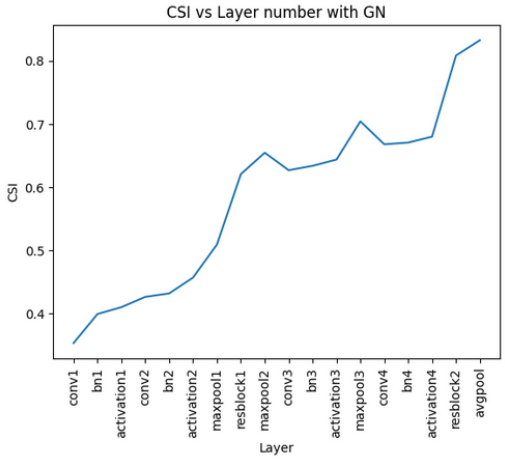
جدول ۵ - مقادیر si برای هر لایه برای داده های ترین و تست

si_train	si_test
[('conv1', 0.22592592239379883), ('bn1', 0.32543209195137024), ('activation1', 0.3338271677494049), ('conv2', 0.2893827259540558), ('bn2', 0.36320987343788147), ('activation2', 0.3708641827106476), ('maxpool1', 0.43530863523483276), ('resblock1', 0.5162962675094604), ('maxpool2', 0.5809876322746277), ('conv3', 0.5624691247940063), ('bn3', 0.5674074292182922), ('activation3', 0.5595061779022217), ('maxpool3', 0.6469135880470276), ('conv4', 0.6281481385231018), ('bn4', 0.6200000047683716), ('activation4', 0.6266666650772095), ('resblock2', 0.7345678806304932), ('avgpool', 0.7580246925354004)]	[('conv1', 0.1711111217737198), ('bn1', 0.28777778148651123), ('activation1', 0.2922222316265106), ('conv2', 0.2622222304344177), ('bn2', 0.324444442987442), ('activation2', 0.3266666829586029), ('maxpool1', 0.4000000059604645), ('resblock1', 0.4911111295223236), ('maxpool2', 0.551111102104187), ('conv3', 0.5200000405311584), ('bn3', 0.5366666913032532), ('activation3', 0.5066666603088379), ('maxpool3', 0.6033333539962769), ('conv4', 0.5711111426353455), ('bn4', 0.5800000429153442), ('activation4', 0.5844444632530212), ('resblock2', 0.6788889169692993), ('avgpool', 0.7066667079925537)]

- منحنی CSI بر روی تمامی لایه ها:



شکل ۲۰ - منحنی CSI بر روی تمامی لایه ها برای داده های test



شکل ۱۹ -منحنی CSI بر روی تمامی لایه ها برای داده های train

جدول ٦ – مقادیر csi برای هر لایه برای داده های ترین و تست

csi_train	csi_test
[('conv1', 0.35333332419395447), ('bn1', 0.3992592692375183), ('activation1', 0.41012346744537354), ('conv2', 0.42641976475715637), ('bn2', 0.431851863861084), ('activation2', 0.4570370316505432), ('maxpool1', 0.5093827247619629), ('resblock1', 0.6207407116889954), ('maxpool2', 0.6545678973197937), ('conv3', 0.6269136071205139), ('bn3', 0.6338271498680115), ('activation3', 0.6437036991119385), ('maxpool3', 0.7041975259780884), ('conv4', 0.6679012179374695), ('bn4', 0.6706172823905945), ('activation4', 0.6800000071525574), ('resblock2', 0.8083950877189636), ('avgpool', 0.8325926065444946)]	[('conv1', 0.42888888716697693), ('bn1', 0.5055555701255798), ('activation1', 0.5433333516120911), ('conv2', 0.5722222328186035), ('bn2', 0.5755555629730225), ('activation2', 0.59333336353302), ('maxpool1', 0.6388888955116272), ('resblock1', 0.7433333396911621), ('maxpool2', 0.7433333396911621), ('conv3', 0.7277777791023254), ('bn3', 0.7377777695655823), ('activation3', 0.7599999904632568), ('maxpool3', 0.753333330154419), ('conv4', 0.7444444894790649), ('bn4', 0.746666669845581), ('activation4', 0.7511111497879028), ('resblock2', 0.7988889217376709), ('avgpool', 0.8044444918632507)]

۳. پیاده سازی layer normalization:

```
class MyResNetBlock(nn.Module):
    def __init__(self, in_channels, out_channels ,size):
        super(MyResNetBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3,
padding=1)
        self.norm1 = nn.LayerNorm([out_channels,size,size])  # Add Layer
Normalization
        self.activation1 = nn.ELU()
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3,
padding=1)
        self.norm2 = nn.LayerNorm([out_channels,size,size])  # Add Layer
Normalization

    def forward(self, x):
        shortcut = x
        x = self.conv1(x)
        x = self.norm1(x)  # Apply Layer Normalization
        x = self.activation1(x)
        x = self.conv2(x)
        x = self.norm2(x)  # Apply Layer Normalization
        x += shortcut
        x = self.activation1(x)
        return x

class MyResNet(nn.Module):
    def __init__(self):
        super(MyResNet, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, padding=1)
        self.norm1 = nn.LayerNorm([32, 32, 32])  # Add Layer Normalization
        self.activation1 = nn.ELU()
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.norm2 = nn.LayerNorm([64, 32, 32])  # Add Layer Normalization
        self.activation2 = nn.ELU()
        self.maxpool1 = nn.MaxPool2d(kernel_size=2)

        self.resblock1 = MyResNetBlock(64, 64, 16)
        self.maxpool2 = nn.MaxPool2d(kernel_size=2)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, padding=1)
        self.norm3 = nn.LayerNorm([128, 16, 16])  # Add Layer Normalization
        self.activation3 = nn.ELU()
        self.maxpool3 = nn.MaxPool2d(kernel_size=2)
        self.conv4 = nn.Conv2d(128, 256, kernel_size=3, padding=1)
        self.norm4 = nn.LayerNorm([256, 8, 8])  # Add Layer Normalization
        self.activation4 = nn.ELU()
        self.resblock2 = MyResNetBlock(256, 256 ,4)
```

```
        self.avgpool = nn.AvgPool2d(kernel_size=3, stride=2)
        self.flatten = nn.Flatten()
        self.dense1 = nn.Linear(256, 256)
        self.dense2 = nn.Linear(256, 10)
        self.softmax = nn.Softmax(dim=1)

    def forward(self, x):
        x = self.conv1(x)
        x = self.norm1(x)  # Apply Layer Normalization
        x = self.activation1(x)
        x = self.conv2(x)
        x = self.norm2(x)  # Apply Layer Normalization
        x = self.activation2(x)
        x = self.maxpool1(x)

        x = self.resblock1(x)
        x = self.conv3(x)
        x = self.norm3(x)  # Apply Layer Normalization
        x = self.activation3(x)
        x = self.maxpool2(x)

        x = self.conv4(x)
        x = self.norm4(x)  # Apply Layer Normalization
        x = self.activation4(x)
        x = self.maxpool3(x)
        x = self.resblock2(x)
        x = self.avgpool(x)
        x = self.flatten(x)
        x = self.dense1(x)
        x = self.dense2(x)
        x = self.softmax(x)
        return x
```

این کد به جای Normalization دسته ای از نرمال سازی لایه LN استفاده می کند. اجزای اصلی شامل کلاس MyResNetBlock است که نشان دهنده یک بلوک residual با دو لایه کانولوشن، توابع فعال سازی ELU و نرمال سازی لایه است که بعد از هر لایه کانولوشن اعمال می شود. کلاس MyResNet مدل اصلی را تشکیل می‌دهد که دارای لایه‌های کانولوشنال اولیه با Normalization لایه و فعال‌سازی ELU و به دنبال آن max-pooling است. سپس این مدل از دو نمونه از بلوک ResNet تعریف شده MyResNetBlock استفاده می کند، که هر کدام از لایه های کانولوشن، نرمال سازی لایه، فعال سازی و max-pooling را دنبال می کند. این مدل با average pooling، flattern کردن و لایه‌های کاملاً متصل به پایان می رسد. نرمال سازی لایه برای نرمال سازی فعال سازی ها، ایجاد ثبات در

طول آموزش و ارائه جایگزینی برای نرمال سازی دسته ای استفاده می شود. استفاده از نرمال سازی لایه به کاهش مسائل مربوط به تغییر متغیر داخلی کمک می کند و آموزش شبکه های عصبی عمیق را برای طبقه بندی تصاویر امکان پذیر می کند.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 32, 32, 32]             896
       LayerNorm-2          [-1, 32, 32, 32]          65,536
             ELU-3          [-1, 32, 32, 32]               0
            Conv2d-4          [-1, 64, 32, 32]          18,496
       LayerNorm-5          [-1, 64, 32, 32]         131,072
             ELU-6          [-1, 64, 32, 32]               0
       MaxPool2d-7          [-1, 64, 16, 16]               0
            Conv2d-8          [-1, 64, 16, 16]          36,928
       LayerNorm-9          [-1, 64, 16, 16]          32,768
             ELU-10          [-1, 64, 16, 16]               0
           Conv2d-11          [-1, 64, 16, 16]          36,928
      LayerNorm-12          [-1, 64, 16, 16]          32,768
             ELU-13          [-1, 64, 16, 16]               0
  MyResNetBlock-14          [-1, 64, 16, 16]               0
           Conv2d-15         [-1, 128, 16, 16]          73,856
      LayerNorm-16         [-1, 128, 16, 16]          65,536
             ELU-17         [-1, 128, 16, 16]               0
      MaxPool2d-18           [-1, 128, 8, 8]               0
           Conv2d-19           [-1, 256, 8, 8]         295,168
      LayerNorm-20           [-1, 256, 8, 8]          32,768
             ELU-21           [-1, 256, 8, 8]               0
      MaxPool2d-22           [-1, 256, 4, 4]               0
           Conv2d-23           [-1, 256, 4, 4]         590,080
      LayerNorm-24           [-1, 256, 4, 4]           8,192
             ELU-25           [-1, 256, 4, 4]               0
           Conv2d-26           [-1, 256, 4, 4]         590,080
      LayerNorm-27           [-1, 256, 4, 4]           8,192
             ELU-28           [-1, 256, 4, 4]               0
  MyResNetBlock-29           [-1, 256, 4, 4]               0
      AvgPool2d-30           [-1, 256, 1, 1]               0
         Flatten-31                 [-1, 256]               0
          Linear-32                 [-1, 256]          65,792
          Linear-33                  [-1, 10]           2,570
         Softmax-34                  [-1, 10]               0
================================================================
Total params: 2,087,626
Trainable params: 2,087,626
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 4.69
Params size (MB): 7.96
Estimated Total Size (MB): 12.67
----------------------------------------------------------------
```

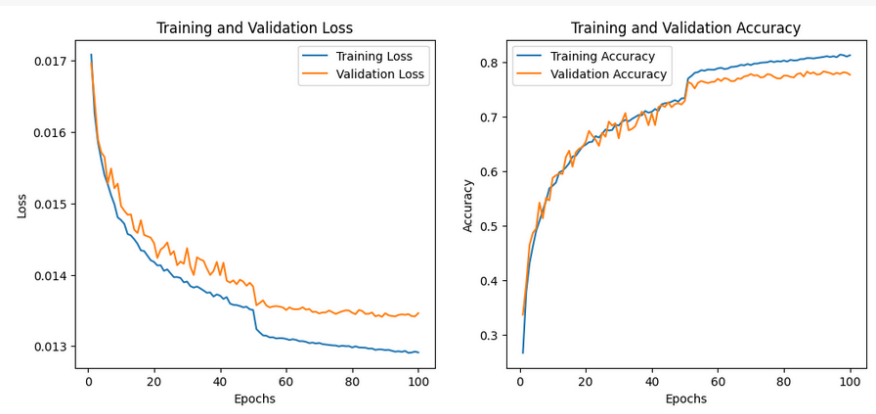
شکل ۲۱ – خلاصه ای از مدل پیاده سازی شده

مدل را آموزش می دهیم و به نتایج زیر دست پیدا کردیم:

دقت نهایی بر روی test :

Test set: Average loss: 0.0132, Accuracy: 0.78

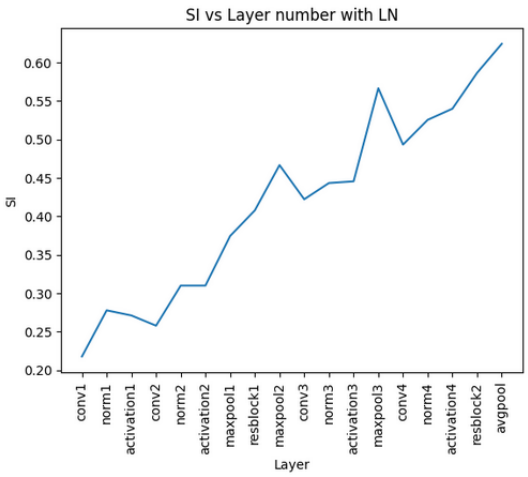
منحنی های دقت و loss:



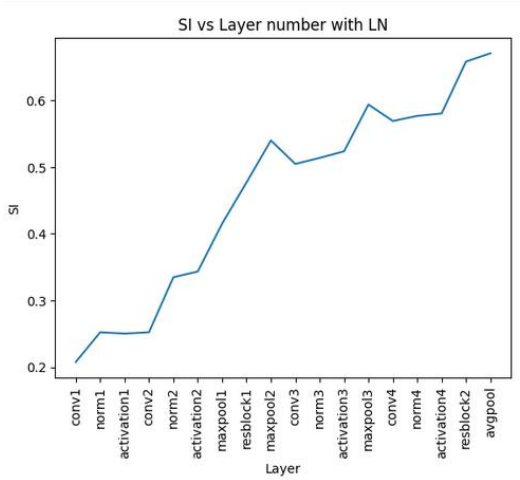
شکل ۲۲- منحني هاي دقت و loss برای مدل اول

نتایج متریک ها:

- منحنی SI بر روی تمامی لایه ها:



شکل ۲۳ - منحنی SI بر روی تمامی لایه ها برای داده های test



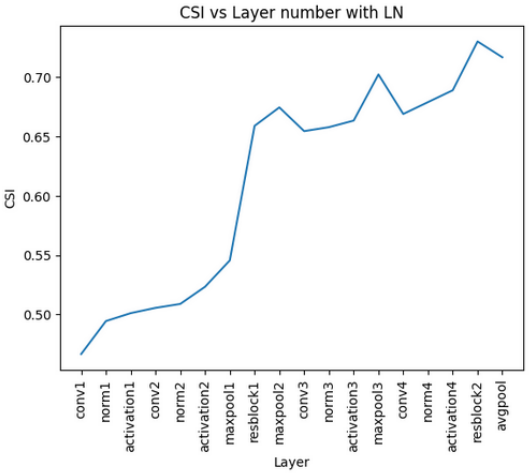
شکل ۲۴ -منحنی SI بر روی تمامی لایه ها برای داده های train

منحنی ها روند صعودی دارند. در conv3 باز مشاهده می کنیم که SI کاهش پیدا کرده است. ولی اسن کاهش کتر از حالت قبل است.

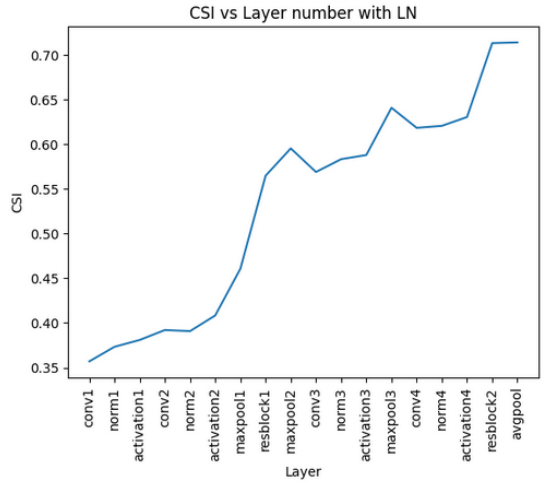
جدول ۷ - مقادیر si برای هر لایه برای داده های ترین و تست

si_test	si_train
[('conv1', 0.2177777886390686), ('norm1', 0.2777777910232544), ('activation1', 0.2711111307144165), ('conv2', 0.25777778029441833), ('norm2', 0.3100000023841858), ('activation2', 0.3100000023841858), ('maxpool1', 0.37444445490837097), ('resblock1', 0.4077777862548828), ('maxpool2', 0.46666666865348816), ('conv3', 0.42222222685813904), ('norm3', 0.44333335757255554), ('activation3', 0.44555556774139404), ('maxpool3', 0.5666666626930237), ('conv4', 0.4933333396911621), ('norm4', 0.5255555510520935), ('activation4', 0.5400000214576721), ('resblock2', 0.5866667032241821), ('avgpool', 0.6244444847106934)]	[('conv1', 0.2079012393951416), ('norm1', 0.2523456811904907), ('activation1', 0.2503703832626343), ('conv2', 0.2523456811904907), ('norm2', 0.33481481671333313), ('activation2', 0.34345677495002747), ('maxpool1', 0.4153086543083191), ('resblock1', 0.4770370423793793), ('maxpool2', 0.5400000214576721), ('conv3', 0.5046913623809814), ('norm3', 0.5138271450996399), ('activation3', 0.5237036943435669), ('maxpool3', 0.5938271880149841), ('conv4', 0.5691357851028442), ('norm4', 0.5767900943756104), ('activation4', 0.5804938077926636), ('resblock2', 0.6582716107368469), ('avgpool', 0.6706172823905945)]

- منحنی CSI بر روی تمامی لایه ها:



شکل ۲۶ - منحنی CSI بر روی تمامی لایه ها برای داده های test



شکل ۲۵ -منحنی CSI بر روی تمامی لایه ها برای داده های train

جدول ۸ – مقادیر csi برای هر لایه برای داده های ترین و تست

csi_train	csi_test
[('conv1', 0.3570370376110077), ('norm1', 0.3733333349227905), ('activation1', 0.38098764419555664), ('conv2', 0.3920987546443939), ('norm2', 0.39086419343948364), ('activation2', 0.40839505195617676), ('maxpool1', 0.46074074506759644), ('resblock1', 0.5646913647651672), ('maxpool2', 0.5953086614608765), ('conv3', 0.5688889026641846), ('norm3', 0.5832098722457886), ('activation3', 0.58790123462677), ('maxpool3', 0.6407407522201538), ('conv4', 0.6182715892791748), ('norm4', 0.6204938292503357), ('activation4', 0.6303703784942627), ('resblock2', 0.7130864262580872), ('avgpool', 0.7138271331787109)]	[('conv1', 0.46666666865348816), ('norm1', 0.49444445967674255), ('activation1', 0.5011111497879028), ('conv2', 0.5055555701255798), ('norm2', 0.5088889002799988), ('activation2', 0.5233333706855774), ('maxpool1', 0.545555591583252), ('resblock1', 0.6588888764381409), ('maxpool2', 0.6744444370269775), ('conv3', 0.6544444561004639), ('norm3', 0.6577777862548828), ('activation3', 0.6633333563804626), ('maxpool3', 0.7022222280502319), ('conv4', 0.6688889265060425), ('norm4', 0.6788889169692993), ('activation4', 0.6888889074325562), ('resblock2', 0.7300000190734863), ('avgpool', 0.7166666984558105)]

مقایسه این ۳ نوع normalization:

جدول ۹ - مقایسه با normalization هاي متفاوت

Model	Test acc	SI-train	SI-test	CSI-train	CSI-test
MyResnet	0.77	0.67	0.6	0.74	0.7
BN	0.82	0.76	0.68	0.82	0.77
GN	0.8	0.75	0.7	0.83	0.8
LN	0.78	0.67	0.62	0.713	0.716

با توجه به جدول بالا متوجه میشویم که با افزودن لایه های normalization دقت افزایش پیادا میکند همچنین شاخص ها نیز مقدار آن ها بیشتر می شود. از بین لایه های normalization مشاهده می کنیم که BN در اینجا به دقت بالاتری رسیده است. اگر augmentationهای دیگری استفاده می کردیم احتمالا به نتایج دیگری دست پیدا میکردیم.

د)

در کد قسمت الف کافیست که قسمت add را حذف کنیم و به کد زیر که بدون skip connection هست میرسیم.

```
# Define a ResNet block
class MyResNetBlock(nn.Module):
    # Constructor with input and output channel parameters
    def __init__(self, in_channels, out_channels):
        super(MyResNetBlock, self).__init__()
        # First convolutional layer
        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3,
padding=1)
        # ELU activation function
        self.activation1 = nn.ELU()
        # Second convolutional layer
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3,
padding=1)

    # Forward method for the block
    def forward(self, x):
        x = self.conv1(x)
        x = self.activation1(x)
```

```
        x = self.conv2(x)
        x = self.activation1(x)
        return x

# Define the main ResNet model
class MyResNet(nn.Module):
    # Constructor
    def __init__(self):
        super(MyResNet, self).__init__()
        # Initial convolutional layers
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, padding=1)
        self.activation1 = nn.ELU()
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.activation2 = nn.ELU()
        self.maxpool1 = nn.MaxPool2d(kernel_size=2)

        # Residual blocks
        self.resblock1 = MyResNetBlock(64, 64)
        self.maxpool2 = nn.MaxPool2d(kernel_size=2)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, padding=1)
        self.activation3 = nn.ELU()
        self.maxpool3 = nn.MaxPool2d(kernel_size=2)
        self.conv4 = nn.Conv2d(128, 256, kernel_size=3, padding=1)
        self.activation4 = nn.ELU()
        self.resblock2 = MyResNetBlock(256, 256)

        # Average pooling and fully connected layers
        self.avgpool = nn.AvgPool2d(kernel_size=3, stride=2)
        self.flatten = nn.Flatten()
        self.dense1 = nn.Linear(256, 256)
        self.dense2 = nn.Linear(256, 10)
        self.softmax = nn.Softmax(dim=1)

    # Forward method for the model
    def forward(self, x):
        x = self.conv1(x)
        x = self.activation1(x)
        x = self.conv2(x)
        x = self.activation2(x)
        x = self.maxpool1(x)

        x = self.resblock1(x)
        x = self.conv3(x)
        x = self.activation3(x)
        x = self.maxpool2(x)

        x = self.conv4(x)
        x = self.activation4(x)
```

```
        x = self.maxpool3(x)
        x = self.resblock2(x)
        x = self.avgpool(x)
        x = self.flatten(x)
        x = self.dense1(x)
        x = self.dense2(x)
        x = self.softmax(x)
        return x
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 32, 32, 32]             896
               ELU-2          [-1, 32, 32, 32]               0
            Conv2d-3          [-1, 64, 32, 32]          18,496
               ELU-4          [-1, 64, 32, 32]               0
         MaxPool2d-5          [-1, 64, 16, 16]               0
            Conv2d-6          [-1, 64, 16, 16]          36,928
               ELU-7          [-1, 64, 16, 16]               0
            Conv2d-8          [-1, 64, 16, 16]          36,928
               ELU-9          [-1, 64, 16, 16]               0
    MyResNetBlock-10          [-1, 64, 16, 16]               0
           Conv2d-11         [-1, 128, 16, 16]          73,856
              ELU-12         [-1, 128, 16, 16]               0
        MaxPool2d-13           [-1, 128, 8, 8]               0
           Conv2d-14           [-1, 256, 8, 8]         295,168
              ELU-15           [-1, 256, 8, 8]               0
        MaxPool2d-16           [-1, 256, 4, 4]               0
           Conv2d-17           [-1, 256, 4, 4]         590,080
              ELU-18           [-1, 256, 4, 4]               0
           Conv2d-19           [-1, 256, 4, 4]         590,080
              ELU-20           [-1, 256, 4, 4]               0
    MyResNetBlock-21           [-1, 256, 4, 4]               0
        AvgPool2d-22           [-1, 256, 1, 1]               0
          Flatten-23                 [-1, 256]               0
           Linear-24                 [-1, 256]          65,792
           Linear-25                  [-1, 10]           2,570
          Softmax-26                  [-1, 10]               0
================================================================
Total params: 1,710,794
Trainable params: 1,710,794
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 3.26
Params size (MB): 6.53
Estimated Total Size (MB): 9.79
----------------------------------------------------------------
```

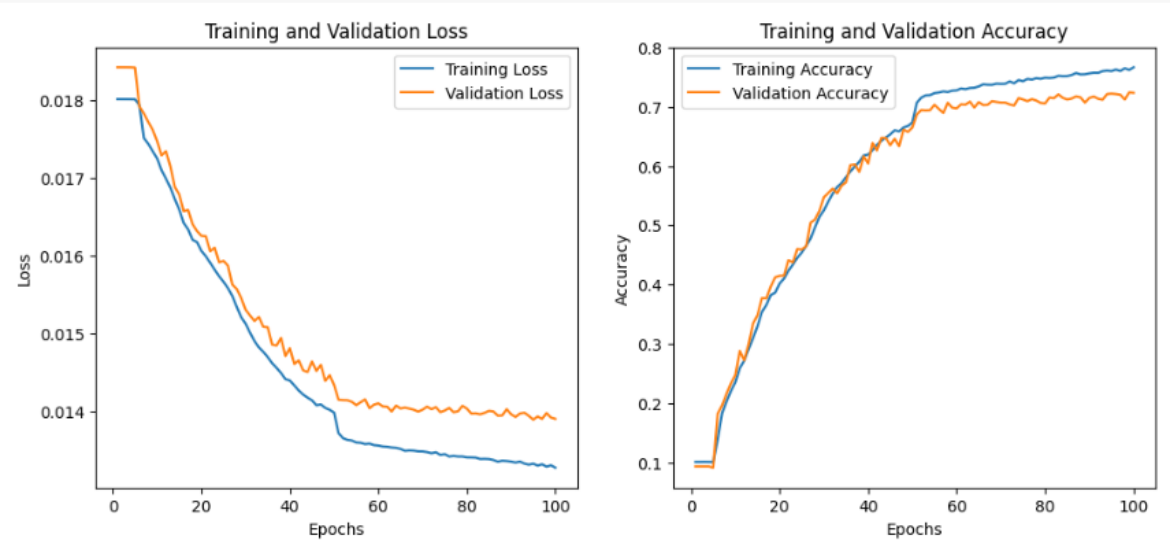
شکل ۲۷ - خلاصه مدل

مدل را آموزش می دهیم و به نتایج زیر دست پیدا کردیم:

دقت نهایی بر روی test :

Test set: Average loss: 0.0136, Accuracy: 0.73

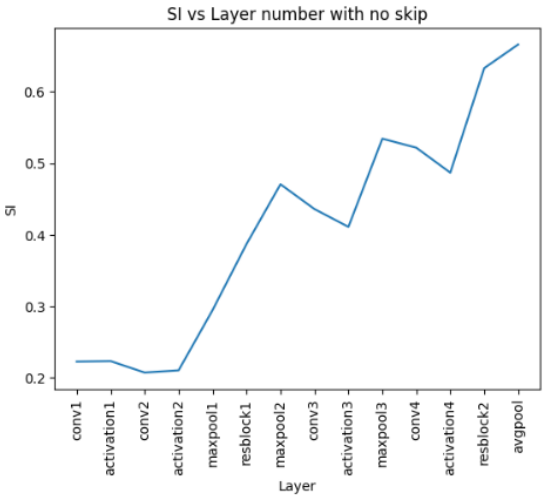
منحنی های دقت و loss:



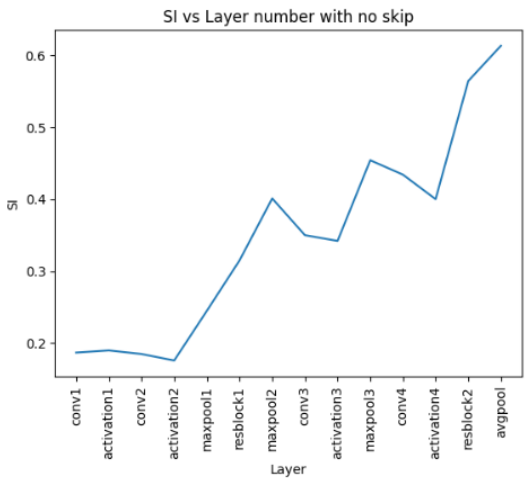
شکل ۲۸- منحني هاي دقت و loss برای مدل اول

نتایج متریک ها:

- منحنی SI بر روی تمامی لایه ها:



شکل ۳۰ -منحنی SI بر روی تمامی لایه ها برای داده های train



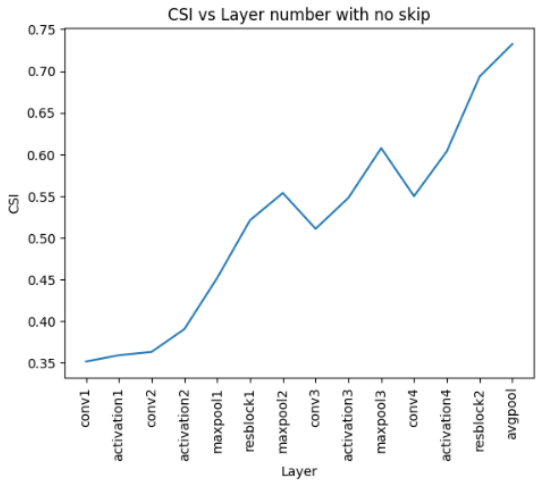
شکل ۲۹ - منحنی SI بر روی تمامی لایه ها برای داده های test

منحنی ها روند صعودی دارند. در conv3 باز مشاهده می کنیم که SI کاهش پیدا کرده است. ولی اسن کاهش کتر از حالت قبل است.

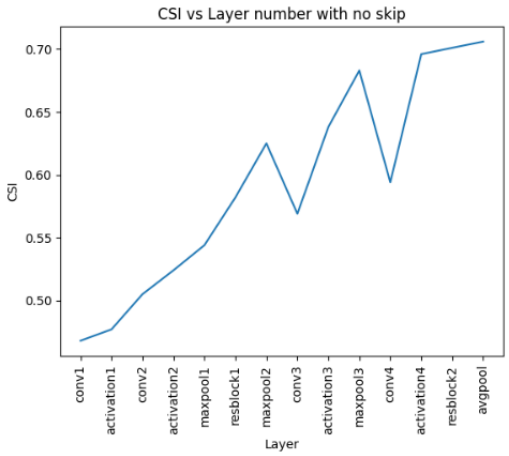
جدول ۱۰- مقادیر si برای هر لایه برای داده های ترین و تست

si_train	si_test
[('conv1', 0.2226666659116745), ('activation1', 0.22333332896232605), ('conv2', 0.20733334124088287), ('activation2', 0.21044445037841797), ('maxpool1', 0.29466667771339417), ('resblock1', 0.3871111273765564), ('maxpool2', 0.47066667675971985), ('conv3', 0.4360000193119049), ('activation3', 0.41111111640930176), ('maxpool3', 0.5344444513320923), ('conv4', 0.5217778086662292), ('activation4', 0.4868888854980469), ('resblock2', 0.6331111192703247), ('avgpool', 0.6662222146987915)]	[('conv1', 0.18700000643730164), ('activation1', 0.1900000125169754), ('conv2', 0.1850000023841858), ('activation2', 0.17600001394748688), ('maxpool1', 0.24500000476837158), ('resblock1', 0.3150000274181366), ('maxpool2', 0.4010000228881836), ('conv3', 0.3500000238418579), ('activation3', 0.34200000762939453), ('maxpool3', 0.4540000259876251), ('conv4', 0.43400001525878906), ('activation4', 0.4000000059604645), ('resblock2', 0.5640000104904175), ('avgpool', 0.6130000352859497)]

- منحنی CSI بر روی تمامی لایه ها:



شکل ۳۱ -منحنی CSI بر روی تمامی لایه ها برای داده های train



شکل ۳۲ - منحنی CSI بر روی تمامی لایه ها برای داده های test

جدول ۱۱ – مقادیر csi برای هر لایه برای داده های ترین و تست

csi_train	csi_test
[('conv1', 0.3517777919769287), ('activation1', 0.359333336353302), ('conv2', 0.3633333444595337), ('activation2', 0.390666663646698), ('maxpool1', 0.4519999921321869),	[('conv1', 0.46800002455711365), ('activation1', 0.47700002789497375), ('conv2', 0.5049999952316284), ('activation2', 0.5240000486373901), ('maxpool1', 0.5440000295639038),

```
('resblock1', 0.5211111307144165),
('maxpool2', 0.5537777543067932),
('conv3', 0.5108888745307922),
('activation3', 0.5479999780654907),
('maxpool3', 0.6075555682182312),
('conv4', 0.550000011920929),
('activation4', 0.6037777662277222),
('resblock2', 0.6933333277702332),
('avgpool', 0.7324444651603699)]
```

```
('resblock1', 0.5820000171661377),
('maxpool2', 0.625),
('conv3', 0.5690000057220459),
('activation3', 0.6380000114440918),
('maxpool3', 0.6830000281333923),
('conv4', 0.5940000414848328),
('activation4', 0.6960000395774841),
('resblock2', 0.7010000348091125),
('avgpool', 0.706000030040741)]
```

جدول ۱۲ – مقایسه مدل با و بدون skip connection

Model	Test acc	SI-train	SI-test	CSI-train	CSI-test
MyResnet	0.77	0.67	0.6	0.74	0.7
No skip	0.73	0.66	0.61	0.73	0.7

مشاهده می کنیم که با حذف skip connection دقت مدل با کاهش چشمگیری مواجه هست و مقادیر SI هم کاهش پیدا کرده است و نشان می دهد که مدل توانایی جداکردن را به خوبی ندارد.

د)

کانولوشن های ۱در ۱ معمولاً در معماری ResNet برای کاهش ابعاد و افزایش غیرخطی بودن استفاده می شوند. افزودن پیچش های ۱x۱ می تواند به بهبود ظرفیت مدل بدون افزایش قابل توجه هزینه محاسباتی کمک کند.

افزودن لایه‌های کانولوشنال ۱×۱ بعد از لایه‌های max-pooling یا قبل از لایه‌های کاملاً متصل نهایی، اهداف خاصی را در معماری شبکه‌های عصبی عمیق، به ویژه در زمینه مدل‌هایی مانند ResNet انجام می‌دهد.

۱. بعد از Max-Pooling Layers:

- کاهش ابعاد: لایه‌های Max-pooling ابعاد فضایی feature maps را کاهش می‌دهند و مهم‌ترین اطلاعات را حفظ می‌کنند. پیروی از max-pooling با یک لایه کانولوشن ۱در ۱ می‌تواند به کاهش تعداد کانال‌ها کمک کند و در عین حال ویژگی‌های مهم را حفظ کند و نوعی کاهش ابعاد را ارائه دهد.

- افزایش غیرخطی بودن: معرفی غیرخطی پس از max-pooling می‌تواند به ثبت الگوهای پیچیده‌تر در نقشه‌های ویژگی downsampledشده کمک کند.

۲. قبل از آخرین لایه های کاملا متصل:

- Global Information Aggregation: پیچیدگی‌های ۱در ۱ قبل از آخرین لایه‌های کاملاً متصل اعمال می‌شوند تا بتوانند اطلاعات جهانی را از کل نقشه ویژگی جمع‌آوری کنند. این می تواند برای کارهایی که روابط فضایی بین ویژگی ها مهم است مفید باشد.

- کاهش پارامتر: کاهش تعداد کانال ها قبل از لایه های کاملا متصل به کنترل تعداد پارامترهای مدل کمک می کند که می تواند برای کارایی از نظر محاسبات و استفاده از حافظه بسیار مهم باشد.

در هر دو مورد، لایه‌های کانولوشنال ۱x۱ به‌عنوان شکلی از لایه گلوگاه عمل می‌کنند و بین ظرفیت مدل و راندمان محاسباتی تعادل ایجاد می‌کنند. این به مدل اجازه می دهد تا با حفظ بیانی، یک نمایش فشرده را بیاموزد.

ما در اینجا قبل از آخرین لایه های کاملا متصل لایه را جایگزین میکنیم. کد این قسمت به شکل زیر است:

```
# Define a ResNet block
class MyResNetBlock(nn.Module):
    # Constructor with input and output channel parameters
    def __init__(self, in_channels, out_channels):
        super(MyResNetBlock, self).__init__()
        # First convolutional layer
        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3,
padding=1)
        # ELU activation function
        self.activation1 = nn.ELU()
        # Second convolutional layer
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3,
padding=1)

    # Forward method for the block
    def forward(self, x):
        shortcut = x  # Preserve the input for the shortcut connection
        x = self.conv1(x)
        x = self.activation1(x)
        x = self.conv2(x)
        x += shortcut  # Add the input to the output (residual connection)
        x = self.activation1(x)
        return x
```

```
# Define the main ResNet model
class MyResNet(nn.Module):
    # Constructor
    def __init__(self):
        super(MyResNet, self).__init__()
        # Initial convolutional layers
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, padding=1)
        self.activation1 = nn.ELU()
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.activation2 = nn.ELU()
        self.maxpool1 = nn.MaxPool2d(kernel_size=2)

        # Residual blocks
        self.resblock1 = MyResNetBlock(64, 64)
        self.maxpool2 = nn.MaxPool2d(kernel_size=2)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, padding=1)
        self.activation3 = nn.ELU()
        self.maxpool3 = nn.MaxPool2d(kernel_size=2)
        self.conv1x1 = nn.Conv2d(128, 256, kernel_size=1, padding=1)  # 1x1
convolution
        self.activation4 = nn.ELU()
        self.resblock2 = MyResNetBlock(256, 256)

        # Average pooling and fully connected layers
        self.avgpool = nn.AvgPool2d(kernel_size=3, stride=2)
        self.flatten = nn.Flatten()
        self.dense1 = nn.Linear(1024, 256)
        self.dense2 = nn.Linear(256, 10)
        self.softmax = nn.Softmax(dim=1)

    # Forward method for the model
    def forward(self, x):
        x = self.conv1(x)
        x = self.activation1(x)
        x = self.conv2(x)
        x = self.activation2(x)
        x = self.maxpool1(x)

        x = self.resblock1(x)
        x = self.conv3(x)
        x = self.activation3(x)
        x = self.maxpool2(x)

        x = self.conv1x1(x)
        x = self.activation4(x)
        x = self.maxpool3(x)
        x = self.resblock2(x)
```

```
        x = self.avgpool(x)
        x = self.flatten(x)
        x = self.dense1(x)
        x = self.dense2(x)
        x = self.softmax(x)
        return x
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 32, 32, 32]             896
               ELU-2           [-1, 32, 32, 32]               0
            Conv2d-3           [-1, 64, 32, 32]          18,496
               ELU-4           [-1, 64, 32, 32]               0
         MaxPool2d-5           [-1, 64, 16, 16]               0
            Conv2d-6           [-1, 64, 16, 16]          36,928
               ELU-7           [-1, 64, 16, 16]               0
            Conv2d-8           [-1, 64, 16, 16]          36,928
               ELU-9           [-1, 64, 16, 16]               0
    MyResNetBlock-10           [-1, 64, 16, 16]               0
           Conv2d-11          [-1, 128, 16, 16]          73,856
              ELU-12          [-1, 128, 16, 16]               0
        MaxPool2d-13           [-1, 128, 8, 8]               0
           Conv2d-14          [-1, 256, 10, 10]          33,024
              ELU-15          [-1, 256, 10, 10]               0
        MaxPool2d-16            [-1, 256, 5, 5]               0
           Conv2d-17            [-1, 256, 5, 5]         590,080
              ELU-18            [-1, 256, 5, 5]               0
           Conv2d-19            [-1, 256, 5, 5]         590,080
              ELU-20            [-1, 256, 5, 5]               0
    MyResNetBlock-21            [-1, 256, 5, 5]               0
        AvgPool2d-22            [-1, 256, 2, 2]               0
          Flatten-23                 [-1, 1024]               0
           Linear-24                  [-1, 256]         262,400
           Linear-25                   [-1, 10]           2,570
          Softmax-26                   [-1, 10]               0
================================================================
Total params: 1,645,258
Trainable params: 1,645,258
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 3.51
Params size (MB): 6.28
Estimated Total Size (MB): 9.80
----------------------------------------------------------------
```

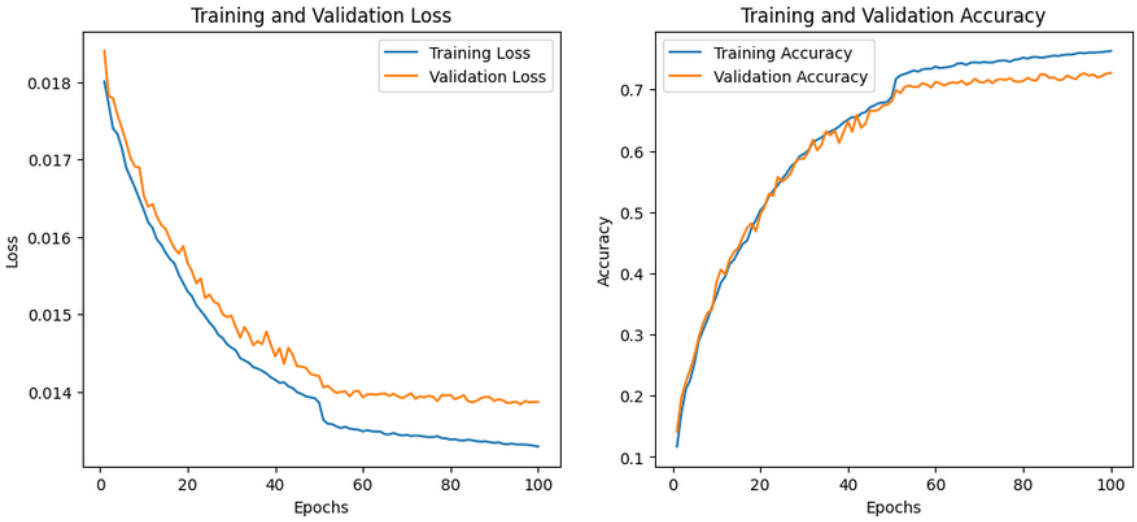
شکل ۳۳ - خلاصه مدل

مدل را آموزش می دهیم و به نتایج زیر دست پیدا کردیم:

دقت نهایی بر روی test :

Test set: Average loss: 0.0136, Accuracy: 0.73

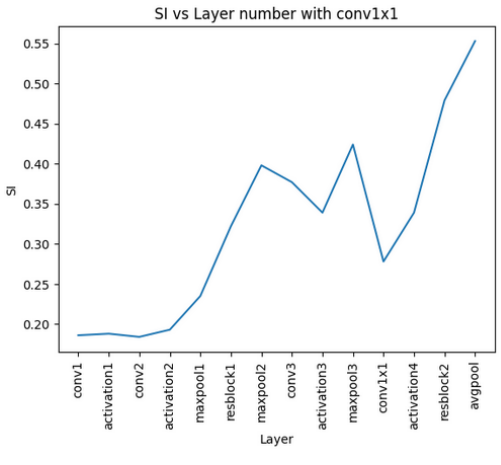
منحنی های دقت و loss:



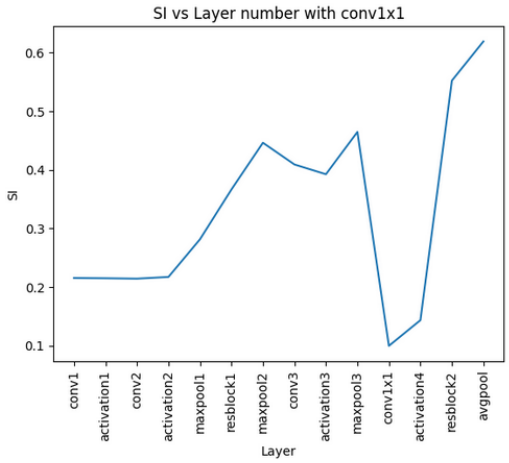
شکل ۳٤- منحني هاي دقت و loss برای مدل اول

نتایج متریک ها:

- منحنی SI بر روی تمامی لایه ها:



شکل ۳۵ - منحنی SI بر روی تمامی لایه ها برای داده های test



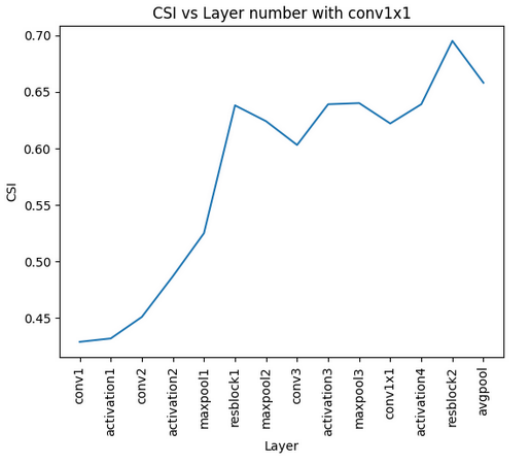
شکل ۳٦ -منحنی SI بر روی تمامی لایه ها برای داده های train

منحنی ها روند صعودی دارند. در conv3 باز مشاهده می کنیم که SI کاهش پیدا کرده است. ولی اسن کاهش کتر از حالت قبل است. ولی میبینیم که در cov1-1 که گذاشتیم si به شدت کاهش پیادا کرده است که مطلوب ما نیست.

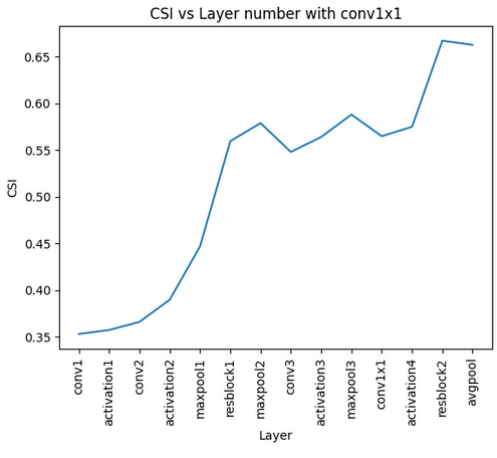
جدول ۱۳ - مقادیر si برای هر لایه برای داده های ترین و تست

si_train	si_test
[('conv1', 0.2155555635690689), ('activation1', 0.21511110663414001), ('conv2', 0.21444444358348846), ('activation2', 0.2173333317041397), ('maxpool1', 0.28155556321144104), ('resblock1', 0.36666667461395264), ('maxpool2', 0.44644445180892944), ('conv3', 0.40933334827423096), ('activation3', 0.39266666769981384), ('maxpool3', 0.46488890051841736), ('conv1x1', 0.09977778047323227), ('activation4', 0.14355555176734924), ('resblock2', 0.5524444580078125), ('avgpool', 0.6193333268165588)]	[('conv1', 0.1860000044107437), ('activation1', 0.18800000846385956), ('conv2', 0.18400001525878906), ('activation2', 0.19300000369548798), ('maxpool1', 0.23500001430511475), ('resblock1', 0.32200002670288086), ('maxpool2', 0.398000031709671), ('conv3', 0.37700000405311584), ('activation3', 0.33900001645088196), ('maxpool3', 0.4240000247955322), ('conv1x1', 0.27800002694129944), ('activation4', 0.33900001645088196), ('resblock2', 0.4790000319480896), ('avgpool', 0.5530000329017639)]

- منحنی CSI بر روی تمامی لایه ها:



شکل ۳۸ - منحنی CSI بر روی تمامی لایه ها برای داده های test



شکل ۳۷ -منحنی CSI بر روی تمامی لایه ها برای داده های train

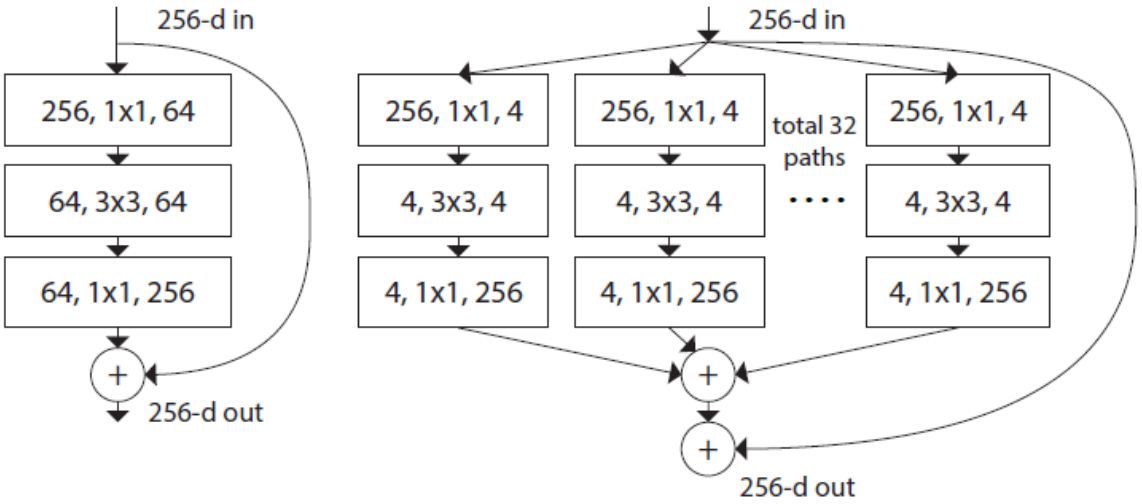
جدول ۱۴ – مقادیر csi برای هر لایه برای داده های ترین و تست

csi_train	csi_test
[('conv1', 0.3531111180782318), ('activation1', 0.35733333230018616), ('conv2', 0.3659999966621399), ('activation2', 0.3897777795791626),	[('conv1', 0.42900002002716064), ('activation1', 0.4320000112056732), ('conv2', 0.45100003480911255), ('activation2', 0.4870000183582306),

('maxpool1', 0.44688889384269714), ('resblock1', 0.5595555901527405), ('maxpool2', 0.5788888931274414), ('conv3', 0.5479999780654907), ('activation3', 0.5640000104904175), ('maxpool3', 0.5879999995231628), ('conv1x1', 0.5648888945579529), ('activation4', 0.5748888850212097), ('resblock2', 0.6671110987663269), ('avgpool', 0.6626666784286499)]	('maxpool1', 0.5250000357627869), ('resblock1', 0.6380000114440918), ('maxpool2', 0.6240000128746033), ('conv3', 0.6030000448226929), ('activation3', 0.6390000581741333), ('maxpool3', 0.64000004529953), ('conv1x1', 0.6220000386238098), ('activation4', 0.6390000581741333), ('resblock2', 0.6950000524520874), ('avgpool', 0.6580000519752502)]

در این حالت عملکرد ضعیفتری نسبت به حالت های قبل داریم. همچنین میتوانیم بعد از Max-Pooling Layers لایه را بگذاریم و نتیجه را با این حالت مقایسه کنیم.

و)



شکل ۳۹– معماری ResNext

برای path =2 به شکل زیر مدل را پیاده سازی میکنیم:

```
# Define a block
class GP_Block(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(GP_Block, self).__init__()
```

```
        # Path 1
        self.conv1_path1 = nn.Conv2d(in_channels, out_channels, kernel_size=3,
padding=1)
        self.activation = nn.ELU()
        self.conv2_path1 = nn.Conv2d(out_channels, out_channels, kernel_size=3,
padding=1)

        # Path 2
        self.conv1_path2 = nn.Conv2d(in_channels, out_channels, kernel_size=3,
padding=1)
        self.conv2_path2 = nn.Conv2d(out_channels, out_channels, kernel_size=3,
padding=1)

    def forward(self, x):
        shortcut = x  # Save the input as a shortcut for the residual connection

        # Path 1
        x1 = self.conv1_path1(x)
        x1 = self.activation(x1)
        x1 = self.conv2_path1(x1)

        # Path 2
        x2 = self.conv1_path2(x)
        x2 = self.activation(x2)
        x2 = self.conv2_path2(x2)

        # Combine paths (element-wise addition)
        x = x1 + x2

        # Final combination with the shortcut (residual connection)
        x += shortcut
        x = self.activation(x)  # Apply activation function to the final output

        return x

# Define the main ResNet model
class MyResNext(nn.Module):
    # Constructor
    def __init__(self):
        super(MyResNext, self).__init__()
        # Initial convolutional layers
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, padding=1)
        self.activation1 = nn.ELU()
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.activation2 = nn.ELU()
        self.maxpool1 = nn.MaxPool2d(kernel_size=2)
```

```
        # Residual blocks
        self.resblock1 = GP_Block(64, 64)
        self.maxpool2 = nn.MaxPool2d(kernel_size=2)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, padding=1)
        self.activation3 = nn.ELU()
        self.maxpool3 = nn.MaxPool2d(kernel_size=2)
        self.conv4 = nn.Conv2d(128, 256, kernel_size=3, padding=1)
        self.activation4 = nn.ELU()
        self.resblock2 = GP_Block(256, 256)

        # Average pooling and fully connected layers
        self.avgpool = nn.AvgPool2d(kernel_size=3, stride=2)
        self.flatten = nn.Flatten()
        self.dense1 = nn.Linear(256, 256)
        self.dense2 = nn.Linear(256, 10)
        self.softmax = nn.Softmax(dim=1)

    # Forward method for the model
    def forward(self, x):
        x = self.conv1(x)
        x = self.activation1(x)
        x = self.conv2(x)
        x = self.activation2(x)
        x = self.maxpool1(x)

        x = self.resblock1(x)
        x = self.conv3(x)
        x = self.activation3(x)
        x = self.maxpool2(x)

        x = self.conv4(x)
        x = self.activation4(x)
        x = self.maxpool3(x)
        x = self.resblock2(x)
        x = self.avgpool(x)
        x = self.flatten(x)
        x = self.dense1(x)
        x = self.dense2(x)
        x = self.softmax(x)
        return x
```

این کد یک معماری ResNeXt را تعریف می‌کند که به عنوان MyResNext برای طبقه‌بندی تصاویر نوشته شده است. بلوک مرکزی، GP_Block ، **دو مسیر** را در هر بلوک باقیمانده ترکیب می کند. هر مسیر از یک جفت لایه کانولوشن با توابع فعال سازی ELU تشکیل شده است که مدل را قادر می سازد تا ویژگی های سلسله مراتبی پیچیده را ثبت کند. این دو مسیر مسیرهای استخراج ویژگی های متنوعی را

نشان می دهند که توانایی مدل را برای یادگیری طیف وسیع تری از نمایش ها افزایش می دهد. خروجی نهایی هر بلوک با جمع عنصری خروجی های دو مسیر و به دنبال آن افزودن ورودی اصلی از طریق اتصال باقیمانده به دست می آید. این ساختار به مدل اجازه می دهد تا هم ویژگی های خاص و هم ویژگی های مشترک را بیاموزد و تنوع ویژگی ها را در شبکه ارتقا دهد.

مدل اصلی، MyResNext با لایه‌های کانولوشن اولیه و فعال‌سازی‌های ELU شروع می‌شود و پس از آن max-pooling انجام می‌شود. سپس مدل نمونه هایی از GP_Block تعریف شده را به عنوان بلوک های باقیمانده ترکیب می کند. این بلوک‌ها که در نقاط استراتژیک شبکه قرار می‌گیرند، امکان یادگیری ویژگی‌های پیچیده را از طریق مسیرهای موازی و باقی‌مانده فراهم می‌کنند. این معماری با average pooling و لایه‌های کاملاً متصل برای طبقه‌بندی، و به دنبال آن یک تابع فعال‌سازی softmax برای تولید احتمالات کلاس به پایان می‌رسد. **استفاده از بلوک‌های باقی‌مانده با مسیرهای دوگانه، ظرفیت شبکه را برای گرفتن الگوهای پیچیده افزایش می‌دهد و آن را برای کارهای طبقه‌بندی تصویر مناسب می‌سازد.**

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 32, 32, 32]             896
              ELU-2          [-1, 32, 32, 32]               0
            Conv2d-3          [-1, 64, 32, 32]          18,496
              ELU-4          [-1, 64, 32, 32]               0
         MaxPool2d-5          [-1, 64, 16, 16]               0
            Conv2d-6          [-1, 64, 16, 16]          36,928
              ELU-7          [-1, 64, 16, 16]               0
            Conv2d-8          [-1, 64, 16, 16]          36,928
            Conv2d-9          [-1, 64, 16, 16]          36,928
             ELU-10          [-1, 64, 16, 16]               0
           Conv2d-11          [-1, 64, 16, 16]          36,928
             ELU-12          [-1, 64, 16, 16]               0
       GP_Block-13          [-1, 64, 16, 16]               0
           Conv2d-14         [-1, 128, 16, 16]          73,856
             ELU-15         [-1, 128, 16, 16]               0
        MaxPool2d-16          [-1, 128, 8, 8]               0
           Conv2d-17          [-1, 256, 8, 8]         295,168
             ELU-18          [-1, 256, 8, 8]               0
        MaxPool2d-19          [-1, 256, 4, 4]               0
           Conv2d-20          [-1, 256, 4, 4]         590,080
             ELU-21          [-1, 256, 4, 4]               0
           Conv2d-22          [-1, 256, 4, 4]         590,080
           Conv2d-23          [-1, 256, 4, 4]         590,080
             ELU-24          [-1, 256, 4, 4]               0
           Conv2d-25          [-1, 256, 4, 4]         590,080
             ELU-26          [-1, 256, 4, 4]               0
       GP_Block-27          [-1, 256, 4, 4]               0
       AvgPool2d-28          [-1, 256, 1, 1]               0
          Flatten-29                [-1, 256]               0
           Linear-30                [-1, 256]          65,792
           Linear-31                 [-1, 10]           2,570
          Softmax-32                 [-1, 10]               0
================================================================
Total params: 2,964,810
Trainable params: 2,964,810
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 3.72
Params size (MB): 11.31
Estimated Total Size (MB): 15.05
----------------------------------------------------------------
```

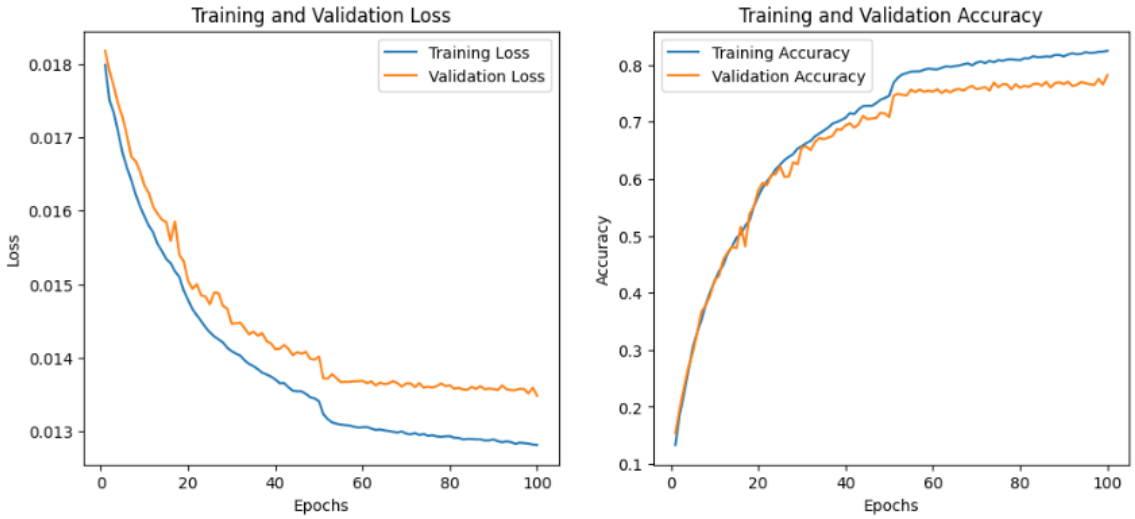
شکل ٤٠– معماری این شبکه

مدل را آموزش می دهیم و به نتایج زیر دست پیدا کردیم:

دقت نهایی بر روی test :

Test set: Average loss: 0.0132, Accuracy: 0.78

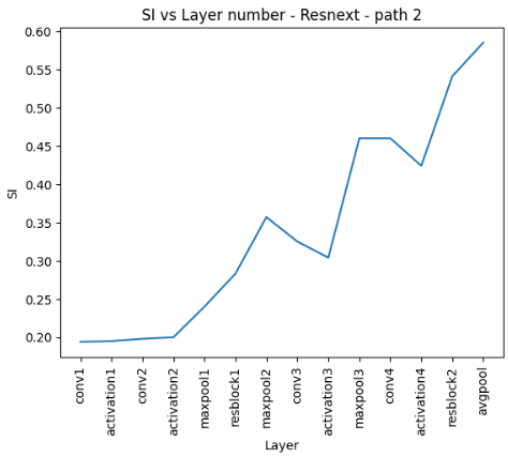
منحنی های دقت و loss:



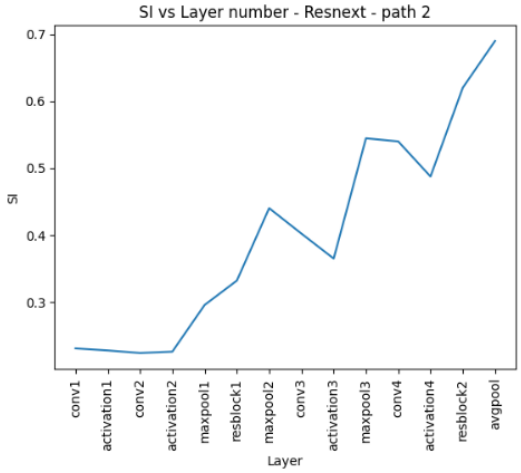
شکل ٤١- منحني هاي دقت و loss برای مدل اول

نتایج متریک ها:

- منحنی SI بر روی تمامی لایه ها:



شکل ٤٢ - منحنی SI بر روی تمامی لایه ها برای داده های test



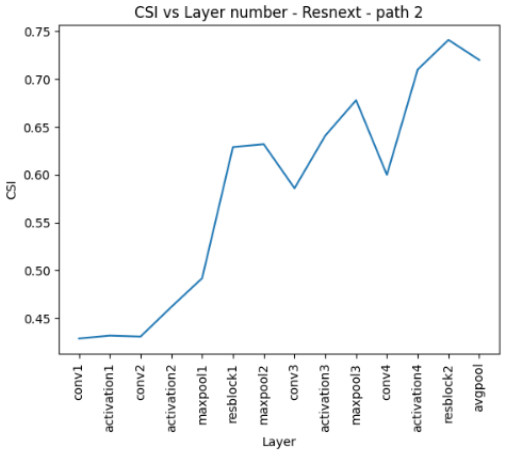
شکل ٤٣ -منحنی SI بر روی تمامی لایه ها برای داده های train

در conv3 باز کاهش مشاهده میکنیم.

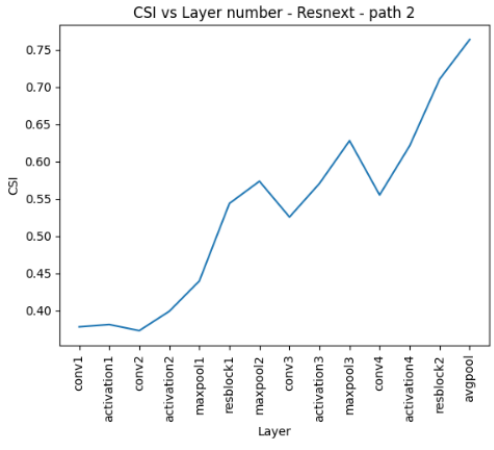
جدول ۱۵ - مقادیر si برای هر لایه برای داده های ترین و تست

si_train	si_test
[('conv1', 0.23044444620609283), ('activation1', 0.22733333706855774), ('conv2', 0.2235555648803711), ('activation2', 0.22555555403232574), ('maxpool1', 0.2953333258628845), ('resblock1', 0.33177778124809265), ('maxpool2', 0.4399999976158142), ('conv3', 0.4020000100135803), ('activation3', 0.3646666705608368), ('maxpool3', 0.5444444417953491), ('conv4', 0.5397777557373047), ('activation4', 0.4873333275318146), ('resblock2', 0.6200000047683716), ('avgpool', 0.6899999976158142)]	[('conv1', 0.1940000057220459), ('activation1', 0.19500000774860382), ('conv2', 0.1980000138282776), ('activation2', 0.20000000298023224), ('maxpool1', 0.24000000953674316), ('resblock1', 0.28300002217292786), ('maxpool2', 0.3570000231266022), ('conv3', 0.32500001788139343), ('activation3', 0.30400002002716064), ('maxpool3', 0.46000000834465027), ('conv4', 0.46000000834465027), ('activation4', 0.4240000247955322), ('resblock2', 0.5410000085830688), ('avgpool', 0.5850000381469727)]

- منحنی CSI بر روی تمامی لایه ها:



شکل ٤٥ - منحنی CSI بر روی تمامی لایه ها برای داده های test



شکل ٤٤ -منحنی CSI بر روی تمامی لایه ها برای داده های train

جدول ۱۶ – مقادیر csi برای هر لایه برای داده های ترین و تست

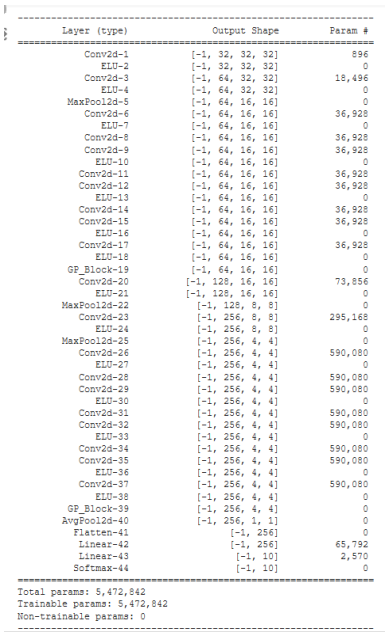
csi_train	csi_test
[('conv1', 0.3779999911785126), ('activation1', 0.38111111521720886), ('conv2', 0.3728888928890228), ('activation2', 0.39888888597488403), ('maxpool1', 0.4395555555820465),	[('conv1', 0.42900002002716064), ('activation1', 0.4320000112056732), ('conv2', 0.4310000240802765), ('activation2', 0.4620000123977661), ('maxpool1', 0.492000013589859),

('resblock1', 0.5435555577278137), ('maxpool2', 0.5733333230018616), ('conv3', 0.5253333449363708), ('activation3', 0.5704444646835327), ('maxpool3', 0.6277777552604675), ('conv4', 0.554888904094696), ('activation4', 0.620888888835907), ('resblock2', 0.710444450378418), ('avgpool', 0.7635555863380432)]	('resblock1', 0.6290000081062317), ('maxpool2', 0.6320000290870667), ('conv3', 0.5860000252723694), ('activation3', 0.6410000324249268), ('maxpool3', 0.6780000329017639), ('conv4', 0.6000000238418579), ('activation4', 0.7100000381469727), ('resblock2', 0.7410000562667847), ('avgpool', 0.7200000286102295)]

برای path=4 هم مشابه حالت بالا مدل را پیاده سازی میکنیم و تغییرات زیر را اعمال میکنیم:

```
class GP_Block(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(GP_Block, self).__init__()

        # Path 1
        self.conv1_path1 = nn.Conv2d(in_channels, out_channels, kernel_size=3,
padding=1)
        self.activation = nn.ELU()
        self.conv2_path1 = nn.Conv2d(out_channels, out_channels, kernel_size=3,
padding=1)

        # Path 2
        self.conv1_path2 = nn.Conv2d(in_channels, out_channels, kernel_size=3,
padding=1)
        self.conv2_path2 = nn.Conv2d(out_channels, out_channels, kernel_size=3,
padding=1)

        # Path 3
        self.conv1_path3 = nn.Conv2d(in_channels, out_channels, kernel_size=3,
padding=1)
        self.conv2_path3 = nn.Conv2d(out_channels, out_channels, kernel_size=3,
padding=1)

        # Path 4
        self.conv1_path4 = nn.Conv2d(in_channels, out_channels, kernel_size=3,
padding=1)
        self.conv2_path4 = nn.Conv2d(out_channels, out_channels, kernel_size=3,
padding=1)

    def forward(self, x):
        shortcut = x  # Save the input as a shortcut for the residual connection

        # Path 1
```

```
        x1 = self.conv1_path1(x)
        x1 = self.activation(x1)
        x1 = self.conv2_path1(x1)

        # Path 2
        x2 = self.conv1_path2(x)
        x2 = self.activation(x2)
        x2 = self.conv2_path2(x2)

        # Path 3
        x3 = self.conv1_path3(x)
        x3 = self.activation(x3)
        x3 = self.conv2_path3(x3)

        # Path 4
        x4 = self.conv1_path4(x)
        x4 = self.activation(x4)
        x4 = self.conv2_path4(x4)

        # Combine paths (element-wise addition)
        x = x1 + x2 + x3 + x4

        # Final combination with the shortcut (residual connection)
        x += shortcut
        x = self.activation(x)  # Apply activation function to the final output

        return x
```



Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 32, 32, 32]	896
ELU-2	[-1, 32, 32, 32]	0
Conv2d-3	[-1, 64, 32, 32]	18,496
ELU-4	[-1, 64, 32, 32]	0
MaxPool2d-5	[-1, 64, 16, 16]	0
Conv2d-6	[-1, 64, 16, 16]	36,928
ELU-7	[-1, 64, 16, 16]	0
Conv2d-8	[-1, 64, 16, 16]	36,928
Conv2d-9	[-1, 64, 16, 16]	36,928
ELU-10	[-1, 64, 16, 16]	0
Conv2d-11	[-1, 64, 16, 16]	36,928
Conv2d-12	[-1, 64, 16, 16]	36,928
ELU-13	[-1, 64, 16, 16]	0
Conv2d-14	[-1, 64, 16, 16]	36,928
Conv2d-15	[-1, 64, 16, 16]	36,928
ELU-16	[-1, 64, 16, 16]	0
Conv2d-17	[-1, 64, 16, 16]	36,928
ELU-18	[-1, 64, 16, 16]	0
GP_Block-19	[-1, 64, 16, 16]	0
Conv2d-20	[-1, 128, 16, 16]	73,856
ELU-21	[-1, 128, 16, 16]	0
MaxPool2d-22	[-1, 128, 8, 8]	0
Conv2d-23	[-1, 256, 8, 8]	295,168
ELU-24	[-1, 256, 8, 8]	0
MaxPool2d-25	[-1, 256, 4, 4]	0
Conv2d-26	[-1, 256, 4, 4]	590,080
ELU-27	[-1, 256, 4, 4]	0
Conv2d-28	[-1, 256, 4, 4]	590,080
Conv2d-29	[-1, 256, 4, 4]	590,080
ELU-30	[-1, 256, 4, 4]	0
Conv2d-31	[-1, 256, 4, 4]	590,080
Conv2d-32	[-1, 256, 4, 4]	590,080
ELU-33	[-1, 256, 4, 4]	0
Conv2d-34	[-1, 256, 4, 4]	590,080
Conv2d-35	[-1, 256, 4, 4]	590,080
ELU-36	[-1, 256, 4, 4]	0
Conv2d-37	[-1, 256, 4, 4]	590,080
ELU-38	[-1, 256, 4, 4]	0
GP_Block-39	[-1, 256, 4, 4]	0
AvgPool2d-40	[-1, 256, 1, 1]	0
Flatten-41	[-1, 256]	0
Linear-42	[-1, 256]	65,792
Linear-43	[-1, 10]	2,570
Softmax-44	[-1, 10]	0

Total params: 5,472,842
Trainable params: 5,472,842
Non-trainable params: 0

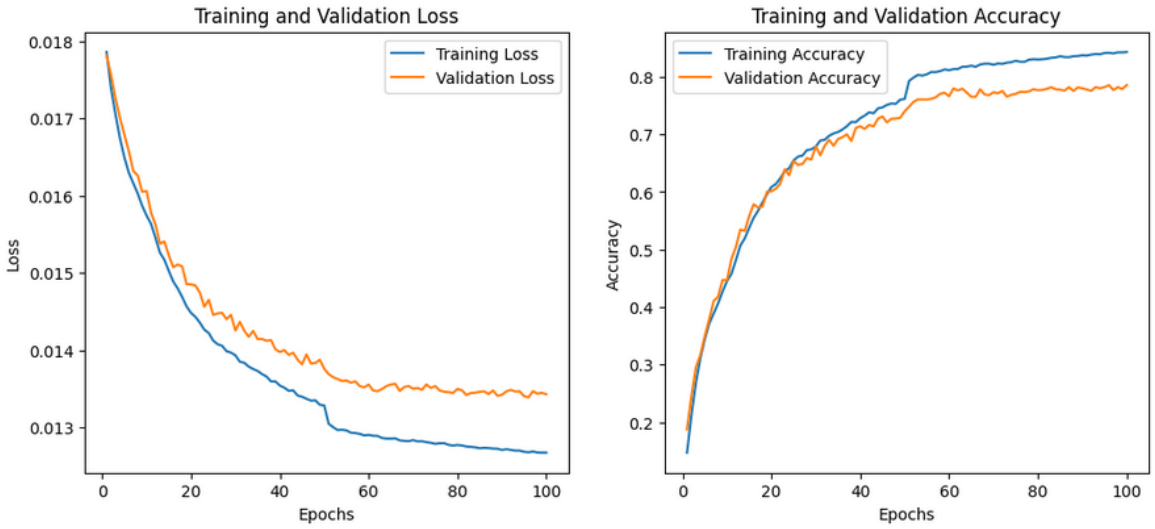
شکل ٤٦ ــ خلاصه مدل

مدل را آموزش می دهیم و به نتایج زیر دست پیدا کردیم:

دقت نهایی بر روی test :

Test set: Average loss: 0.0132, Accuracy: 0.79

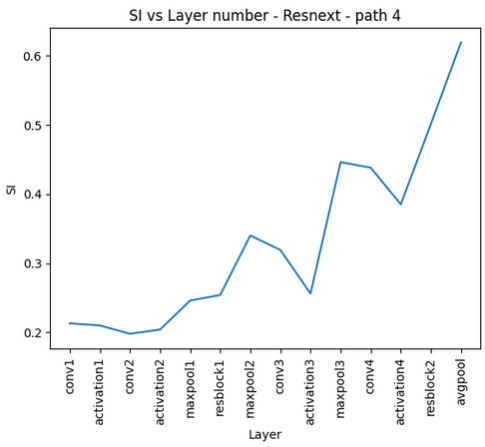
منحنی های دقت و loss:



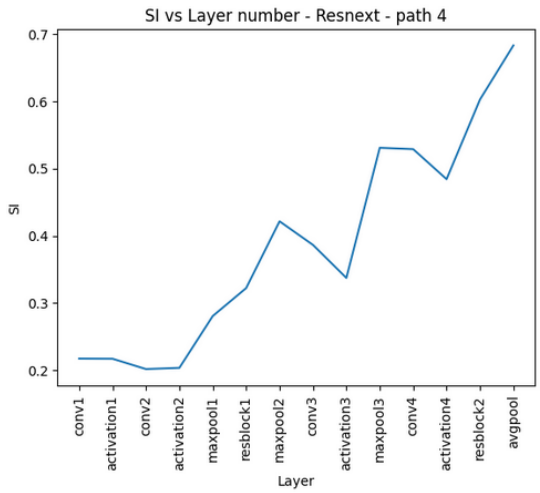
شکل ٤٧- منحني هاي دقت و loss برای مدل اول

نتایج متریک ها:

- منحنی SI بر روی تمامی لایه ها:



شکل ٤٨ - منحنی SI بر روی تمامی لایه ها برای داده های test



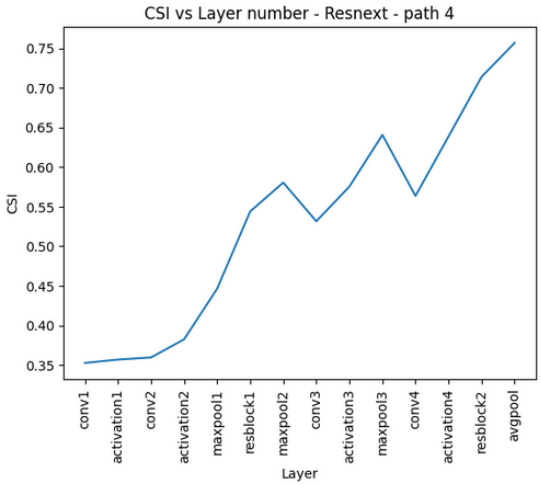
شکل ٤٩ -منحنی SI بر روی تمامی لایه ها برای داده های train

در conv3 باز کاهش مشاهده میکنیم.

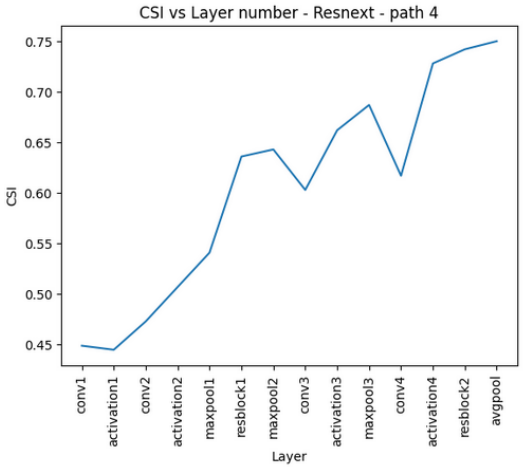
جدول ۱۷ - مقادیر si برای هر لایه برای داده های ترین و تست

si_train	si_test
[('conv1', 0.2173333317041397), ('activation1', 0.21711111068725586), ('conv2', 0.20177778601646423), ('activation2', 0.20355555415153503), ('maxpool1', 0.28066667914390564), ('resblock1', 0.3222222328186035), ('maxpool2', 0.4215555489063263), ('conv3', 0.38644444942474365), ('activation3', 0.3373333513736725), ('maxpool3', 0.5308889150619507), ('conv4', 0.5288888812065125), ('activation4', 0.4842222332954407), ('resblock2', 0.6026666760444641), ('avgpool', 0.6831111311912537)]ô	[('conv1', 0.21300001442432404), ('activation1', 0.21000000834465027), ('conv2', 0.1980000138282776), ('activation2', 0.20400001108646393), ('maxpool1', 0.2460000067949295), ('resblock1', 0.2540000081062317), ('maxpool2', 0.3400000035762787), ('conv3', 0.3190000057220459), ('activation3', 0.25600001215934753), ('maxpool3', 0.44600000977516174), ('conv4', 0.43800002336502075), ('activation4', 0.3850000202655792), ('resblock2', 0.5010000467300415), ('avgpool', 0.6190000176429749)]

- منحنی CSI بر روی تمامی لایه ها:



شکل ۵۰ -منحنی CSI بر روی تمامی لایه ها برای داده های train



شکل ۵۱ - منحنی CSI بر روی تمامی لایه ها برای داده های test

جدول ۱۸ – مقادیر csi برای هر لایه برای داده های ترین و تست

csi_train	csi_test
[('conv1', 0.3408888876438141), ('activation1', 0.34111112356185913), ('conv2', 0.3479999899864197), ('activation2', 0.37444445490837097), ('maxpool1', 0.433777779340744),	[('conv1', 0.4490000307559967), ('activation1', 0.445000022649765), ('conv2', 0.47300001978874207), ('activation2', 0.5070000290870667), ('maxpool1', 0.5410000085830688),

```
('resblock1', 0.6360000371932983),
('maxpool2', 0.6430000066757202),
('conv3', 0.6030000448226929),
('activation3', 0.6620000600814819),
('maxpool3', 0.687000036239624),
('conv4', 0.6170000433921814),
('activation4', 0.7280000448226929),
('resblock2', 0.7420000433921814),
('avgpool', 0.7500000596046448)]
```

```
('resblock1', 0.538444459438324),
('maxpool2', 0.5680000185966492),
('conv3', 0.5206666588783264),
('activation3', 0.5577777624130249),
('maxpool3', 0.6348888874053955),
('conv4', 0.5522222518920898),
('activation4', 0.629111111164093),
('resblock2', 0.7246666550636292),
('avgpool', 0.758222222328186)]
```

مقایسه این دو روش:

جدول ۱۹ – مقایسه بین این روش ها

Model	Test acc	SI-train	SI-test	CSI-train	CSI-test
MyResnet	0.77	0.67	0.6	0.74	0.7
Path2	0.78	0.68	0.58	0.76	0.72
Path 4	0.79	0.68	0.61	0.75	0.75

مشاهده میکنیم که در group convها نتایج بهتری داریم

در مدل‌های ResNeXt که از کانولوشن‌های گروهی استفاده می‌کنند، تعداد مسیرها به تعداد مسیرهای کانولوشن موازی در هر بلوک اشاره دارد. هر مسیر به طور مستقل داده های ورودی را پردازش می کند و خروجی های آنها از طریق جمع عناصر ترکیب می شوند. تأثیر تغییر تعداد مسیرها در مدل ResNeXt با پیچیدگی های گروهی بسیار مهم است و چندین پیامد دارد:

افزایش ظرفیت مدل: تعداد مسیرهای بیشتر به مدل امکان می دهد مجموعه ای از ویژگی های متنوع تری را ثبت کند. هر مسیر، نمایش‌های متفاوتی از داده‌های ورودی را می‌آموزد و ظرفیت کلی مدل را برای درک الگوهای پیچیده و تغییرات درون داده‌ها افزایش می‌دهد. این ظرفیت افزایش یافته برای کارهایی که نیازمند تبعیض جزئیات دقیق هستند، سودمند است.

تنوع ویژگی های بهبود یافته: با مسیرهای بیشتر، مدل می تواند طیف وسیع تری از ویژگی ها را بیاموزد و آن را قادر می سازد تا با ویژگی های مختلف موجود در مجموعه داده سازگار شود. این تنوع به ویژه هنگامی که با مجموعه داده‌های حاوی الگوهای بصری متنوع سروکار داریم یا زمانی که مجموعه داده بزرگ است و تنوع قابل‌توجهی را نشان می‌دهد، ارزشمند است.

اثر **Regularization**: داشتن چندین مسیر، نوعی منظم‌سازی ضمنی را معرفی می‌کند، زیرا مدل یاد می‌گیرد ویژگی‌ها را از مسیرهای مختلف ترکیب کند. این می تواند به جلوگیری از تطبیق بیش از حد و بهبود تعمیم مدل به داده های دیده نشده کمک کند.

هزینه محاسباتی: از جنبه منفی، افزایش تعداد مسیرها همچنین با هزینه محاسباتی بالاتر در طول آموزش و استنتاج همراه است. ماهیت موازی این مسیرها به محاسبات و حافظه بیشتری نیاز دارد که به طور بالقوه بر کارایی مدل تأثیر می گذارد.

سوال دوم : شبکه تشخیص اشیا

مرحله ۱:

الف) مقادیر **ground truth** را به شکل زیر استخراج میکنیم:

قبل از آن برای سرعت بخشیدن به پردازش، عکسهایی که با **cat** شروع می شدند را به فولدر دیگری منتقل کردیم.

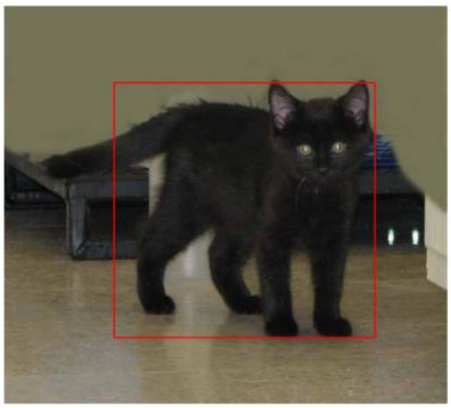
```
# Step 2: Function to parse XML files and extract information
def parse_xml(xml_file):
    # Parse the XML file using ElementTree
    tree = ET.parse(xml_file)
    root = tree.getroot()
    # Extract image ID from the 'filename' tag
    image_id = root.find('filename').text
    # Extract bounding box and label information for 'cat' objects
    cat_boxes = []  # List to store bounding boxes for 'cat' objects
    labels = []
    # Iterate through each 'object' tag in the XML file
    for obj in root.findall('object'):
        # Extract label from the 'name' tag inside the 'object' tag
        label = obj.find('name').text

        # Check if the label is 'cat'
        if label == 'cat':
            # Extract bounding box coordinates from the 'bndbox' tag inside the
'object' tag
            bbox = obj.find('bndbox')
            xmin = round(float(bbox.find('xmin').text))
            ymin = round(float(bbox.find('ymin').text))
            xmax = round(float(bbox.find('xmax').text))
            ymax = round(float(bbox.find('ymax').text))

            # Append bounding box coordinates to the list for 'cat' objects
            cat_boxes.append((xmin, ymin, xmax, ymax))
            labels.append(label)

    # Return the extracted information: image ID and bounding boxes for 'cat'
objects
    return image_id, cat_boxes, labels
```

کد ارائه شده، parse_xml که مسیر یک فایل XML را به عنوان ورودی می‌گیرد. از ماژول ElementTree برای تجزیه محتوای XML استفاده می کند و اطلاعات مربوط به اشیاء cat را از فایل استخراج می کند. به طور خاص، شناسه تصویر را از تگ filename بازیابی می کند و از طریق هر تگ object در XML تکرار می شود، برچسب ها و مختصات جعبه را برای اشیایی که با عنوان cat برچسب گذاری شده اند استخراج می کند. اطلاعات جعبه در لیستی به نام cat_boxes ذخیره می شود، که در آن هر ورودی یک tuple است که مختصات xmin، ymin، xmax، ymax یک جعبه محدود در اطراف یک شی cat را نشان می دهد. علاوه بر این، برچسب های مربوطه در لیستی به نام "برچسب" ذخیره می شوند. سپس این تابع شناسه تصویر، لیست کادرها و لیست برچسب‌های اشیاء cat را برمی‌گرداند.

```
# Step 3: Process all XML files in the dataset
xml_folder = '/content/cat'
graound_truth  = []

for xml_file in os.listdir(xml_folder):
    if xml_file.endswith('.xml'):
        xml_path = os.path.join(xml_folder, xml_file)
        image_id, boxes, labels = parse_xml(xml_path)
        graound_truth.append({'image_id': image_id, 'boxes': boxes, 'labels':
labels})
```

خروجی کد بالا را برای یک مثال مشاهده میکنید:

```
Image ID: cat.3880.jpg
   Labels: ['cat']
   Bounding Boxes:
- (123, 85, 418, 372)
```



شکل ۵۲ – خروجی کد بالا

ب)

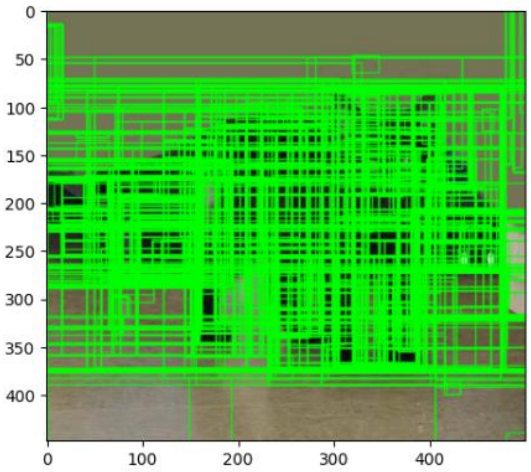
```
# Function to perform Selective Search
def selective_search(image):
    # Create a Selective Search segmentation object
    cv2.setUseOptimized(True);
    ss = cv2.ximgproc.segmentation.createSelectiveSearchSegmentation()
    # Set the input image for selective search
    ss.setBaseImage(image)
    # Switch to the fast but less accurate mode
    ss.switchToSelectiveSearchQuality()
    # Run selective search
    rects = ss.process()
    # Return the region proposals
    return rects
```

تابعی به نام **selective_search** را تعریف می کند که الگوریتم جستجوی انتخابی را برای تولید پیشنهاد منطقه پیاده سازی می کند. جستجوی انتخابی الگوریتمی است که برای شناسایی مناطق بالقوه شی در یک تصویر استفاده می شود. این تابع با ایجاد یک شی تقسیم بندی جستجوی انتخابی از ماژول های **OpenCV**، تنظیم تصویر ورودی برای تجزیه و تحلیل، و پیکربندی الگوریتم برای اولویت دادن به کیفیت بر سرعت شروع می شود. سپس فرآیند جستجوی انتخابی واقعی با فراخوانی **ss.process()** اجرا می‌شود که در نتیجه فهرستی از پیشنهادات منطقه مستطیلی ذخیره شده در متغیر **rects** ایجاد می‌شود. هر مستطیل نشان دهنده یک منطقه کاندید است که الگوریتم آن را به طور بالقوه حاوی یک شی می داند. این پیشنهادات منطقه را می توان در مراحل بعدی خطوط لوله تشخیص اشیاء برای تمرکز بر مناطق خاص مورد علاقه در تصویر مورد استفاده قرار داد و تشخیص کارآمدتر و دقیق‌تر اشیاء را تسهیل کرد. جستجوی انتخابی با ترکیب استراتژی‌های تقسیم‌بندی متنوع، مانند رنگ، بافت و اندازه، برای ایجاد مجموعه‌ای جامع از پیشنهادات منطقه‌ای که مقیاس‌های مختلف و ظاهر شی در یک تصویر را پوشش می‌دهد، به این امر دست می‌یابد.

در مقاله گفته شده است که تعداد پیشنهادات ۲۰۰۰ است، برای پیاده سازی آن به شکل زیر عمل میکنیم:

```
def selective_search(image, max_proposals=2000):
    # Create a Selective Search segmentation object
    cv2.setUseOptimized(True)
    ss = cv2.ximgproc.segmentation.createSelectiveSearchSegmentation()
    # Set the input image for selective search
    ss.setBaseImage(image)
```

```
    # Switch to the fast but less accurate mode
    ss.switchToSelectiveSearchQuality()
    # Run selective search
    rects = ss.process()
    # Ensure the number of proposals does not exceed the limit
    if len(rects) > max_proposals:
        rects = rects[:max_proposals]
    # Return the region proposals
    return rects
```



شکل ۵۳ – نمونه ای از خروجی الگوریتم

ج)

```
def calculate_iou(box1, box2):
    x1, y1, w1, h1 = box1
    x2, y2, w2, h2 = box2

    x_intersection = max(0, min(x1 + w1, x2 + w2) - max(x1, x2))
    y_intersection = max(0, min(y1 + h1, y2 + h2) - max(y1, y2))
    intersection = x_intersection * y_intersection

    area_box1 = w1 * h1
    area_box2 = w2 * h2
    union = area_box1 + area_box2 - intersection

    iou = intersection / union
    return iou
```

«calculate_iou»، که تقاطع روی IoU Union را بین دو کادر محدود که با مختصات آنها (x، y، عرض، ارتفاع) نشان داده شده اند، محاسبه می کند. IoU یک متریک است که معمولاً در وظایف تشخیص

اشیا برای اندازه‌گیری همپوشانی بین دو جعبه مرزی استفاده می‌شود. این تابع ابتدا مختصات جعبه های ورودی را استخراج می کند و با یافتن ناحیه همپوشانی در امتداد محورهای x و y، سطح تقاطع را محاسبه می کند. سپس مساحت اتحاد را با جمع کردن تک تک نواحی جعبه ها و کم کردن سطح تقاطع محاسبه می کند. در نهایت، IoU با تقسیم منطقه تقاطع بر ناحیه اتحادیه تعیین می شود. مقدار IoU حاصل از ۰ (بدون همپوشانی) تا ۱ (همپوشانی کامل) متغیر است، که معیاری از شباهت بین دو جعبه مرزی را ارائه می دهد.

د)

هدف کلی این کد تولید نمونه‌های آموزشی برای مدل تشخیص شی با برش مناطق مورد علاقه از تصاویر بر اساس ground truth annotations است. نمونه‌های مثبت مناطقی را نشان می‌دهند که دارای اشیاء مورد علاقه هستند (با همپوشانی قابل‌توجه با ground truth bounding boxes)، در حالی که نمونه‌های منفی مناطقی را نشان می‌دهند که شامل اشیاء هدف نیستند. این مرحله آماده‌سازی داده‌ها برای آموزش مدل‌های تشخیص اشیا بسیار مهم است، و آنها را قادر می‌سازد تا از نشانه‌های بصری مرتبط به شیوه‌ای تحت نظارت یاد بگیرند.

```
# Step 1d: Crop images using region proposals and save as training images
def crop_and_save_images(image, proposals, gt_boxes, gt_labels, save_folder):
    os.makedirs(save_folder, exist_ok=True)

    positive_samples = 0
    negative_samples = 0

    for i, proposal in enumerate(proposals):

        for gt_box, gt_label in zip(gt_boxes, gt_labels):

            if gt_box:
                iou = calculate_iou(gt_box, proposal)
            else:
                iou = 0

            if iou > 0.5:
                # Positive sample
                x, y, w, h = proposal
                cropped_image = image[y:y+h, x:x+w]
                # resized_image = cv2.resize(cropped_image, (224, 224),
interpolation=cv2.INTER_AREA)
                save_path = os.path.join(save_folder,
f"positive_{positive_samples}.png")
```

```
                cv2.imwrite(save_path, cv2.cvtColor(cropped_image,
cv2.COLOR_RGB2BGR))
                positive_samples += 1

            else:
                # Negative sample
                x, y, w, h = proposal
                cropped_image = image[y:y+h, x:x+w]
                # resized_image = cv2.resize(cropped_image, (224, 224),
interpolation=cv2.INTER_AREA)
                save_path = os.path.join(save_folder,
f"negative_{negative_samples}.png")
                cv2.imwrite(save_path, cv2.cvtColor(cropped_image,
cv2.COLOR_RGB2BGR))
                negative_samples += 1

# Path to your dataset folder
dataset_folder = '/content/cat'
# Path to save cropped training images
save_folder = '/content/train'

# Step 1: Iterate through each image and perform the steps
for file_name in tqdm(os.listdir(dataset_folder)):
    if file_name.endswith('.xml'):
        xml_path = os.path.join(dataset_folder, file_name)
        image_id, gt_boxes, gt_labels = parse_xml(xml_path)

        # Load the corresponding image
        image_path = os.path.join(dataset_folder, image_id)
        image = cv2.imread(image_path)

        # Convert BGR to RGB
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

        # Step 1b: Generate region proposals
        proposals = selective_search(image)

        # Step 1c: Check region proposals with ground truth values using IoU
        crop_and_save_images(image, proposals, gt_boxes, gt_labels, save_folder)

print("Cropping and saving completed.")
```

از طریق مجموعه داده ای از تصاویر و فایل های XML متناظر حاوی جعبه ها و truth bounding boxes، تکرار می شود. برای هر تصویر، تصویر را بارگیری می‌کند، آن را از فرمت BGR به فرمت RGB تبدیل می‌کند، پیشنهادهای منطقه‌ای را با استفاده از جستجوی انتخابی ایجاد می‌کند، و سپس همپوشانی بین این پیشنهادات و ground truth bounding boxes را با استفاده از Intersection over Union

(IoU) تعیین می‌کند. بر اساس مقادیر IoU، کد به طور انتخابی نمونه های مثبت که IoU > 0.5 و نمونه های منفی که IoU <= 0.5 را در پوشه های جداگانه ذخیره می کند، یک مجموعه داده مناسب برای آموزش مدل های تشخیص اشیا ایجاد می کند. نمونه های مثبت نواحی حاوی اشیاء مورد علاقه هستند، در حالی که نمونه های منفی مناطق پس زمینه را نشان می دهند. کد از OpenCV برای پردازش تصویر استفاده می کند و یک ساختار مجموعه داده خاص را با فایل های XML فرض می کند که اطلاعات gt را ارائه می دهد.

نمونه ای از خروجی های این بخش به شکل زیر است:

شکل ٥٤ – نمونه های positive

شکل ٥٥ – نمونه های negative

مرحله 2:

توجه باید داشته باشیم که دیتاست ما بالانس نیست، درنتیجه باید از class weight هنگام آموزش استفاده کنیم.

```
Number of positive images: 15921
Number of negative images: 1099938
```

```
positive_count = 15921
negative_count = 1099938
total_samples = positive_count + negative_count
# Calculate class frequencies
frequency_class_0 = positive_count / total_samples
frequency_class_1 = negative_count / total_samples
# Calculate inverse of class frequencies as class weights
```

```
weight_for_class_0 = 1 / frequency_class_0
weight_for_class_1 = 1 / frequency_class_1
print("Weight for Class 0:", weight_for_class_0)
print("Weight for Class 1:", weight_for_class_1)
```

```
Weight for Class 0: 70.0872432636141
Weight for Class 1: 1.014474452196396
```

این اعداد قابل پیشبینی است چون برای هر تصویر ما ۲۰۰۰region proposal خواهیم داشت، تعداد عکس ها ۵۵۰ تا عکس گربه و درکل باید ۱۱۰۰۰۰۰ نمونه داشته باشیم. در ادامه ما sampleهای مثبت و منفی را به دو فولدر جداگانه میبریم.

```
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor()
])
dataset = ImageFolder(root='content/cat_dataset/', transform=transform)
```

داده هارا به شکل بالا میخوانیم و در پایین داده ها را به ۸۰-۲۰ درصد تقسیم میکنیم برای داده های ترین و ارزیابی:

```
# Split the dataset into train and validation sets
train_size = int(0.8 * len(dataset))
val_size = len(dataset) - train_size
train_dataset, val_dataset = random_split(dataset, [train_size, val_size])

# Create DataLoader for training set
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True,
num_workers=1)

# Create DataLoader for validation set
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False, num_workers=1)
```

حلقه train:

```
# Assuming 'device' is defined somewhere in your code
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
# Load the pre-trained MobileNet model
model = models.mobilenet_v2(pretrained=True)
# Move the model to the device
model = model.to(device)
```

```
# Freeze all the parameters in the feature network
for param in model.features.parameters():
    param.requires_grad = False
# Move the model parameters to the same device
model = model.to(device)
# Modify the classifier for binary classification
num_features = model.classifier[1].in_features

classifier_layers = [
    nn.Dropout(0.2),
    nn.Linear(num_features, 128),
    nn.ReLU(),
    nn.Dropout(0.2),
    nn.Linear(128, 2),
    nn.Sigmoid(dim=1)
]
# Move the classifier layers to the same device
classifier = nn.Sequential(*classifier_layers).to(device)
model.classifier = classifier

#Define class weights
class_weights = torch.tensor([weight_for_class_0, weight_for_class_1],
dtype=torch.float32).to(device)

# Define the criterion with class weights
criterion = nn.CrossEntropyLoss(weight=class_weights)

# Define the optimizer
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)

# Number of training epochs
num_epochs = 10
```

این کد یک مدل طبقه بندی تصویر باینری را با استفاده از معماری MobileNetV2 از پیش آموزش دیده را تنظیم می کند. مدل MobileNetV2 بارگذاری شده و به دستگاه مشخص شده منتقل می شود. پارامترها در شبکه ویژگی مدل منجمد می شوند تا وزن های از پیش آموزش داده شده خود را حفظ کنند و کل مدل مجدداً برای سازگاری به دستگاه منتقل می شود.

بخش طبقه‌بندی‌کننده مدل برای طبقه‌بندی باینری با جایگزینی لایه‌های خطی موجود با یک پیکربندی سفارشی اصلاح می‌شود. به طور خاص، دو لایه کاملاً متصل با توابع فعال‌سازی ReLU و dropout regularization اضافه می‌شوند. تعداد ویژگی های ورودی برای اولین لایه خطی توسط ساختار طبقه بندی کننده اصلی MobileNetV2 تعیین می شود. طبقه‌بندی‌کننده به‌دست‌آمده به دستگاه مشابه مدل منتقل می‌شود. وزن کلاس ها بر اساس وزن های مشخص شده برای هر کلاس تعریف می شود و معیار

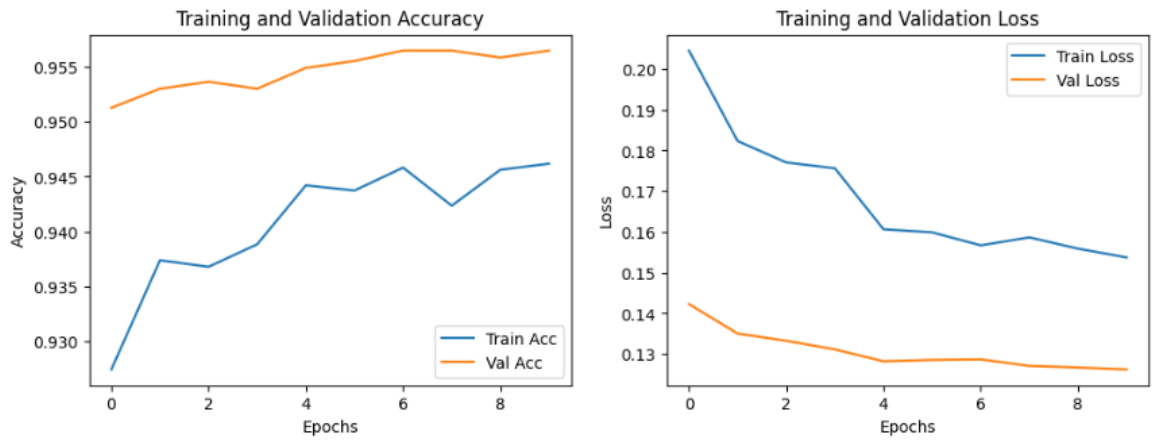
CrossEntropyLoss با این وزن های کلاس نمونه سازی می شود. بهینه سازی توسط Stochastic Gradient Descent (SGD) با نرخ یادگیری ۰,۰۰۱ و تکانه ۰,۹ انجام می شود.

فرآیند آموزش به گونه‌ای پیکربندی شده است که برای ۱۰ دوره اجرا شود، که در طی آن مدل با استفاده از وزن‌های کلاس، معیار و بهینه‌ساز بر روی یک کار طبقه‌بندی باینری آموزش داده می‌شود.

```
# Lists to store history for plotting
train_loss_history = []
val_loss_history = []
train_acc_history = []
val_acc_history = []
# Training loop
for epoch in tqdm(range(num_epochs)):
    print('Epoch {}/{}'.format(epoch, num_epochs - 1))
    # Training phase
    model.train()

    running_loss = 0.0
    running_corrects = 0

    for images, labels in tqdm(train_loader):
        images, labels = images.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(images)
        # Ensure labels are on the same device as outputs
        labels = labels.to(device)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        # Statistics
        running_loss += loss.item() * images.size(0)
        _, preds = torch.max(outputs, 1)
        running_corrects += torch.sum(preds == labels.data)

    epoch_loss = running_loss / len(train_loader.dataset)
    epoch_acc = running_corrects.double() / len(train_loader.dataset)

    print(f'train Loss: {epoch_loss:.4f}, Acc: {epoch_acc:.4f}')

    # Validation phase
    model.eval()  # Set the model to evaluation mode

    with torch.no_grad():
        total_correct = 0
        total_samples = 0
```

```
        val_running_loss = 0.0

        for val_images, val_labels in tqdm(val_loader):
            val_images, val_labels = val_images.to(device), val_labels.to(device)

            val_outputs = model(val_images)
            _, predicted = torch.max(val_outputs, 1)
            total_samples += val_labels.size(0)
            total_correct += (predicted == val_labels).sum().item()

            # Calculate validation loss
            val_loss = criterion(val_outputs, val_labels)
            val_running_loss += val_loss.item() * val_images.size(0)

        val_accuracy = total_correct / total_samples
        val_epoch_loss = val_running_loss / len(val_loader.dataset)

        print(f'val Loss: {val_epoch_loss:.4f}, Acc: {val_accuracy:.4f}')

    # Save values for plotting
    train_loss_history.append(epoch_loss)
    val_loss_history.append(val_epoch_loss)
    train_acc_history.append(epoch_acc)
    val_acc_history.append(val_accuracy)
```

مدل را آموزش می دهیم و به نتایج زیر دست پیدا میکنیم:



شکل ٥٦ – نمودار دقت و loss شبکه

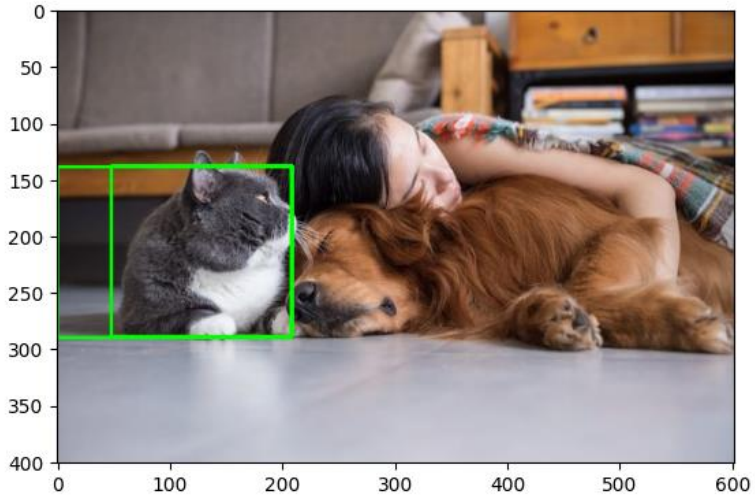
خروجی های داده های تست به صورت زیر می باشد:

برای نمایش خروجی ها به شکل زیر عمل می کنیم:

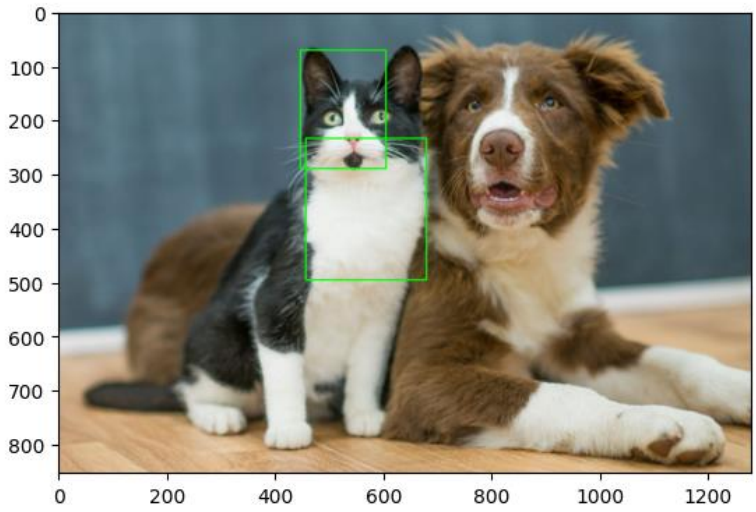
```
# Read the test image
img = cv2.imread('test1.jpg')

# Create a selective search segmentation object
cv2.setUseOptimized(True)
ss = cv2.ximgproc.segmentation.createSelectiveSearchSegmentation()
ss.setBaseImage(img)
ss.switchToSelectiveSearchFast()
ssresults = ss.process()

threshold = 0.8

for e, result in enumerate(ssresults):
    x, y, w, h = result
    timage = img[y:y+h, x:x+w]
    resized = cv2.resize(timage, (224, 224), interpolation=cv2.INTER_AREA)
    resized = np.expand_dims(resized, axis=0)
    input_tensor = torch.from_numpy(resized).permute(0, 3, 1,
2).float().to('cuda')
   # Forward pass
    with torch.no_grad():
         out = model(input_tensor)

            # Apply softmax to get probabilities
    probs = F.softmax(out[0], dim=0)

    if probs.max().item() > threshold:
          cv2.rectangle(imOut, (x, y), (x+w, y+h), (0, 255, 0), 2)
plt.figure()
plt.imshow(img)
plt.show()
```

این کد با استفاده از ترکیبی از تقسیم بندی جستجوی انتخابی و یک مدل تشخیص شی را انجام می دهد. کد با خواندن تصاویر تست با استفاده از کتابخانه OpenCV شروع می شود و سپس یک شی تقسیم بندی جستجوی انتخابی از ماژول **ximgproc** را مقداردهی اولیه می کند. الگوریتم جستجوی انتخابی، پیشنهادات منطقه ای را در تصویر بر اساس رنگ، بافت و سایر ویژگی ها ایجاد می کند. این پیشنهادها سپس برای پردازش بیشتر مورد استفاده قرار می گیرند.

در مرحله بعد، کد روی پیشنهادات منطقه تولید شده تکرار می شود و هر منطقه را از تصویر اصلی استخراج می کند. اندازه هر ناحیه استخراج‌شده به ۲۲۴ در ۲۴۴ پیکسل تغییر می‌کند و برای مطابقت با فرمت ورودی مورد انتظار یک مدل یادگیری عمیق، پیش پردازش شده است. برای هر ناحیه، یک گذر رو به جلو از مدل انجام می شود و تابع **softmax** برای بدست آوردن احتمالات کلاس اعمال می شود. اگر حداکثر احتمال از یک آستانه از پیش تعریف شده (در این مورد ۰,۸) فراتر رود، با استفاده از **OpenCV** یک کادر محدود در اطراف ناحیه روی تصویر اصلی ترسیم می شود. در نهایت با استفاده از **Matplotlib** تصویر اصلی با کادرهای مرزی ترسیم شده نمایش داده می شود. (تمام **region**ها را بدست آورده و با خروجی مدل مقایسه می کنیم آن هایی که بیشتر از **threshold** بودند را رسم میکنیم.) (خروجی ها می تئانند بهتر شوند، یکی از راه کار ها تین است که معیار انتخاب **IOU** را بیشتر کنیم ۰,۵ نباشد)



شکل ۵۷ – خروجی test2



شکل ۵۸ – خروجی test1