به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین اول

نام و نام خانوادگی	مهسا راستی نجف آبادی – مینا سیدی
شماره دانشجویی	810398084 - 810198393
تاریخ ارسال گزارش	۱۴۰۱.۰۷.۰3

فهرست

شکل‌ها

جدول‌ها

پاسخ 1. تاثیر تغییر رزولوشن در طبقه بندی در شبکه CNN

۱-۱. دست گرمی

فیلتر اول:

0	1	0
1	1	1
0	1	0

فیلتر دوم:

1	1	1
1	0	1
1	1	1

تصویر اول از کلاس اول:

0	0	0	0
1	1	1	1
1	0	0	1
1	1	1	1

خروجی فیلتر شده با بایاس -2 و تابع فعال سازی رلو:

1	1
1	1

خروجی فیلتر اول

1

خروجی maxpooling

1	1
5	5

خروجی فیلتر دوم

5

خروجی maxpooling

تصویر دوم از کلاس اول:

1	1	1	0
1	0	1	0
1	1	1	0
0	0	0	1

خروجی فیلتر شده با بایاس -2 و تابع فعال سازی رلو:

2	1
1	1

خروجی فیلتر اول

2

خروجی maxpooling

6	2
2	1

خروجی فیلتر دوم

6

خروجی maxpooling

تصویر اول از کلاس دوم:

0	0	0	0
0	1	1	1
0	0	1	0
0	0	0	0

خروجی فیلتر شده با بایاس -2 و تابع فعال سازی رلو:

0	2
0	0

خروجی فیلتر اول

2

خروجی maxpooling

0	1
1	1

خروجی فیلتر دوم

1

خروجی maxpooling

تصویر دوم از کلاس دوم:

0	0	0	1
0	1	0	0
1	1	1	1
0	1	0	0

خروجی فیلتر شده با بایاس -2 و تابع فعال سازی رلو:

0	0
3	1

خروجی فیلتر اول

3

خروجی maxpooling

1	3
2	2

خروجی فیلتر دوم

3

خروجی maxpooling

CIFAR10 1.2

Import کردن کتابخانه های مورد نیاز در طول پروژه:

```
!pip install torchmetrics
import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision
import torchvision.transforms as transforms
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from datetime import datetime
from keras.datasets import cifar10
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
import itertools
import time
import torch.utils.data as data_utils
from torchmetrics.classification import BinaryF1Score, BinaryAccuracy, BinaryPrecision
```

شکل 1. Libraries

دانلود مجموعه داده از کتابخانه Keras:

```
Downloading cifar10 Dataset (32x32)

[2] (x_train32, y_train32), (x_test32, y_test32) = cifar10.load_data()

    Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
    170498071/170498071 [==============================] - 2s 0us/step

[3] np.shape(x_train32)

    (50000, 32, 32, 3)
```

شکل 2. لود دیتاست

الف

مجموعه داده ای که دانلود کردیم به صورت دیفالت استایل 32x32 است.

پس استایل های 16x16 و 8x8 را به استفاده از tf.image.reshape() تهیه میکنیم.

16x16 resolution style

```
[8]  x_train16=tf.image.resize(
       images=x_train32,
       size=[16,16],
       method=tf.image.ResizeMethod.NEAREST_NEIGHBOR,
       preserve_aspect_ratio=False,
       antialias=False,
       name=None
       ).numpy()
     np.shape(x_train16)

     x_test16=tf.image.resize(
       images=x_test32,
       size=[16,16],
       method=tf.image.ResizeMethod.NEAREST_NEIGHBOR,
       preserve_aspect_ratio=False,
       antialias=False,
       name=None
       ).numpy()
     print(np.shape(x_test16))
```

```
(10000, 16, 16, 3)
```

شکل 3. استایل **16x16**

8x8 resolution style

```
x_train8=tf.image.resize(
  images=x_train32,
  size=[8,8],
  method=tf.image.ResizeMethod.NEAREST_NEIGHBOR,
  preserve_aspect_ratio=False,
  antialias=False,
  name=None
  ).numpy()
np.shape(x_train8)

x_test8=tf.image.resize(
  images=x_test32,
  size=[8,8],
  method=tf.image.ResizeMethod.NEAREST_NEIGHBOR,
  preserve_aspect_ratio=False,
  antialias=False,
  name=None
  ).numpy()
print(np.shape(x_test8))
```

```
(10000, 16, 16, 3)
```

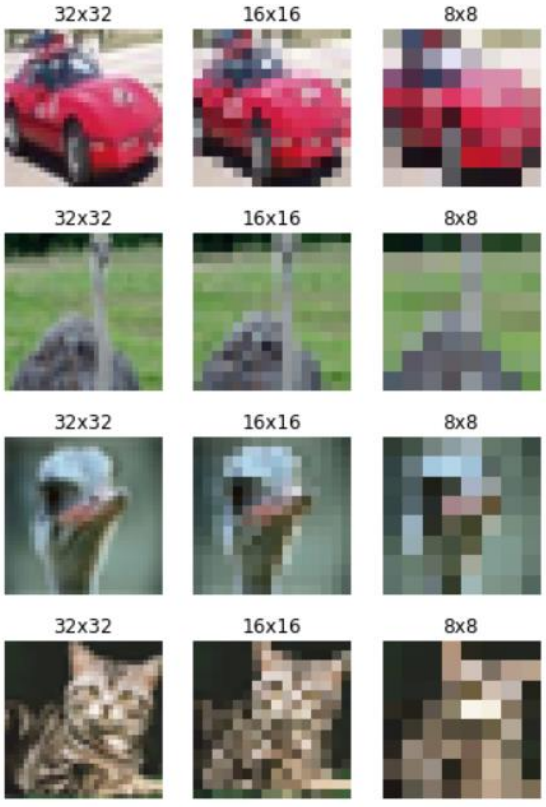
شکل 4. استایل **8x8**

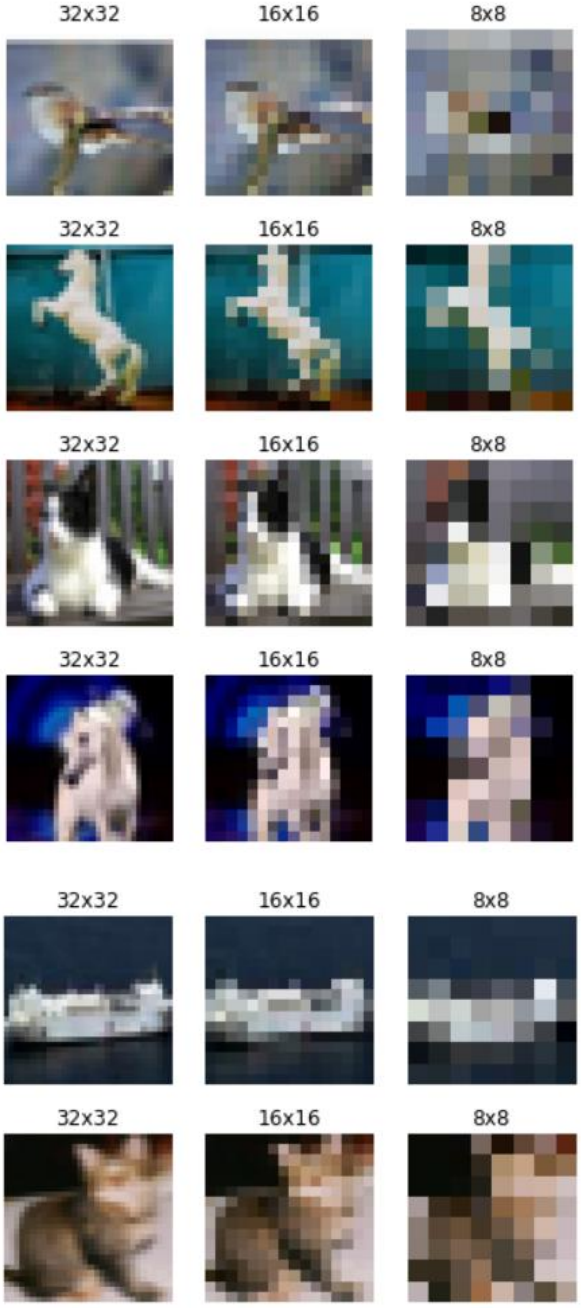
سپس 10 تصویر به را به صورت رندم در هر سه استایل نمایش میدهیم:

A) Showing different resolution styles for 10 random images

```
samples=np.random.randint(len(x_train32),size=(10))
for i,index in zip(range(10),samples):
  ax1=plt.subplot(1,3,1)
  plt.imshow(x_train32[index])
  plt.title("32x32")
  plt.axis('off')
  ax2=plt.subplot(1,3,2)
  plt.imshow(x_train16[index])
  plt.title("16x16")
  plt.axis('off')
  ax3=plt.subplot(1,3,3)
  plt.imshow(x_train8[index])
  plt.title("8x8")
  plt.axis('off')
  plt.show()
```

شکل 5. نمایش 10 تصویر رندم در 3 استایل متفاوت





شکل 6. نمایش 10 تصویر رندم در 3 استایل متفاوت**CIFAR10**

ب

دو روش برای تقسیم بندی داده ها به سه دسته آموزش، ارزیابی و تست وجود دارد:

1. Splitting Randomly:

شما نمی توانید عملکرد پیش بینی یک مدل را با همان داده هایی که برای آموزش استفاده کرده اید ارزیابی کنید. بهتر است مدل را با داده های جدیدی ارزیابی کنید که قبلاً توسط مدل دیده نشده است. تقسیم تصادفی داده ها رایج ترین روش مورد استفاده برای آن ارزیابی بایاس نشده است.

2. Splitting using the temporal component:

هر زمان که مجموعه داده شامل متغیر تاریخ باشد، و ما بخواهیم چیزی را در آینده پیش بینی کنیم، استفاده از متغیر زمانی روش قابل اعتماد تری برای تقسیم مجموعه داده ها است. از این رو باید از آخرین نمونه ها برای ایجاد مجموعه داده ارزیابی و تست استفاده کنیم.

برای هر دو روش TOTV و TVTV به داده های ارزیابی به 3 استایل 32x32, 16x16, 8x8 نیاز داریم.

از آنجا که معماری شبکه طراحی شده ورودی های 32x32 میگیرد، باید داده های 16x16 و 8x8 را دوباره به سایز 32x32 برسانیم.

```
x_train16=tf.image.resize(
  images=x_train16,
  size=[32,32],
  method=tf.image.ResizeMethod.NEAREST_NEIGHBOR,
  preserve_aspect_ratio=False,
  antialias=False,
  name=None
  ).numpy()
np.shape(x_train16)

x_test16=tf.image.resize(
  images=x_test16,
  size=[32,32],
  method=tf.image.ResizeMethod.NEAREST_NEIGHBOR,
  preserve_aspect_ratio=False,
  antialias=False,
  name=None
  ).numpy()
```

```
x_train8=tf.image.resize(
  images=x_train8,
  size=[32,32],
  method=tf.image.ResizeMethod.NEAREST_NEIGHBOR,
  preserve_aspect_ratio=False,
  antialias=False,
  name=None
  ).numpy()
np.shape(x_train8)

x_test8=tf.image.resize(
  images=x_test8,
  size=[32,32],
  method=tf.image.ResizeMethod.NEAREST_NEIGHBOR,
  preserve_aspect_ratio=False,
  antialias=False,
  name=None
  ).numpy()
```

شکل 7. تغییر سایز داده های **16x16, 8x8x** به **32x32**

سپس داده ها را به Tensor تبدیل میکنیم.

changing datatypes from numpy to tensor

```
#32x32
x_train32=torch.tensor(x_train32, dtype=torch.float32)
y_train32=torch.tensor(y_train32, dtype=torch.float32)

x_test32=torch.tensor(x_test32, dtype=torch.float32)
y_test32=torch.tensor(y_test32, dtype=torch.float32)

#16x16
x_train16=torch.tensor(x_train16, dtype=torch.float32)
y_train16=torch.tensor(y_train32, dtype=torch.float32)

x_test16=torch.tensor(x_test16, dtype=torch.float32)
y_test16=torch.tensor(y_test32, dtype=torch.float32)

#8x8
x_train8=torch.tensor(x_train8, dtype=torch.float32)
y_train8=torch.tensor(y_train32, dtype=torch.float32)

x_test8=torch.tensor(x_test8, dtype=torch.float32)
y_test8=torch.tensor(y_test32, dtype=torch.float32)
```

شکل 8. تبدیل داده ها از **numpy** به **tensor**

پس از آن یک کلاس دیتاست تعریف میکنیم تا بتوان با کمک آن داده ها را به Dataloader داده و داده ها را به صورت batch به شبکه داد. همچنین برای داده های آموزش داده ها را به صورت shuffle شده به شبکه میدهیم.

```
class Dataset():
  def __init__(self, x, y, k):
    self.x = x
    self.yy = np.zeros((self.x.shape[0], k))
    for i in range(self.x.shape[0]):
      self.yy[i][int(y[i])] = 1

  def __len__(self):
    return self.x.shape[0]

  def __getitem__(self, index):
    return self.x[index], self.yy[index]
```

شکل 9. کلاس دیتاست

```
K=len(np.unique(y_train32))

#32x32
training_set32=Dataset(x_train32,y_train32,K)
training_generator32 = torch.utils.data.DataLoader(training_set32,batch_size=2000,shuffle=True)

validation_set32=Dataset(x_test32,y_test32,K)
validation_generator32 = torch.utils.data.DataLoader(validation_set32,batch_size=2000,shuffle=False)

# 16x16
training_set16=Dataset(x_train16,y_train16,K)
training_generator16 = torch.utils.data.DataLoader(training_set16,batch_size=2000,shuffle=True)

validation_set16=Dataset(x_test16,y_test16,K)
validation_generator16 = torch.utils.data.DataLoader(validation_set16,batch_size=2000,shuffle=False)

# 8x8
training_set8=Dataset(x_train8,y_train8,K)
training_generator8 = torch.utils.data.DataLoader(training_set8,batch_size=2000,shuffle=True)

validation_set8=Dataset(x_test8,y_test8,K)
validation_generator8 = torch.utils.data.DataLoader(validation_set8,batch_size=2000,shuffle=False)
```

شکل 10. دادن دیتاها به **dataloader**

سپس شبکه CNN را مطابق با معماری مقاله طراحی میکنیم.

```
class CNN(nn.Module):
  def __init__(self,K):
    super(CNN,self).__init__()
    self.conv1 = nn.Conv2d(3,32,kernel_size=3)
    self.conv2 = nn.Conv2d(32,32,kernel_size=3)
    self.conv3 = nn.Conv2d(32,32,kernel_size=3)
    self.conv4 = nn.Conv2d(32,64,kernel_size=3)
    self.conv5 = nn.Conv2d(64,64,kernel_size=3)
    self.conv6 = nn.Conv2d(64,64,kernel_size=3)
    self.maxpool1=nn.MaxPool2d((2,2))
    self.maxpool2=nn.MaxPool2d((2,2))
    self.lin1 = nn.Linear(64 * 3 * 3, 512)
    self.lin2 = nn.Linear(512, K)
    self.drp1 = nn.Dropout(p=0.25)
    self.drp2 = nn.Dropout(p=0.5)
  def forward(self,x):
    x = F.relu(self.conv1(x))
    x = F.relu(self.conv2(x))
    x = F.relu(self.conv3(x))
    x = self.maxpool1(x)
    x = self.drp1(x)#
    x = F.relu(self.conv4(x))
    x = F.relu(self.conv5(x))
    x = F.relu(self.conv6(x))
    x = self.maxpool2(x)
    x = self.drp1(x)
    x = x.contiguous().view(-1,64 * 3 * 3)
    x = F.relu(self.lin1(x))
    x = self.drp2(x)
    x = self.lin2(x)
    return x
```

شکل 11. تعریف معماری شبکه

مدل های مجزا برای حالات داده های تست و ترین (32x32, 16x16, 8x8) تعریف میکنیم.

```
Defining needed models for both TVTV and TOTV methods

cifar10_model=CNN(K)
cifar10_model16=CNN(K)
cifar10_model8=CNN(K)
cifar10_model3216=CNN(K)
cifar10_model328=CNN(K)
```

شکل 12. مدل‌های مورد نیاز برای **TOVT**

مدل ها را روی gpu ترین میکنیم.

```
Use gpu for Trainnig

use_cuda = torch.cuda.is_available()
device = torch.device("cuda:0" if use_cuda else "cpu")
torch.backends.cudnn.benchmark = True
print(device)
cifar10_model.to(device)
cifar10_model16.to(device)
cifar10_model8.to(device)
cifar10_model3216.to(device)
cifar10_model328.to(device)

cuda:0
CNN(
  (conv1): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1))
  (conv2): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
  (conv3): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
  (conv4): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
  (conv5): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1))
  (conv6): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1))
  (maxpool1): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1, ceil_mode=False)
  (maxpool2): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1, ceil_mode=False)
  (lin1): Linear(in_features=576, out_features=512, bias=True)
  (lin2): Linear(in_features=512, out_features=10, bias=True)
  (drp1): Dropout(p=0.25, inplace=False)
  (drp2): Dropout(p=0.5, inplace=False)
)
```

شکل 13. Train بر روی **GPU**

توابع optimization و loss را تعریف میکنیم.

```
Optimization and criterion for both TVTV and TOTV methods

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(cifar10_model.parameters(),lr=0.001)
optimizer16 = torch.optim.Adam(cifar10_model16.parameters(),lr=0.001)
optimizer8 = torch.optim.Adam(cifar10_model8.parameters(),lr=0.001)
optimizer3216 = torch.optim.Adam(cifar10_model3216.parameters(),lr=0.001)
optimizer328 = torch.optim.Adam(cifar10_model328.parameters(),lr=0.001)
metricF1=BinaryF1Score()
metricAcc=BinaryAccuracy()
metricPre=BinaryPrecision()
softmax = nn.Softmax(dim=1)
```

شکل 14. توابع **optimizer** و **loss**

تابع training مدل را تعریف میکنیم.

Trainig the model

```
def train_loop(model, criterion, optimizer, train_loader, valid_loader, epochs):
  train_losses=np.zeros(epochs)
  valid_losses=np.zeros(epochs)
  Accuracy=np.zeros(epochs)

  for i in range(epochs):
    t0=datetime.now()
    train_loss=[]
    valid_loss=[]
    f1 = []
    accuracy = []
    precision=[]
```

```
  model.train()
  for datas, targets in train_loader:
    datas, targets= datas.to(device), targets.to(device)
    datas=torch.permute(datas,(0,3,1,2))
    optimizer.zero_grad()
    outputs=model(datas)
    loss=criterion(outputs, targets)
    loss.backward()
    optimizer.step()

    train_loss.append(loss.item())
  train_loss=np.mean(train_loss)
  with torch.no_grad():
    model.eval()
    for datas, targets in valid_loader:
      datas, targets= datas.to(device), targets.to(device)
      datas=torch.permute(datas,(0,3,1,2))
      outputs=model(datas)
      outputs = softmax(outputs)
      loss=criterion(outputs, targets)
      valid_loss.append(loss.item())
      f1.append(metricF1(outputs.cpu(),targets.cpu()))
      accuracy.append(metricAcc(outputs.cpu(),targets.cpu()))
      precision.append(metricPre(outputs.cpu(),targets.cpu()))

  valid_loss=np.mean(valid_loss)
  f1=np.mean(f1)
  accuracy=np.mean(accuracy)
  precisionn=np.mean(precision)
```

```
  train_losses[i]=train_loss
  valid_losses[i]=valid_loss
  Accuracy[i]=accuracy

  delta_t=datetime.now()-t0

  print(f'epoch {i+1}/{epochs}, train loss: {train_loss:.4f}, \
        validation loss: {valid_loss:.4f}, duration: {delta_t}')
  print(f'epoch {i+1}/{epochs}, f1: {f1:.4f}, \
        accuracy: {accuracy:.4f}, precision: {precisionn:.4f}')

return train_losses, valid_losses, Accuracy
```

شکل 15. تابع **training**

ج

روش TOTV

```
TOTV and TVTV 32x32

                (variable) valid_losses32: ndarray
train_losses32, valid_losses32, accuracy32 = train_loop(cifar10_model, criterion, optimizer,
                                                        training_generator32,validation_generator32, epochs=75)
```

شکل 16. مدل **TOTV** و **TVTV** با داده های آموزش **32x32** و تست **32x32**

```
epoch 1/75, f1: 0.0135,          accuracy: 0.9003,          precision: 0.6509
epoch 2/75, f1: 0.0569,          accuracy: 0.9013,          precision: 0.6313
epoch 3/75, f1: 0.2101,          accuracy: 0.9050,          precision: 0.6222
epoch 4/75, f1: 0.2932,          accuracy: 0.9082,          precision: 0.6374
epoch 5/75, f1: 0.3671,          accuracy: 0.9132,          precision: 0.6775
epoch 6/75, f1: 0.3991,          accuracy: 0.9174,          precision: 0.7327
epoch 7/75, f1: 0.4501,          accuracy: 0.9216,          precision: 0.7541
epoch 8/75, f1: 0.4770,          accuracy: 0.9225,          precision: 0.7323
epoch 9/75, f1: 0.5135,          accuracy: 0.9247,          precision: 0.7258
epoch 10/75, f1: 0.5193,          accuracy: 0.9265,          precision: 0.7506
epoch 11/75, f1: 0.5518,          accuracy: 0.9299,          precision: 0.7649
epoch 12/75, f1: 0.5591,          accuracy: 0.9311,          precision: 0.7754
epoch 13/75, f1: 0.5803,          accuracy: 0.9333,          precision: 0.7833
epoch 14/75, f1: 0.5799,          accuracy: 0.9334,          precision: 0.7853
epoch 15/75, f1: 0.6151,          accuracy: 0.9362,          precision: 0.7754
epoch 16/75, f1: 0.5892,          accuracy: 0.9338,          precision: 0.7758
epoch 17/75, f1: 0.6245,          accuracy: 0.9369,          precision: 0.7711
epoch 18/75, f1: 0.6376,          accuracy: 0.9396,          precision: 0.7961
epoch 19/75, f1: 0.6463,          accuracy: 0.9412,          precision: 0.8118
epoch 20/75, f1: 0.6615,          accuracy: 0.9424,          precision: 0.8021
epoch 21/75, f1: 0.6582,          accuracy: 0.9426,          precision: 0.8134
epoch 22/75, f1: 0.6770,          accuracy: 0.9446,          precision: 0.8122
epoch 23/75, f1: 0.6809,          accuracy: 0.9447,          precision: 0.8051
epoch 24/75, f1: 0.6743,          accuracy: 0.9440,          precision: 0.8051
epoch 25/75, f1: 0.6920,          accuracy: 0.9463,          precision: 0.8115
epoch 26/75, f1: 0.6969,          accuracy: 0.9462,          precision: 0.7973
```

```
epoch 27/75, f1: 0.6957,          accuracy: 0.9470,          precision: 0.8156
epoch 28/75, f1: 0.7013,          accuracy: 0.9474,          precision: 0.8116
epoch 29/75, f1: 0.7118,          accuracy: 0.9491,          precision: 0.8211
epoch 30/75, f1: 0.7182,          accuracy: 0.9497,          precision: 0.8160
epoch 31/75, f1: 0.7202,          accuracy: 0.9503,          precision: 0.8243
epoch 32/75, f1: 0.7066,          accuracy: 0.9485,          precision: 0.8219
epoch 33/75, f1: 0.7211,          accuracy: 0.9505,          precision: 0.8249
epoch 34/75, f1: 0.7183,          accuracy: 0.9500,          precision: 0.8223
epoch 35/75, f1: 0.7164,          accuracy: 0.9495,          precision: 0.8166
epoch 36/75, f1: 0.7197,          accuracy: 0.9500,          precision: 0.8179
epoch 37/75, f1: 0.7234,          accuracy: 0.9504,          precision: 0.8187
epoch 38/75, f1: 0.7357,          accuracy: 0.9523,          precision: 0.8253
epoch 39/75, f1: 0.7369,          accuracy: 0.9530,          precision: 0.8370
epoch 40/75, f1: 0.7301,          accuracy: 0.9512,          precision: 0.8159
epoch 41/75, f1: 0.7396,          accuracy: 0.9529,          precision: 0.8274
epoch 42/75, f1: 0.7404,          accuracy: 0.9530,          precision: 0.8277
epoch 43/75, f1: 0.7469,          accuracy: 0.9541,          precision: 0.8334
epoch 44/75, f1: 0.7388,          accuracy: 0.9528,          precision: 0.8264
epoch 45/75, f1: 0.7512,          accuracy: 0.9548,          precision: 0.8347
epoch 46/75, f1: 0.7407,          accuracy: 0.9530,          precision: 0.8258
epoch 47/75, f1: 0.7488,          accuracy: 0.9540,          precision: 0.8258
epoch 48/75, f1: 0.7484,          accuracy: 0.9540,          precision: 0.8256
epoch 49/75, f1: 0.7461,          accuracy: 0.9536,          precision: 0.8247
epoch 50/75, f1: 0.7552,          accuracy: 0.9552,          precision: 0.8330
epoch 51/75, f1: 0.7488,          accuracy: 0.9540,          precision: 0.8241
epoch 52/75, f1: 0.7395,          accuracy: 0.9527,          precision: 0.8222
epoch 53/75, f1: 0.7571,          accuracy: 0.9556,          precision: 0.8365
epoch 54/75, f1: 0.7516,          accuracy: 0.9545,          precision: 0.8281
epoch 55/75, f1: 0.7543,          accuracy: 0.9547,          precision: 0.8251
epoch 56/75, f1: 0.7592,          accuracy: 0.9557,          precision: 0.8318
epoch 57/75, f1: 0.7605,          accuracy: 0.9558,          precision: 0.8302
epoch 58/75, f1: 0.7580,          accuracy: 0.9555,          precision: 0.8300
epoch 59/75, f1: 0.7518,          accuracy: 0.9547,          precision: 0.8325
epoch 60/75, f1: 0.7558,          accuracy: 0.9550,          precision: 0.8266
epoch 61/75, f1: 0.7588,          accuracy: 0.9554,          precision: 0.8266
```

```
epoch 62/75, f1: 0.7541,          accuracy: 0.9548,          precision: 0.8260
epoch 63/75, f1: 0.7637,          accuracy: 0.9560,          precision: 0.8252
epoch 64/75, f1: 0.7576,          accuracy: 0.9552,          precision: 0.8251
epoch 65/75, f1: 0.7641,          accuracy: 0.9562,          precision: 0.8289
epoch 66/75, f1: 0.7704,          accuracy: 0.9574,          precision: 0.8352
epoch 67/75, f1: 0.7644,          accuracy: 0.9561,          precision: 0.8247
epoch 68/75, f1: 0.7657,          accuracy: 0.9562,          precision: 0.8222
epoch 69/75, f1: 0.7697,          accuracy: 0.9572,          precision: 0.8333
epoch 70/75, f1: 0.7731,          accuracy: 0.9575,          precision: 0.8286
epoch 71/75, f1: 0.7600,          accuracy: 0.9559,          precision: 0.8334
epoch 72/75, f1: 0.7708,          accuracy: 0.9571,          precision: 0.8276
epoch 73/75, f1: 0.7717,          accuracy: 0.9573,          precision: 0.8289
epoch 74/75, f1: 0.7672,          accuracy: 0.9565,          precision: 0.8243
epoch 75/75, f1: 0.7698,          accuracy: 0.9568,          precision: 0.8245
```

شکل 17. نتایج مدل **TOTV** و **TVTV** با داده های آموزش **32x32** و تست **32x32**

TOTV 32x32->16x16

```
train_losses_32, valid_losses_16, accuracy2 = train_loop(cifar10_model3216, criterion, optimizer3216,
                                    training_generator32,validation_generator16, epochs=100)
```

شکل 18. مدل **TOTV** با داده های آموزش **32x32** و تست **16x16**

```
epoch 1/100, f1: 0.0002,          accuracy: 0.9000,          precision: 0.2000
epoch 2/100, f1: 0.1362,          accuracy: 0.9025,          precision: 0.5966
epoch 3/100, f1: 0.2283,          accuracy: 0.9061,          precision: 0.6413
epoch 4/100, f1: 0.2397,          accuracy: 0.9073,          precision: 0.6655
epoch 5/100, f1: 0.2595,          accuracy: 0.9096,          precision: 0.7190
epoch 6/100, f1: 0.3525,          accuracy: 0.9133,          precision: 0.6951
epoch 7/100, f1: 0.3944,          accuracy: 0.9148,          precision: 0.6818
epoch 8/100, f1: 0.4100,          accuracy: 0.9156,          precision: 0.6806
epoch 9/100, f1: 0.4405,          accuracy: 0.9163,          precision: 0.6644
epoch 10/100, f1: 0.4367,          accuracy: 0.9107,          precision: 0.5915
epoch 11/100, f1: 0.4737,          accuracy: 0.9195,          precision: 0.6840
epoch 12/100, f1: 0.4791,          accuracy: 0.9206,          precision: 0.6962
epoch 13/100, f1: 0.4531,          accuracy: 0.9114,          precision: 0.5921
epoch 14/100, f1: 0.4494,          accuracy: 0.9084,          precision: 0.5629
epoch 15/100, f1: 0.4717,          accuracy: 0.9125,          precision: 0.5955
epoch 16/100, f1: 0.5019,          accuracy: 0.9185,          precision: 0.6456
epoch 17/100, f1: 0.4653,          accuracy: 0.9114,          precision: 0.5863
epoch 18/100, f1: 0.4708,          accuracy: 0.9099,          precision: 0.5702
epoch 19/100, f1: 0.4718,          accuracy: 0.9097,          precision: 0.5682
epoch 20/100, f1: 0.4839,          accuracy: 0.9135,          precision: 0.6000
epoch 21/100, f1: 0.4982,          accuracy: 0.9174,          precision: 0.6351
epoch 22/100, f1: 0.4724,          accuracy: 0.9097,          precision: 0.5680
epoch 23/100, f1: 0.5018,          accuracy: 0.9169,          precision: 0.6264
epoch 24/100, f1: 0.4940,          accuracy: 0.9140,          precision: 0.5997
epoch 25/100, f1: 0.5123,          accuracy: 0.9168,          precision: 0.6196
epoch 26/100, f1: 0.4528,          accuracy: 0.9065,          precision: 0.5462
epoch 27/100, f1: 0.4635,          accuracy: 0.9065,          precision: 0.5435
epoch 28/100, f1: 0.4523,          accuracy: 0.9052,          precision: 0.5355
epoch 29/100, f1: 0.4854,          accuracy: 0.9113,          precision: 0.5782
epoch 30/100, f1: 0.4790,          accuracy: 0.9111,          precision: 0.5784
epoch 31/100, f1: 0.4704,          accuracy: 0.9071,          precision: 0.5469
epoch 32/100, f1: 0.4769,          accuracy: 0.9092,          precision: 0.5626
epoch 33/100, f1: 0.4751,          accuracy: 0.9075,          precision: 0.5490
```

```
epoch 34/100, f1: 0.4210,          accuracy: 0.8965,          precision: 0.4776
epoch 35/100, f1: 0.5090,          accuracy: 0.9157,          precision: 0.6099
epoch 36/100, f1: 0.4795,          accuracy: 0.9091,          precision: 0.5607
epoch 37/100, f1: 0.5026,          accuracy: 0.9141,          precision: 0.5970
epoch 38/100, f1: 0.4124,          accuracy: 0.8942,          precision: 0.4637
epoch 39/100, f1: 0.4334,          accuracy: 0.8979,          precision: 0.4868
epoch 40/100, f1: 0.4595,          accuracy: 0.9038,          precision: 0.5241
epoch 41/100, f1: 0.4709,          accuracy: 0.9050,          precision: 0.5314
epoch 42/100, f1: 0.4568,          accuracy: 0.9039,          precision: 0.5255
epoch 43/100, f1: 0.4590,          accuracy: 0.9028,          precision: 0.5175
epoch 44/100, f1: 0.4417,          accuracy: 0.8995,          precision: 0.4966
epoch 45/100, f1: 0.4635,          accuracy: 0.9047,          precision: 0.5306
epoch 46/100, f1: 0.4486,          accuracy: 0.9017,          precision: 0.5109
epoch 47/100, f1: 0.4343,          accuracy: 0.8984,          precision: 0.4897
epoch 48/100, f1: 0.4391,          accuracy: 0.8993,          precision: 0.4954
epoch 49/100, f1: 0.4425,          accuracy: 0.9001,          precision: 0.5008
epoch 50/100, f1: 0.4495,          accuracy: 0.9006,          precision: 0.5034
epoch 51/100, f1: 0.3963,          accuracy: 0.8894,          precision: 0.4362
epoch 52/100, f1: 0.4282,          accuracy: 0.8959,          precision: 0.4751
epoch 53/100, f1: 0.4545,          accuracy: 0.9019,          precision: 0.5116
epoch 54/100, f1: 0.3828,          accuracy: 0.8872,          precision: 0.4227
epoch 55/100, f1: 0.3977,          accuracy: 0.8898,          precision: 0.4387
epoch 56/100, f1: 0.4529,          accuracy: 0.9012,          precision: 0.5073
epoch 57/100, f1: 0.4049,          accuracy: 0.8922,          precision: 0.4521
epoch 58/100, f1: 0.4621,          accuracy: 0.9029,          precision: 0.5183
epoch 59/100, f1: 0.4403,          accuracy: 0.8989,          precision: 0.4932
epoch 60/100, f1: 0.4018,          accuracy: 0.8912,          precision: 0.4462
epoch 61/100, f1: 0.4332,          accuracy: 0.8965,          precision: 0.4787
epoch 62/100, f1: 0.4475,          accuracy: 0.9000,          precision: 0.4999
epoch 63/100, f1: 0.4022,          accuracy: 0.8899,          precision: 0.4402
epoch 64/100, f1: 0.4315,          accuracy: 0.8961,          precision: 0.4763
epoch 65/100, f1: 0.4396,          accuracy: 0.8972,          precision: 0.4835
epoch 66/100, f1: 0.4162,          accuracy: 0.8916,          precision: 0.4508
```

```
epoch 67/100, f1: 0.4601,          accuracy: 0.9011,          precision: 0.5067
epoch 68/100, f1: 0.4746,          accuracy: 0.9053,          precision: 0.5329
epoch 69/100, f1: 0.4190,          accuracy: 0.8927,          precision: 0.4569
epoch 70/100, f1: 0.4324,          accuracy: 0.8955,          precision: 0.4734
epoch 71/100, f1: 0.4478,          accuracy: 0.8994,          precision: 0.4961
epoch 72/100, f1: 0.4069,          accuracy: 0.8896,          precision: 0.4397
epoch 73/100, f1: 0.4263,          accuracy: 0.8950,          precision: 0.4701
epoch 74/100, f1: 0.4448,          accuracy: 0.8992,          precision: 0.4952
epoch 75/100, f1: 0.4802,          accuracy: 0.9051,          precision: 0.5310
```

شکل 19. نتایج مدل **TOTV** با داده های آموزش **32x32** و تست **16x16**

```
TOTV 32x32-> 8x8

Loading...
train_losses_32_, valid_losses_8, accuracy3 = train_loop(cifar10_model328, criterion, optimizer328,
                                    training_generator32,validation_generator8, epochs=75)
```

شکل 20. مدل **TOTV** با داده های آموزش **32x32** و تست **8x8**

```
epoch 1/100, f1: 0.0000,          accuracy: 0.9000,          precision: 0.0000
epoch 2/100, f1: 0.0087,          accuracy: 0.9002,          precision: 0.6193
epoch 3/100, f1: 0.0515,          accuracy: 0.9010,          precision: 0.6164
epoch 4/100, f1: 0.1979,          accuracy: 0.9034,          precision: 0.5832
epoch 5/100, f1: 0.2275,          accuracy: 0.9060,          precision: 0.6369
epoch 6/100, f1: 0.2685,          accuracy: 0.9071,          precision: 0.6323
epoch 7/100, f1: 0.3048,          accuracy: 0.9053,          precision: 0.5730
epoch 8/100, f1: 0.3354,          accuracy: 0.9062,          precision: 0.5752
epoch 9/100, f1: 0.3515,          accuracy: 0.9028,          precision: 0.5285
epoch 10/100, f1: 0.3434,          accuracy: 0.8962,          precision: 0.4674
epoch 11/100, f1: 0.3440,          accuracy: 0.8987,          precision: 0.4876
epoch 12/100, f1: 0.2976,          accuracy: 0.8819,          precision: 0.3673
epoch 13/100, f1: 0.3248,          accuracy: 0.8909,          precision: 0.4263
epoch 14/100, f1: 0.3356,          accuracy: 0.8941,          precision: 0.4504
epoch 15/100, f1: 0.3049,          accuracy: 0.8872,          precision: 0.3972
epoch 16/100, f1: 0.3377,          accuracy: 0.8942,          precision: 0.4515
epoch 17/100, f1: 0.2770,          accuracy: 0.8762,          precision: 0.3328
epoch 18/100, f1: 0.2883,          accuracy: 0.8784,          precision: 0.3474
epoch 19/100, f1: 0.2743,          accuracy: 0.8748,          precision: 0.3262
epoch 20/100, f1: 0.3252,          accuracy: 0.8854,          precision: 0.3954
epoch 21/100, f1: 0.3087,          accuracy: 0.8809,          precision: 0.3677
epoch 22/100, f1: 0.2767,          accuracy: 0.8750,          precision: 0.3281
epoch 23/100, f1: 0.2851,          accuracy: 0.8772,          precision: 0.3411
epoch 24/100, f1: 0.2618,          accuracy: 0.8680,          precision: 0.2969
epoch 25/100, f1: 0.3176,          accuracy: 0.8803,          precision: 0.3694
epoch 26/100, f1: 0.2832,          accuracy: 0.8712,          precision: 0.3191
epoch 27/100, f1: 0.2798,          accuracy: 0.8717,          precision: 0.3188
epoch 28/100, f1: 0.3108,          accuracy: 0.8791,          precision: 0.3614
epoch 29/100, f1: 0.2892,          accuracy: 0.8752,          precision: 0.3360
epoch 30/100, f1: 0.2861,          accuracy: 0.8725,          precision: 0.3250
epoch 31/100, f1: 0.2810,          accuracy: 0.8713,          precision: 0.3183
epoch 32/100, f1: 0.2738,          accuracy: 0.8708,          precision: 0.3125
epoch 33/100, f1: 0.2682,          accuracy: 0.8681,          precision: 0.3011
epoch 34/100, f1: 0.2526,          accuracy: 0.8630,          precision: 0.2780
```

```
epoch 34/100, f1: 0.2526,          accuracy: 0.8630,          precision: 0.2780
epoch 35/100, f1: 0.2747,          accuracy: 0.8679,          precision: 0.3046
epoch 36/100, f1: 0.2633,          accuracy: 0.8655,          precision: 0.2911
epoch 37/100, f1: 0.2597,          accuracy: 0.8637,          precision: 0.2842
epoch 38/100, f1: 0.2651,          accuracy: 0.8651,          precision: 0.2910
epoch 39/100, f1: 0.2395,          accuracy: 0.8584,          precision: 0.2587
epoch 40/100, f1: 0.2717,          accuracy: 0.8651,          precision: 0.2954
epoch 41/100, f1: 0.2787,          accuracy: 0.8678,          precision: 0.3066
epoch 42/100, f1: 0.2716,          accuracy: 0.8654,          precision: 0.2959
epoch 43/100, f1: 0.2627,          accuracy: 0.8634,          precision: 0.2853
epoch 44/100, f1: 0.2440,          accuracy: 0.8592,          precision: 0.2634
epoch 45/100, f1: 0.2320,          accuracy: 0.8572,          precision: 0.2510
epoch 46/100, f1: 0.2452,          accuracy: 0.8574,          precision: 0.2604
epoch 47/100, f1: 0.2515,          accuracy: 0.8596,          precision: 0.2693
epoch 48/100, f1: 0.2578,          accuracy: 0.8623,          precision: 0.2795
epoch 49/100, f1: 0.2398,          accuracy: 0.8581,          precision: 0.2582
epoch 50/100, f1: 0.2386,          accuracy: 0.8580,          precision: 0.2572
epoch 51/100, f1: 0.2470,          accuracy: 0.8583,          precision: 0.2637
epoch 52/100, f1: 0.2333,          accuracy: 0.8554,          precision: 0.2483
epoch 53/100, f1: 0.2457,          accuracy: 0.8585,          precision: 0.2632
epoch 54/100, f1: 0.2483,          accuracy: 0.8593,          precision: 0.2666
epoch 55/100, f1: 0.2530,          accuracy: 0.8595,          precision: 0.2701
epoch 56/100, f1: 0.2786,          accuracy: 0.8681,          precision: 0.3076
epoch 57/100, f1: 0.2409,          accuracy: 0.8562,          precision: 0.2551
epoch 58/100, f1: 0.2308,          accuracy: 0.8547,          precision: 0.2452
epoch 59/100, f1: 0.2428,          accuracy: 0.8566,          precision: 0.2573
epoch 60/100, f1: 0.2429,          accuracy: 0.8575,          precision: 0.2592
epoch 61/100, f1: 0.2586,          accuracy: 0.8599,          precision: 0.2747
epoch 62/100, f1: 0.2642,          accuracy: 0.8613,          precision: 0.2813
epoch 63/100, f1: 0.2353,          accuracy: 0.8549,          precision: 0.2487
epoch 64/100, f1: 0.2335,          accuracy: 0.8552,          precision: 0.2480
epoch 65/100, f1: 0.2374,          accuracy: 0.8541,          precision: 0.2486
epoch 66/100, f1: 0.2508,          accuracy: 0.8578,          precision: 0.2650
epoch 67/100, f1: 0.2376,          accuracy: 0.8568,          precision: 0.2541
```

```
epoch 68/100, f1: 0.2579,          accuracy: 0.8606,          precision: 0.2756
epoch 69/100, f1: 0.2313,          accuracy: 0.8534,          precision: 0.2431
epoch 70/100, f1: 0.2481,          accuracy: 0.8569,          precision: 0.2614
epoch 71/100, f1: 0.2326,          accuracy: 0.8533,          precision: 0.2438
epoch 72/100, f1: 0.2603,          accuracy: 0.8597,          precision: 0.2752
epoch 73/100, f1: 0.2379,          accuracy: 0.8550,          precision: 0.2508
epoch 74/100, f1: 0.2525,          accuracy: 0.8576,          precision: 0.2658
epoch 75/100, f1: 0.2628,          accuracy: 0.8606,          precision: 0.2790
```

شکل 21. نتایج مدل **TOTV** با داده های آموزش **32x32** و تست **8x8**

جدول 1. مقایسه نتایج استفاده از روش **TOVT**

CIFAR10 Dataset resolution	TOTV		
	Accuracy	Precision	F1 score
32x32	95.68%	82.45%	76.98%
16x16	90.51%	53.10%	48.02%
8x8	86.06%	27.9%	26.28%

د

روش TVTV:

```
TVTV 16x16

[33] train_losses16, valid_losses16, accuracy16 = train_loop(cifar10_model16, criterion, optimizer16,
                                                training_generator16,validation_generator16, epochs=100)
```

شکل 22. مدل **TVTV** با داده های آموزش و تست **16x16**

```
epoch 1/100, f1: 0.0018,          accuracy: 0.8999,          precision: 0.3690
epoch 2/100, f1: 0.1442,          accuracy: 0.9024,          precision: 0.5869
epoch 3/100, f1: 0.2510,          accuracy: 0.9078,          precision: 0.6703
epoch 4/100, f1: 0.3499,          accuracy: 0.9112,          precision: 0.6536
epoch 5/100, f1: 0.3801,          accuracy: 0.9157,          precision: 0.7167
epoch 6/100, f1: 0.4538,          accuracy: 0.9200,          precision: 0.7158
epoch 7/100, f1: 0.4574,          accuracy: 0.9211,          precision: 0.7330
epoch 8/100, f1: 0.4932,          accuracy: 0.9254,          precision: 0.7698
epoch 9/100, f1: 0.5183,          accuracy: 0.9263,          precision: 0.7484
epoch 10/100, f1: 0.5140,          accuracy: 0.9249,          precision: 0.7283
epoch 11/100, f1: 0.5490,          accuracy: 0.9305,          precision: 0.7817
epoch 12/100, f1: 0.5735,          accuracy: 0.9310,          precision: 0.7503
epoch 13/100, f1: 0.5649,          accuracy: 0.9315,          precision: 0.7745
epoch 14/100, f1: 0.5783,          accuracy: 0.9328,          precision: 0.7768
epoch 15/100, f1: 0.5841,          accuracy: 0.9322,          precision: 0.7557
epoch 16/100, f1: 0.5709,          accuracy: 0.9332,          precision: 0.7993
epoch 17/100, f1: 0.6051,          accuracy: 0.9351,          precision: 0.7720
epoch 18/100, f1: 0.5955,          accuracy: 0.9342,          precision: 0.7727
epoch 19/100, f1: 0.6046,          accuracy: 0.9349,          precision: 0.7706
epoch 20/100, f1: 0.6123,          accuracy: 0.9371,          precision: 0.7975
epoch 21/100, f1: 0.6231,          accuracy: 0.9376,          precision: 0.7872
epoch 22/100, f1: 0.6288,          accuracy: 0.9383,          precision: 0.7884
epoch 23/100, f1: 0.6306,          accuracy: 0.9388,          precision: 0.7948
epoch 24/100, f1: 0.6329,          accuracy: 0.9389,          precision: 0.7930
epoch 25/100, f1: 0.6452,          accuracy: 0.9401,          precision: 0.7912
epoch 26/100, f1: 0.6234,          accuracy: 0.9367,          precision: 0.7695
epoch 27/100, f1: 0.6480,          accuracy: 0.9407,          precision: 0.7974
epoch 28/100, f1: 0.6481,          accuracy: 0.9402,          precision: 0.7879
epoch 29/100, f1: 0.6520,          accuracy: 0.9404,          precision: 0.7832
epoch 30/100, f1: 0.6497,          accuracy: 0.9404,          precision: 0.7873
epoch 31/100, f1: 0.6669,          accuracy: 0.9423,          precision: 0.7895
epoch 32/100, f1: 0.6613,          accuracy: 0.9420,          precision: 0.7943
```

```
epoch 33/100, f1: 0.6576,          accuracy: 0.9417,          precision: 0.7970
epoch 34/100, f1: 0.6629,          accuracy: 0.9429,          precision: 0.8083
epoch 35/100, f1: 0.6666,          accuracy: 0.9424,          precision: 0.7911
epoch 36/100, f1: 0.6679,          accuracy: 0.9422,          precision: 0.7859
epoch 37/100, f1: 0.6685,          accuracy: 0.9422,          precision: 0.7844
epoch 38/100, f1: 0.6704,          accuracy: 0.9432,          precision: 0.7995
epoch 39/100, f1: 0.6724,          accuracy: 0.9431,          precision: 0.7925
epoch 40/100, f1: 0.6746,          accuracy: 0.9436,          precision: 0.7971
epoch 41/100, f1: 0.6730,          accuracy: 0.9434,          precision: 0.7963
epoch 42/100, f1: 0.6707,          accuracy: 0.9427,          precision: 0.7877
epoch 43/100, f1: 0.6825,          accuracy: 0.9440,          precision: 0.7889
epoch 44/100, f1: 0.6838,          accuracy: 0.9438,          precision: 0.7818
epoch 45/100, f1: 0.6863,          accuracy: 0.9450,          precision: 0.7986
epoch 46/100, f1: 0.6875,          accuracy: 0.9447,          precision: 0.7912
epoch 47/100, f1: 0.6780,          accuracy: 0.9431,          precision: 0.7804
epoch 48/100, f1: 0.6900,          accuracy: 0.9447,          precision: 0.7846
epoch 49/100, f1: 0.6876,          accuracy: 0.9443,          precision: 0.7834
epoch 50/100, f1: 0.6835,          accuracy: 0.9448,          precision: 0.8007
epoch 51/100, f1: 0.6876,          accuracy: 0.9454,          precision: 0.8029
epoch 52/100, f1: 0.6845,          accuracy: 0.9451,          precision: 0.8049
epoch 53/100, f1: 0.6859,          accuracy: 0.9445,          precision: 0.7904
epoch 54/100, f1: 0.6854,          accuracy: 0.9452,          precision: 0.8046
epoch 55/100, f1: 0.6847,          accuracy: 0.9444,          precision: 0.7905
epoch 56/100, f1: 0.6929,          accuracy: 0.9458,          precision: 0.7996
epoch 57/100, f1: 0.6919,          accuracy: 0.9454,          precision: 0.7947
epoch 58/100, f1: 0.6864,          accuracy: 0.9448,          precision: 0.7956
epoch 59/100, f1: 0.6871,          accuracy: 0.9449,          precision: 0.7960
epoch 60/100, f1: 0.6928,          accuracy: 0.9455,          precision: 0.7946
epoch 61/100, f1: 0.6977,          accuracy: 0.9458,          precision: 0.7890
epoch 62/100, f1: 0.6871,          accuracy: 0.9445,          precision: 0.7883
epoch 63/100, f1: 0.6933,          accuracy: 0.9454,          precision: 0.7917
```

```
epoch 64/100, f1: 0.6924,          accuracy: 0.9453,          precision: 0.7906
epoch 65/100, f1: 0.6873,          accuracy: 0.9444,          precision: 0.7849
epoch 66/100, f1: 0.6988,          accuracy: 0.9462,          precision: 0.7937
epoch 67/100, f1: 0.6967,          accuracy: 0.9454,          precision: 0.7843
epoch 68/100, f1: 0.7006,          accuracy: 0.9463,          precision: 0.7920
epoch 69/100, f1: 0.6960,          accuracy: 0.9456,          precision: 0.7884
epoch 70/100, f1: 0.6929,          accuracy: 0.9452,          precision: 0.7889
epoch 71/100, f1: 0.6989,          accuracy: 0.9459,          precision: 0.7888
epoch 72/100, f1: 0.6981,          accuracy: 0.9459,          precision: 0.7906
epoch 73/100, f1: 0.7045,          accuracy: 0.9469,          precision: 0.7944
epoch 74/100, f1: 0.6991,          accuracy: 0.9457,          precision: 0.7833
epoch 75/100, f1: 0.6961,          accuracy: 0.9454,          precision: 0.7854
```

شکل 23. نتایج مدل **TVTV** با داده های آموزش و تست **16x16**

TVTV 8x8

```
train_losses8, valid_losses8, accuracy8 = train_loop(cifar10_model8, criterion, optimizer8,
                                                     training_generator8,validation_generator8, epochs=100)
```

شکل 24. مدل **TVTV** با داده های آموزش و تست **8x8**

```
epoch 1/100, f1: 0.0002,          accuracy: 0.9000,          precision: 0.2000
epoch 2/100, f1: 0.0279,          accuracy: 0.9005,          precision: 0.6119
epoch 3/100, f1: 0.1299,          accuracy: 0.9013,          precision: 0.5496
epoch 4/100, f1: 0.2380,          accuracy: 0.9062,          precision: 0.6351
epoch 5/100, f1: 0.2087,          accuracy: 0.9069,          precision: 0.6929
epoch 6/100, f1: 0.2544,          accuracy: 0.9085,          precision: 0.6868
epoch 7/100, f1: 0.2954,          accuracy: 0.9101,          precision: 0.6821
epoch 8/100, f1: 0.3306,          accuracy: 0.9121,          precision: 0.6940
epoch 9/100, f1: 0.3513,          accuracy: 0.9140,          precision: 0.7141
epoch 10/100, f1: 0.3599,          accuracy: 0.9144,          precision: 0.7129
epoch 11/100, f1: 0.3900,          accuracy: 0.9159,          precision: 0.7089
epoch 12/100, f1: 0.4180,          accuracy: 0.9157,          precision: 0.6755
epoch 13/100, f1: 0.4019,          accuracy: 0.9175,          precision: 0.7311
epoch 14/100, f1: 0.4580,          accuracy: 0.9201,          precision: 0.7116
epoch 15/100, f1: 0.4364,          accuracy: 0.9188,          precision: 0.7133
epoch 16/100, f1: 0.4484,          accuracy: 0.9205,          precision: 0.7311
epoch 17/100, f1: 0.4753,          accuracy: 0.9217,          precision: 0.7200
epoch 18/100, f1: 0.4569,          accuracy: 0.9217,          precision: 0.7447
epoch 19/100, f1: 0.4782,          accuracy: 0.9224,          precision: 0.7290
epoch 20/100, f1: 0.4730,          accuracy: 0.9220,          precision: 0.7298
epoch 21/100, f1: 0.4855,          accuracy: 0.9237,          precision: 0.7454
epoch 22/100, f1: 0.4965,          accuracy: 0.9242,          precision: 0.7391
epoch 23/100, f1: 0.5044,          accuracy: 0.9245,          precision: 0.7344
epoch 24/100, f1: 0.5035,          accuracy: 0.9243,          precision: 0.7310
epoch 25/100, f1: 0.5090,          accuracy: 0.9252,          precision: 0.7403
epoch 26/100, f1: 0.5049,          accuracy: 0.9250,          precision: 0.7433
epoch 27/100, f1: 0.5187,          accuracy: 0.9263,          precision: 0.7475
epoch 28/100, f1: 0.5240,          accuracy: 0.9262,          precision: 0.7372
epoch 29/100, f1: 0.5181,          accuracy: 0.9259,          precision: 0.7401
epoch 30/100, f1: 0.5241,          accuracy: 0.9255,          precision: 0.7251
epoch 31/100, f1: 0.5381,          accuracy: 0.9264,          precision: 0.7230
epoch 32/100, f1: 0.5374,          accuracy: 0.9271,          precision: 0.7358
epoch 33/100, f1: 0.5363,          accuracy: 0.9276,          precision: 0.7456
epoch 34/100, f1: 0.5442,          accuracy: 0.9278,          precision: 0.7388
```

```
epoch 35/100, f1: 0.5444,          accuracy: 0.9271,          precision: 0.7253
epoch 36/100, f1: 0.5397,          accuracy: 0.9272,          precision: 0.7340
epoch 37/100, f1: 0.5434,          accuracy: 0.9281,          precision: 0.7446
epoch 38/100, f1: 0.5463,          accuracy: 0.9279,          precision: 0.7359
epoch 39/100, f1: 0.5512,          accuracy: 0.9279,          precision: 0.7294
epoch 40/100, f1: 0.5586,          accuracy: 0.9278,          precision: 0.7190
epoch 41/100, f1: 0.5519,          accuracy: 0.9279,          precision: 0.7289
epoch 42/100, f1: 0.5477,          accuracy: 0.9278,          precision: 0.7336
epoch 43/100, f1: 0.5635,          accuracy: 0.9293,          precision: 0.7369
epoch 44/100, f1: 0.5531,          accuracy: 0.9279,          precision: 0.7281
epoch 45/100, f1: 0.5622,          accuracy: 0.9282,          precision: 0.7203
epoch 46/100, f1: 0.5519,          accuracy: 0.9277,          precision: 0.7264
epoch 47/100, f1: 0.5591,          accuracy: 0.9278,          precision: 0.7186
epoch 48/100, f1: 0.5620,          accuracy: 0.9284,          precision: 0.7241
epoch 49/100, f1: 0.5635,          accuracy: 0.9289,          precision: 0.7305
epoch 50/100, f1: 0.5702,          accuracy: 0.9296,          precision: 0.7314
epoch 51/100, f1: 0.5580,          accuracy: 0.9286,          precision: 0.7318
epoch 52/100, f1: 0.5626,          accuracy: 0.9288,          precision: 0.7293
epoch 53/100, f1: 0.5626,          accuracy: 0.9281,          precision: 0.7174
epoch 54/100, f1: 0.5629,          accuracy: 0.9286,          precision: 0.7262
epoch 55/100, f1: 0.5705,          accuracy: 0.9289,          precision: 0.7199
epoch 56/100, f1: 0.5655,          accuracy: 0.9292,          precision: 0.7315
epoch 57/100, f1: 0.5643,          accuracy: 0.9288,          precision: 0.7273
epoch 58/100, f1: 0.5615,          accuracy: 0.9281,          precision: 0.7194
epoch 59/100, f1: 0.5703,          accuracy: 0.9290,          precision: 0.7229
epoch 60/100, f1: 0.5660,          accuracy: 0.9287,          precision: 0.7226
epoch 61/100, f1: 0.5777,          accuracy: 0.9293,          precision: 0.7170
epoch 62/100, f1: 0.5616,          accuracy: 0.9280,          precision: 0.7176
epoch 63/100, f1: 0.5749,          accuracy: 0.9296,          precision: 0.7248
epoch 64/100, f1: 0.5723,          accuracy: 0.9285,          precision: 0.7126
epoch 65/100, f1: 0.5562,          accuracy: 0.9271,          precision: 0.7111
epoch 66/100, f1: 0.5695,          accuracy: 0.9281,          precision: 0.7093
epoch 67/100, f1: 0.5801,          accuracy: 0.9296,          precision: 0.7180
epoch 68/100, f1: 0.5744,          accuracy: 0.9295,          precision: 0.7242
```

```
epoch 69/100, f1: 0.5669,          accuracy: 0.9284,          precision: 0.7171
epoch 70/100, f1: 0.5692,          accuracy: 0.9280,          precision: 0.7090
epoch 71/100, f1: 0.5754,          accuracy: 0.9285,          precision: 0.7083
epoch 72/100, f1: 0.5693,          accuracy: 0.9283,          precision: 0.7121
epoch 73/100, f1: 0.5761,          accuracy: 0.9288,          precision: 0.7123
epoch 74/100, f1: 0.5777,          accuracy: 0.9295,          precision: 0.7195
epoch 75/100, f1: 0.5750,          accuracy: 0.9283,          precision: 0.7062
```

شکل 25. نتایج مدل **TVTV** با داده های آموزش و تست **8x8**

جدول 2. مقایسه نتایج استفاده از روش **TVTV**

CIFAR10 Dataset resolution	TVTV		
	Accuracy	Precision	F1 score
32x32	95.68%	82.45%	76.98%
16x16	94.54%	78.54%	69.61%
8x8	92.83%	70.62%	57.50%

پاسخ ۲ – آشنایی با معماری CNN

2-1. لود دیتاست مقاله

در مقاله "CNN Model for Image Classification on MNIST and Fashion Dataset"، 5 معماری مختلف برای شبکه CNN معرفی شده است. برای Load کردن داده‌های Fashion MNIST، از کتابخانه keras.dataset استفاده می‌کنیم:

```
!pip install torchmetrics

import gc
import torch
import numpy as np
import torch.nn as nn
import matplotlib.pyplot as plt
import torch.nn.functional as F
from keras.datasets import fashion_mnist
from torch.utils.data import DataLoader, Dataset
from torchmetrics.classification import BinaryF1Score
from torchmetrics.classification import BinaryAccuracy
from torchmetrics.classification import BinaryPrecision
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting torchmetrics
  Downloading torchmetrics-0.10.3-py3-none-any.whl (529 kB)
     |████████████████████████████████| 529 kB 28.5 MB/s
Requirement already satisfied: torch>=1.3.1 in /usr/local/lib/python3.7/dist-packages (from torchmetrics) (1.12.1+cu113)
Requirement already satisfied: packaging in /usr/local/lib/python3.7/dist-packages (from torchmetrics) (21.3)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-packages (from torchmetrics) (4.1.1)
Requirement already satisfied: numpy>=1.17.2 in /usr/local/lib/python3.7/dist-packages (from torchmetrics) (1.21.6)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.7/dist-packages (from packaging->torchm
Installing collected packages: torchmetrics
Successfully installed torchmetrics-0.10.3
```

شکل 26. Import کردن کتابخانه های مختلف

بهتر است قبل از اجرای برنامه در colab، Cache را پاک کنیم:

```
torch.cuda.empty_cache()
gc.collect()
torch.cuda.memory_summary(device=None, abbreviated=False)
```

```
'|===========================================================
---------------------------------------------------------
===========================================================
---------------------------------------------------------
m large pool |       0 B  |       0 B  |       0 B  |       0
---------------------------------------------------------
e pool |       0 B  |       0 B  | ...'
```

شکل 27. پاک کردن **cache** در **Calab**

سپس داده‌ها را ازکتابخانه keras.dataset، وارد برنامه می‌کنیم:

```
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()

x_train = torch.tensor(x_train, dtype=torch.float32)
y_train = torch.tensor(y_train, dtype=torch.float32)

x_test = torch.tensor(x_test, dtype=torch.float32)
y_test = torch.tensor(y_test, dtype=torch.float32)

Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
29515/29515 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26421880/26421880 [==============================] - 2s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
5148/5148 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4422102/4422102 [==============================] - 0s 0us/step
```

شکل 28. Load کردن داده‌ها

Datatype های داده‌هایی که وارد برنامه کرده ایم، از نوع torch هستند. به همین دلیل با کمک کلاسی به نام My_Data، 3 تابع __init__، __len__ و __getitem__ را تعریف می‌کنیم:

```
class My_Data():
  def __init__(self, x, y, k):
    self.x = x
    self.z = np.zeros((self.x.shape[0], k))
    for i in range(self.x.shape[0]):
      self.z[i][int(y[i])] = 1

  def __len__(self):
    return self.x.shape[0]

  def __getitem__(self, index):
    return self.x[index], self.z[index]
```

شکل 29. تعریف توابع **__init__**، **__len__** و **__getitem__** در کلاس **My_Data**

2-2. انتخاب معماری

از مقاله، معماری‌های دوم و چهارم را انتخاب می‌کنیم. Optimum Parameter در معماری دوم و چهارم، batchsizeهای 128تایی، تابع فعالسازی softmax، Adam Optimizer، droupout 0.25، 50 اپک و ابعاد کرنل 2*2 است:

```python
class CNN_Arc2(nn.Module):
  def __init__(self, K):
    super(CNN_Arc2, self).__init__()

    self.conv1 = nn.Conv2d(1, 64, kernel_size = 2)
    self.conv2 = nn.Conv2d(64, 64, kernel_size = 2)
    self.maxpool = nn.MaxPool2d((2, 2))
    self.dropout = nn.Dropout(p = 0.25)
    self.linear1 = nn.Linear(9216, 64)
    self.linear2 = nn.Linear(64, 10)

  def forward(self, x):
    x = F.softmax(self.conv1(x), dim = 1)
    x = self.maxpool(x)
    x = self.dropout(x)

    x = F.softmax(self.conv2(x), dim = 1)
    x = self.dropout(x)

    x = x.view(x.size(0), -1)
    x=self.linear1(x)
    x = self.dropout(x)

    x = self.linear2(x)
    return x
```

شکل 30. معماری دوم

```python
class CNN_Arc4(nn.Module):
  def __init__(self, K):
    super(CNN_Arc4, self).__init__()

    self.conv1 = nn.Conv2d(1, 64, kernel_size = 2)
    self.conv2 = nn.Conv2d(64, 64, kernel_size = 2)
    self.maxpool = nn.MaxPool2d((2, 2))
    self.dropout = nn.Dropout(p = 0.1)
    self.linear1 = nn.Linear(64,64)
    self.linear2 = nn.Linear(64,10)

  def forward(self, x):
    x = F.softmax(self.conv1(x), dim = 1)
    x = self.maxpool(x)
    x = self.dropout(x)

    x = F.softmax(self.conv2(x), dim = 1)
    x = self.maxpool(x)
    x = self.dropout(x)

    x = F.softmax(self.conv2(x), dim = 1)
    x = self.maxpool(x)
    x = self.dropout(x)

    x = F.softmax(self.conv2(x), dim = 1)
    x = self.dropout(x)

    x = x.view(x.size(0), -1)
    x = self.linear1(x)
    x = self.dropout(x)

    x = self.linear2(x)
    return x
```

شکل 31. معماری چهارم

2-3. توضیح لایه‌های مختلف معماری

معماری دوم، از 3 لایه ساخته شده است. در لایه اول، ورودی‌ها با ابعاد 1x64 از یک فیلتر softmax با سایز کرنل 2x2، عبور می‌کنند. سپس تابع maxpool، داده‌ها را به ابعاد 64x64 تبدیل کرده و تابع dropout، 25% داده‌ها را عبور می‌دهد. در لایه دوم، داده‌ها با ابعاد 64x64 از فیلتر softmax عبور می‌کند و دوباره 25% داده‌ها با تابع dropout وارد لایه بعدی می‌شود. در لایه آخر، ورودی‌ها را flat کرده که به ابعاد 9216x64 تبدیل می‌شود. مجددا 25% این داده‌ها از تابع dropout عبور کرده و به ابعاد 64x10 در می‌آید که درواقع به 10 کلاس تقسیم می‌شود.

معماری چهارم، از 5 لایه ساخته شده است. در لایه اول، ورودی‌ها با ابعاد 1x64 از یک فیلتر softmax با سایز کرنل 2x2، عبور می‌کنند. سپس تابع maxpool، داده‌ها را به ابعاد 64x64 تبدیل کرده و تابع dropout، 25% داده‌ها را عبور می‌دهد. در لایه دوم و سوم، داده‌ها با ابعاد 64x64 از فیلتر softmax عبور می‌کند و دوباره 25% داده‌ها با تابع dropout وارد لایه بعدی می‌شود. در لایه چهارم، داده‌های لایه قبلی، از فیلتر softmax عبور کرده و 25% آنها وارد لایه بعدی می‌شوند. در لایه آخر، ورودی‌ها را flat کرده که به ابعاد 64x64 تبدیل می‌شود. مجددا 25% این داده‌ها از تابع dropout عبور کرده و به ابعاد 64x10 در می‌آید که درواقع به 10 کلاس تقسیم می‌شود.

2-4. مقایسه نتایج دو معماری مختلف

برای training داده‌ها، ابتدا داده‌های train و test را وارد dataloader می‌کنیم:

```
train_data = My_Data(x_train, y_train, 10)
test_data = My_Data(x_test, y_test, 10)

train = torch.utils.data.DataLoader(train_data, 128, True)
test = torch.utils.data.DataLoader(test_data, 128, False)
```

شکل 32. وارد کردن دیتاهای train و test به dataloader

تعداد کلاس‌های classification و تعداد اپک‌های training را از روی optimum parameter مشخص می‌کنیم:

```
k = len(torch.unique(y_train))
Fashion_MNIST = CNN_Arc2(k)
epochs_number = 50
```

شکل 33. مشخص کردن تعداد کلاس‌ها و اپک‌ها

Optimizer، Loss Function، دقت، صحت و f1 را به تعریف می‌کنیم:

```
optimizer = torch.optim.Adam(Fashion_MNIST.parameters(), lr = 0.01)
loss_function = nn.CrossEntropyLoss()

f1sc = BinaryF1Score()
acc = BinaryAccuracy()
pre = BinaryPrecision()
softmax = nn.Softmax(dim = 1)

f1score = []
accuracy = []
precision = []
```

شکل 34. معرفی Optimizer، Loss Function، دقت، صحت و f1

معماری‌های مختلف را وارد GPU می‌کنیم:

```
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
loss_function=loss_function.to(device)
Fashion_MNIST.to(device)
acc.to(device)
pre.to(device)
f1sc.to(device)
```

شکل 35. واردکردن معماری‌ها به GPU

درنهایت، معماری‌های مختلف را train می‌کنیم:

```
for epoch in range(epochs_number):
  train_loss = []
  test_loss = []
  Fashion_MNIST.train()
  for data, pred in train:
    data = data.unsqueeze(1)
    data = data.to(device)
    pred = pred.to(device)
    optimizer.zero_grad()
    output = Fashion_MNIST(data)
    loss = loss_function(output.to(device), pred)
    loss.backward()
    train_loss.append(loss.item())
    optimizer.step()

  Fashion_MNIST.eval()
  with torch.no_grad():
    for data, pred in test:
      data = data.unsqueeze(1)
      data = data.to(device)
      pred = pred.to(device)
      outputs = Fashion_MNIST(data)
      outputs = softmax(outputs)
      los = loss_function(outputs.to(device), pred)
      test_loss.append(los.item())
      accuracy.append(acc(outputs, pred))
      precision.append(pre(outputs, pred))
      f1score.append(f1sc(outputs, pred))
  print(f'epoch {epoch} : Accuracy {sum(accuracy)/len(accuracy)*100}   Precision {sum(precision)/len(precision)*100}   f1score {sum(f1score)/len
```

شکل 36. Train کردن شبکه CNN

دقت، صحت و f1 در معماری‌های دوم و چهارم به صورت زیر است:

```
epoch 0 : Accuracy 97.16773223876953   Precision 87.61808013916016   f1score 85.48558044433594
epoch 1 : Accuracy 97.3185806274414   Precision 88.47347259521484   f1score 86.24827575683594
epoch 2 : Accuracy 97.43936920166016   Precision 88.88602447509766   f1score 86.90110778808594
epoch 3 : Accuracy 97.51832580566406   Precision 89.2293472290039   f1score 87.31656646728516
epoch 4 : Accuracy 97.5419692993164   Precision 89.21279907226562   f1score 87.45767974853516
epoch 5 : Accuracy 97.59183502197266   Precision 89.42129516601562   f1score 87.71934509277344
epoch 6 : Accuracy 97.62126159667969   Precision 89.53276062011719   f1score 87.87480926513672
epoch 7 : Accuracy 97.63480377197266   Precision 89.5618667602539   f1score 87.9498519897461
epoch 8 : Accuracy 97.63731384277344   Precision 89.54386901855469   f1score 87.96755981445312
epoch 9 : Accuracy 97.62726593017578   Precision 89.45609283447266   f1score 87.92222595214844
```

```
epoch 40 : Accuracy 97.70689392089844   Precision 89.52838897705078   f1score 88.37897491455078
epoch 41 : Accuracy 97.71025085449219   Precision 89.53797912597656   f1score 88.39723205566406
epoch 42 : Accuracy 97.71321868896484   Precision 89.54833984375   f1score 88.41303253173828
epoch 43 : Accuracy 97.70460510253906   Precision 89.49769592285156   f1score 88.37044525146484
epoch 44 : Accuracy 97.70934295654297   Precision 89.51594543457031   f1score 88.39525604248047
epoch 45 : Accuracy 97.71381378173828   Precision 89.53571319580078   f1score 88.4184799194336
epoch 46 : Accuracy 97.7154769897461   Precision 89.53936004638672   f1score 88.42778778076172
epoch 47 : Accuracy 97.71257781982422   Precision 89.51643371582031   f1score 88.41439056396484
epoch 48 : Accuracy 97.71275329589844   Precision 89.51060485839844   f1score 88.41654968261719
epoch 49 : Accuracy 97.71359252929688   Precision 89.50965118408203   f1score 88.42169189453125
```

شکل 37. دقت، صحت و f1 در معماری دوم با Adam Optimizer

```
epoch 0 : Accuracy 95.46282196044922   Precision 82.44061279296875   f1score 75.32402801513672
epoch 1 : Accuracy 95.79161834716797   Precision 82.88712310791016   f1score 77.54132080078125
epoch 2 : Accuracy 96.01167297363281   Precision 83.7554702758789   f1score 78.82400512695312
epoch 3 : Accuracy 96.17955780029297   Precision 84.4595718383789   f1score 79.77676391601562
epoch 4 : Accuracy 96.3134994506836   Precision 85.06595611572266   f1score 80.524169921875
epoch 5 : Accuracy 96.41565704345703   Precision 85.5537338256836   f1score 81.08706665039062
epoch 6 : Accuracy 96.49454498291016   Precision 85.9190673828125   f1score 81.52306365966797
epoch 7 : Accuracy 96.54953002929688   Precision 86.23954772949219   f1score 81.81332397460938
epoch 8 : Accuracy 96.60457611083984   Precision 86.42800903320312   f1score 82.13040924072266
epoch 9 : Accuracy 96.64170837402344   Precision 86.5517349243164   f1score 82.34498596191406
```

```
epoch 40 : Accuracy 97.06580352783203   Precision 87.93140411376953   f1score 84.77153015136719
epoch 41 : Accuracy 97.07357025146484   Precision 87.9539566040039   f1score 84.81562042236328
epoch 42 : Accuracy 97.07960510253906   Precision 87.9698486328125   f1score 84.85025787353516
epoch 43 : Accuracy 97.08565521240234   Precision 87.98733520507812   f1score 84.88456726074219
epoch 44 : Accuracy 97.08966064453125   Precision 88.00314331054688   f1score 84.90665435791016
epoch 45 : Accuracy 97.09628295898438   Precision 88.02688598632812   f1score 84.94329833984375
epoch 46 : Accuracy 97.10391998291016   Precision 88.04801940917969   f1score 84.98664093017578
epoch 47 : Accuracy 97.1084213256836   Precision 88.05848693847656   f1score 85.0126724243164
epoch 48 : Accuracy 97.1136474609375   Precision 88.08382415771484   f1score 85.0406265258789
epoch 49 : Accuracy 97.11924743652344   Precision 88.09518432617188   f1score 85.07331085205078
```

شکل 38. دقت، صحت و f1 در معماری چهارم با Adam Optimizer

```
epoch 0 : Accuracy 90.00006866455078   Precision 0.0   f1score 0.0
epoch 1 : Accuracy 90.00003814697266   Precision 0.0   f1score 0.0
epoch 2 : Accuracy 89.99981689453125   Precision 0.0   f1score 0.0
epoch 3 : Accuracy 89.99971771240234   Precision 0.0   f1score 0.0
epoch 4 : Accuracy 89.99964904785156   Precision 0.0   f1score 0.0
epoch 5 : Accuracy 89.99960327148438   Precision 0.0   f1score 0.0
epoch 6 : Accuracy 89.99958038330078   Precision 0.0   f1score 0.0
epoch 7 : Accuracy 89.99986267089844   Precision 0.0   f1score 0.0
epoch 8 : Accuracy 90.0001449584961   Precision 0.0   f1score 0.0
epoch 9 : Accuracy 90.00038146972656   Precision 0.0   f1score 0.0
```

```
epoch 40 : Accuracy 92.85530090332031   Precision 48.23674392700195   f1score 39.178951263427734
epoch 41 : Accuracy 92.92776489257812   Precision 49.12368392944336   f1score 40.092952728271484
epoch 42 : Accuracy 92.99153137207031   Precision 49.93922424316406   f1score 40.9350471496582
epoch 43 : Accuracy 93.0589599609375   Precision 50.770050048828125   f1score 41.77084732055664
epoch 44 : Accuracy 93.12336730957031   Precision 51.54970932006836   f1score 42.57292938232422
epoch 45 : Accuracy 93.18586730957031   Precision 52.293766021728516   f1score 43.34736633300781
epoch 46 : Accuracy 93.24596405029297   Precision 53.009098052978516   f1score 44.08974838256836
epoch 47 : Accuracy 93.30347442626953   Precision 53.69551086425781   f1score 44.800323486328125
epoch 48 : Accuracy 93.35865020751953   Precision 54.361026763916016   f1score 45.479827880859375
epoch 49 : Accuracy 93.41170501708984   Precision 54.98793029785156   f1score 46.1367073059082
```

شکل 39. دقت، صحت و f1 در معماری دوم با SGD Optimizer

```
epoch 0 : Accuracy 90.00006866455078   Precision 0.0   f1score 0.0
epoch 1 : Accuracy 90.00003814697266   Precision 0.0   f1score 0.0
epoch 2 : Accuracy 89.99981689453125   Precision 0.0   f1score 0.0
epoch 3 : Accuracy 89.99971771240234   Precision 0.0   f1score 0.0
epoch 4 : Accuracy 89.99964904785156   Precision 0.0   f1score 0.0
epoch 5 : Accuracy 89.99960327148438   Precision 0.0   f1score 0.0
epoch 6 : Accuracy 89.99958038330078   Precision 0.0   f1score 0.0
epoch 7 : Accuracy 89.99986267089844   Precision 0.0   f1score 0.0
epoch 8 : Accuracy 90.0001449584961   Precision 0.0   f1score 0.0
epoch 9 : Accuracy 90.00038146972656   Precision 0.0   f1score 0.0
```

```
epoch 40 : Accuracy 89.9983139038086   Precision 0.0   f1score 0.0
epoch 41 : Accuracy 89.99811553955078   Precision 0.0   f1score 0.0
epoch 42 : Accuracy 89.99793243408203   Precision 0.0   f1score 0.0
epoch 43 : Accuracy 89.99775695800781   Precision 0.0   f1score 0.0
epoch 44 : Accuracy 89.99759674072266   Precision 0.0   f1score 0.0
epoch 45 : Accuracy 89.99742889404297   Precision 0.0   f1score 0.0
epoch 46 : Accuracy 89.99727630615234   Precision 0.0   f1score 0.0
epoch 47 : Accuracy 89.99712371826172   Precision 0.0   f1score 0.0
epoch 48 : Accuracy 89.99699401855469   Precision 0.0   f1score 0.0
epoch 49 : Accuracy 89.99685668945312   Precision 0.0   f1score 0.0
```

شکل 40. دقت، صحت و f1 در معماری چهارم با SGD Optimizer

5-2. مقایسه نتایج استفاده از بهینه‌سازهای مختلف

جدول 3. مقایسه نتایج استفاده از بهینه‌سازهای مختلف در معماری‌های مختلف

Architectures	Optimizer	Accuracy	Precision	F1score
2th	Adam	97.71%	89.50%	88.42%
	SGD	93.41%	54.98%	45.13%
4th	Adam	97.11%	88.09%	85.07%
	SGD	89.99%	0%	0%

6-2. استفاده از Dropout

در هنگام آموزش، برخی از نورون‌ها را رها می‌شود تا از Overfit شدن شبکه، جلوگیری شود. با رها کردن اتفاقی برخی نورون‌ها در شبکه، در هر اپک، نورون‌های متفاوتی در شبکه train می‌شود اما همه نورون‌ها در test وجود دارد و این سبب میشود شبکه دیرتر Overfit شود.