به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

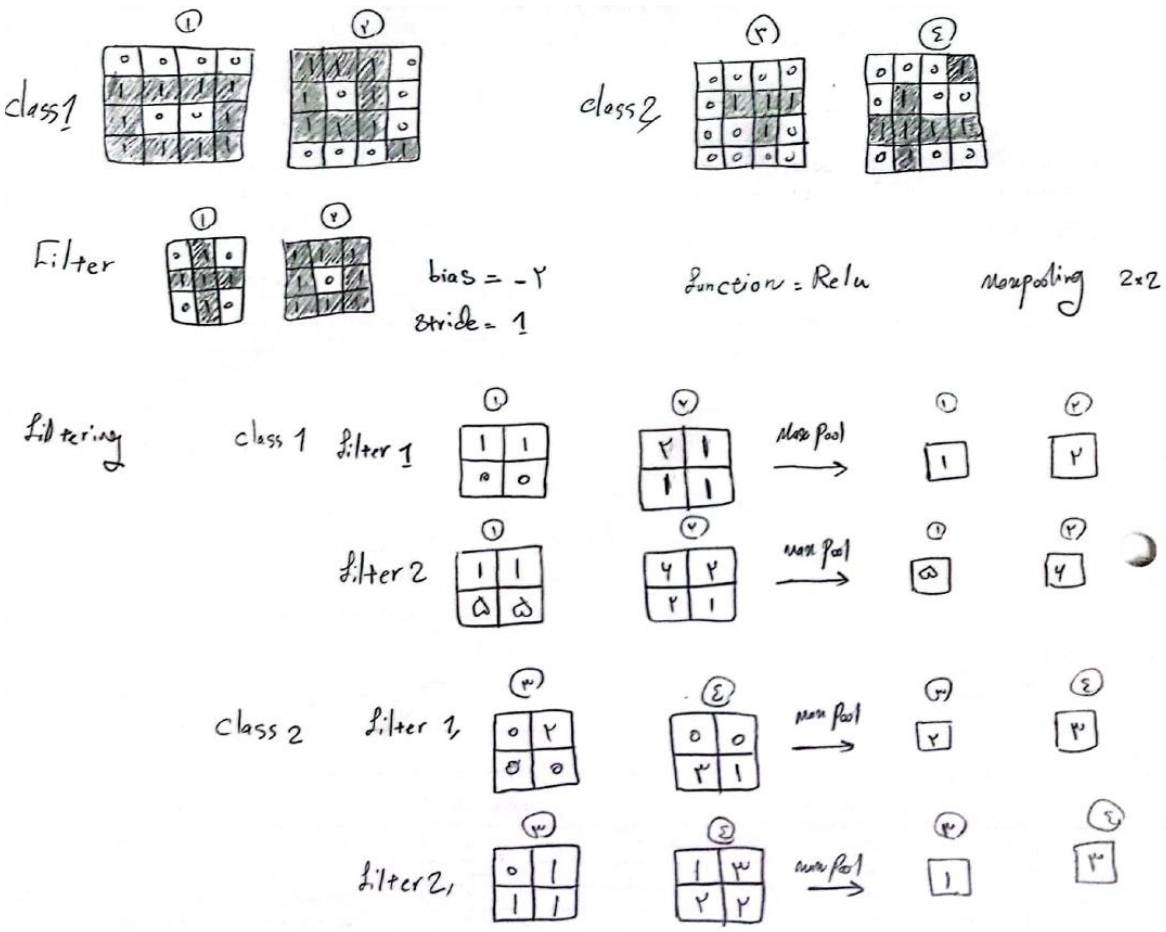
تمرین اول

نام و نام خانوادگی	فاطمه نائینیان – محمد عبائیانی
شماره دانشجویی	810198432 - 810198479
تاریخ ارسال گزارش	1401-09-03

فهرست

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

1-1. دست گرمی



شکل 1 دست گرمی: **filter and max pooling**

2-1. الف)

ابتدا دیتاست داده شده را لود میکنیم. سپس label های متناظر ان را به صورت one hot در می آوریم.

```
(x_train32,y_train),(x_test32,y_test32) = cifar10.load_data()
y_train = to_categorical(y_train)
y_test32 = to_categorical(y_test32)
```

شکل 2 لود کردن دیتاست

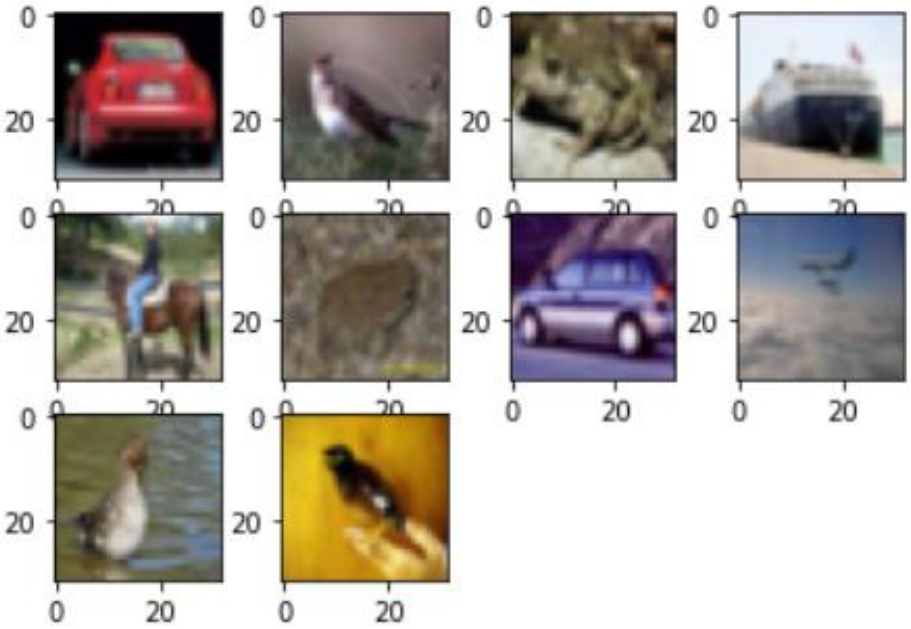
سپس با دستورresize که در کتابخانه cv2 وجود دارد تصاویر را به رزولوشن 16*16 و 8*8 میبریم. البته به دلیل اینکه در بخش های بعد نیاز داریم تا سایز تصاویر یکشان باشد، پس دوباره یک resize با ابعاد 32*32 میزنیم.

```
x_train16 = [0]*x_train32.shape[0]
x_train8 = [0]*x_train32.shape[0]
x_test16 = [0]*x_test32.shape[0]
x_test8 = [0]*x_test32.shape[0]

for i in range(0,x_train32.shape[0]):
  x_train16[i] = cv2.resize(x_train32[i],(16,16))
  x_train16[i] = cv2.resize(x_train16[i],(32,32))
  x_train8[i] = cv2.resize(x_train32[i],(8,8))
  x_train8[i] = cv2.resize(x_train8[i],(32,32))

for i in range(0,x_test32.shape[0]):
  x_test16[i] = cv2.resize(x_test32[i],(16,16))
  x_test16[i] = cv2.resize(x_test16[i],(32,32))
  x_test8[i] = cv2.resize(x_test32[i],(8,8))
  x_test8[i] = cv2.resize(x_test8[i],(32,32))

x_train16 = np.array(x_train16)
x_train8 = np.array(x_train8)
x_test16 = np.array(x_test16)
x_test8 = np.array(x_test8)
```

شکل 3 تغییر سایز تصاویر دیتاست

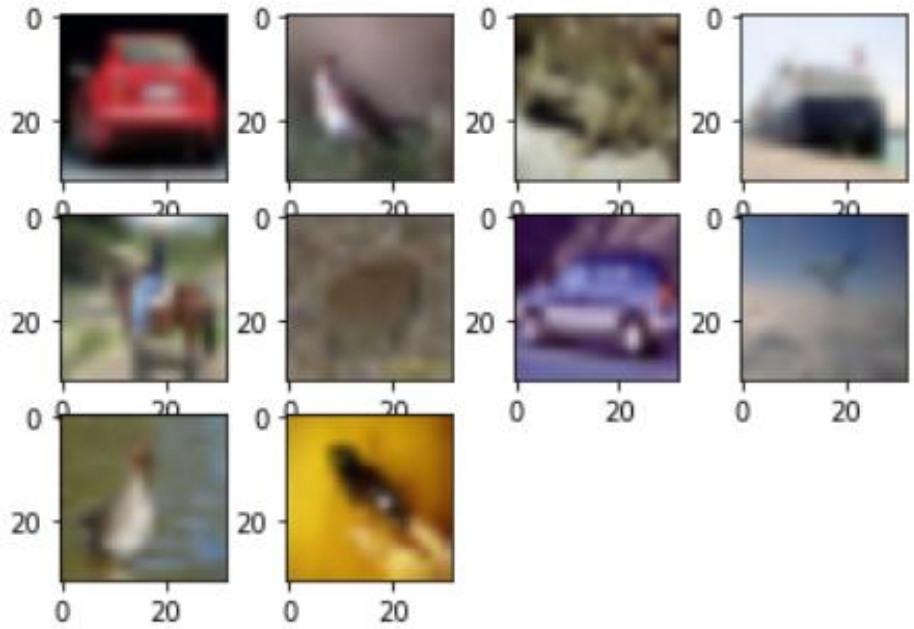
حال با کتابخانه رندوم، 10 ایندکس تصادفی انتخاب میکنیم.

سپس تصاویر با رزولوشن 32*32 را نمایش میدهیم.



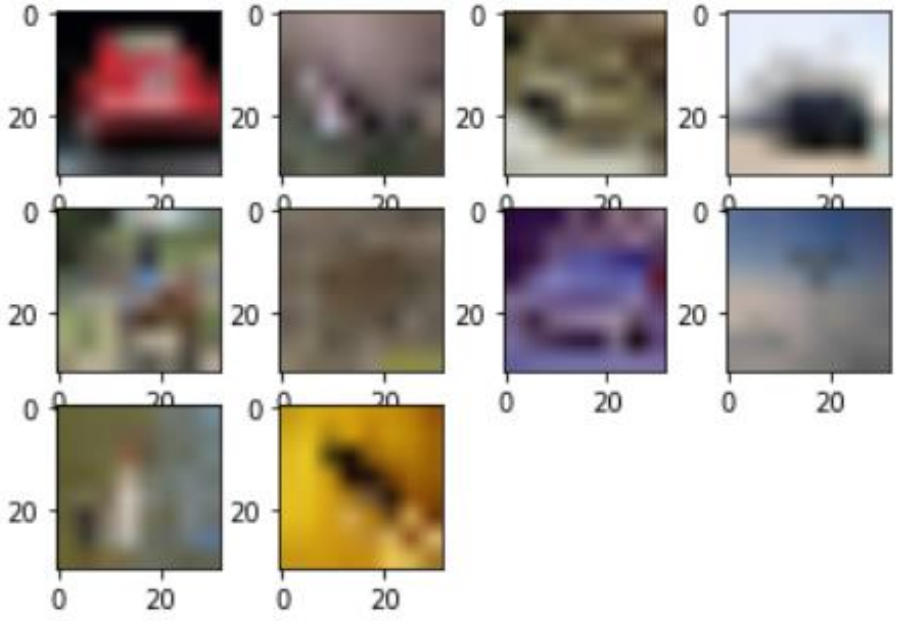
شکل 4 نمایش 10 تصویر تصادفی با رزولوشن 32*32

برای تصاویر 16*16 داریم.



شکل 5 نمایش 10 تصویر تصادفی با رزولوشن 16*16

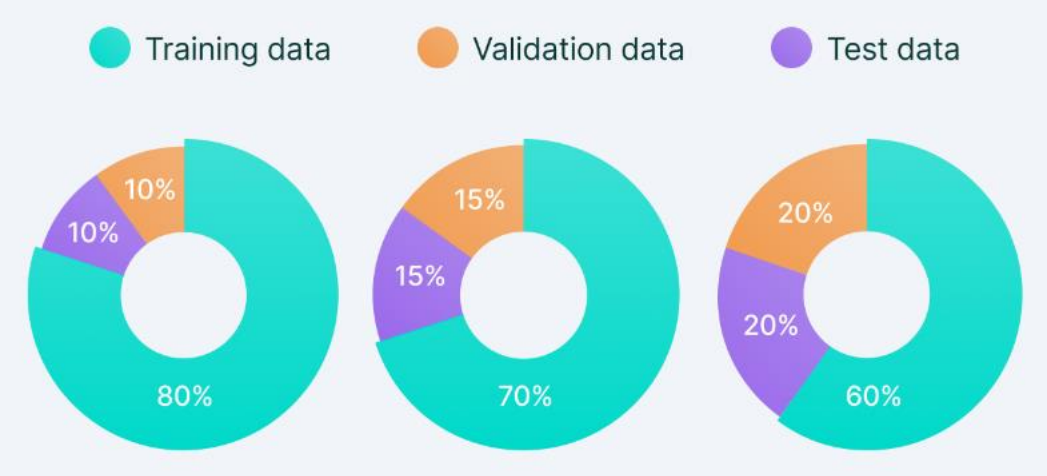
تصاویر 8*8 نیز به صورت زیر هستند.



شکل 6 نمایش 10 تصویر تصادفی با رزولوشن 8*8

3-1. ب)

روش های مختلفی برای جداسازی داده های تست و اموزش وجود دارد. اما در حالت کلی هیچ درصد بهینه ای برای ان وجود ندارد و بستگی به نوع داده و شبکه دارد. مهمترین نکته ای که در این جداسازی باید به ان توجه داشت این است که در این جداسازی داده های تست و اموزش بایاس نداشته باشند و به خوبی از همه انواع داده های در تست و اموزش وجود داشته باشد. در حالت کلی سه روش مرسوم برای این جداسازی وجود دارد.



شکل 7 سه روش مرسوم **split data**

که ما از روش وسط استفاده کرده ایم. در نتیجه به صورت زیر داده ها را به **test, train** و **validation** تقسیم کرده ایم.

```
x_train32 = x_train32.astype('float32')
x_train16 = x_train16.astype('float32')
x_train8 = x_train8.astype('float32')

x_test32 = x_test32.astype('float32')
x_test16 = x_test16.astype('float32')
x_test8 = x_test8.astype('float32')

x_train32 = x_train32/255
x_train16 = x_train16/255
x_train8 = x_train8/255

x_test32 = x_test32/255
x_test16 = x_test16/255
x_test8 = x_test8/255

x_train32, x_valid32, y_train32, y_valid32 = train_test_split(x_train32, y_train, test_size=0.2, random_state=4)
x_train16, x_valid16, y_train16, y_valid16 = train_test_split(x_train16, y_train, test_size=0.2, random_state=4)
x_train8, x_valid8, y_train8, y_valid8 = train_test_split(x_train8, y_train, test_size=0.2, random_state=4)
```

شکل 8 جداسازی دیتا با کمک **train_test_split**

4-1. ج)

ابتدا یک شبکه cnn را با کمک class میزنیم و سپس الگوریتم های خواسته شده را شرح میدهیم.

با توجه به مقاله شبکه cnn دارای لایه های زیر است:

CNN Architecture for CIFAR10		
Layers	**Layers Parameter**	**Activation Function**
Conv2D	32,size=(3,3)	Relu
Conv2D	32,size=(3,3)	Relu
Conv2D	32,size=(3,3)	Relu
Maxpooling2D	Size=(2,2)	
Dropout	0.25	
Conv2D	64,size=(3,3)	Relu
Conv2D	64,size=(3,3)	Relu
Conv2D	64,size=(3,3)	Relu
Maxpooling2D	Size=(2,2)	
Dropout	0.25	
Dense	512	Relu
Dropout	0.5	
Dense	10	Softmax

شکل 9 معماری شبکه **cnn** برای **cifar10**

بنابراین در تابع __init__ کلاس، به صورت زیر شبکه را تعریف میکنیم.

```
class cnn:
  def __init__(self):
    self.model = models.Sequential()

    self.model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
    self.model.add(layers.Conv2D(32, (3, 3), activation='relu',))
    self.model.add(layers.Conv2D(32, (3, 3), activation='relu',))
    self.model.add(layers.MaxPooling2D((2, 2)))
    self.model.add(layers.Dropout(.25))

    self.model.add(layers.Conv2D(64, (3, 3), activation='relu',))
    self.model.add(layers.Conv2D(64, (3, 3), activation='relu',))
    self.model.add(layers.Conv2D(64, (3, 3), activation='relu',))
    self.model.add(layers.MaxPooling2D((2, 2)))
    self.model.add(layers.Dropout(.25))

    self.model.add(layers.Flatten())
    self.model.add(layers.Dense(512, activation="relu",kernel_initializer='he_uniform'))
    self.model.add(layers.Dropout(.5))

    self.model.add(layers.Dense(10,activation="softmax"))

    self.model.compile(optimizer= "adam", loss='categorical_crossentropy', metrics=['accuracy'])

    self.model.summary()
```

شکل 10 معماری شبکه با کمک کلاس

در تابع fit نیز داده های اموزش و ولیدیشن را به شبکه میدهیم:

```
def fit(self , x_train , y_train , x_valid , y_valid ):
  self.history = self.model.fit(x_train, y_train, epochs=40 , batch_size=32, validation_data=(x_valid, y_valid))
```

شکل 11 تابع **fit**

در تابع بعدی یعنی accuracy_plot نموداری از دقت اموزش و ولیدیشن را بر حسب ایپاک ها نمایش میدهیم.

```
def accuracy_plot(self):
  fig = plt.figure()
  plt.plot(self.history.history['accuracy'],'r')
  plt.plot(self.history.history['val_accuracy'],'b')
  plt.title('model accuracy')
  plt.ylabel('accuracy')
  plt.xlabel('epoch')
  plt.legend(['train', 'validation'])
  plt.grid()
```

شکل 12 تابع **accuracy_plot**

در تابع loss_plot نموداری از loss اموزش و ولیدیشن را بر حسب ایپاک ها نمایش میدهیم.

```
def loss_plot(self):
  fig = plt.figure()
  plt.plot(self.history.history['loss'],'r')
  plt.plot(self.history.history['val_loss'],'b')
  plt.title('model loss')
  plt.ylabel('loss')
  plt.xlabel('epoch')
  plt.legend(['train', 'validation'])
  plt.grid()
```

شکل 13 تابع **loss_plot**

در تابع report مقدار نهایی loss ، accuracy ، Percision ، recall Score وF1 Score را نمایش میدهیم:

```
def report(self , x_test , y_test ):
  Y_pred = self.model.predict(x_test)
  y_pred = np.argmax(Y_pred, axis=1)
  Y_test = np.argmax(y_test, axis=1)

  test_loss,test_accuracy = self.model.evaluate(x_test,y_test)
  print('test loss = %f' % test_loss)
  print('test accuracy = %f' % test_accuracy)
  # accuracy: (tp + tn) / (p + n)
  accuracy = accuracy_score(Y_test, y_pred)
  print('Accuracy: %f' % accuracy)
  # precision tp / (tp + fp)
  precision = precision_score(Y_test, y_pred, average='macro')
  print('Precision: %f' % precision)
  # recall: tp / (tp + fn)
  recall = recall_score(Y_test, y_pred, average='macro')
  print('Recall: %f' % recall)
  # f1: 2 tp / (2 tp + fp + fn)
  f1 = f1_score(Y_test, y_pred, average='macro')
  print('F1 score: %f' % f1)
```

شکل 14 تابع **report**

در روش TOTV شبکه عصبی را روی داده های با رزولوشن 32*32 آموزش میدهیم. سپس در مرحله تست، علاوه بر داده های 32*32 ، داده های 16*16 و 8*8 را نیز تست میکنیم.

بنابراین ابتدا داده های اموزش و ولیدیشن 32*32 را به مدل میدهیم و فیت میکنیم.

PART 1 : TOTV

```
[17]  1 model = cnn()
      2 model.fit(x_train32, y_train32, x_valid32, y_valid32)
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 32)        896

 conv2d_1 (Conv2D)           (None, 28, 28, 32)        9248

 conv2d_2 (Conv2D)           (None, 26, 26, 32)        9248

 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
 )

 dropout (Dropout)           (None, 13, 13, 32)        0

 conv2d_3 (Conv2D)           (None, 11, 11, 64)        18496

 conv2d_4 (Conv2D)           (None, 9, 9, 64)          36928

 conv2d_5 (Conv2D)           (None, 7, 7, 64)          36928

 max_pooling2d_1 (MaxPooling  (None, 3, 3, 64)         0
 2D)

 dropout_1 (Dropout)         (None, 3, 3, 64)          0

 flatten (Flatten)           (None, 576)               0

 dense (Dense)               (None, 512)               295424

 dropout_2 (Dropout)         (None, 512)               0

 dense_1 (Dense)             (None, 10)                5130

=================================================================
Total params: 412,298
Trainable params: 412,298
Non-trainable params: 0
```

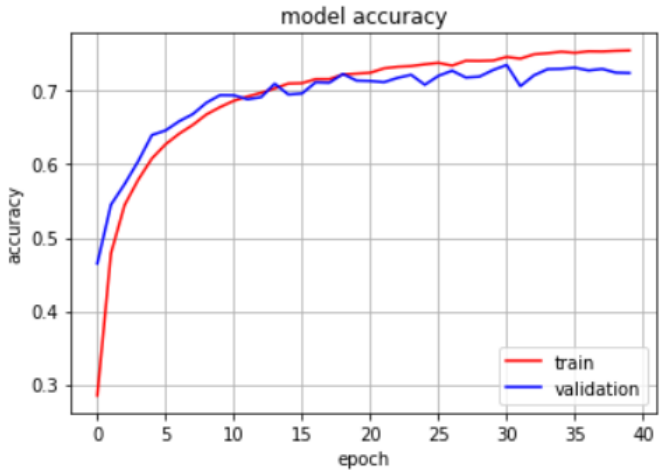
شکل 15 فیت کردن با داده های 32*32

سپس به اندازه 40 ایپاک ان را اموزش میدهیم. برای نمونه دو ایپاک اخر به شکل زیر می شود.

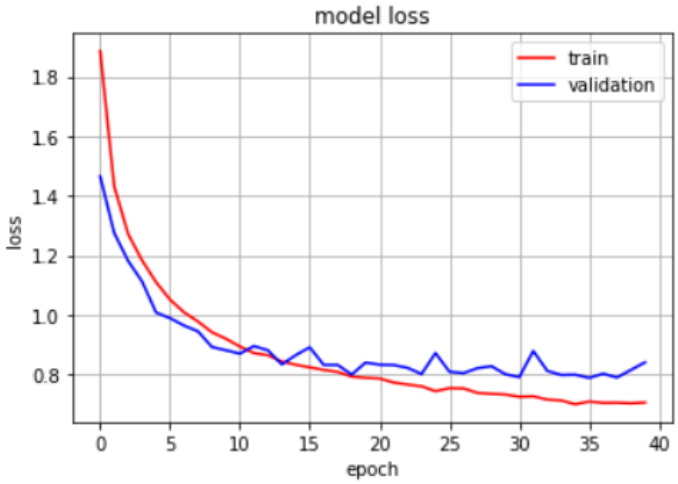
```
Epoch 39/40
1250/1250 [==============================] - 8s 6ms/step - loss: 0.7032 - accuracy: 0.7546 - val_loss: 0.8156 - val_accuracy: 0.7249
Epoch 40/40
1250/1250 [==============================] - 7s 6ms/step - loss: 0.7061 - accuracy: 0.7552 - val_loss: 0.8410 - val_accuracy: 0.7244
```

شکل 16 اموزش با 32*32 به اندازه 40 ایپاک

نمودار دقت و loss ان نیز به صورت زیر می شود.



شکل 17 نمودار دقت به ازای داده اموزش 32*32



شکل 18 نمودار **loss** به ازای داده اموزش 32*32

خروجی report به ازای داده های تست 32*32 به صورت زیر است:

```
test loss = 0.843062
test accuracy = 0.722400
Accuracy: 0.722400
Precision: 0.728394
Recall: 0.722400
F1 score: 0.719111
```

شکل 19 مقدار دقت و **loss** و **Precision** و **F1 score** برای داده های تست 32*32

خروجی report به ازای داده های تست 16*16 به صورت زیر است:

```
test loss = 2.026375
test accuracy = 0.378600
Accuracy: 0.378600
Precision: 0.541996
Recall: 0.378600
F1 score: 0.369880
```

شکل 20 مقدار دقت و**loss** و **Precision** و **F1 score** برای داده های تست 16*16

خروجی report به ازای داده های تست 8*8 به صورت زیر است:

```
test loss = 2.689169
test accuracy = 0.225200
Accuracy: 0.225200
Precision: 0.384229
Recall: 0.225200
F1 score: 0.179374
```

شکل 21 مقدار دقت و**loss** و **Precision** و **F1 score** برای داده های تست 8*8

5-1. د)

در روش TVTV در این روش به صورت جداگانه بر روی هر رزولوشن شبکه را اموزش میدهیم و تست میکنیم.

بنابراین ابتدا به سراغ داده های 32*32 میرویم:

```
model2 = cnn()
model2.fit(x_train32, y_train32, x_valid32, y_valid32)
```

شکل 22 فیت کردن مدل با داده های 32*32

```
Epoch 39/40
1250/1250 [==============================] - 7s 6ms/step - loss: 0.6792 - accuracy: 0.7645 - val_loss: 0.8032 - val_accuracy: 0.7270
Epoch 40/40
1250/1250 [==============================] - 7s 6ms/step - loss: 0.6723 - accuracy: 0.7653 - val_loss: 0.8039 - val_accuracy: 0.7387
```

شکل 23 فیت کردن به مدت 40 ایپاک با داده های 32*32

سپس به نتایج تست زیر میرسیم:

```
test loss = 0.820891
test accuracy = 0.734900
Accuracy: 0.734900
Precision: 0.740647
Recall: 0.734900
F1 score: 0.734097
```

شکل 24 نتایج تست مدل 32*32 با داده های 32*32

حال با داده های 16*16 اموزش میدهیم:

```
model3 = cnn()
model3.fit(x_train16, y_train16, x_valid16, y_valid16)
```

شکل 25 فیت کردن مدل با داده های 16*16

```
Epoch 39/40
1250/1250 [==============================] - 7s 6ms/step - loss: 0.9614 - accuracy: 0.6597 - val_loss: 1.0492 - val_accuracy: 0.6349
Epoch 40/40
1250/1250 [==============================] - 7s 6ms/step - loss: 0.9553 - accuracy: 0.6627 - val_loss: 1.0366 - val_accuracy: 0.6417
```

شکل 26 فیت کردن به مدت 40 ایپاک با داده های 16*16

سپس در تست به نتایج زیر میرسیم:

```
test loss = 1.029921
test accuracy = 0.644700
Accuracy: 0.644700
Precision: 0.646886
Recall: 0.644700
F1 score: 0.642353
```

شکل 27 نتایج تست مدل 16*16 با داده های 16*16

حال با داده های 8*8 اموزش میدهیم:

```
model4 = cnn()
model4.fit(x_train8, y_train8, x_valid8, y_valid8)
```

شکل 28 فیت کردن مدل با داده های 8*8

```
Epoch 39/40
1250/1250 [==============================] - 7s 6ms/step - loss: 1.2671 - accuracy: 0.5452 - val_loss: 1.3050 - val_accuracy: 0.5332
Epoch 40/40
1250/1250 [==============================] - 7s 6ms/step - loss: 1.2641 - accuracy: 0.5462 - val_loss: 1.3134 - val_accuracy: 0.5342
```

شکل 29 فیت کردن به مدت 40 ایپاک با داده های 8*8

نتایج ارزیابی ان نیز به صورت زیر است:

```
test loss = 1.311263
test accuracy = 0.531800
Accuracy: 0.531800
Precision: 0.540302
Recall: 0.531800
F1 score: 0.531979
```

شکل 30 نتایج تست مدل 8*8 با داده های 8*8

حال همه نتایج را در یک جدول به شکل زیر نمایش می دهیم تا مقایسه و جمع بندی اسان تر شود.

CIFAR10	TOTV			TVTV		
	ACCURACY	PERCISION	F1 SCORE	ACCURACY	PERCISION	F1 SCORE
32*32	0.722400	0.728294	0.719111	0.734900	0.740647	0.734097
16*16	0.378600	0.541996	0.369880	0.644700	0.646886	0.642353
8*8	0.225200	0.384229	0.179274	0.531800	0.540302	0.531979

جدول 1 جمع بندی دو روش **TOTV** و **TVTV**

جمع بندی : در روش TOTV وقتی با مدلی که روی عکس های 32*32 فیت شده، عکس هایی با رزولوشن کمتر را بررسی کنیم، نتایج بسیار غیر قابل اطمینان می شود و طبقه بند به خوبی نمیتواند داده ها را تفکیک کند. اما در روش TVTV وقتی داده های اموزش و تست یک مدل رزولوشن یکسانی داشته باشند، دقت افزایش می یابد. اما باز هم به دلیل اینکه کاهش رزولوشن سبب از دست رفتن ویژگی های عکس می شود، در رزولوشن های پایین، دقت پایینی خواهیم داشت و هرچه رزولوشن بیشتر شود شبکه میتواند با اطمینان بیشتری داده ها را تفکیک کند.

پاسخ ۲ – آشنایی با معماری شبکه CNN

2-1. لود دیتاست مقاله

ابتدا دیتاست را لود میکنیم.

```
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
X = np.concatenate([x_train, x_test], axis=0)
Y = np.concatenate([y_train, y_test], axis=0)
Y = to_categorical(Y)
```

شکل 31 لود کردن دیتاست

مقادیر Y را به صورت one hot میکنیم، سپس داده های اموزش و تست را با نسبت 80 و 20 جدا میکنیم.

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=4)
```

شکل 32 جداسازی دیتای اموزش و تست با **train_test_split**

2-2. انتخاب معماری

در مقاله میبینیم 5 نوع معماری برای شبکه نشان داده شده است که لایه های ان به صورت زیر است.

Architecture 1	Architecture 2	Architecture 3	Architecture 4	Architecture 5
only one input layer and two fully connected layers	2 convolutional layers with (2 x 2) filter size and 2 fully connected layers	3 convolutional layers with (2 x 2) filter size and 2 fully connected layers	4 convolutional layers with (2 x 2) filter size and 2 fully connected layers	4 convolutional layers with (3 x 3) filter size and 2 fully connected layers
(1) INPUT:28×28×1 (2) FC:10 Output Classes	(1) INPUT:28×28×1 (2) FC:10 Output Classes	(1) INPUT:28×28×1 (2) FC:10 Output Classes	(1) INPUT:28×28×1 (2) FC:10 Output Classes	(1) INPUT:28×28×1 (2) FC:10 Output Classes
(3) FC:128 Hidden Neurons	(3) CONV2D:2×2 size,64 filters (4) POOL:2×2 size (5) DROPOUT: = 0.25 (6) CONV2D :2×2 size,64 filters (7) DROPOUT: = 0.25 (8) FC:64 Hidden Neurons (9) DROPOUT: = 0.25	(3) CONV2D:2×2 size,64 filters (4) POOL:2×2 size (5) DROPOUT: = 0.25 (6) CONV2D:2×2 size,64 filters (7) POOL:2×2 size (8) DROPOUT: = 0.25 (9) CONV2D :2×2 size,64 filters (10) DROPOUT: = 0.25 (11) FC:64 Hidden Neurons (12) DROPOUT: = 0.25	(3) CONV2D:2×2 size,64 filters (4) POOL:2×2 size (5) DROPOUT: = 0.25 (6) CONV2D:2×2 size,64 filters (7) POOL:2×2 size (8) DROPOUT: = 0.25 (9) CONV2D:2×2 size,64 filters (10) POOL:2×2 size (11) DROPOUT: = 0.25 (12) CONV2D :2×2 size,64 filters (13) DROPOUT: = 0.25 (14) FC:64 Hidden Neurons (15) DROPOUT: = 0.25	(3) CONV2D:3×3 size,32 filters (4) CONV2D:3×3 size,32 filters (4) POOL:2×2 size (5) DROPOUT: = 0.25 (6) CONV2D:3×3 size,64 filters (7) CONV2D:3×3 size,64 filters (8) POOL:2×2 size (9) DROPOUT: = 0.25 (10) FC:512 Hidden Neurons (11) DROPOUT: = 0.5

شکل 33 جزئیات معماری های ذکر شده در مقاله

در مقایسه این پنج معماری میبینیم معماری اول بسیار ساده است و هیچگونه فیلتر یا لایه کانوولوشنال ندارد و فقط لایه fully connected دارد. از معماری دوم تا معماری پنجم، معماری ها پیچیده تر می شود و تعداد لایه های کانوولوشنال و فیلتر ها بیشتر می شود.

در مقایسه optimal parameters اول به optimizer میپردازیم. همانطور که در جدول دوم مقاله میبینیم، در 50 ایپاک، optimizer Adam به دقت اموزش 97.38 دست یافته است، در صورتی که بقیه دقت پایین تری دارند. دقت تست نیز برای ایپاک های مختلف و اپتیمایزر های مختلف، تفاوت اشکاری ندارد. فقط اینکه در بین همه، SGD به صورت خیلی ضعیف تری عمل میکند.

در مقایسه Batch size میبینیم اینکه مقدار batch را چقدر بگیریم تفاوت اشکاری ندارد. با این حال بیشترین دقت برای batch size =100 است.

در مقایسه learning rate میبینیم بهترین دقت با مقدار learning rate = 0.002 حاصل شده است.

حال دو معماری را انتخاب میکنیم، معماری اول و معماری چهارم.

با توجه به نتایج مقاله، معماری اول بیشترین دقت اموزش و کمترین دقت تست را می دهد. معماری سوم نیز بیشترین دقت تست را میدهد. بنابراین انتظار داریم دقت تست معماری چهارم از دقت تست معماری اول بیشتر باشد.

حال همه معماری ها را مدل میکنیم.

```
def model1(x_train, y_train, x_test, y_test, optimizer):
  model1 = models.Sequential()
  model1.add(layers.Flatten(input_shape=(28, 28)))
  model1.add(layers.Dense(128, activation="relu"))
  model1.add(layers.Dense(10,activation="softmax"))
  model1.compile(optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
  history1 = model1.fit(x_train, y_train, epochs=10, batch_size=32, validation_data=(x_test, y_test))
  report(model1, x_test, y_test)
```

شکل 34 تابع معماری اول

```python
def model2(x_train, y_train, x_test, y_test, optimizer ):
  model2 = models.Sequential()
  model2.add(layers.Conv2D(64, (2, 2), activation='relu', input_shape=(28, 28, 1)))
  model2.add(layers.MaxPooling2D((2, 2)))
  model2.add(layers.Dropout(.25))
  model2.add(layers.Conv2D(64, (2, 2), activation='relu'))
  model2.add(layers.Dropout(.25))
  model2.add(layers.Flatten())
  model2.add(layers.Dense(64, activation="relu"))
  model2.add(layers.Dropout(.25))
  model2.add(layers.Dense(10,activation="softmax"))
  model2.compile(optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
  history2 = model2.fit(x_train, y_train, epochs=10, batch_size=32, validation_data=(x_test, y_test))
  report(model2, x_test, y_test)
```

شکل 35 تابع معماری دوم

```python
def model3(x_train, y_train, x_test, y_test, optimizer ):
  model3 = models.Sequential()
  model3.add(layers.Conv2D(64, (2, 2), activation='relu', input_shape=(28, 28, 1)))
  model3.add(layers.MaxPooling2D((2, 2)))
  model3.add(layers.Dropout(.25))
  model3.add(layers.Conv2D(64, (2, 2), activation='relu'))
  model3.add(layers.MaxPooling2D((2, 2)))
  model3.add(layers.Dropout(.25))
  model3.add(layers.Conv2D(64, (2, 2), activation='relu'))
  model3.add(layers.Dropout(.25))
  model3.add(layers.Flatten())
  model3.add(layers.Dense(64, activation="relu"))
  model3.add(layers.Dropout(.25))
  model3.add(layers.Dense(10,activation="softmax"))
  model3.compile(optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
  history3 = model3.fit(x_train, y_train, epochs=10, batch_size=32, validation_data=(x_test, y_test))
  report(model3, x_test, y_test)
```

شکل 36 تابع معماری سوم

```python
def model4(x_train, y_train, x_test, y_test, optimizer ):
  model4 = models.Sequential()
  model4.add(layers.Conv2D(64, (2, 2), activation='relu', input_shape=(28, 28, 1)))
  model4.add(layers.MaxPooling2D((2, 2)))
  model4.add(layers.Dropout(.25))
  model4.add(layers.Conv2D(64, (2, 2), activation='relu'))
  model4.add(layers.MaxPooling2D((2, 2)))
  model4.add(layers.Dropout(.25))
  model4.add(layers.Conv2D(64, (2, 2), activation='relu'))
  model4.add(layers.MaxPooling2D((2, 2)))
  model4.add(layers.Dropout(.25))
  model4.add(layers.Conv2D(64, (2, 2), activation='relu'))
  model4.add(layers.Dropout(.25))
  model4.add(layers.Flatten())
  model4.add(layers.Dense(64, activation="relu"))
  model4.add(layers.Dropout(.25))
  model4.add(layers.Dense(10,activation="softmax"))
  model4.compile(optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
  history4 = model4.fit(x_train, y_train, epochs=10, batch_size=32, validation_data=(x_test, y_test))
  report(model4, x_test, y_test)
```

شکل 37 تابع معماری چهارم

```python
def model5(x_train, y_train, x_test, y_test, optimizer ):
  model5 = models.Sequential()
  model5.add(layers.Conv2D(64, (3, 3), activation='relu', input_shape=(28, 28, 1)))
  model5.add(layers.Conv2D(64, (3, 3), activation='relu'))
  model5.add(layers.MaxPooling2D((2, 2)))
  model5.add(layers.Dropout(.25))
  model5.add(layers.Conv2D(64, (3, 3), activation='relu'))
  model5.add(layers.Conv2D(64, (3, 3), activation='relu'))
  model5.add(layers.MaxPooling2D((2, 2)))
  model5.add(layers.Dropout(.25))
  model5.add(layers.Flatten())
  model5.add(layers.Dense(512, activation="relu"))
  model5.add(layers.Dropout(.5))
  model5.add(layers.Dense(10,activation="softmax"))
  model5.compile(optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
  history5 = model5.fit(x_train, y_train, epochs=10, batch_size=32, validation_data=(x_test, y_test))
  report(model5, x_test, y_test)
```

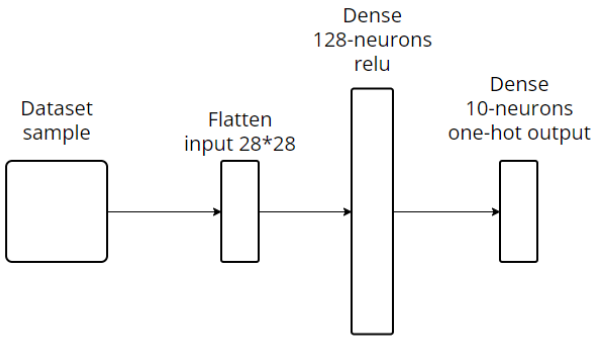
شکل 38 تابع معماری پنجم

3-2. توضیح لایه های مختلف معماری

در معماری اول تنها دو لایه ی fully-connected (به همراه یک لایه برای flatکردن)استفاده میشود.

در گام اول ورودی توسط یک لایه flat شده تا قابلیت تغذیه به شبکه را داشته باشد.

شبکه از 128 نورون در لایه ی پنهان بهره میبرد و در خروجی نیز 10 نورون وجود دارند که به صورت one hot عمل میکنند.



شکل 39 فلوچارت معماری اول

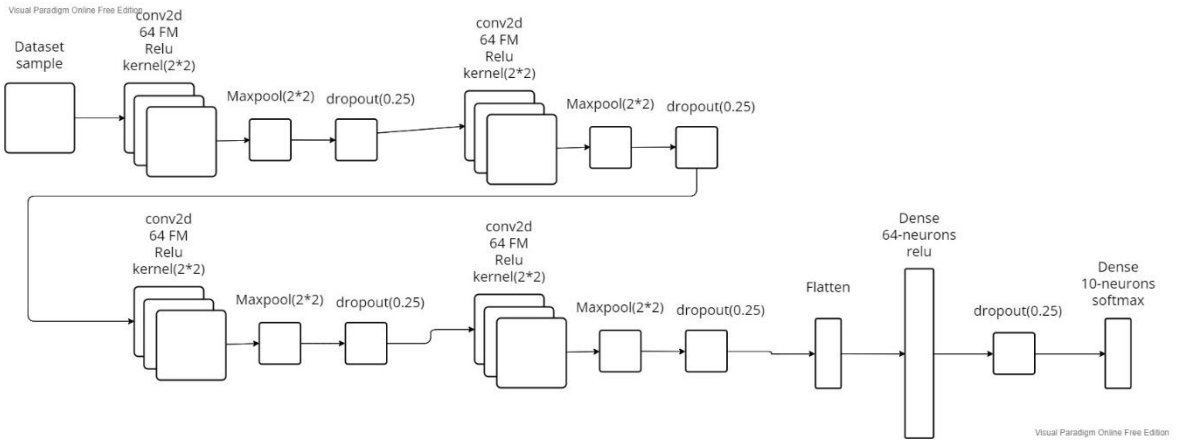
```
def model1(x_train, y_train, x_test, y_test, optimizer):
  model1 = models.Sequential()
  model1.add(layers.Flatten(input_shape=(28, 28)))
  model1.add(layers.Dense(128, activation="relu"))
  model1.add(layers.Dense(10,activation="softmax"))
  model1.compile(optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
  history1 = model1.fit(x_train, y_train, epochs=10, batch_size=32, validation_data=(x_test, y_test))
  report(model1, x_test, y_test)
```

شکل40 تابع معماری اول

در معماری چهارم ورودی به یک لایه ی کانولوشن داده شده و از آن 64 feature map ساخته میشود(با سایز کرنل 2 در 2 و اکیتیویتور relu) سپس به یک لایه ی maxpooling(2*2) داده شده و در نهایت 0.25 درصد dropout روی آن انجام میشود.

پس از چهار بار انجام عملیات بالا خروجی flat شده و وارد یک شبکه ی fully connected با 64 نورون پنهان و 10 نورون خروجی به شکل one-hot و اکتیویتور softmax میشود.

شماتیک معماری به شکل زیر است:



شکل 39 فلوچارت معماری چهارم

پیاده سازی معماری:

```python
def model4(x_train, y_train, x_test, y_test, optimizer ):
  model4 = models.Sequential()
  model4.add(layers.Conv2D(64, (2, 2), activation='relu', input_shape=(28, 28, 1)))
  model4.add(layers.MaxPooling2D((2, 2)))
  model4.add(layers.Dropout(.25))
  model4.add(layers.Conv2D(64, (2, 2), activation='relu'))
  model4.add(layers.MaxPooling2D((2, 2)))
  model4.add(layers.Dropout(.25))
  model4.add(layers.Conv2D(64, (2, 2), activation='relu'))
  model4.add(layers.MaxPooling2D((2, 2)))
  model4.add(layers.Dropout(.25))
  model4.add(layers.Conv2D(64, (2, 2), activation='relu'))
  model4.add(layers.Dropout(.25))
  model4.add(layers.Flatten())
  model4.add(layers.Dense(64, activation="relu"))
  model4.add(layers.Dropout(.25))
  model4.add(layers.Dense(10,activation="softmax"))
  model4.compile(optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
  history4 = model4.fit(x_train, y_train, epochs=10, batch_size=32, validation_data=(x_test, y_test))
  report(model4, x_test, y_test)
```

شکل 40 تابع معماری چهارم

در مجموع معماری اول معماری بسیار ساده ای است که تنها یک شبکه ی fully-connected را شامل میشود، اما معماری چهارم با اضافه کردن 4 لایه ی کانولوشن قبل از ورود به شبکه دارای پیچیدگی و قدرت بیشتری میباشد.

4-2. مقایسه نتایج دو معماری مختلف

ابتدا داده ها را با نسبت 80 درصد اموزش و 20 درصد تست جداسازی میکنیم. سپس با کمک مدلسازی معماری ها، مدل ها را اموزش میدهیم. نتایج زیر به دست امده است.

	Adam			SGD		
	Accuracy	Precision	F1 score	Accuracy	Precision	F1 score
Model 1	0.841142	0.849838	0.842095	0.661929	0.709880	0.606599
Model 4	0.853500	0.851736	0.847881	0.801357	0.797339	0.779934

جدول 2 مقایسه معماری یک و چهار

مشاهده میشود که معماری چهارم در مجموع عملکرد بهتری از معماری اول دارد، پیچیدگی بیشتر معماری چهارم باعث میشود که این مدل feature های داده ها را بهتر تشخیص داده که باعث بالا رفتن دقت پیش بینی می شود که در f1_score قابل مشاهده است. همچنین در طول آموزش نیز با سرعت بیشتری به دقت مورد نظر همگرا میشود که با مقایسه ی accuracy و precision دو مدل میتوان ب این پی برد. این تفاوت سرعت قبل از همگرا شدن محسوس تر می باشد که در اپتیمایزر SGD قابل مشاهده است.

5-2. مقایسه نتایج استفاده بهینه ساز های مختلف

در این تمرین هر پنج نوع معماری را انجام دادیم و نتایج زیر در 40 ایپاک حاصل گردید.

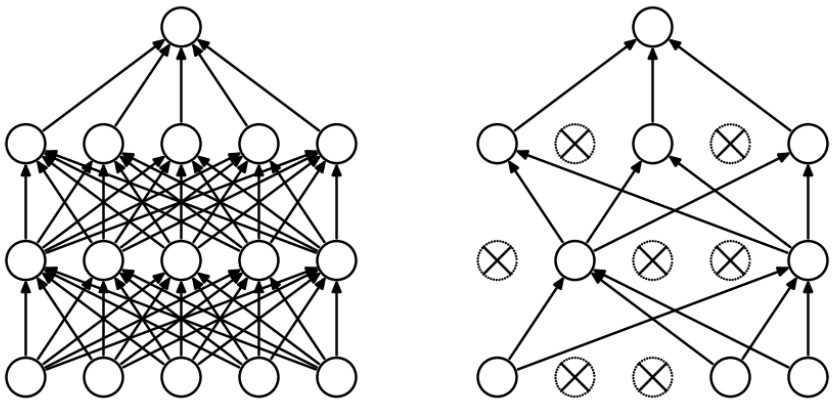
	Adam			SGD		
	Accuracy	Precision	F1 score	Accuracy	Precision	F1 score
Model 1	0.841142	0.849838	0.842095	0.661929	0.709880	0.606599
Model 2	0.900571	0.904001	0.900499	0.887429	0.887674	0.886596
Model 3	0.894857	0.894665	0.893987	0.859786	0.859742	0.856366
Model 4	0.853500	0.851736	0.847881	0.801357	0.797339	0.779934
Model 5	0.899500	0.901216	0.897066	0.889429	0.888924	0.889019

جدول 3 مقایسه هر پنج معماری

از مقایسه این دو بهنیه ساز نتیجه میگیریم که بهینه ساز Adam در تعداد ایپاک ثابت، نتایج بهتری را نسبت به بهینه ساز SGD به دست می اورد

6-2. استفاده از Dropout

Dropout به معنی بیرون انداختن واحد های عملیاتی و محاسباتی از درون لایه های مخفی است. بنابراین در طی عملیات یادگیری به صورت موقتی آن نورون و ارتباطات آن را قطع میکنیم . این عملیات به صورت کاملا تصادفی انجام می شود. یعنی به صورت تصادفی مقداری از داده ها را(که با rate مشخص میشود) برابر صفر گذاشته و بقیه ی داده ها را اسکیل میکند.



شکل 41 نحوه عملکرد **Dropout**

با کمک این روش از بایاس شدن شبکه به سمت نورون های خاص جلوگیری میکنیم و یادگیری را پخش میکنیم. این امر سبب می شود تا خطا کاهش بیابد.

تصادفی بودن این عملیات باعث می شود که نورون ها به ورودی خاصی وابسته نشوند و همچنین redundancy ورودی ها را کاهش داده و باعث میشود شبکه با داده های کم اهمیت train نشود.

تمامی اتفاقات بالا با این هدف انجام میشود که احتمال overfit شدن مدل در طول آموزش کاهش یابد.