به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین دوم

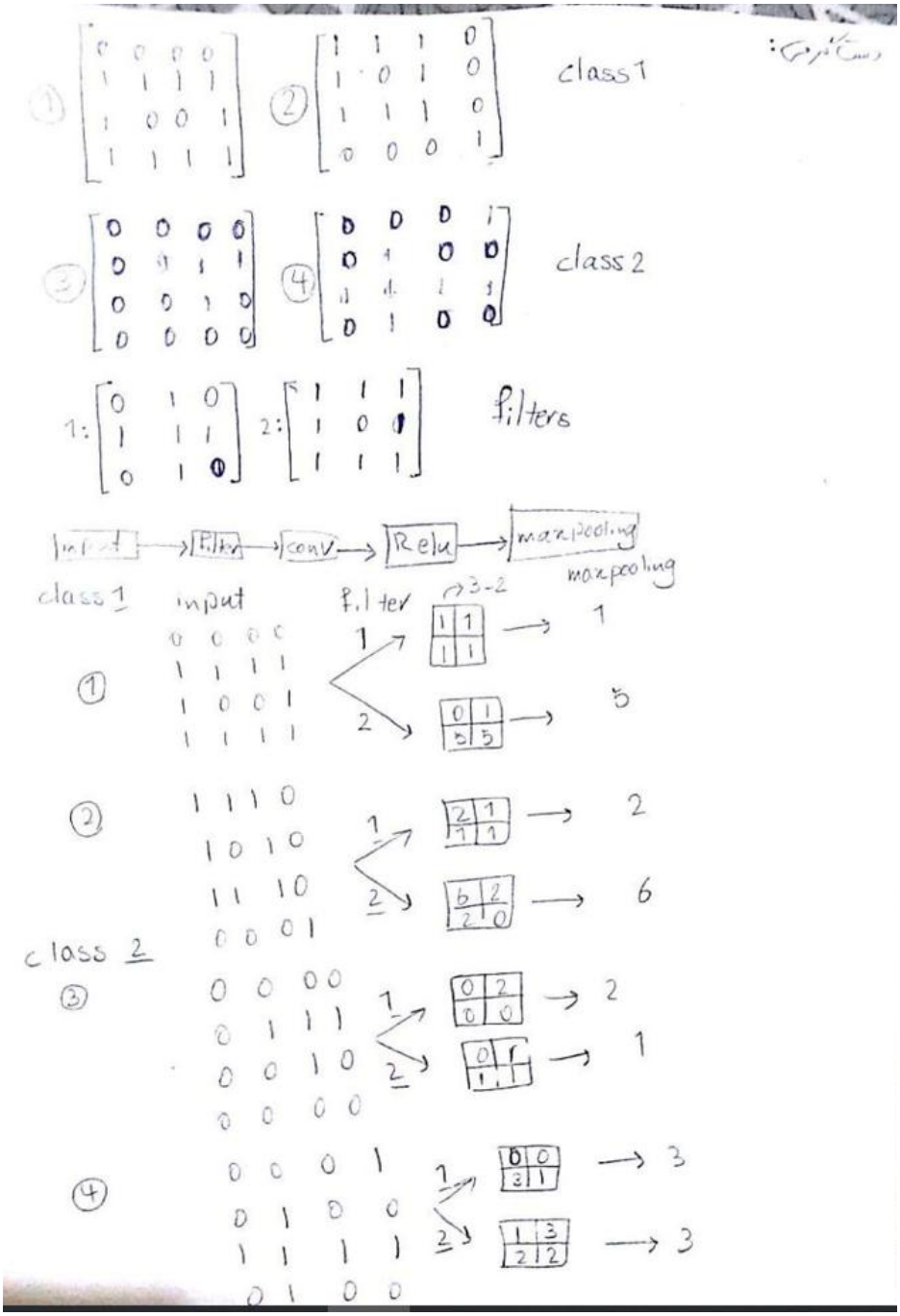
نام و نام خانوادگی	مهسا ندافی قهنویه
شماره دانشجویی	810100490
تاریخ ارسال گزارش	1401/9/2

فهرست

شکل‌ها

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

1-1 دست گرمی



شکل 1: دست گرمی

2-2 شبیه سازی مقاله

```
#import librarys
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from keras.utils import to_categorical
import cv2
import tensorflow as tf
import keras
import seaborn as sns
from keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import ConfusionMatrixDisplay, classification_re
port, confusion_matrix, accuracy_score, f1_score, precision_score, re
call_score
from keras import regularizers
```

الف) در ابتدا با استفاده از کد زیر دیتاست را خوانده و ابعاد بخش ترین و تست را نمایش داده ایم:

```
#load dataset
from keras.datasets import cifar10
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
# one hot encode target values
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
shape of x_train => (50000, 32, 32, 3)
shape of y_train => (50000, 1)
shape of x_test => (10000, 32, 32, 3)
shape of y_test => (10000, 1)
```

همانطور که مشخص است 16.7 درصد از داده به تست و 83.3 درصد از داده به ترین اختصاص داده شده است.

حال اقدام به تولید دیتاست با رزولوشن های مختلف 16*16 و 8*8 میکنیم البته با توجه به تصویر 3 مقاله ابتدا رزولوشن تصویر را کم کرده و سپس آن را به حالت 32*32 بازمیگردانیم چرا که ورودی شبکه تصاویر 32 بیتی می باشد:

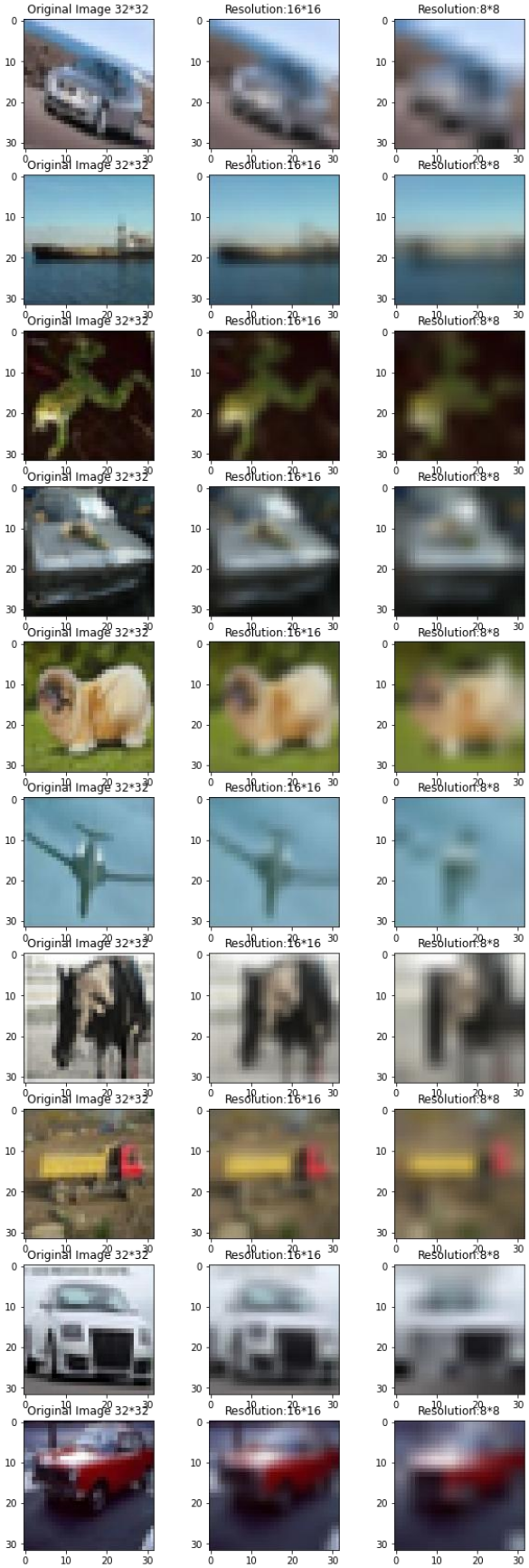
```
#normalization of data
x_train.astype("float32")
x_test=x_test.astype("float32")
# normalize to range 0-1
x_test = x_test/np.max(x_train)
x_train = x_train/np.max(x_train)
#make different resolution of dataset
x_train16 = []
x_train8 = []
for i in range(0,len(x_train)):
    image16 = cv2.resize(cv2.resize(x_train[i],(16,16)),(32,32))
```

```
    image8 = cv2.resize(cv2.resize(x_train[i],(8,8)),(32,32))
    x_train16.append(image16)
    x_train8.append(image8)
x_train16 = np.array(x_train16)
x_train8 = np.array(x_train8)

x_test16 = []
x_test8 = []
for i in range(0,len(x_test)):
    image16 = cv2.resize(cv2.resize(x_test[i],(16,16)),(32,32))
    image8 = cv2.resize(cv2.resize(x_test[i],(8,8)),(32,32))
    x_test16.append(image16)
    x_test8.append(image8)
x_test16 = np.array(x_test16)
x_test8 = np.array(x_test8)
```

و بصورت رندوم 10 تصویر از قسمت ترین داده را با این سه رزولوشن نمایش میدهیم :

```
#plot 10 random image in different resolution
fig, axs = plt.subplots(10, 3,figsize = (10,30))
for i in range(10):
    random = np.random.randint(len(x_train),size=1)
    axs[i,0].imshow(x_train[int(random)])
    axs[i,0].set_title("Original Image 32*32")
    axs[i,1].imshow(x_train16[int(random)])
    axs[i,1].set_title("Resolution:16*16")
    axs[i,2].imshow(x_train8[int(random)])
    axs[i,2].set_title("Resolution:8*8")
```

همانطور که مشخص است با تغییر رزولوشن به خوبی رزولوشن تصاویر کاهش یافته است.دقت شود که نرمالیزه کردن دیتا باید بعد از نمایش تصاویر انجام پذیرد.



شکل 2: 10 تصویر رندوم با 3 رزولوشن متفاوت از داده

ب) تعمیم (generalization)، معمولا به توانایی یک الگوریتم در پاسخ دادن به ورودی های مختلف و البته جدید اطلاق میشود و این بدان معناست که عملکرد در یک الگوریتم یادگیری ماشین، عملکرد الگوریتم بر اساس دیتایی که با آن آموزش دیده (train set) قابل ارزیابی درست نیست.

برای یادگیری ماشین از داده‌های آموزشی (Train) استفاده می‌شود و برای پایش (Monitoring) و بعضاً قطع کردن یادگیری مدل، می‌توان از داده‌های اعتبارسنجی (Validation) استفاده کرد. برای بخش دوم این فرآیند نیز از داده‌های آزمایشی (Test) استفاده می‌شود. از بین این 3 دسته داده، می‌توان داده‌های Validationرا استفاده نکرد؛ هرچند وجود آن‌ها به تنظیم بهتر برخی از آَبرپارامترها (Hyperparameters)کمک شایانی می‌کند.ابتدا می‌توان داده‌های ترین را جدا کرد و سپس در مرحله بعد جداسازی داده‌های اعتبارسنجی را از داده‌های تست انجام داد.

اعتبارسنجی متقابل تکنیکی برای ارزیابی یک مدل یادگیری ماشین و تست عملکرد آن است. این کار به مقایسه و انتخاب یک مدل مناسب برای مساله مدل‌سازی کمک می‌کند. به منظور ارزیابی مدل های طراحی شده لازم است که ابتدا داده ها را به دو قسمت ترین و تست تقسیم بندی کرده و سپس مدل را با استفاده از داده های ترین آموزش داده و سپس evaluate را در مجموعه تست انجام میدهیم.

روشهای cross validation شامل Hold-out ، K-folds ، Leave-one-out ، Leave-p-out، Stratified K-folds، Repeated K-folds ، Nested K-folds و Complete هستند.

در ادامه چند روش معروف توضیح داده شده است:

روش Hold-out: در این روش ارزیابی را برای کل مذلهای تست انجام میدهیم و به عنوان نتایج ارزیابی ذخیره میکنیم این روش ممکن است برای داده هایی که توزیع آنها یکنواخت نیست دچار مشکل شود .

روش K-fold: ابتداد داده را به k دسته ی یکسان تقسیم میکنیم k-1 دسته به عنوان آموزش و 1 دسته به عنوان تست در نظر گرفته میشود در K مرحله آموش بر روی داده های ترین و ارزیابی بر روی داداه های تست بطور مستقل انجام میشود و سپس از تمام آنها میانگین گرفته میشود.

بصورت کلی، در اکثر مواقع، استفاده از kfold نتیجه ی بهتری نسبت به holdout به شما خواهد داد. در شرایط برابر، نتایج ارائه شده توسط kfoldبسیار قابل اطمینان تر و پایدار تر میباشد چرا که آموزش و آزمایش برروی چندین بخش مختلف مجموعه ی داده انجام میشود. همچنین اگر مقدار k را افزایش دهیم، تا مدل برروی مجموعه های مختلف تست کنیم، نتایج و امتیاز کلی قوی تری نیز دریافت خواهیم کرد.

اما با این حال، روش kfold یک نقطه ضعف دارد و آن هم این است که اگر مقدارk بزرگ باشد، فرآیند آموزش بسیار پرهزینه و زمان بر خواهد شد.

روش leave one out: مانند روش k-fold میباشد که k برابر با تعداد داده های آموزش است و سپس میانگین گیری انجام میشود.

بزرگترین مزیت این روش این است که دیتای زیادی هدر نمیدهد. در واقع ما فقط از یک نمونه به عنوان دیتای تست استفاده میکنیم و از بقیه ی نمونه ها در فرآیند آموزش استفاده میکنیم.

در مقایسه kfold با این روش از آنجایی که این روش از nدسته برای اجرای الگوریتم خود استفاده می‌کند و احتمالا مقدار nاز k خیلی بیشتر خواهد بود، لذا این روش هزینه ی زیادی را نیز به دنبال خواهد داشت.

بنابراین بر اساس شواهد تجربی و تحقیقات مختلفی که صورت پذیرفته است، جامعه ی علوم داده به این نتیجه رسیده اند که kfold با k برابر 5 یا 10 بر این روش ترجیح داده میشود.

روش leave p out: p نمونه داده به عنوان داده تست و n-p به عنوان داده ترین انتخاب میشود. و سپس مدل ترین شده و ارزیابی بر روی داده های تست انجام میپذیرد در نهایت از k بار انجام آن میانگین گرفته میشود.

پس با توجه به توضیحات بالا مناسب ترین روش k-fold می باشد که باید k در آن به درستی انتخاب شود.

ج) در روش TOTV مرحله آموزش با تصاویر با رزولوشن اصلی 32*32 صورت میپذیرد و تست با رزولوشن های مختلف انجام میپذیرد.

طبق جدول زیر مدل برای این دیتا طراحی میشود:

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 3: مشخصات هر لایه از شبکه

ابتدا مدل را به شکل زیر طراحی میکنیم:

```
CNN=keras.models.Sequential([
    keras.layers.Conv2D(filters=32, kernel_size=(3,3),
                        activation='relu', input_shape=x_train.shape[
1:],
                        padding="same",
                        kernel_initializer=tf.keras.initializers.he_n
ormal(seed=None)),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=32, kernel_size=(3,3),
                        activation='relu', padding="same",
                        kernel_initializer=tf.keras.initializers.he_n
ormal(seed=None)),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=32, kernel_size=(3,3),
                        activation='relu', padding="same",
                        kernel_initializer=tf.keras.initializers.he_n
ormal(seed=None)),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(pool_size=(2,2)),
    keras.layers.Dropout(0.25),
    #-------------------------------------------------------------------
    keras.layers.Conv2D(filters=64, kernel_size=(3,3),
                        activation='relu', padding="same",
```

```
                        kernel_initializer=tf.keras.initializers.he_n
ormal(seed=None)),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=64, kernel_size=(3,3),
                        activation='relu', padding="same",
                        kernel_initializer=tf.keras.initializers.he_n
ormal(seed=None)),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=64, kernel_size=(3,3),
                        activation='relu', padding="same",
                        kernel_initializer=tf.keras.initializers.he_n
ormal(seed=None)),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(pool_size=(2,2)),
    keras.layers.Dropout(0.25),
    keras.layers.Flatten(),
    #------------------------------------------------------------------
    keras.layers.Dense(512,activation='relu'),
    keras.layers.Dropout(0.5),
    #------------------------------------------------------------------
    keras.layers.Dense(10,activation='softmax')
])

CNN.summary()
```

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 32)        896

batch_normalization (BatchN  (None, 32, 32, 32)        128
ormalization)

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

batch_normalization_1 (Batc  (None, 32, 32, 32)        128
hNormalization)

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

batch_normalization_2 (Batc  (None, 32, 32, 32)        128
hNormalization)

max_pooling2d (MaxPooling2D  (None, 16, 16, 32)        0
)

dropout (Dropout)            (None, 16, 16, 32)        0

conv2d_3 (Conv2D)            (None, 16, 16, 64)        18496

batch_normalization_3 (Batc  (None, 16, 16, 64)        256
hNormalization)

conv2d_4 (Conv2D)            (None, 16, 16, 64)        36928

batch_normalization_4 (Batc  (None, 16, 16, 64)        256
hNormalization)

conv2d_5 (Conv2D)            (None, 16, 16, 64)        36928

batch_normalization_5 (Batc  (None, 16, 16, 64)        256
hNormalization)

max_pooling2d_1 (MaxPooling  (None, 8, 8, 64)          0
2D)

dropout_1 (Dropout)          (None, 8, 8, 64)          0

flatten (Flatten)            (None, 4096)              0

dense (Dense)                (None, 512)               2097664

dropout_2 (Dropout)          (None, 512)               0

dense_1 (Dense)              (None, 10)                5130

=================================================================
Total params: 2,215,690
Trainable params: 2,215,114
Non-trainable params: 576
```

شکل 4: تعداد پارامتر و انواع لایه های شبکه

حال به فیت کردن مدل میپردازیم:

```
#TOTV
CNN.compile(loss='categorical_crossentropy',optimizer=tf.optimizers.A
dam(learning_rate=0.001),metrics=['accuracy'])
```

```
history=CNN.fit(x_train, y_train, batch_size=128, epochs=100, validat
ion_split=0.1)
```

همانطور که مشخص است از optimizer، Adam استفاده شده است و loss، categorical crossentropy در نظر گرفته شده است.

کد مورد استفاده برای بدست آوردن پارامتر های accuracy، f1 و precision :

```
accuracy_score(np.argmax(y_test,axis=1), np.argmax(y_pred,axis=1))
precision_score(np.argmax(y_test,axis=1), np.argmax(y_pred,axis=1),av
erage='macro')
f1_score(np.argmax(y_test,axis=1), np.argmax(y_pred,axis=1),average='
macro')
```

حال شروع به تست داده ها با رزولوشن های مختلف میکنیم:

1.رزولوشن 32*32:

```
y_pred = CNN.predict(x_test)
testing by 32*32 resolution
accuracy =  0.8462
precision =  0.8474297594392357
f1 =  0.8457695681669589
```

2.رزولوشن 16*16:

```
y_pred = CNN.predict(x_test16)
testing by 16*16 resolution
accuracy =  0.3828
precision =  0.6130068355545142
f1 =  0.3715507016504463
```

3.رزولوشن 8*8:

```
y_pred = CNN.predict(x_test8)
testeing by 8*8 resolution
accuracy =  0.2295
precision =  0.3659240310108548
f1 =  0.15799102154699246
```

همانطور که مشخص است با کاهش رزولوشن دقت شبکه رفته رفته کاهش می یابد.

د) در این بخش روش TVTV مورد بررسی قرار گرفته است و نحوه آموزش و تست شبکه به این صورت است که هر مدل با داده های نظیر به نظیر تست و ترین میشوند.

کلیه کد و پارامترهای مورد استفاده در این بخش نیز مانند بخش قبل می باشد:

1.رزولوشن 32*32:

```
#training by 32*32 resolution
CNN.compile(loss='categorical_crossentropy',optimizer=tf.optimizers.A
dam(learning_rate=0.001),metrics=['accuracy'])
history=CNN.fit(x_train, y_train, batch_size=128, epochs=100, validat
ion_split=0.1)
#testeing by 32*32 resolution
y_pred = CNN.predict(x_test)
testing by 32*32 resolution
accuracy =  0.8429
precision =  0.8436937792071209
f1 =  0.8426375722418046
```

2.رزولوشن 16*16:

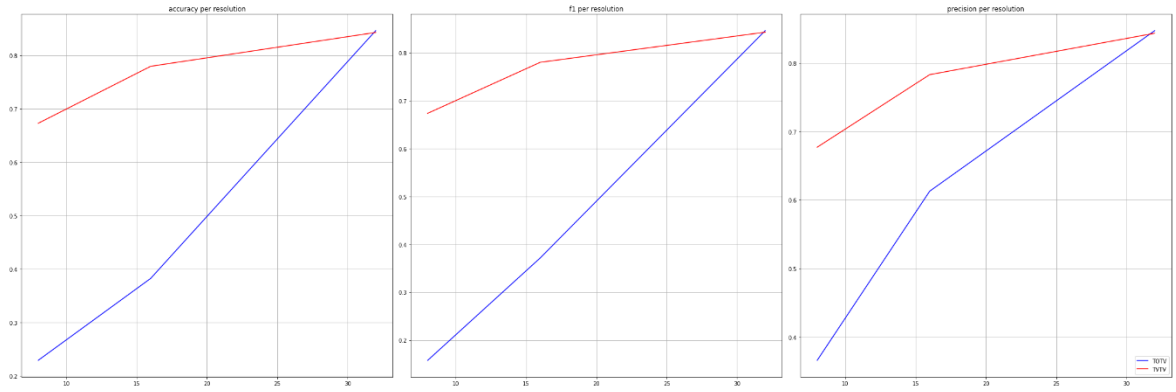
```
#training by 16*16 resolution
CNN.compile(loss='categorical_crossentropy',optimizer=tf.optimizers.A
dam(learning_rate=0.001),metrics=['accuracy'])
history=CNN.fit(x_train16, y_train, batch_size=128, epochs=100, valid
ation_split=0.1)
#testeing by 16*16 resolution
y_pred = CNN.predict(x_test16
testing by 16*16 resolution
accuracy =  0.7796
precision =  0.7832300552853674
f1 =  0.779767169089988
```

2.رزولوشن 8*8:

```
#training by 8*8 resolution
CNN.compile(loss='categorical_crossentropy',optimizer=tf.optimizers.A
dam(learning_rate=0.001),metrics=['accuracy'])
history=CNN.fit(x_train8, y_train, batch_size=128, epochs=100, valida
tion_split=0.1)
#testeing by 8*8 resolution
y_pred = CNN.predict(x_test8)
testing by 8*8 resolution
accuracy =  0.673
precision =  0.6772539125004767
f1 =  0.673399236318124
```

مشخص است که اگر هر شبکه با رزولوشن نظیر به نظیر خود ترین شود و سپس با همان رزولوشن تست شود دقت بهتری خواهیم داشت نسبت به حالتی که آموزش شبکه با رزولوشن متفاوتی صورت گرفته باشد.

نمودار 3 معیار خواسته شده بر حسب رزولوشن در دو روش TOTV,TVTV در زیر آورده شده است:



شکل 5: نمودار **f1,accuracy , precision** در جهت افزایش رزولوشن

پاسخ ۲ – عنوان پرسش دوم به فارسی

برای این سوال از محیط Colab استفاده می کنیم همچنین از GPU runtime برای تسریع سرعت آموزش استفاده می کنیم. قابل ذکر است که یک بار با CPU هم امتحان کردیم و تفاوت زمان آموزش مدل های زیر از ۴ ثانیه در epoch به ۸۰ ثانیه در هر epoch افزایش یافت که نشان دهند قدرت GPU در شبکه های CNN می باشد.

۲-۱) لود کردن دیتاست

ابتدا با استفاده از کتاب خانه keras در داخل tensorflow مجموعه داده fashion mnist را لود می کنیم:

```
Fashion Mnist dataset loaded
X_train shape: (60000, 28, 28)
y_train shape: (60000,)
X_test shape: (10000, 28, 28)
y_test shape: (10000,)
```

۲-۲) انتخاب دو معماری دلخواه

از میان معماری های موجود در مقاله ارائه شده ما دو مورد زیر را که هایلایت کرده ایم را پیاده سازی و بررسی خواهیم کرد. این معماری ها شامل دومین و سومین معماری ارائه شده هستند.

Table 1. Different CNN Architecture for image classification

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 6: معماری های موجود در مقاله

بر اساس توضیحات این مقاله اطلاعات زیر قابل برداشت است:

برای معماری شماره ۲:

برای مجموعه داده fashion mnist در مجموعه داده آموزش و تست به ترتیب به دقت های ۹۲.۰۲ و ۹۲.۷۶ درصد دست یافته اند. بهترین پارامتر های به دست امده برای این مدل از قرار زیر است:

Batch size: 128

softmax activation function

adam optimizer

0.25 dropout after each pooling layer

50 epoch

2x2 kernel size

و اما برای معماری شماره ۳:

برای مجموعه داده fashion mnist در مجموعه داده آموزش و تست به ترتیب به دقت های ۹۳.۰۹ و ۹۳.۶۵ درصد دست یافته اند. بهترین پارامتر های به دست امده برای این مدل از قرار زیر است:

Batch size: 128

softmax activation function

adam optimizer

0.25 dropout after each pooling layer

50 epoch

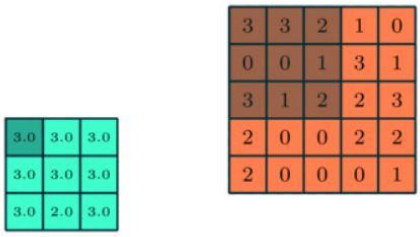
2x2 kernel size

۲-۳) توضیح لایه های به کار رفته در مدل‌ها:

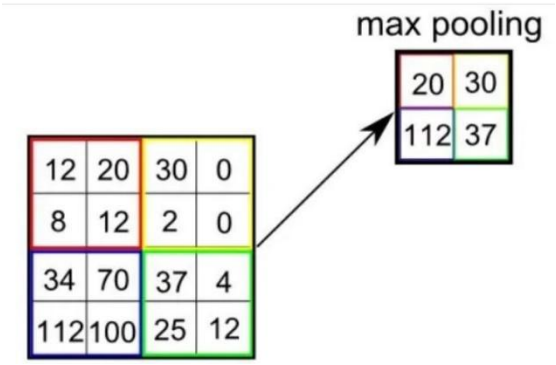
لایه ورودی (input): این لایه صرفا یک لایه برای معرفی ابعاد ورودی شبکه است. ما میدانیم ابعاد تصاویر ورودی ۲۸ در ۲۸ می باشد پس با اعلام کردن این موضوع به مدل شبکه عصبی، این امکان را فراهم می کنیم تا ابعاد لایه های بعدی را پیشبینی کرده و کل مدل را ایجاد کند. در این لایه هیچ محاسباتی رخ نمی دهد.

لایه conv2d: این لایه لایه اصلی معماری پیچشی (کانولوشنی) می باشد. دلیل وجود 2d در عنوان آن نشان دهنده این است که بر روی ورودی های ۲ بعدی عمل می کند. این لایه که بر روی یک تصویر (یا یک ماتریس دو بعدی) عمل می کند ما یک هسته (کرنل) داریم که یک پنجره ثابت (در این لایه ۲ در ۲) می باشد که وزن های مشخصی دارد. این پنجره ۲در۲ روی تصویر ورودی قرار داده می شود و المان

های آن در المان های تصویر نظیر به نظیر ضرب می شوند و سپس حاصل جمع همه این ۴ عدد حاصل در خانه خروجی نوشته می شود (ممکن است این حاصل جمع از یک تابع فعال ساز نیز عبور کنند). سپس این پنجره یک واحد جابجا می شود تا ۴ خانه مجاور (که ۲ خانه آنها تکراری هستند) را پوشش دهد و سپس دوباره عملیات بالا تکرار می شود. به این ترتیب برای مثال فعلی، که تصویری ۲۸ در ۲۸ داریم، با شیف دادن این پنجره روی همه حالات ممکن یک تصویر جدید خواهیم داشت با ابعاد ۲۷ در ۲۷ که حاصل پیچش این پنجره با تصویر اولیه هستند. برای تفهیم بهتر موضوع یک مثال با پنجره ۳در۳ را در نظر بگیرید (تصویر زیر) همانطور که مشخص است با ضرب کردن یک پنجره (که با رنگ تیره مشخص شده) روی تصویر ورودی (که نارنجی است) یک مقدار خروجی دریافت شده است که در خانه تیره آبی رنگ نوشته شده. سپس با جابجا کردن این پنجره می توان المان های دیگر ماتریس آبی را نیز حساب کرد.



لایه maxpool2d: این لایه در واقع یک روش برای کاهش ابعاد یک ورودی ۲ بعدی است. روش های مختلفی برای کاهش بعد یک تصویر وجود دارد اما در maxpool فقط **مهمترین** (بزرگترین) عدد موجود در پنجره مورد برسی نگه داشته می شود. در این لایه ما یک پنجره (کرنل یا هسته) داریم که همانند لایه پیچشی بالا روی تصویر حرکت می کند (با این تفاوت که حرکاتش overlap ندارد) و در هر مرحله بزرگترین عدد قابل مشاهده را یادداشت می کند وباقی اعداد را دور میریزد. تصویر زیر یک نمونه از این اتفاق را روی یک ماتریس ورودی ۴در۴ نشان میدهد که پس از اعمال maxpool به یک ماتریس ۲در۲ کاهش حجم پیدا کرده است.



شکل 7 : **maxpooling**

لایه dropout: این لایه (که یک روش برای کاهش بیش برازش یا overfitting است) به این صورت عمل می کند که در هر بار feedforward شبکه به صورت تصادفی تعدادی از ورودی های این لایه را صفر کرده و اثر آنها را از میان می برد. برای مثال در شبکه های فعلی که نرخ dropout 25% را داریم، هر زمان یک ماتریس ورودی با ابعاد دلخواه (مثلا ۱۰در۱۰) به آن برسد به صورت تصادفی ۲۵ درصد ازین ورودی ها که معادل است با ۲۵ المان تصادفی از ماتریس ورودی را صفر کرده و اثر آنها را از بین می برد. در بخش آخر این گزارش در خصوص دلیل استفاده از این لایه بیشتر توضیح داده ایم.

لایه flatten: این لایه عملیات محاسباتی انجام نمی دهد و صرفا ورودی های چند بعدی را یک بعدی می کند تا برای پردازش های آتی مناسب باشند.

لایه dense: این لایه که پایه ای ترین لایه مورد استفاده در شبکه های عصبی است و ما نیز در ساخت MLP از همین لایه استفاده می کنیم، المان های ورودی را در وزن هایی ضرب می کند و با هم جمع می کند و در خروجی قرار میدهد (ممکن است این حاصل جمع از یک تابع فعال ساز نیز عبور کنند).

دو مدل مذکور تا حد زیادی شبیه هم هستند، با این تفاوت که معماری شماره ۳ تعداد لایه های conv2d و maxpool2d و dropuout بیشتری دارد. در بسیاری از شبکه های cnn دیده شده که افزایش تعداد لایه ها منجر به عملکرد بهتر مدل می شود و این موضوع متناظر با این حقیقت است که هرچه تعداد این لایه های پیچشی بیشتر می شود مدل شاخصه های بیشتر و دقیق تری را از تصاویر ورودی به دست می آورد.

در نهایت می توان گفت تفاوت اصلی این دو مدل در یک بلاک اضافه شامل سه لایه ذکر شده در بالا در معماری شماره ۳ می باشد.

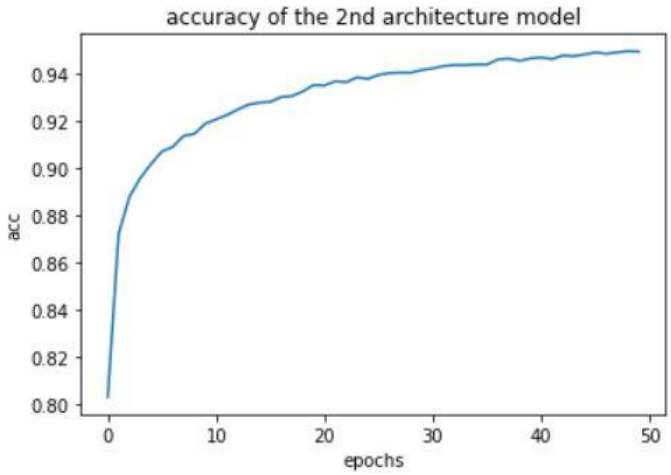
۲-۴) مقایسه نتایج دو معماری بالا:

پیاده سازی معماری دوم:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 27, 27, 64)        320

 max_pooling2d (MaxPooling2D  (None, 13, 13, 64)       0
 )

 dropout (Dropout)           (None, 13, 13, 64)        0

 conv2d_1 (Conv2D)           (None, 12, 12, 64)        16448

 dropout_1 (Dropout)         (None, 12, 12, 64)        0

 dense (Dense)               (None, 12, 12, 64)        4160

 dropout_2 (Dropout)         (None, 12, 12, 64)        0

 dense_1 (Dense)             (None, 12, 12, 10)        650

=================================================================
Total params: 21,578
Trainable params: 21,578
Non-trainable params: 0
_________________________________________________________________
```

آموزش این مدل:

```
469/469 [==============================] - 4s 8ms/step - loss: 0.1458 - accuracy: 0.9452
Epoch 40/50
469/469 [==============================] - 4s 8ms/step - loss: 0.1420 - accuracy: 0.9463
Epoch 41/50
469/469 [==============================] - 4s 8ms/step - loss: 0.1427 - accuracy: 0.9467
Epoch 42/50
469/469 [==============================] - 4s 8ms/step - loss: 0.1429 - accuracy: 0.9460
Epoch 43/50
469/469 [==============================] - 4s 8ms/step - loss: 0.1414 - accuracy: 0.9475
Epoch 44/50
469/469 [==============================] - 4s 8ms/step - loss: 0.1397 - accuracy: 0.9472
Epoch 45/50
469/469 [==============================] - 4s 8ms/step - loss: 0.1382 - accuracy: 0.9480
Epoch 46/50
469/469 [==============================] - 4s 8ms/step - loss: 0.1355 - accuracy: 0.9488
Epoch 47/50
469/469 [==============================] - 4s 8ms/step - loss: 0.1367 - accuracy: 0.9482
Epoch 48/50
469/469 [==============================] - 4s 8ms/step - loss: 0.1364 - accuracy: 0.9488
Epoch 49/50
469/469 [==============================] - 4s 8ms/step - loss: 0.1349 - accuracy: 0.9493
Epoch 50/50
469/469 [==============================] - 4s 8ms/step - loss: 0.1356 - accuracy: 0.9492
```



شکل 8: **accuracy** مدل دوم

محاسبه معیار های مد نظر:

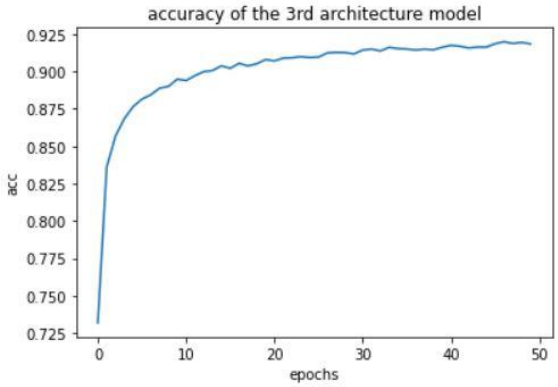
```
Results of the 2nd Architecture of the paper:
On train set:
        precision (avrage): 0.9747685127423139
        accuracy: 0.9747333333333333
        f1_score (avrage): 0.9747008322509355
On test set:
        precision (avrage): 0.9253990062326896
        accuracy: 0.9246
        f1_score (avrage): 0.9247034727532526
```

پیاده سازی معماری سوم:

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 27, 27, 64)        320

 max_pooling2d_2 (MaxPooling  (None, 13, 13, 64)       0
 2D)

 dropout_6 (Dropout)         (None, 13, 13, 64)        0

 conv2d_5 (Conv2D)           (None, 12, 12, 64)        16448

 max_pooling2d_3 (MaxPooling  (None, 6, 6, 64)         0
 2D)

 dropout_7 (Dropout)         (None, 6, 6, 64)          0

 conv2d_6 (Conv2D)           (None, 5, 5, 64)          16448

 dropout_8 (Dropout)         (None, 5, 5, 64)          0

 dense_4 (Dense)             (None, 5, 5, 64)          4160

 dropout_9 (Dropout)         (None, 5, 5, 64)          0

 dense_5 (Dense)             (None, 5, 5, 10)          650

=================================================================
Total params: 38,026
Trainable params: 38,026
Non-trainable params: 0
_________________________________________________________________
```

آموزش این مدل:

```
469/469 [==============================] - 4s 8ms/step - loss: 0.2304 - accuracy: 0.9144
Epoch 40/50
469/469 [==============================] - 4s 8ms/step - loss: 0.2242 - accuracy: 0.9161
Epoch 41/50
469/469 [==============================] - 4s 8ms/step - loss: 0.2235 - accuracy: 0.9174
Epoch 42/50
469/469 [==============================] - 4s 8ms/step - loss: 0.2215 - accuracy: 0.9168
Epoch 43/50
469/469 [==============================] - 4s 8ms/step - loss: 0.2253 - accuracy: 0.9157
Epoch 44/50
469/469 [==============================] - 4s 8ms/step - loss: 0.2225 - accuracy: 0.9163
Epoch 45/50
469/469 [==============================] - 4s 8ms/step - loss: 0.2229 - accuracy: 0.9163
Epoch 46/50
469/469 [==============================] - 4s 8ms/step - loss: 0.2202 - accuracy: 0.9185
Epoch 47/50
469/469 [==============================] - 4s 8ms/step - loss: 0.2177 - accuracy: 0.9199
Epoch 48/50
469/469 [==============================] - 4s 8ms/step - loss: 0.2191 - accuracy: 0.9187
Epoch 49/50
469/469 [==============================] - 4s 8ms/step - loss: 0.2160 - accuracy: 0.9194
Epoch 50/50
469/469 [==============================] - 4s 8ms/step - loss: 0.2181 - accuracy: 0.9184
```



شکل 9: **accuracy** مدل سوم

محاسبه معیار های مد نظر:

```
Results of the 3rd Architecture of the paper:
On train set:
        precision (avrage): 0.9446087153505516
        accuracy: 0.9447166666666666
        f1_score (avrage): 0.9445600804159653
On test set:
        precision (avrage): 0.917793555379706
        accuracy: 0.9181
        f1_score (avrage): 0.9177171859783646
```

برای این بخش، یک بار دیگر نتایج بخش قبل را مرور می کنیم:

```
Results of the 2nd Architecture of the paper:
On train set:
        precision (avrage): 0.9747685127423139
        accuracy: 0.9747333333333333
        f1_score (avrage): 0.9747008322509355
On test set:
        precision (avrage): 0.9253990062326896
        accuracy: 0.9246
        f1_score (avrage): 0.9247034727532526
```

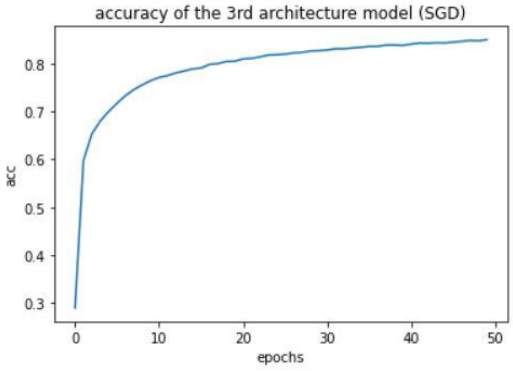
```
Results of the 3rd Architecture of the paper:
On train set:
        precision (avrage): 0.9446087153505516
        accuracy: 0.9447166666666666
        f1_score (avrage): 0.9445600804159653
On test set:
        precision (avrage): 0.917793555379706
        accuracy: 0.9181
        f1_score (avrage): 0.9177171859783646
```

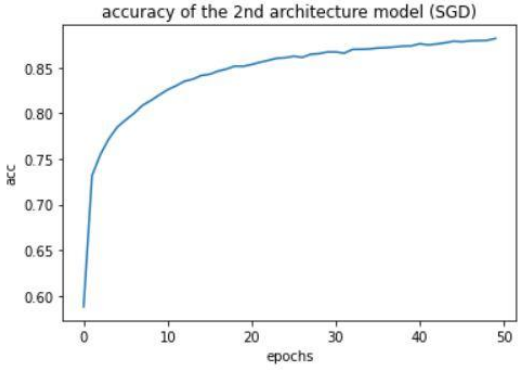
نتایج روی مجموعه تست، تا حد خیلی زیادی شبیه نتایجی است که نویسندگان مقاله به آن دست یافتند (با تفاوت کمتر از ۱ درصد) با این تفاوت که مدل شماره ۳ بجای آنکه اندکی بهتر عمل کند، بدتر عمل کرده است (۲ درصد) آنها به دقت ۹۳ درصد رسیدند اما ما به دقت ۹۱ درصد رسیدیم. البته قابل توجه است که اگر آموزش را چند epoch دیگر ادامه می دادیم احتمالا به این دقت می رسیدیم.

منتهی یک نقطه قابل توجه تفاوت نتایج در مجموعه **آموزش** است که به نظر می رسد مدل شماره ۲ بهتر از مدل شماره ۳ عمل کرده است. با اینکه تعداد لایه های مدل سوم بیشتر از مدل دوم است، به نظر می رسد برای این مسئله (که ورودی ها ۲۸در۲۸ هستند) شاید الزاما افزایش تعداد لایه های مفید نباشد.

۲-۵) مقایسه نتایج بهینه ساز های مختلف:

دو مدل بالا را یک بار دیگر با بهینه ساز SGD اجرا می کنیم. **دقت شود که طبق پارامتر های ذکر شده توسط نویسندگان مقاله بهینه ساز adam در اجرا های بالا استفاده شده بود.**






```
Results of the 2nd Architecture of the paper:
On train set:
        precision (avrage): 0.8953757417591218
        accuracy: 0.8938
        f1_score (avrage): 0.8941885310794732
On test set:
        precision (avrage): 0.8866864966009981
        accuracy: 0.885
        f1_score (avrage): 0.8853818749215415
```

```
Results of the 3rd Architecture of the paper:
On train set:
        precision (avrage): 0.8732262777806146
        accuracy: 0.8735166666666667
        f1_score (avrage): 0.8731160506313176
On test set:
        precision (avrage): 0.8660608914314404
        accuracy: 0.8662
        f1_score (avrage): 0.8657672318812517
```

همانطور که ملاحظه می شود، دقت ها به صورت کلی کاهش پیدا کرده اند. و نکته قابل توجه آنکه همچنان معماری شماره ۲ بهتر از معماری شماره ۳ عمل می کند.

دلیل برتری Adam که از adaptive moment estimation نشات می گیرد نسبت به SGD این است که این بهینه ساز همزمان از تکنیک های حفظ momentum (باعث می شود تغیر ناگهانی گرادیان ها سرعت کلی همگرایی را کاهش ندهد) و تغییر نرخ adaptive (که با توجه به گرادیان های به دست امده برای هر وزن نرخ یادگیری متناظر با ان وزن را تغییر میدهد) استفاده می کند. بهینه ساز stochastic gradient decent از هیچ کدام ازین تکنیک ها استفاده نمی کند به همین خاطر سرعت همگرایی

کمتری دارد. برسی های زیادی روی بهینه ساز های مختلف انجام شده و در اکثر آنها بهینه ساز Adam بهتر از باقی عمل می کند.

۲-۶) دلیل استفاده از dropout:

لایه dropout در واقع یک نوع رگیولایزر (regularizer) است که به منظور کاهش بیش برازش (over-fitting) استفاده می شود. نحوه اثر لایه دراپاوت را در توضیح این لایه در بخش پیش توضیح دادیم. این کار باعث میشود که اثر بخشی از ورودی های این لایه خنثی شوند و عملا در خروجی موثر نباشند. آزمایش نشان داده است که گاهی اوقات که مدل دچار بیش برازش می شود می توان با در نظر نگرفتن برخی ورودی ها به صورت تصادفی، به نوعی در روند آموزش اثری بگذاریم که مدل دچار بیش برازش نشود، لذا از این لایه ها در شبکه های عصبی به خصوص موارد با ابعاد بالا مثل CNN استفاده می شود.