به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین دوم

نام و نام خانوادگی	حمید نعمتی – مهرداد نوربخش
شماره دانشجویی	810100495 – 810100492
تاریخ ارسال گزارش	۱۴۰۱.۰۹.۰۲

فهرست

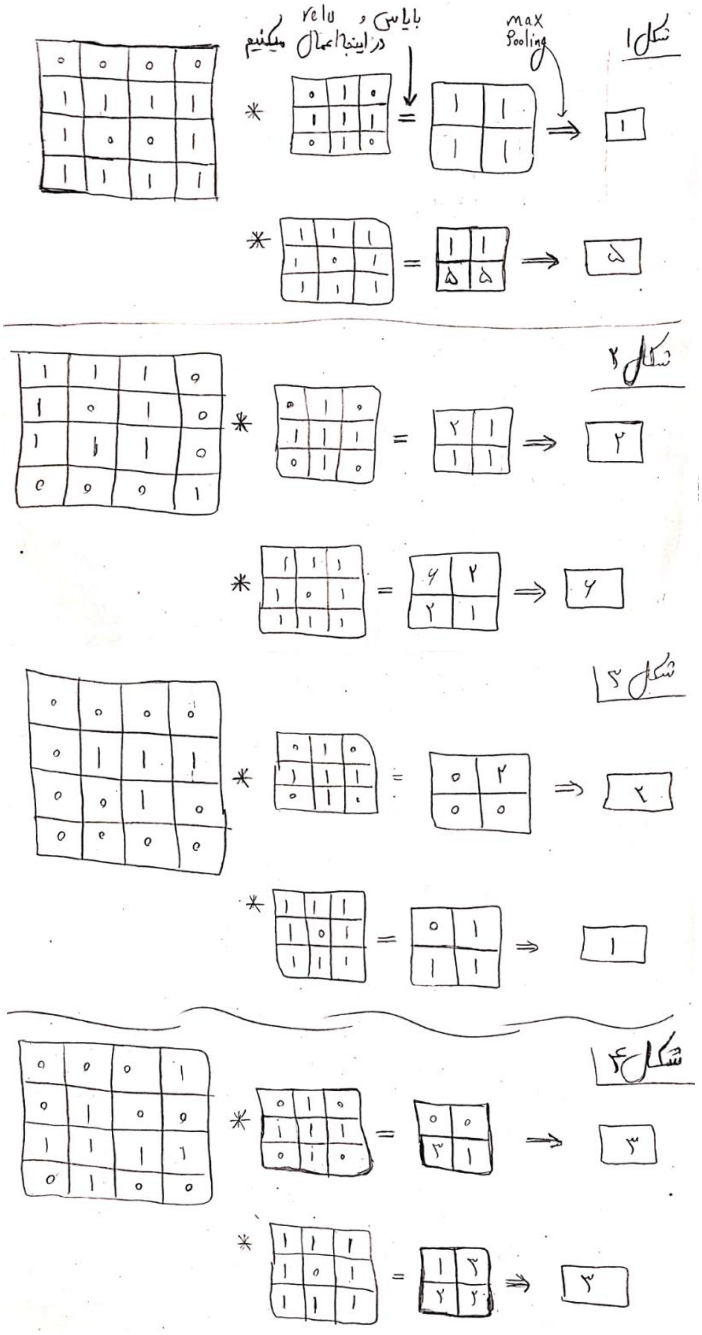
شکل‌ها

جدول‌ها

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

۱-۱. دست گرمی

برای حل این سوال از آنچه در اسلایدهای درس گفته شد بهره میگیریم و به سادگی مسئله را حل میکنیم. ابتدا فیلتر را اعمال سپس بایاس را به ان اضافه میکنیم و Relu را روی حاصل اعمال میکنیم و بزرگترین خانه را به عنوان نتیجه Max-Pooling برمیگردانیم.



شکل 1- دست گرمی با **CNN**

۱-۲. الف

آدرس کولب حاوی کدها:

NN_HW2_Q1.ipynb - Colaboratory (google.com)

بعد از انتخاب ۱۰ عکس با استفاده از resize رزولوشن عکس را تغییر میدهیم. عکس‌ها فقط از داده‌های Train انتخاب شدند.

```
#Generate 10 random numbers between 10 and 30
randomlist = random.sample(range(0, 50000), 10)
resolution = [32, 16, 8]

for res in resolution:
    print("resolution:", res)
    fig = plt.figure(figsize=(15, 6))
    for i, num in enumerate(randomlist):
        ax = fig.add_subplot(1, 10, i + 1, xticks=[], yticks=[])
        ax.set_title(i + 1)
        img = x_train[i]
        img = cv2.resize(img, (res,res))
        plt.imshow(img)
    plt.show()
```

شکل 2- ده عکس انتخاب شده به صورت تصادفی با ۳ رزولوشن مختلف

۱-۳. ب

در تمام مسائل supervised برای اطمینان از دقت مدل در صنعت از train-test-split استفاده میکنیم. از داده train برای اموزش مدل و از test برای ارزیابی مدل روی داده‌های که تا به حال ندیده استفاده میشود. معمولا در مدل‌های یادگیری ماشین تعدادی Hyperparameter وجود دارد که انها هم باید تنظیم شوند که مدل دقتش به حداکثر مقدار ممکنش برسد به همین دلیل قسمت از داده Train را به عنوان validation جدا میکنند تا این ابرپارامترها تنظیم شوند. در مسائلی که به اندازه کافی داده وجود داشته باشد میتوان از train-test-split بهره گرفت، زیرا که داده‌ها باید یک نمونه مناسب از جمعیت واقعی باشند در غیر این صورت تقسیم داده ممکن نیست و باید به سراغ روش‌هایی چون استنباط آماری برویم. اگر داده موجود زیاد نباشد میتوان با استفاده از روش‌هایی چون k-fold cross validation استفاده کنیم تا از داده

نهایت استفاده را ببریم. نمیتوان یک مرز یکسان برای تقسیم داده به اموزش و تست برای همه مسائل تعریف کرد و بستگی به مسئله و میزان داده موجود دارد ولی معمولا درصدهای زیر رایج هستند:

- ۸۰ درصد آموزش و ۲۰ درصد تست
- ۶۷ درصد آموزش و ۳۳ درصد تست
- ۵۰ درصد آموزش و ۵۰ درصد تست

امکان اینکه یکی از کلاس‌ها در داده تست یا آموزش بیشتر از دیگری باشد وجود دارد پس بهتر است از روش stratified استفاده کنیم تا متمن شویم که تقسیم متوازنی از داده‌ها داشته ایم.

برای اینکه بتوانیم نتایج به دست امده را دوباره تکرار کنیم باید random state را هم set کنیم زیرا که به دلیل تقسیم شدن تصادفی داده‌ها هر بار مجموعه متفاوتی به دست میاوریم.

قبل از تقسیم به دلیل شیوه خاص مرتب شدن داده‌ها در صنعت نیاز به shuffle کردن انها وجود دارد. معمولا در حین اموزش مدل مجموعه‌های آموزش و تست باید مرتب shuffle شوند دوباره تا مدل سریعتر همگرا شود.

معمولا از Sklearn.model_selection.train_test_split استفاده میشود که داده را به آموزش و تست تقسیم بندی کنیم. سپس از keras استفاده میکنیم که در داخل تابع fit میتوان با مقدار دادن به validation_split تقسیم داده به اموزش و ارزیابی را انجام داد.

در این مسئله مشکل کمبود داده نداریم پس میتوان نسبت‌های مختلفی به آموزش و تست داد. متاسفانه در مقاله داده شده این نسبت‌ها بیان نشده‌اند و ما با دیفالت خود مسئله جلو میرویم. وقتی داده را دانلود میکنیم قسمت Train و Test جدا میشود پس ما با همان حالت پیشفرض جلو میرویم.

```
[2]  (x_train, y_train), (x_test, y_test) = cifar10.load_data()

     print("x_train:", x_train.shape)
     print("y_train:", y_train.shape)
     print("y_test :", x_test .shape)
     print("y_test :", y_test .shape)

     Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
     170498071/170498071 [==============================] - 2s 0us/step
     x_train: (50000, 32, 32, 3)
     y_train: (50000, 1)
     y_test : (10000, 32, 32, 3)
     y_test : (10000, 1)
```

شکل 3- نسبت داده آموزش به تست

۱-۴. ج

برای تغییر رزولوشن عکس‌ها ابتدا انها را سایز مورد نظر میبریم مثلا 8*8 سپس انها را به 32*32 برمیگردانیم. برگرداندن به سایز اصلی به این دلیل است که ورودی شبکه سایز ثابتی دارد. نمونه ای از این تغییر رزولشن را در زیر میبینیم:

```
for res in resolution:
    print("resolution:", res)
    fig = plt.figure(figsize=(15, 6))
    for i, num in enumerate(randomlist):
        ax = fig.add_subplot(1, 10, i + 1, xticks=[], yticks=[])
        ax.set_title(i + 1)
        img = x_train[i]
        img = cv2.resize(img, (res,res))
        img = cv2.resize(img, (32 ,32 ))
        plt.imshow(img)
    plt.show()
```

شکل 4- ده عکس تصادفی با سه رزولوشن متفاوت

این دقیقا چیزی است که خود مقاله به ان اشاره کرده است:

A. Preparing Varying resolution images

For generating varying resolution sets of the original image, image rescale/resize operation is performed on an original image with a defined set of lower resolutions. For 32x32 resolution image, varying resolution image set contains 8x8, 16x16, 24x24 and 32x32 pixel resolution image. Afterwards, each image is resized into original size for the sake of input tensor of Convolutional Neural Network. Original image with varying resolution images is shown in Figure 3. In this methodology, varying resolution of standard image dataset MNIST and CIFAR10 are prepared.

شکل 5- اشاره مقاله به عمل **resize** کردن

حال باید این عکس‌ها نرمالیزه شوند و به بازه 0 و 1 بیایند زیرا که شبکه در این فضای حالت بهتر میتواند آموزش ببیند. از انجا که رنج عددی هر پیکسل از 0 تا 255 است پس هر پیکسل را بر 255 تقسیم میکنیم و حاصل عکس‌های نرمالایز شده هستند. (البته بهتر بود یک 0.5 هم از حاصل کم بشود تا به جای بازه 0تا1 به بازه0.5تا-0.5 برویم)

```
x_train    = (x_train   /255)
x_train_32 = (x_train_32/255)
x_train_16 = (x_train_16/255)
x_train_8  = (x_train_8 /255)

x_test     = (x_test    /255)
x_test_32  = (x_test_32 /255)
x_test_16  = (x_test_16 /255)
x_test_8   = (x_test_8  /255)
```

شکل 6- نحوه‌ی نرمالایز کردن تصاویر

همچین ستون y را باید به حالت one hot دربیاوریم تا قادر به استفاده از Softmax باشیم.

```
from keras.utils import to_categorical #one-hot encode target column

y_train   = to_categorical(y_train)
y_train_2 = np.argmax(y_test, axis=-1)
y_test    = to_categorical(y_test)
y_test_2  = np.argmax(y_test, axis=-1)

y_train[0]
```

```
array([0., 0., 0., 0., 0., 0., 1., 0., 0., 0.], dtype=float32)
```

شکل 7- تبدیل برچسب **Categorical** به **one-hot**

سپس خود شبکه عصبی را طراحی میکنیم با توجه به آنچه خود مقاله گفته است:

```
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Flatten, MaxPooling2D, Dropout

model = Sequential()

model.add(Conv2D(32, kernel_size=(3,3), activation='relu', input_shape=(32,32,3)))
model.add(Conv2D(32, kernel_size=(3,3), activation='relu'))
model.add(Conv2D(32, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.25))

model.add(Conv2D(64, kernel_size=(3,3), activation='relu'))
model.add(Conv2D(64, kernel_size=(3,3), activation='relu'))
model.add(Conv2D(64, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(10, activation='softmax'))

model.summary()
```

شکل 8- طراحی اولین شبکه با توجه به مقاله

```
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 9- نشان دادن خلاصه مدل

سپس مدل را کامپایل میکنیم و تنظیماتی که وارد کرده‌ایم بعد چند بار آزمایش و خطا انتخاب شده و بهترین نتایج را میداد.

```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=5e-4), loss='categorical_crossentropy', metrics=['accuracy'])
```

```
from keras.callbacks import EarlyStopping
early_stopping = EarlyStopping(patience=10, monitor='val_loss', verbose=1, mode='min', restore_best_weights=True)

start = time()

history = model.fit(x_train_32, y_train, validation_split=0.2, epochs=60, batch_size=64, verbose=1, callbacks=[early_stopping])

print('\n\ntime: ')
print( time() - start )
```

شکل 10- تنظیمات مدل اول

سپس مدل را برای اموزش آماده میکنیم و 20٪ از 50000 داده اموزشی را که یعنی همان مقداری که برای تست کردن داریم را به عنوان داده validation در نظر میگیرم. سایر تنظیمات مدل هم به صورت آزمایش و خطا بعد بارها امتحان کردن به دست امده است.

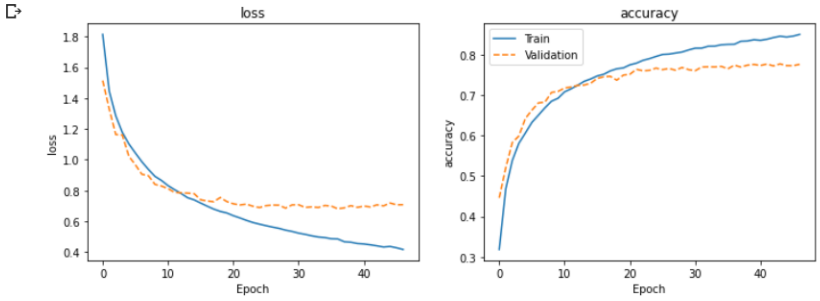
مدل بعد چند دقیقه آموزش خود را متوقف میکند.

```
625/625 [==============================] - 5s 8ms/step - loss: 0.4289 - accuracy: 0.8460 - val_loss: 0.7078 - val_accuracy: 0.7728
Epoch 47/60
622/625 [============================>.] - ETA: 0s - loss: 0.4177 - accuracy: 0.8505Restoring model weights from the end of the best epoch: 37.
625/625 [==============================] - 5s 8ms/step - loss: 0.4181 - accuracy: 0.8503 - val_loss: 0.7088 - val_accuracy: 0.7762
Epoch 47: early stopping


time:
257.1944079399109
```

شکل 11- پایان آموزش مدل اول

برای اطمینان از درستی نمودار دقت و خطا را رسم میکنم:

```
fig     = plt.figure(figsize=(12, 4))
metrics = ['loss', 'accuracy']

for i, metric in enumerate(metrics):
    plt.subplot(1, 2, i+1)
    plt.plot   (history.epoch, history.history[metric], label='Train')
    plt.plot   (history.epoch, history.history[f"val_{metric}"], linestyle="--", label='Validation')
    plt.xlabel ('Epoch')
    plt.ylabel (metric)
    plt.title  (metric)

plt.legend()
plt.show()
```



شکل 12- نمودار دقت و خطا روی داده آموزش و اعتبار سنجی مدل اول

برای حالت ۳۲*۳۲:

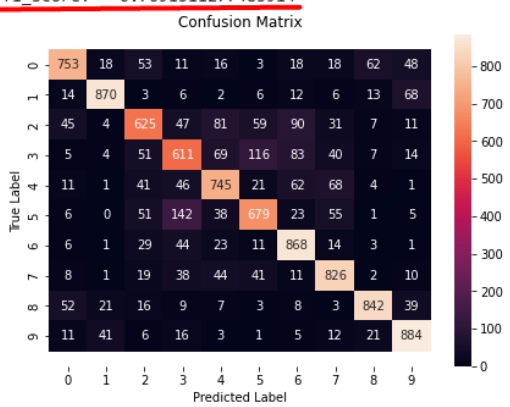
Trained on 32 * 32 and Tested On 32 * 32

```
[ ] [test_loss, test_acc] = model.evaluate(x_test_32, y_test)

    print("Test Loss:", test_loss, "Test Accuracy:", test_acc)
```

```
313/313 [==============================] - 1s 4ms/step - loss: 0.7061 - accuracy: 0.7703
Test Loss: 0.7061281204223633 Test Accuracy: 0.7702999711036682
```

```
[ ] test_preds   = np.argmax(model.predict(x_test_32), axis=-1)

    print("Accuracy:  " , test_acc)
    print("precision: " , precision_score(y_test_2, test_preds, average="macro"))
    print("f1_score:  " , f1_score       (y_test_2, test_preds, average="macro"))

    plot_cm(y_test_2, test_preds)
```

```
313/313 [==============================] - 2s 6ms/step
Accuracy:   0.7702999711036682
precision:  0.7706340551165393
f1_score:   0.7691511277485914
```



شکل 13- نتایج مدل اول روی رزولوشن ۳۲*۳۲

برای حالت ۱۶*۱۶:

به وضوح میتوان تغیرات و افزایش تعداد اشتباه‌ها را در ماتریس کانفوژن دید.

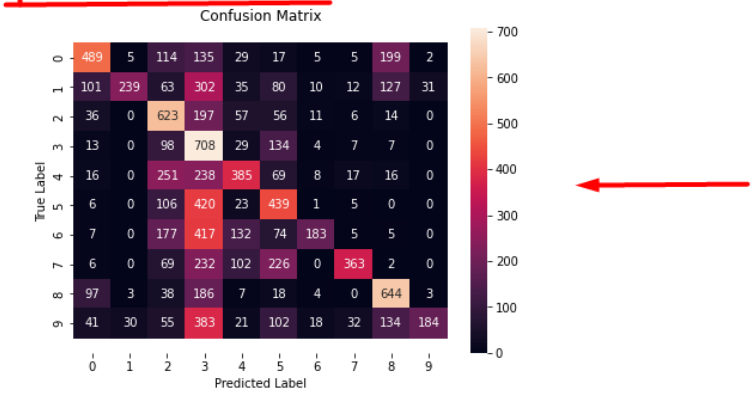
Trained on 32 * 32 and Tested On 16 * 16

```
[ ] [test_loss, test_acc] = model.evaluate(x_test_16, y_test)

    print("Test Loss:", test_loss, "Test Accuracy:", test_acc)
```

```
313/313 [==============================] - 2s 6ms/step - loss: 2.0490 - accuracy: 0.4257
Test Loss: 2.048978805541992 Test Accuracy: 0.42570000886917114
```

```
[ ] test_preds = np.argmax(model.predict(x_test_16), axis=-1)

    print("Accuracy:  " , test_acc)
    print("precision: " , precision_score(y_test_2, test_preds, average="macro"))
    print("f1_score:  " , f1_score       (y_test_2, test_preds, average="macro"))

    plot_cm(y_test_2, test_preds)
```

```
313/313 [==============================] - 2s 6ms/step
Accuracy:   0.42570000886917114
precision:  0.5857151374574888
f1_score:   0.42450375776001364
```



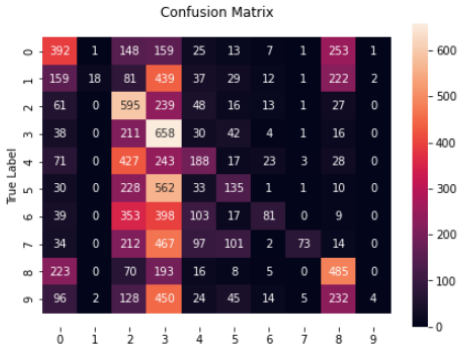
شکل 14- نتایج مدل اول روی رزولوشن ۱۶*۱۶

برای حالت ۸*۸:

Trained on 32 * 32 and Tested On 8 * 8

```
[ ] [test_loss, test_acc] = model.evaluate(x_test_8, y_test)

    print("Test Loss:", test_loss, "Test Accuracy:", test_acc)

    313/313 [==============================] - 1s 4ms/step - loss: 2.8100 - accuracy: 0.2629
    Test Loss: 2.8099567890167236 Test Accuracy: 0.2628999948501587
```

```
    test_preds = np.argmax(model.predict(x_test_8), axis=-1)

    print("\nAccuracy:  " , test_acc)
    print("precision: "   , precision_score(y_test_2, test_preds, average="macro"))
    print("f1_score:  "   , f1_score       (y_test_2, test_preds, average="macro"))
    print("\n")

    plot_cm(y_test_2, test_preds)

    313/313 [==============================] - 1s 3ms/step

    Accuracy:   0.2628999948501587
    precision:  0.4541909327597244
    f1_score:   0.2148304736334028
```



شکل 15- نتایج مدل اول روی رزولوشن ۸*۸

حال که نتایج این قسمت را به صورت کامل نشان دادیم به سراغ قسمت د میرویم و در انتها نتیجه گیری هر دو بخش را به صورت یکجا خواهیم داشت.

قبل از اتمام این بخش یک نگاه هم به مقاله بکنیم:

1) Training with original resolution image dataset and testing with varying resolution image dataset (TOTV):

This training and testing method is used to analyse how the reduction of image resolution affects the performance of classifier which trained on higher resolution images. In this method, Classifier is trained on original resolution train image dataset and evaluated on a set of varying resolution test image dataset. For 32x32 image dataset, Classifier is trained on 32x32 resolution image dataset and evaluated on separately 8x8, 16x16, 24x24 and 32x32 resolution image dataset.

شکل 16- گفته مقاله راجع به روش **TOTV**

دقیقا ما هم همین مراحل را رفته ایم. پس این بخش هم تمام میشود.

۱-۵. د

یک نگاه به خود مقاله میکنیم:

2) Training and testing with each Varying resolution image dataset separately (TVTV):

This training testing method analyses the performance of CNN based image classifier for training and testing with lower resolution images. In this method, Classifiers separately trained on each varying resolution train image dataset and evaluated on the corresponding resolution test image dataset. For 32x32 image dataset, Classifier is trained separately on 8x8, 16x16, 24x24 and 32x32 resolution image dataset and evaluated on corresponding 8x8, 16x16, 24x24 and 32x32 resolution image dataset.

شکل 17- گفته مقاله راجع به روش **TVTV**

ابهام کوچکی وجود دارد این است که آیا وقتی روی ۸*۸ اموزش میدهیم آیا این داده باید به ۳۲*۳۲ resize شود یا نه. ما فرض میکنیم که این کار باید انجام شود و ورودی تمام شبکه‌ها ۳۲*۳۲ است.

برای حالت ۳۲*۳۲:

این قسمت با حالت اول قسمت ج که قبلا حساب کردیم هیچ تفاوتی ندارد.

برای حالت ۱۶*۱۶:

برای این حالت باید دوباره مدل جدیدی آموزش دهیم که با داده رزولشن ۱۶*۱۶ اموزش میدهیم. البته سایز ورودی را همان 32*32 نگه میداریم.

مدل جدید:

Trained on 16 * 16 and Tested On 16 * 16

```
[ ]  from keras.models import Sequential
     from keras.layers import Dense, Conv2D, Flatten, MaxPooling2D, Dropout

     model2 = Sequential()

     model2.add(Conv2D(32, kernel_size=(3,3), activation='relu', input_shape=(32,32,3)))
     model2.add(Conv2D(32, kernel_size=(3,3), activation='relu'))
     model2.add(Conv2D(32, kernel_size=(3,3), activation='relu'))
     model2.add(MaxPooling2D(pool_size=2))
     model2.add(Dropout(0.25))

     model2.add(Conv2D(64, kernel_size=(3,3), activation='relu'))
     model2.add(Conv2D(64, kernel_size=(3,3), activation='relu'))
     model2.add(Conv2D(64, kernel_size=(3,3), activation='relu'))
     model2.add(MaxPooling2D(pool_size=2))
     model2.add(Dropout(0.25))

     model2.add(Flatten())

     model2.add(Dense(512, activation='relu'))
     model2.add(Dropout(0.5))

     model2.add(Dense(10, activation='softmax'))

     model2.summary()
```

شکل 18- مدل دوم با طراحی مشابه مدل اول

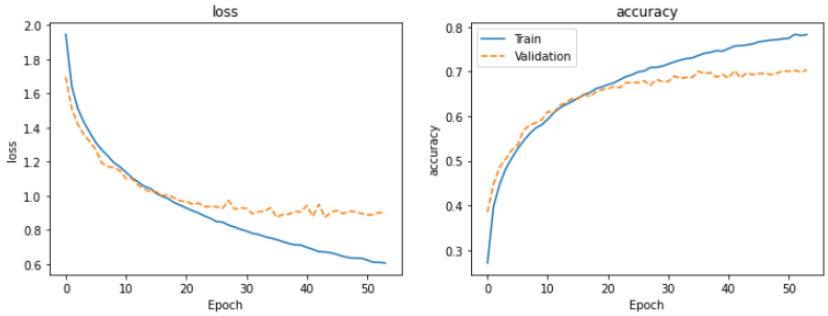
مدل را چند بار اموزش میدهیم و هایپرپارامترهایی که بیشترین دقت را داد انتخاب کردیم و در نهایت برای اطمینان از درستی آموزش نمودار خطا و دقت را رسم میکنیم:

```
model2.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=5e-4), loss='categorical_crossentropy', metrics=['accuracy'])
```

```
from keras.callbacks import EarlyStopping
early_stopping = EarlyStopping(patience=10, monitor='val_loss', verbose=1, mode='min', restore_best_weights=True)

start = time()

history2 = model2.fit(x_train_16, y_train, validation_split=0.3, epochs=60, batch_size=128, verbose=1, callbacks=[early_stopping])

print('\n\ntime: ')
print( time() - start )
```

شکل 19- تنظیمات مدل دوم

```
fig     = plt.figure(figsize=(12, 4))
metrics = ['loss', 'accuracy']

for i, metric in enumerate(metrics):
    plt.subplot(1, 2, i+1)
    plt.plot   (history2.epoch, history2.history[metric], label='Train')
    plt.plot   (history2.epoch, history2.history[f"val_{metric}"], linestyle="--", label='Validation')
    plt.xlabel ('Epoch')
    plt.ylabel (metric)
    plt.title  (metric)

plt.legend()
plt.show()
```



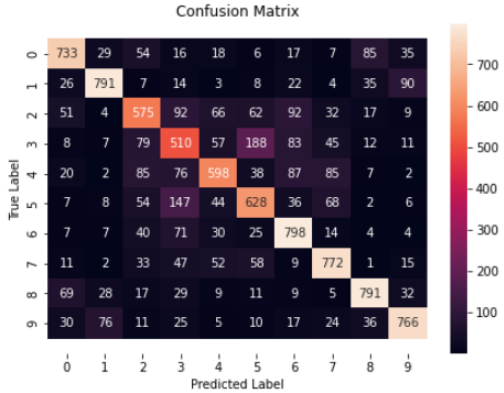
شکل 20- نمودار دقت و خطا روی داده آموزش و اعتبار سنجی مدل دوم

```
[test_loss, test_acc] = model2.evaluate(x_test_16, y_test)

print("Test Loss:", test_loss, "Test Accuracy:", test_acc)
```

```
313/313 [==============================] - 1s 4ms/step - loss: 0.8935 - accuracy: 0.6962
Test Loss: 0.8934670090675354 Test Accuracy: 0.6962000131607056
```

```
test_preds = np.argmax(model2.predict(x_test_16), axis=-1)

print("\nAccuracy:  " , test_acc)
print("precision: "   , precision_score(y_test_2, test_preds, average="macro"))
print("f1_score:  "   , f1_score       (y_test_2, test_preds, average="macro"))
print("\n")

plot_cm(y_test_2, test_preds)
```

```
313/313 [==============================] - 1s 2ms/step

Accuracy:   0.6962000131607056
precision:  0.6976928120197712
f1_score:   0.6960353127194416
```



شکل 21- نتایج مدل دوم روی رزولوشن ۱۶*۱۶

برای حالت ۸*۸:

برای این حالت باید دوباره مدل جدیدی آموزش دهیم که با داده رزولشن ۸*۸ اموزش میدهیم. البته سایز ورودی را همان 32*32 نگه میداریم.

Trained on 8 * 8 and Tested On 8 * 8

```
[ ] from keras.models import Sequential
    from keras.layers import Dense, Conv2D, Flatten, MaxPooling2D, Dropout

    model3 = Sequential()

    model3.add(Conv2D(32, kernel_size=(3,3), activation='relu', input_shape=(32,32,3)))
    model3.add(Conv2D(32, kernel_size=(3,3), activation='relu'))
    model3.add(Conv2D(32, kernel_size=(3,3), activation='relu'))
    model3.add(MaxPooling2D(pool_size=2))
    model3.add(Dropout(0.25))

    model3.add(Conv2D(64, kernel_size=(3,3), activation='relu'))
    model3.add(Conv2D(64, kernel_size=(3,3), activation='relu'))
    model3.add(Conv2D(64, kernel_size=(3,3), activation='relu'))
    model3.add(MaxPooling2D(pool_size=2))
    model3.add(Dropout(0.25))

    model3.add(Flatten())

    model3.add(Dense(512, activation='relu'))
    model3.add(Dropout(0.5))

    model3.add(Dense(10, activation='softmax'))

    model3.summary()
```

شکل 22- مدل سوم با طراحی مشابه مدل اول

تنظیم هایپرپارامترها:

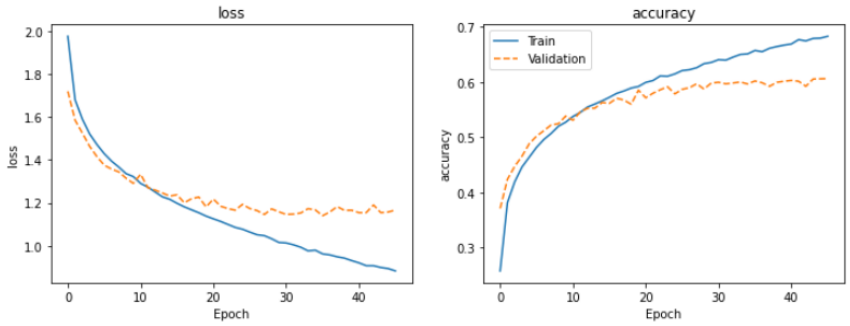
```
model3.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=5e-4), loss='categorical_crossentropy', metrics=['accuracy'])
```

```
from keras.callbacks import EarlyStopping

early_stopping = EarlyStopping(patience=10, monitor='val_loss', verbose=1, mode='min', restore_best_weights=True)

start = time()

history3 = model3.fit(x_train_8, y_train, validation_split=0.2, epochs=60, batch_size=128, verbose=1, callbacks=[early_stopping])

print('\n\ntime: ')
print( time() - start )
```

شکل 23- تنظیمات مدل سوم

تست کردن درستی آموزش:

```
fig     = plt.figure(figsize=(12, 4))
metrics = ['loss', 'accuracy']

for i, metric in enumerate(metrics):
    plt.subplot(1, 2, i+1)
    plt.plot   (history3.epoch, history3.history[metric], label='Train')
    plt.plot   (history3.epoch, history3.history[f"val_{metric}"], linestyle="--", label='Validation')
    plt.xlabel ('Epoch')
    plt.ylabel (metric)
    plt.title  (metric)

plt.legend()
plt.show()
```



شکل 24- نمودار دقت و خطا روی داده آموزش و اعتبار سنجی مدل سوم

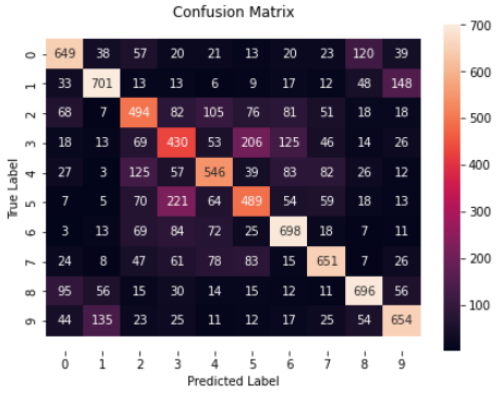
نتایج:

```
[test_loss, test_acc] = model3.evaluate(x_test_8, y_test)

print("Test Loss:", test_loss, "Test Accuracy:", test_acc)
```

```
313/313 [==============================] - 1s 3ms/step - loss: 1.1372 - accuracy: 0.6008
Test Loss: 1.1371606588363647 Test Accuracy: 0.6007999777793884
```

```
test_preds = np.argmax(model3.predict(x_test_8), axis=-1)

print("\nAccuracy:  " , test_acc)
print("precision: "   , precision_score(y_test_2, test_preds, average="macro"))
print("f1_score:  "   , f1_score       (y_test_2, test_preds, average="macro"))
print("\n")

plot_cm(y_test_2, test_preds)
```

```
313/313 [==============================] - 1s 2ms/step

Accuracy:   0.6007999777793884
precision:  0.6008720572600236
f1_score:   0.6005465775876655
```



شکل 25- نتایج مدل سوم روی رزولوشن ۸*۸

تحلیل و نتیجه گیری بخش ج و د

خب نگاهی به دقت‌هایی که خود مقاله به دست اورده میکنیم:

Table 3 Performance result of experimental study on CIFAR10 dataset

CIFAR10 Dataset Resolution	TOTV Trained on original resolution dataset (32x32) and tested on varying resolution dataset (32x32, 24x24, 16x16, 8x8)			TVTV Trained and tested on each varying resolution dataset separately (32x32, 24x24, 16x16, 8x8)		
	Accuracy	Precision	F1 Score	Accuracy	Precision	F1 Score
32x32	0.8752	0.87652	0.87548	0.8752	0.87652	0.87548
24x24	0.6409	0.72365	0.65320	0.6204	0.70501	0.63220
16x16	0.3166	0.48415	0.29897	0.4233	0.62030	0.40654
8x8	0.1855	0.27090	0.13986	0.3020	0.54599	0.24262

شکل 26- خلاصه نتایج مقاله

حال ما هم خلاصه نتایج را در جدولی میاوریم:

جدول 1- خلاصه نتایج ما

CIFAR10	TOTV			TVTV		
Resolution	Accuracy	Precision	F1 Score	Accuracy	Precision	F1 Score
32*32	77	77	77	77	77	77
16*16	43	59	42	69	69	69
8*8	21	45	26	60	60	60

به دلیل راندوم seed یا تفاوت هایپرپارامترها دقیقا به اعدادی که مقاله به دست آورده است نرسیدیم ولی به اعدادی نزدیکی رسیدیم که دقیقا همان نتایج را میتوان گرفت.

دلیل دیگر این مسئله میتواند این باشد که ما از Model Checkpoint استفاده نکرده ایم. Keras به صورت پیشفرض آخرین مدل را برمیگرداند نه بهترین مدل را. هر چند که با توجه به نمودارهایی که رسم کردیم دیدیم که تفاوت چندانی در validation رخ نداده بود از یک جایی به بعد.

همچنین ممکن است مقاله درصدهایی متفاوتی برای داده اموزش و اعتبار سنجی و تست در نظر گرفته باشد که بسیار محتمل است.

حال خود اعداد را اگر بخواهیم تحلیل کنیم میبینیم که افت شدیدی در حالت TOTV داشته ایم ولی این افت دقت در TVTV بسیار کمتر است. این یعنی اگر مدلی را اموزش دهیم و انتظار داشته باشیم همه عکس‌هایی که از ورودی میگیرد را صرفا با resize به رزولوشن مورد نظر ببریم و دقت کافی بگیریم سخت در اشتباه هستیم و مدل باید بر روی رزولوشن‌های مختلف آموزش ببیند. با اینکه عکس‌هایی که رزولوشن ۸*۸ دارند برای انسان هم تشخیص آنها سخت است ولی شبکه عصبی بسیار خوب عمل کرده و دقت فوق العاده‌ای دارد. میتوان نتیجه گرفت که به جای استفاده از عکس‌های با رزولوشن بسیار بالا میتوان با همان رزولوشن متوسط هم به دقت‌های بسیار خوبی رسید و نیازی به اموزش دادن شبکه‌ای با ورودی چندهزار در چندهزار نیست و میتوان با شبکه ای کوچکتر که سریعتر هم اموزش میبینند در زمان کوتاه‌تری به دقت مورد نظر رسید.

هم در مقاله هم در اعداد به دست اورده شده توسط خودمان میبینیم که precision با سرعت کمتری افت میکند که معادل ان است که recall با سرعت بیشتری افت کرده است. پس با کاهش رزولوشن وقتی مدل نظر میدهد هنوز از نظر خودش اطمینان بالایی دارد و این یعنی ماتریس کانفیوژن هنوز در قطر اصلی سنگینی میکند که دقیقا این را در شکل‌ها دیدیم.

خب در اینجا سوال یک به صورت کامل به پایان میرسد :)

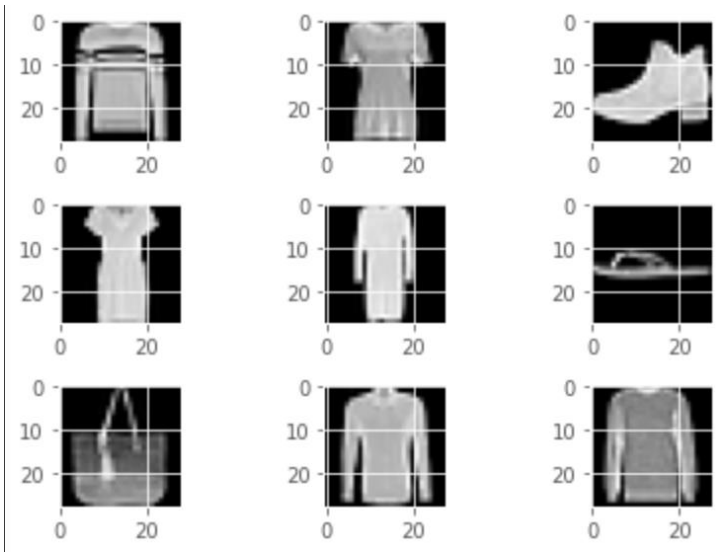
پاسخ ۲ – آشنایی با معماری شبکه CNN

آدرس کولب حاوی کدهای این سوال:

HW2_Q2.ipynb - Colaboratory (google.com)

۲-۱. لود دیتاست مقاله

ابتدا به کمک کتابخانه‌ی keras دیتاست fashion MNIST را لود می‌کنیم. این دیتاست شامل 70 هزار نمونه است که 60 هزار نمونه‌ی آن مربوط به داده‌های train و 10 هزار نمونه‌ی آن مربوط به داده‌های test است. به طور دلخواه چند نمونه از داده‌ها را رسم می‌کنیم. نتیجه در شکل زیر نمایش داده شده است.



شکل 27- چند نمونه از داده‌های **fashion MNIST**

همچنین می‌دانیم که داده‌های موجود در این دیتاست شامل کلاس‌های زیر می‌شود.

```
labels = [
        'T-shirt/top',
        'Trouser',
        'Pullover',
        'Dress',
        'Coat',
        'Sandal',
        'Shirt',
        'Sneaker',
        'Bag',
        'Ankle boot',
    ]
```

شکل 28- کلاس‌های **fashion MNIST**

پیش از آن که به سراغ CNN برویم باید داده‌ها را پیش‌پردازش کنیم. از آنجایی که این تصاویر سیاه و سفید هستند و ابعاد (28,28) دارند، یک بعد جدید به هر عکس اضافه می‌کنیم تا به صورت استاندارد سه کاناله در بیایند. بنابراین ابعاد هر عکس به صورت (28,28,1) خواهد بود. پس از تغییر ابعاد، داده‌ها را نرمال می‌کنیم. می‌دانیم که عکس‌ها Grayscale هستند و بازه‌ای از 0 تا 255 دارند بنابراین برای آن که داده‌ها را به اعداد بین 0 تا 1 تبدیل کنیم آن‌ها را بر 255 تقسیم می‌کنیم.

برای داده‌های target نیز از to_categorical از توابع keras استفاده می‌کنیم تا خروجی را به صورت ماتریس باینری در بیاوریم که در آن خانه‌ای که کلاس واقعی نمونه است، 1 است و باقی خانه‌ها 0 هستند. سایز هر ماتریس نیز به اندازه‌ی تعداد کلاس‌ها یعنی 10 است. پس از انجام پیش‌پردازش می‌توانیم به سراغ طراحی مدل برویم.

۲-۲. انتخاب معماری

در این قسمت دو معماری از معماری‌های معرفی شده در مقاله را انتخاب می‌کنیم و آن‌ها را پیاده‌سازی می‌کنیم. معماری‌های انتخاب شده، معماری‌های شماره 2 و 3 هستند. هر کدام از این معماری‌ها را به همراه پارامترهای بهینه‌ی گفته شده در مقاله پیاده‌سازی می‌کنیم.

معماری 2:

Architecture 2
2 convolutional layers with (2 x 2) filter size and 2 fully connected layers
(1) INPUT:28×28×1 (2) FC:10 Output Classes
(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

شکل 29- ساختار معماری **2** در مقاله

معماری 2 همانطور که دیده می‌شود از 2 لایه‌ی کانولوشن و 2 لایه‌ی FC تشکیل شده است. لایه‌های کانولوشن هر کدام دارای 64 فیلتر و سایز 2*2 هستند. بعد از کانولوشن لایه‌ی اول، یک لایه‌ی MaxPooling با سایز 2*2 داریم و پس از هر لایه نیز Dropout به اندازه‌ی 0.25 داریم. پارامترهای بهینه‌ی ذکر شده در مقاله برای این معماری در شکل زیر نمایش داده شده است. این قسمت با هایلایت آبی مشخص شده است.

	70	96.90%	89.04%	99.94%	98.22%
	100	97.99%	88.87%	99.96%	98.23%
Adam	50	97.38%	88.59%	99.20%	98.12%

epochs and 2x2 kernel size.

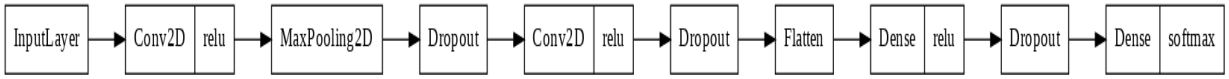
For Fashion-MNIST dataset, best training accuracy and testing accuracy obtained are 92.02% and 92.76% respectively. The best result obtained with 128 Batch size, softmax activation function,

Journal of Scientific Research, Volume 64, Issue 2, 2020

adam optimizer, 0.25 dropout after each pooling layer, 50 epochs and 2x2 kernel size.

شکل 30- پارامترهای بهینه در معماری **2**

علاوه بر این پارامترها، برای loss از categorical_crossentropy استفاده می‌کنیم. از آنجایی که مسئله‌ی ما طبقه‌بندی است این تابع loss مناسب است. metric را نیز accuracy در نظر می‌گیریم. همچنین activation function های لایه‌های میانی را نیز طبق مقاله relu در نظر می‌گیریم. مدل ساخته شده در شکل زیر نمایش داده شده است.



شکل 31- مدل پیاده‌سازی شده برای معماری **2**

همچنین در شکل‌های زیر summary و کد زده شده برای این معماری نیز مشاهده می‌شود.

```
#Architecture 2
model_1 = Sequential()
model_1.add(Conv2D(64, kernel_size=(2,2), activation='relu', input_shape=(28,28,1),padding='same'))
model_1.add(MaxPooling2D(pool_size=(2, 2)))
model_1.add(Dropout(0.25))
model_1.add(Conv2D(64, kernel_size=(2,2), activation='relu',padding='same'))
model_1.add(Dropout(0.25))
model_1.add(Flatten())
model_1.add(Dense(64, activation='relu'))
model_1.add(Dropout(0.25))
model_1.add(Dense(10, activation='softmax'))

model_1.compile(optimizer='adam',metrics=['accuracy'],loss='categorical_crossentropy')
```

شکل 32- کد زده شده برای معماری **2**

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 28, 28, 64)        320

 max_pooling2d_2 (MaxPooling  (None, 14, 14, 64)       0
 2D)

 dropout_6 (Dropout)         (None, 14, 14, 64)        0

 conv2d_5 (Conv2D)           (None, 14, 14, 64)        16448

 dropout_7 (Dropout)         (None, 14, 14, 64)        0

 flatten_2 (Flatten)         (None, 12544)             0

 dense_4 (Dense)             (None, 64)                802880

 dropout_8 (Dropout)         (None, 64)                0

 dense_5 (Dense)             (None, 10)                650

=================================================================
Total params: 820,298
Trainable params: 820,298
Non-trainable params: 0
_________________________________________________________________
```

شکل 33- **summary** مدل زده شده برای معماری **2**

معماری 3:

ساختار معماری 3 در شکل زیر نمایش داده شده است.

Architecture 3
3 convolutional layers with (2 x 2) filter size and 2 fully connected layers
(1) INPUT:28×28×1 (2) FC:10 Output Classes
(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

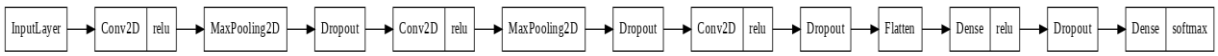
شکل 34- ساختار معماری **3** در مقاله

این معماری نسبت به معماری 2، یک لایه‌ی کانولوشن اضافی و یک لایه‌ی اضافی MaxPooling دارد. اندازه‌های استفاده شده در این معماری نیز مشابه معماری 2 است. پارامترهای بهینه در این معماری در شکل زیر نمایش داده شده است.

For Fashion-MNIST dataset, best training accuracy and testing accuracy obtained are 93.09% and 93.56% respectively. The best result obtained with 128 Batch size, softmax activation function, adam optimizer, 0.25 dropout after each pooling layer, 50 epochs and 2x2 kernel size.

شکل 35- پارامترهای بهینه در معماری **3**

پارامترهای استفاده شده نیز مشابه پارامترهای معماری 2 است. باقی پارامترهای از جمله loss و metric نیز مانند معماری 2 در نظر گرفته شده است. مدل ساخته‌شده در شکل زیر نمایش داده شده است.



شکل 36- مدل پیاده‌سازی شده برای معماری **3**

همچنین در شکل‌های زیر summary و کد زده شده برای این معماری نیز مشاهده می‌شود.

```
#Architecture 3
model_2 = Sequential()
model_2.add(Conv2D(64, kernel_size=(2,2), activation='relu', input_shape=(28,28,1),padding='same'))
model_2.add(MaxPooling2D(pool_size=(2, 2)))
model_2.add(Dropout(0.25))
model_2.add(Conv2D(64, kernel_size=(2,2), activation='relu',padding='same'))
model_2.add(MaxPooling2D(pool_size=(2, 2)))
model_2.add(Dropout(0.25))
model_2.add(Conv2D(64, kernel_size=(2,2), activation='relu',padding='same'))
model_2.add(Dropout(0.25))
model_2.add(Flatten())
model_2.add(Dense(64, activation='relu'))
model_2.add(Dropout(0.25))
model_2.add(Dense(10, activation='softmax'))

model_2.compile(optimizer='adam',metrics=['accuracy'],loss='categorical_crossentropy')
```

شکل 37- کد زده شده برای معماری **3**

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_5 (Conv2D)           (None, 28, 28, 64)        320

 max_pooling2d_3 (MaxPooling  (None, 14, 14, 64)       0
 2D)

 dropout_7 (Dropout)         (None, 14, 14, 64)        0

 conv2d_6 (Conv2D)           (None, 14, 14, 64)        16448

 max_pooling2d_4 (MaxPooling  (None, 7, 7, 64)         0
 2D)

 dropout_8 (Dropout)         (None, 7, 7, 64)          0

 conv2d_7 (Conv2D)           (None, 7, 7, 64)          16448

 dropout_9 (Dropout)         (None, 7, 7, 64)          0

 flatten_2 (Flatten)         (None, 3136)              0

 dense_4 (Dense)             (None, 64)                200768

 dropout_10 (Dropout)        (None, 64)                0

 dense_5 (Dense)             (None, 10)                650

=================================================================
Total params: 234,634
Trainable params: 234,634
Non-trainable params: 0
```

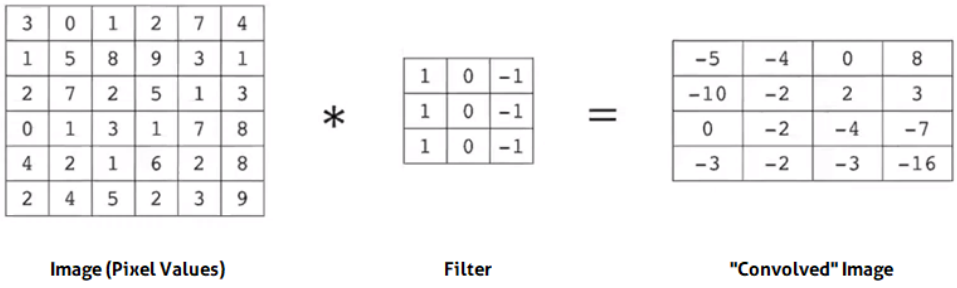
شکل 38- **summary** مدل زده شده برای معماری **3**

۲-۳. توضیح لایه‌های مختلف معماری

دو معماری طراحی شده را مشاهده کردیم و دیدیم که ساختار لایه‌های آن‌ها مشابه یکدیگر بود و تنها در تعداد لایه‌های کانولوشن و MaxPooling تفاوت داشتند. هر کدام از این لایه‌ها را جداگانه بررسی می‌کنیم.

لایه‌ی کانولوشن(convolutional layers):

این لایه اولین لایه‌ی است که در ساختار CNN می‌آید که یک سری فیلتر است که بر روی هر کدام از عکس‌ها اعمال می‌شود. این فیلتر مانند یک پنجره عمل می‌کند که در هر مرحله بر روی عکس حرکت می‌کند و فیلتر را اعمال می‌کند که به این فرایند کانولوشن می‌گویند. هر بار پیکسل‌هایی از عکس که در این پنجره قرار دارند، با استفاده از dot product به یک عدد تبدیل می‌شوند.



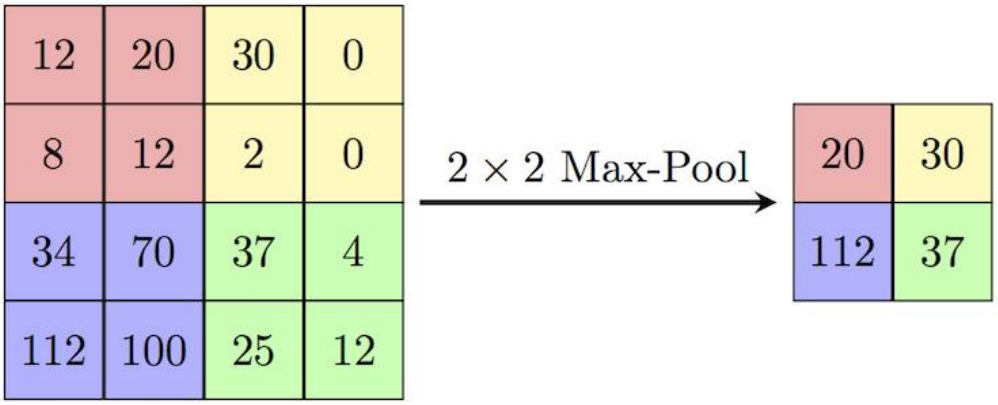
شکل 39- لایه‌ی کانولوشن

با اعمال این فیلتر، ارتباط میان پیکسل‌ها در یک تصویر، ارتباط معنادارتری می‌شود و به مدل کمک می‌کند درک بهتری از تصویر داشته باشد. هدف اصلی در استفاده از لایه‌ی کانولوشن، استخراج ویژگی از تصاویر است که در واقع با حذف پیکسل‌های نامربوط و یافتن پیکسل‌های معنادار، این ویژگی‌ها استخراج می‌شوند. پس از استفاده از کانولوشن، در انتها یک تابع فعالساز مانند relu بر روی نتیجه‌ی بدست آمده اعمال می‌شود تا مدل از حالت خطی خارج شود. استفاده از لایه‌ی کانولوشن علاوه بر استخراج ویژگی، باعث کاهش پارامترهای اضافی و ساده‌تر شدن مدل می‌شود.

لایه‌ی MaxPooling:

پس از لایه‌ی کانولوشن، لایه‌ی MaxPooling می‌آید. لایه‌ی Pooling نیز یک همسایگی از پیکسل‌ها را انتخاب کرده و آن‌ها را در یک گروه قرار داده و یک عدد را به عنوان نماینده‌ی آن گروه قرار می‌دهد.

به طور کلی عملکرد این لایه شبیه به کانولوشن است و در واقع نوعی فیلتر است با این تفاوت که در اینجا فیلتر بر روی ماتریس ویژگی‌ها(خروجی لایه‌ی کانولوشن) حرکت نمی‌کند.



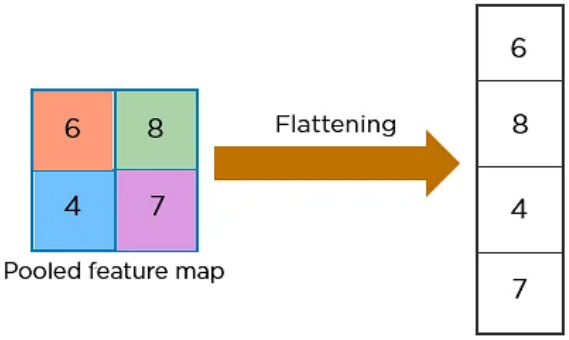
شکل 40- لایه‌ی **MaxPooling**

در MaxPooling نماینده‌ی هر گروه از پیکسل‌ها، ماکزیمم آن گروه است. برای مثال در شکل فوق که یک Pool 2*2 داریم، اعدادی که در پنجره‌ای به سایز 2*2 قرار می‌گیرند با هم یک همسایگی در نظر گرفته می‌شوند و ماکزیمم آن‌ها به عنوان نتیجه برگردانده می‌شود. استفاده از این لایه نیز به طور چشمگیری باعث کاهش بعد دیتای ما و کاهش تعداد پارامترهای مدل می‌شود.

پس از اعمال MaxPooling، نتیجه وارد لایه‌ی Dropout می‌شود که این لایه را در قسمت آخر توضیح خواهیم داد.

لایه‌ی Flatten:

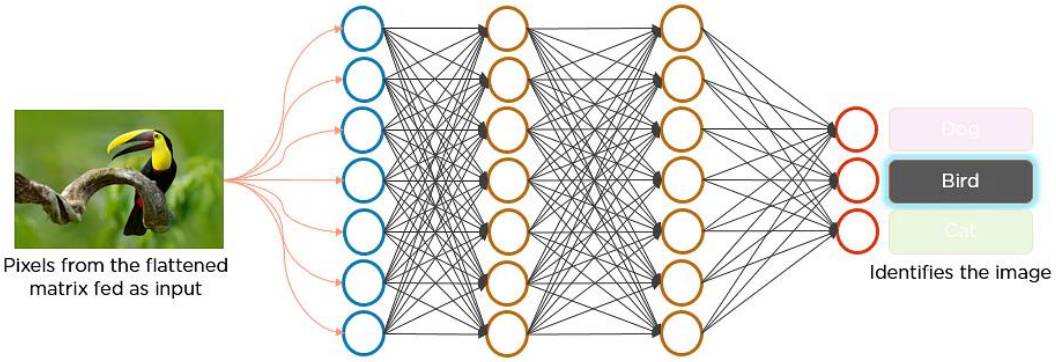
خروجی لایه‌ی MaxPooling پس از اعمال Dropout، وارد لایه‌ی Flatten می‌شود. این لایه ماتریس دو بعد ویژگی‌ها را به یک وکتور خطی طولانی تبدیل می‌کند. این کار برای این انجام می‌شود که بتوان این داده را به عنوان ورودی به لایه‌ی fully connected داد.



شکل 41- لایه‌ی **Flatten**

لایه‌ی Dense یا Fully Connected(FC):

این لایه نیز شامل تعدادی نورون است که این نورون‌ها همه به یکدیگر متصل هستند و به همین دلیل به آن fully connected گفته می‌شود. پس از دریافت ورودی از لایه‌ی Flatten، هر نورون با استفاده از ماتریس وزن، یک linear transformation را بر روی وکتور ورودی اعمال می‌کند. پس از آن یک non-linear transformation از طریق تابع فعال‌ساز مانند relu اعمال می‌شود. در نهایت وزن‌هایی که بدست می‌آیند به مدل کمک می‌کنند تا کلاس داده‌ی ورودی را پیش‌بینی کند. ساختار این قسمت از شبکه‌ی CNN مانند MLP است.



شکل 42- لایه‌ی **FC**

تعداد لایه‌های Dense در شبکه، و تعداد نورون‌های هر لایه می‌تواند بسته به ساختار مسئله متفاوت باشد.

لایه‌ی خروجی:

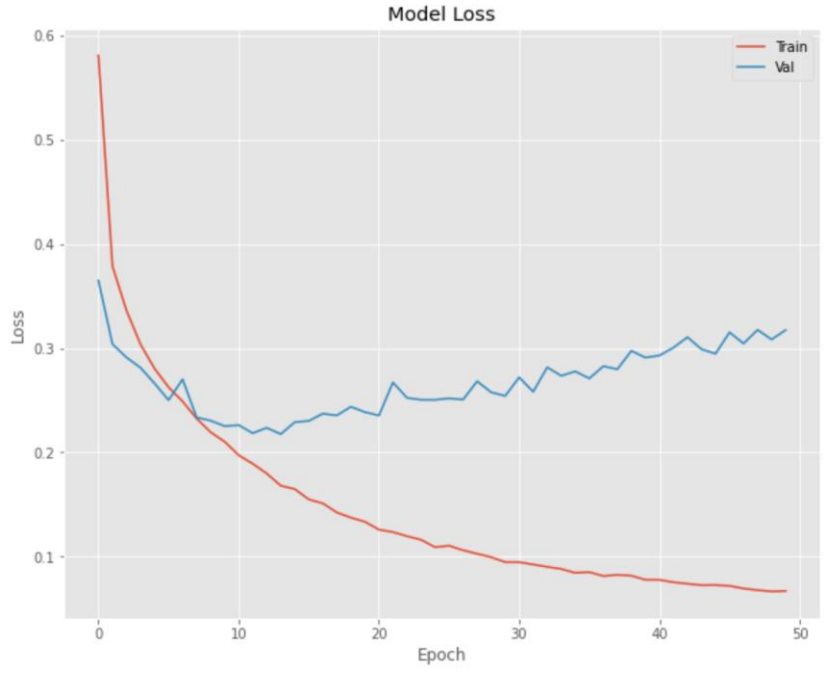
آخرین لایه‌ی شبکه است که در واقع یک لایه‌ی Dense است. تعداد نورون‌های موجود در این لایه به تعداد کلاس‌های موجود در دیتا است. این لایه، خروجی آخرین لایه‌ی hidden را دریافت می‌کند و خروجی مدل را تولید می‌کند. تابع فعال‌ساز در این لایه، softmax است که ورودی را دریافت می‌کند و به ازای هر کلاس در دیتا، یک احتمال را به عنوان خروجی مدل برای آن دیتا در نظر می‌گیرد در نهایت یک وکتور از احتمالات رو به عنوان خروجی برمی‌گرداند. در این وکتور کلاسی که بیشترین مقدار احتمال را داشته باشد به عنوان پیش‌بینی مدل برای آن دیتا انتخاب می‌شود.

دیدیم که در معماری‌های 2 و 3 که آن‌ها را پیاده‌سازی کردیم، همه چیز یکسان بود به جز تعداد لایه‌های کانولوشن و Pooling. در معماری 3 یک لایه‌ی اضافه کانولوشن و Pooling داریم. همانطور که گفتیم از این دو لایه برای استخراج ویژگی از ورودی استفاده می‌شود. زمانی که یک لایه‌ی کانولوشن را بر روی داده‌ی اصلی اعمال می‌کنیم ویژگی‌های low-level مانند گوشه‌های تصویر را استخراج می‌کنیم. زمانی

که بر روی خروجی لایه‌ی کانولوشن اول، مجددا کانولوشن اعمال می‌کنیم، ویژگی‌هایی که استخراج می‌کنیم ترکیبی از ویژگی‌های low-level مرحله‌ی قبل می‌شوند و ویژگی‌های abstract تری استخراج می‌شوند. مثلا از ترکیب گوشه‌ها ممکن است به یک شکل هندسی مانند مستطیل برسیم. هر چه این کار را تکرار کنیم ویژگی‌های استخراج شده پیچیده‌تر می‌شوند. این که چه تعداد لایه‌ی کانولوشن داشته باشیم کاملا بستگی به مسئله و دیتای ما دارد. با افزایش این لایه‌ها لزوما دقت مدل بهتر نمی‌شود چرا که ممکن است مدل، ویژگی‌های پیچیده‌ای را استخراج کند که در کلاس‌های مختلف مشترک‌اند و کمکی به تشخیص این کلاس‌ها از یکدیگر نمی‌کند. در مقاله نیز دیدیم که معماری 4 که 4 لایه‌ی کانولوشن دارد، نسبت به معماری 3 که 3 لایه‌ی کانولوشن دارد، دقت کمتری در داده‌ی آموزش و تست دارد.

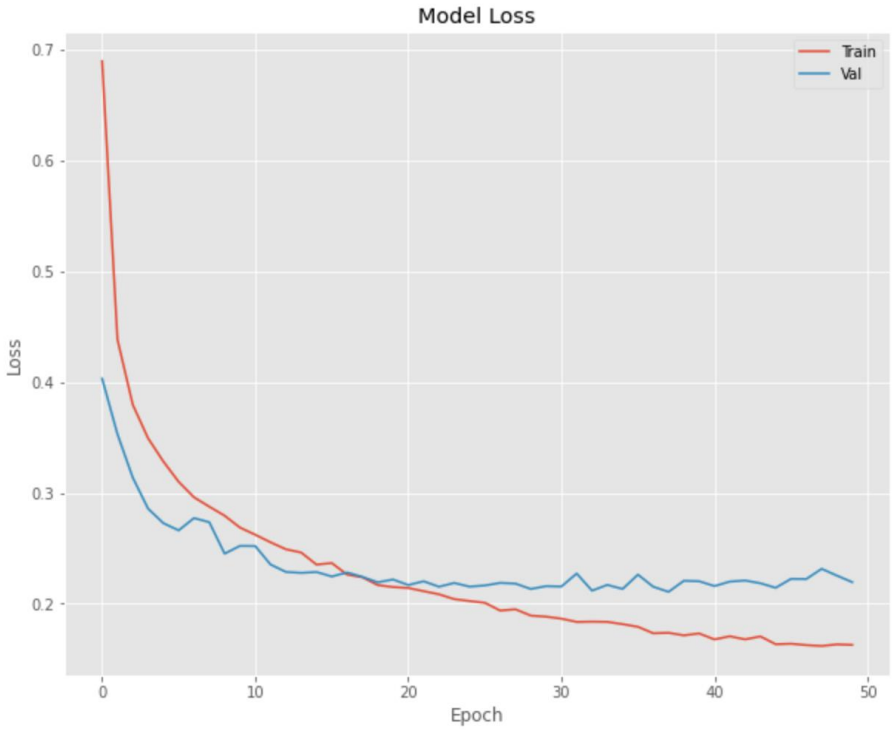
۴-۲. مقایسه‌ی نتایج دو معماری مختلف

مدل‌های طراحی شده را با پارامترهای ذکر شده در قسمت قبل آموزش می‌دهیم. گفتیم که 60 هزار داده‌ی آموزش داریم که از این مقدار 20 درصد را به عنوان داده‌ی validation استفاده می‌کنیم و 80 درصد را نیز داده‌ی آموزش در نظر می‌گیریم. 10 هزار نمونه نیز داده‌های تست هستند که برای ارزیابی مدل از آن‌ها استفاده می‌کنیم. نتیجه‌ی loss مدل برای معماری 2 در شکل زیر نمایش داده شده است.



شکل 43- نمودار **loss** برای معماری **2**

می‌بینیم که مقدار loss برای پس از مدتی برای داده‌های validation، افزایش یافته است که می‌تواند نشان‌دهنده‌ی این باشد که مدل ممکن است به سمت overfit شدن پیش برود و بر روی داده‌هایی که ندیده است به خوبی جواب ندهد و generalization کمتری داشته باشد. هر چند که با توجه به اسکیل نمودار loss و مقادیر محور y، این اختلاف ناچیز است اما احتمال overfit شدن وجود دارد.



شکل 44- نمودار **loss** برای معماری **3**

در معماری 3 می‌بینیم که مدل عملکردی بهتری بر روی داده‌های validation داشته است و کمی generalization بهتری نسبت به معماری 2 دارد.

مقادیر accuracy بدست آمده برای هر کدام از تو مدل نیز در جدول زیر نمایش داده شده است.

جدول 2- دقت در دو معماری مختلف

مدل	Accuracy	Train	Test
معماری 2		98.16 %	92.03%
معماری 3		96.26%	92.29%

مشاهده می‌کنیم که دقت معماری 3 بر روی داده‌های تست کمی بهتر از معماری 2 شده است که بر اساس آن چه در نمودار loss مشاهده کردیم همین انتظار نیز می‌رفت.

برای گزارش مقادیر precision و f1-score از classification_report استفاده می‌کنیم. نتیجه در شکل‌های زیر نمایش داده شده است.

	precision	recall	f1-score	support
T-shirt/top	0.86	0.88	0.87	1000
Trouser	0.99	0.98	0.98	1000
Pullover	0.87	0.90	0.89	1000
Dress	0.92	0.90	0.91	1000
Coat	0.87	0.90	0.88	1000
Sandal	0.99	0.97	0.98	1000
Shirt	0.80	0.74	0.77	1000
Sneaker	0.95	0.98	0.97	1000
Bag	0.98	0.98	0.98	1000
Ankle boot	0.97	0.96	0.97	1000
accuracy			0.92	10000
macro avg	0.92	0.92	0.92	10000
weighted avg	0.92	0.92	0.92	10000

شکل 45- **classification_report** برای معماری **2**

	precision	recall	f1-score	support
T-shirt/top	0.87	0.88	0.88	1000
Trouser	0.99	0.98	0.99	1000
Pullover	0.89	0.88	0.89	1000
Dress	0.93	0.91	0.92	1000
Coat	0.88	0.87	0.87	1000
Sandal	0.99	0.98	0.99	1000
Shirt	0.76	0.78	0.77	1000
Sneaker	0.96	0.99	0.98	1000
Bag	0.99	0.98	0.98	1000
Ankle boot	0.98	0.97	0.97	1000
accuracy			0.92	10000
macro avg	0.92	0.92	0.92	10000
weighted avg	0.92	0.92	0.92	10000

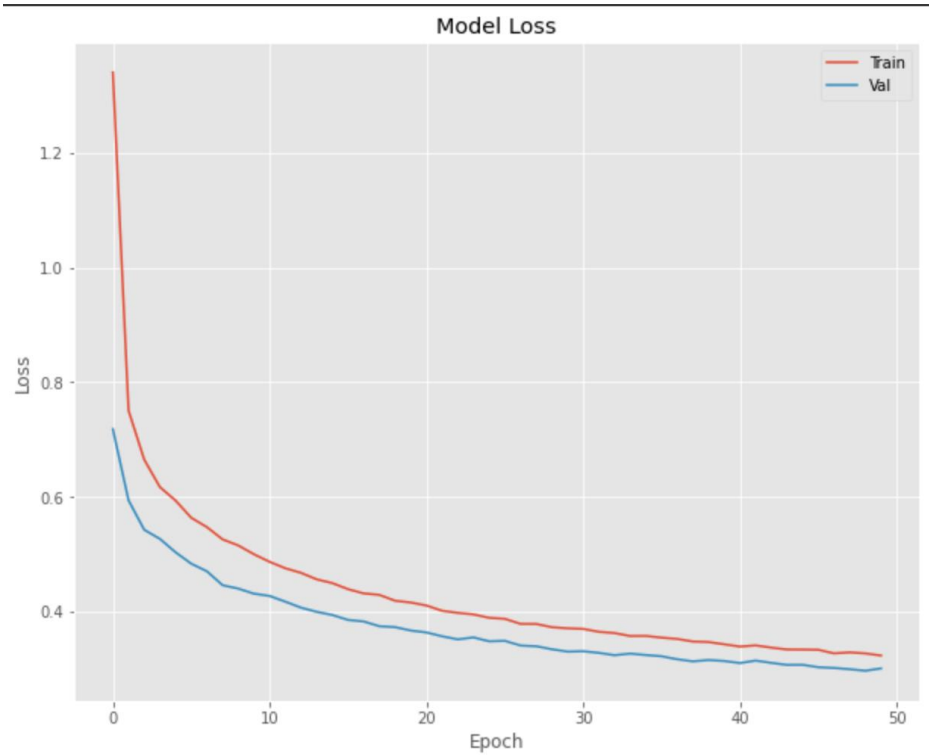
شکل 46- **classification_report** برای معماری **3**

می‌بینیم که نتایج بدست آمده بسیار شبیه به یکدیگر است و تنها مقادیر f1-score و precision در برخی از کلاس‌ها با هم تفاوت دارند. برای مثال اگر به کلاس Shirt در معماری 3 دقت کنیم می‌بینیم که کمترین precision را دارد و در مقایسه با معماری 2 نیز کاهش یافته است. دلیل آن می‌تواند این باشد که ویژگی‌های استخراج شده در معماری 3، پیچیده‌تر بوده‌اند و ممکن است باعث شده باشند که تعدادی از نمونه‌هایی که شبیه این کلاس بوده‌اند مانند T-shirt یا Coat به اشتباه به عنوان Shirt طبقه‌بندی شده باشند. اگر ویژگی‌های استخراج شده را پیچیده‌تر نیز بکنیم احتمالا این مقدار مجددا کاهش یابد. میانگین

مقادیر f1-score نیز در هر دو معماری با یکدیگر برابر است. در مجموع می‌توانیم نتیجه بگیریم که دو معماری بر روی این دیتاست بسیار مشابه یکدیگر عمل می‌کنند با این تفاوت که معماری 3 بر روی دیتای جدید کمی بهتر عمل می‌کند و همچنین با احتمال کمتری ممکن است overfit شود چرا که دقت داده‌های آموزش برای این معماری کمتر از معماری 2 است. این نتیجه با نتایج بدست آمده در مقاله نیز سازگار است.

۲-۵. مقایسه‌ی استفاده از بهینه‌سازهای مختلف

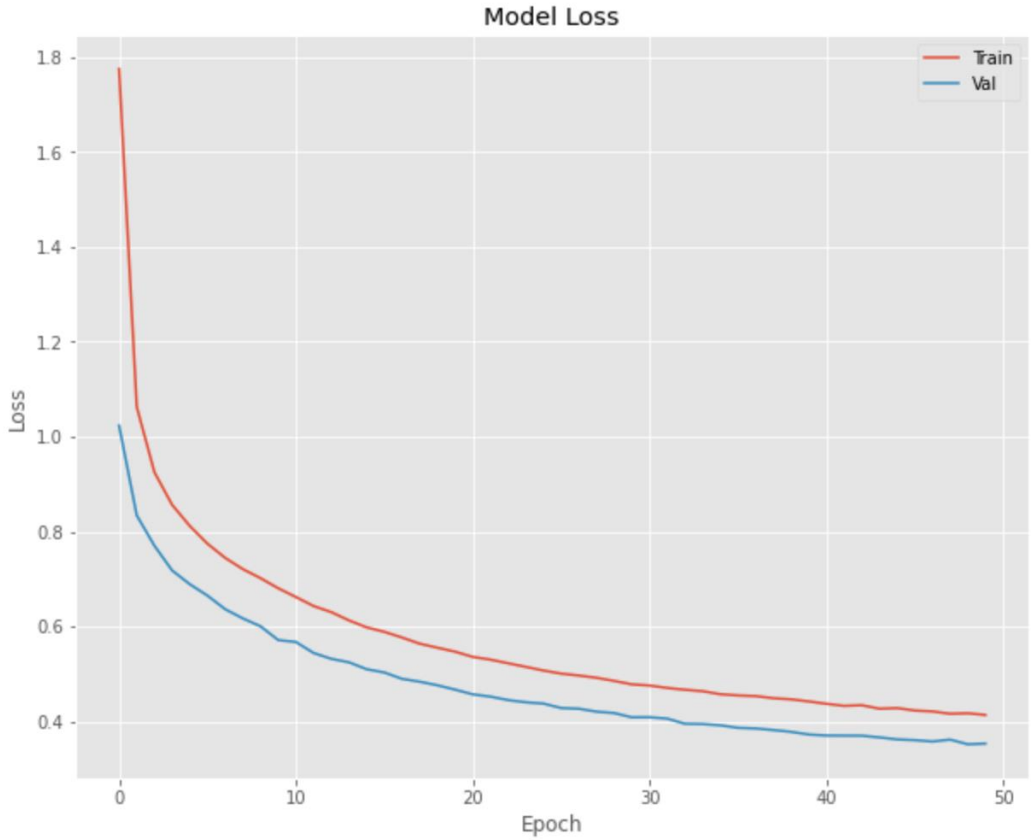
در قسمت قبل دیدیم که بهترین بهینه‌ساز بر اساس نتایج مقاله برای هر دو معماری 2 و 3، adam است. نتایج بدست آمده را نیز مشاهده کردیم. در این قسمت هر دو معماری را با استفاده از SGD نیز آموزش می‌دهیم و نتایج را مقایسه می‌کنیم. لازم به ذکر است که تمام پارامترهای دیگر ثابت هستند و تنها بهینه‌سازها تغییر می‌کنند. نمودار loss برای معماری 2 با بهینه‌ساز SGD در شکل زیر نمایش داده شده است.



شکل 47- نمودار **loss** برای معماری **2** با **SGD**

اگر نتیجه را با نمودار مشابه با بهینه‌ساز adam مقایسه کنیم می‌بینیم که در این حالت نمودرهای loss برای validation و train به یکدیگر نزدیک‌تر هستند که می‌توان نتیجه گرفت مدل آموزش دیده شده generalization مناسبی دارد اما اگر به مقادیر loss در محور y دقت کنیم می‌بینیم که مقدار loss در

حالت adam بعد از 50 epoch کمتر است و مدلی که با بهینه‌ساز adam آموزش دیده است بهتر به دیتای ما فیت شده است. همین نمودار را برای معماری 3 نیز در شکل زیر می‌توانیم مشاهده کنیم.



شکل 48- نمودار **loss** در معماری **3** با **SGD**

این نمودار نیز مانند نمودار قبلی است و اگر چه در انتهای آموزش نمودارهای validation و train به یکدیگر نزدیکتر هستند(در مقایسه با adam) اما مقدار loss در آن حالت کمتر است. دقت مدل‌ها را در حالت SGD برای دو معماری در جدول زیر مشاهده می‌کنیم.

جدول 3- دقت در دو معماری مختلف با **SGD**

مدل	Accuracy	Train	Test
معماری 2		89.86 %	88.73%
معماری 3		87.57%	86.77%

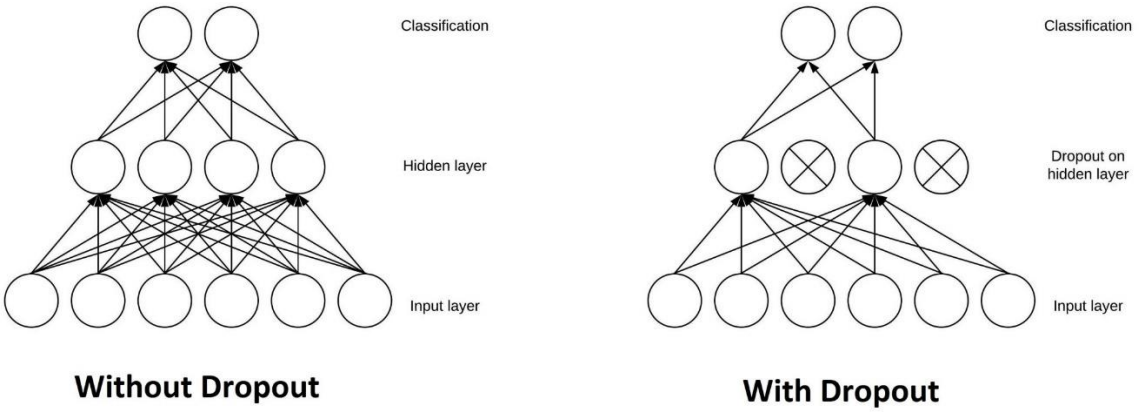
می‌بینیم که دقت مدل‌ها در مقایسه با adam کمتر شده‌اند.

با مقایسه‌ی نمودارهای loss و دقت مدل‌های متوجه می‌شویم که در مجموع adam، loss کمتری بر روی داده‌های train دارد اما این مقدار برای داده‌های validation ممکن است کمتر نباشد. adam سریعتر همگرا

می‌شود و این نتیجه از نمودارهای loss نیز مشخص است. مقادیر loss در نمودارهای adam از مقادیر متناظر در حالت SGD کمتر هستند. برای آن که به مدل بهتری با SGD برسیم باید تعداد epoch بیشتری مدل را آموزش دهیم. همانطور که مشخص است، SGD، generalization بهتری دارد و بر روی داده‌های دیده نشده بهتر عمل می‌کند. علت اینکه دقت‌ها در حالت SGD کمتر هستند این است که مدل را باید بیشتر از 50 epoch آموزش می‌دادیم تا به نتیجه‌ی مطلوب برسیم. به طور کلی مدل آموزش دیده شده با SGD، stable تر است و generalization error کمتری دارد و این از نمودار loss نیز مشخص است.

۲-۶. استفاده از Dropout

Dropout یکی از تکنیک‌های regularization است که احتمال overfit شدن مدل را کاهش می‌دهد و generalizability مدل را بهتر می‌کند. این تکنیک در طول آموزش در هر epoch، درصدی از نورون‌ها را به صورت تصادفی انتخاب می‌کند و خروجی آن‌ها را drop می‌کند و نمی‌گذارد که این نورون‌ها تاثیرگذار باشند.



شکل 49- **Dropout**

این کار باعث می‌شود که نورون‌هایی که خروجی آن‌ها تاثیر نداشته است، تاثیرگذار شوند. این روش مانند این است که چندین شبکه‌ی عصبی را آموزش دهیم و در نهایت از خروجی میانگین بگیریم. اگر در لایه‌ای از شبکه‌ی عصبی اشتباهی صورت بگیرد و لایه‌های بعدی مجبور به جبران آن باشند، dropout این روند را متوقف می‌کند و باعث می‌شود مدل robust شود و دچار overfitting در مراحل اولیه‌ی آموزش نشود.