به نام خدا

# گزارش تمرین ششم

# درس الگوریتم‌های یادگیری ماشین

استاد راهنما: دکتر کمندی


سیده محیا معتمدی

۸۱۰۸۹۷۰۵۳

مینو احمدی

۸۱۰۸۹۷۰۳۲

دانشکده علوم مهندسی

پردیس دانشکده‌های فنی دانشگاه تهران

در این پروژه قصد داریم به کمک لایه های کانولوشنی و دیتاست fashion_mnist مدل classification را پیاده سازی کنیم.

## بخش a)

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

x_train = x_train / 255.0
x_test = x_test / 255.0

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

*Figure ۱- لود کردن داده‌ها*

load_data function [source]

```
tf.keras.datasets.fashion_mnist.load_data()
```

Loads the Fashion-MNIST dataset.

This is a dataset of 60,000 28x28 grayscale images of 10 fashion categories, along with a test set of 10,000 images. This dataset can be used as a drop-in replacement for MNIST.

The classes are:

| Label | Description |
|---|---|
| 0 | T-shirt/top |
| 1 | Trouser |
| 2 | Pullover |
| 3 | Dress |
| 4 | Coat |
| 5 | Sandal |
| 6 | Shirt |
| 7 | Sneaker |
| 8 | Bag |
| 9 | Ankle boot |

**Returns**

- **Tuple of NumPy arrays**: `(x_train, y_train), (x_test, y_test)`.

*Figure ۲- توضیحات مربوط به دیتاست در سایت keras*

با استفاده از دستور load data داده ها را از دیتا ست گرفتیم و برای نرمال کردن داده ها همه داده‌ها را تقسیم بر ۲۵۵ کرده‌ایم ( زیرا رنج تمام داده ها از ۰ تا ۲۵۵ است )

این دیتاست شامل ۷۰ هزار دیتا ۲۸ *۲۸ است که ۶۰ هزار تای آن برای آموزش و ۱۰ هزار تای آن برای تست است که در ده کلاس طبقه بندی شده اند.

در زیر ابعاد تصاویر اموزش و تست را میبینید .

```
print("Train Image shape: " ,x_train.shape)
print("Test Image shape: " ,x_test.shape)
```

```
Train Image shape:  (60000, 28, 28)
Test Image shape:  (10000, 28, 28)
```

*Figure ۳- ابعاد تصاویر داده‌ها آموزش و تست*

ده تصویر از داده‌های آموزش را به عنوان مثال در زیر آورده‌ایم :

```
plt.figure(figsize=(15,15))
for i in range(10):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.imshow(x_train[i], cmap=plt.cm.binary)
    plt.xlabel(classes[y_train[i]])
plt.show()
```

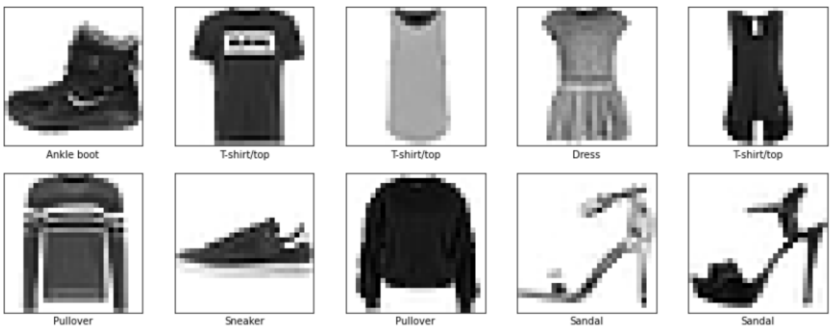


*Figure ۴- تصاویر کلاس‌های مختلف*

## بخش b)

```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Conv2D( filters=32, kernel_size=(3, 3), strides=(1, 1), padding='valid', activation='relu', input_shape=(28, 28, 1)))

model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

model.add(tf.keras.layers.Dropout(rate=0.2))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(units=128,activation='relu'))
model.add( tf.keras.layers.Dense(units=10, activation='softmax'))

model.compile(loss=tf.keras.losses.sparse_categorical_crossentropy, optimizer=tf.keras.optimizers.Adam(), metrics=['accuracy'])

model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_1 (Conv2D)           (None, 26, 26, 32)        320

 max_pooling2d_1 (MaxPooling  (None, 13, 13, 32)       0
 2D)

 dropout_1 (Dropout)         (None, 13, 13, 32)        0

 flatten_1 (Flatten)         (None, 5408)              0

 dense_2 (Dense)             (None, 128)               692352

 dense_3 (Dense)             (None, 10)                1290

=================================================================
Total params: 693,962
Trainable params: 693,962
Non-trainable params: 0
_________________________________________________________________
```

*Figure ۵-معماری مدل*

مدل را با استفاده از دستور Sequential طراحی کرده ایم در ابتدا یک لایه کانولوشن ۲ بعدی با کرنل سایز ۳*۳ و ابعاد input ۲۸*۲۸ و تابع فعالساز relu و تعداد ۳۲ تا فیلتر اضافه کردیم .

در لایه های بعدی از لایه های maxpooling استفاده کرده ایم که یکی از لایه های اصلی در طراحی شبکه کانولوشن است در این لایه بیشترین مقدار داده‌ای که در لایه قبلی توسط کرنل به ازای هر فیلتر به دست امده است، انتخاب میشود.

در مرحله بعدی یک لایه dropout با مقدار ۰.۲ اضافه کرده ایم تا از overfitting جلوگیری شود و شبکه بتواند تعمیم بهتری برای داده تست داشته باشد .

در انتها دو لایه dense (fully connected)اضافه کرده ایم که ابعاد خروجی آن‌ها به ترتیب ۱۲۸ و ۱۰ است در لایه اخر fully connected از اکتیویشن فانکشنsoft-max استفاده کرده‌ایم و تعداد نورون ها لایه آخر برابر ده است چون ده کلاس برای داده ها وجود دارد.

هم چنین از تابع خطا sparse_categorical_crossentropy و بهینه ساز adam استفاده کرده ایم.

کد های مربوط به این بخش و شمای کلی از مدل را میتوانید در بالا مشاهده کنید.

در نهایت مدل را با batch_size=۲۵۶, epochs=۲۰, validation_split=۰.۱ و callback در حالت early stopping اموزش داده ایم .

```python
from gc import callbacks
x_train = np.expand_dims(x_train, -1)
x_test = np.expand_dims(x_test, -1)
callback = tf.keras.callbacks.EarlyStopping(
    monitor="val_loss",
    min_delta=0,
    patience=3,
    verbose=0,
    mode="auto",
    baseline=None,
    restore_best_weights=False,
)
history = model.fit(x_train, y_train, batch_size=256, epochs=20, validation_split=0.1, verbose=1, callbacks = [callback])
```

*Figure ۶-callback و تابع fit*

نتایج epoch های اخر به شرح زیر است :

```
Epoch 12/20
211/211 [==============================] - 26s 123ms/step - loss: 0.1691 - accuracy: 0.9383 - val_loss: 0.2378 - val_accuracy: 0.9137
Epoch 13/20
211/211 [==============================] - 26s 123ms/step - loss: 0.1597 - accuracy: 0.9416 - val_loss: 0.2396 - val_accuracy: 0.9168
Epoch 14/20
211/211 [==============================] - 27s 128ms/step - loss: 0.1530 - accuracy: 0.9440 - val_loss: 0.2374 - val_accuracy: 0.9160
```

*Figure ۷-نتایج ایپاک‌های آخر*

در نمودار های زیر میتوان نمودار پیشرفت دقت و خطا را مشاهده کنید.

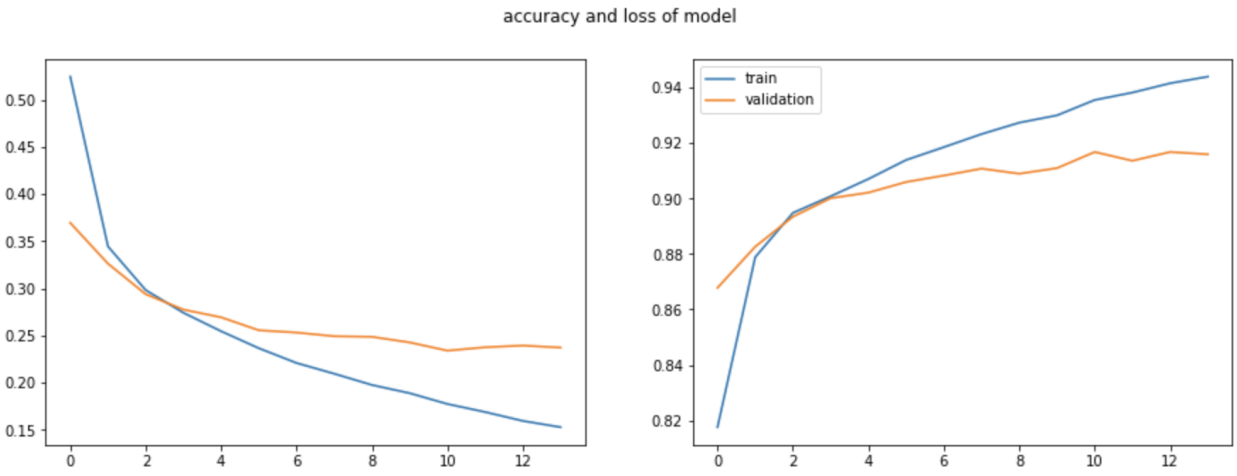


*Figure ۸- نتایج خطا و دقت*

با استفاده است دستور model.predict داده های تست را به مدل اعمال کردیم و خروجی را بدست اوردیم برای اینکه کلاس انتخاب شده توسط مدل را پیدا کنیم از دستور np.argmax کلاس با بیشترین احتمال را میگیریم و در نهایت با استفاده از

دستور classification report محاسبه کرده‌ایم که در هر کلاس، در چند درصد از مواقع مدل عملکرد خوبی از خود نشان میدهد .

داده های مربوط به این بخش در زیر امده است.

```
predicted = model.predict(x_test)
predicted = list(np.argmax(predicted, axis=1))
print(classification_report(y_test, predicted, target_names=classes))
```

```
              precision    recall  f1-score   support

 T-shirt/top       0.87      0.87      0.87      1000
     Trouser       0.99      0.98      0.99      1000
    Pullover       0.88      0.87      0.88      1000
       Dress       0.89      0.96      0.92      1000
        Coat       0.89      0.85      0.87      1000
      Sandal       0.98      0.98      0.98      1000
       Shirt       0.76      0.76      0.76      1000
     Sneaker       0.95      0.98      0.96      1000
         Bag       0.98      0.98      0.98      1000
  Ankle boot       0.98      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
```

*Figure ۹- نتایج به دست آمده از داده‌های تست به تفکیک کلاس*

## بخش c)

همان طور که در بخش قبل ذکر شده در طراحی این مدل از dropout=۰.۲(تا از overfitting جلوگیری شود و شبکه بتواند تعمیم بهتری با داده تست داشته باشد)

```
model.add(tf.keras.layers.Dropout(rate=0.2))
```

*Figure ۱۰- لایه Dropout*

و call back حالت early stopping استفاده شده است که در این بخش داده validation loss را مانیتور میکنیم و مقدار patienceرا هم برابر ۳ قرار میدهیم در نتیجه هر زمان که مقدارvalidation loss در سه epochپشت سر هم کاهش پیدا نکند مدل اموزش را متوقف میکند میتوانیم در قسمت قبل مشاهده کنیم که با اینکه بیست epoch برای اموزش در نظر گرفته بودیم به دلیل قرار دادن callbackمدل تنها چهارده epoch را طی کرده است .

```
callback = tf.keras.callbacks.EarlyStopping(
    monitor="val_loss",
    min_delta=0,
    patience=3,
    verbose=0,
    mode="auto",
    baseline=None,
    restore_best_weights=False,
)
```

*callback - ۱۱ Figure*