عنوان : تمرین اول یادگیری عمیق

نگارنده : سحر داستانی اوغانی

شماره دانشجویی : ۹۹۱۱۲۱۰۸

دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

دانشکده ریاضی و علوم کامپیوتر

بخش اول - دیتاست

معرفی دیتاست

دیتاست Fashion MNIST، متشکل از ۶۰ هزار تصویر grayscale است که هر کدام حاوی ۲۸ × ۲۸ پیکسل از ۱۰ نوع لباس مختلف است. هرنوع از این البسه با یکی از شماره‌های ۰ تا ۹ نمایش داده شده است:

شماره	۰	۱	۲	۳	۴	۵	۶	۷	۸	۹
نوع	تاپ/تیشرت	شلوار	پلیور	پیراهن زنانه	کت	صندل	پیراهن مردانه	کتانی	کیف	بوت کوتاه

تصاویر در این دیتاست، سه ویژگی زیر را دارند:

- presegmented هستند. یعنی هر تصویر تنها حاوی یک آیتم است.
- تمامی تصاویر، مربع‌هایی ۲۸×۲۸ هستند.
- تصاویر grayscale هستند.

فراخوانی

برای فراخوانی این دیتاست، می‌توان از کتابخانه‌ی Fashion_mnist در دیتاست keras استفاده کرد و آن را در دو گروه تست و ترین جا داد.

Flattens

ابتدا دیتاست را reshape می‌کنیم تا یک کانال رنگ داشته باشیم.

هر تصویر در دیتاست، عددهای صحیح بدون علامتی در بازه‌ی سیاه و سفید یا ۰ تا ۲۵۵ است. بنابراین باید داده‌ها را نرمال‌سازی کرد. یعنی بازه‌ی پیکسل تصاویر grayscale را تا [0,1] پایین آورد. برای این‌کار، ابتدا باید داده‌های integer را به float تبدیل کرد و سپس آن‌ها را بر ۲۵۵ که ماکسیمم مقدار است، تقسیم کرد.

One hot encoding

تابعی به نام to_categorical از کتابخانه‌ی keras.utils وجود دارد که عدد صحیح هر کلاس را به یک عدد باینری تبدیل می‌کند.

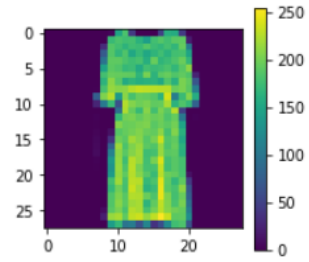
برای موارد بالا، تابعی به نام load_dataset() نوشته شده است. خروجی این تابع به صورت زیر است:

```
====================load_dataset=======================
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
32768/29515 [================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26427392/26421880 [==============================] - 1s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
8192/5148 [===============================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4423680/4422102 [==============================] - 0s 0us/step
--------------------Dataset is loaded
--------------------Dataset is reshaped
--------------------Dataset is converted to float
--------------------Dataset is normalized
--------------------Dataset is onhot-encoded
====================load_dataset is finished============
```

برای درک بهتر ماهیت دیتاست، مواردی از آن را می‌توانید در ادامه در قالب یک تصویر مشاهده کنید:

این تصویر متعلق به داده‌ی سوم است که تصویر یک پیراهن را نمایش می‌دهد.

```
====================show_picture=======================
Colorbar sample of dataset:
```




```
Image (#1902): Which is label number '3', or label 'Dress''
```

تصاویر روبرو، مجموعه‌ای را نمایش می‌دهد که دیتاست در خود جای داده است:


```
====================show_picture is finished============
```

بخش دوم – مدل

مدلی که طراحی می‌کنیم باید دارای دو جنبه‌ی اصلی باشد:

- feature extraction: که خود دارای دو لایه‌ی convolutional و pooling است.

- classifier: که عمل پیش‌بینی را انجام می‌دهد.

علاوه بر این، طبق تعریف صورت مسئله، مدل باید دارای چهار لایه باشد:

- لایه کانولوشن:

در لایه‌ی convolutional از یک feature map که دارای 32 نورون است و یک feature detector با اندازه‌ی 3×3 برای ساخت feture map، استفاده می‌کنیم. این feature detector، اندازه‌ی کرنل را مشخص می‌کند.

این لایه از activation ‘linear’ به جای ReLU استفاده می‌کند. دلیل آن در ادامه توضیح داده شده است:

تابع ReLU عملکردی دارد که طی آن آستانه را از صفر شروع می‌کند. ممکن است در طی آموزش، واحدهای ReLU از بین بروند و به اصطلاح بمیرند. این حادثه زمانی اتفاق می‌افتد که یک گرادیان بزرگ از یک نورون ReLU عبور کند. این عبور کردن باعث می‌شود وزن‌ها به نحوی update شوند که نورون هرگز بر روی هیچ داده‌ی دیگری فعال نشود. اگر این اتفاق بیفتد، گرادیان عبوری از واحد پردازشی برای همیشه صفر باقی می‌ماند. برای جلوگیری از این اتفاق از یک لایه‌ی Leaky ReLUs استفاده می‌کنیم. این لایه باعث می‌شود تابع هرگز صفر نشود ولی دارای شیب منفی باشد.

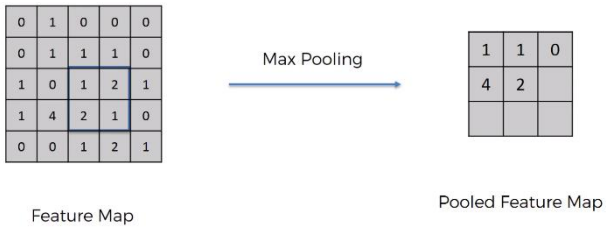
منبع نکات ذکر شده در رابطه با لایه‌ی Leaky ReLUs و متون انگلیسی آن در ادامه آمده است:

https://www.datacamp.com/community/tutorials/convolutional-neural-networks-python

> More specifically, you add Leaky ReLUs because they attempt to fix the problem of dying Rectified Linear Units (ReLUs). The ReLU activation function is used a lot in neural network architectures and more specifically in convolutional networks, where it has proven to be more effective than the widely used logistic sigmoid function. As of 2017, this activation function is the most popular one for deep neural networks. The ReLU function allows the activation to be thresholded at zero. However, during the training, ReLU units can "die". This can happen when a large gradient flows through a ReLU neuron: it can cause the weights to update in such a way that the neuron will never activate on any data point again. If this happens, then the gradient flowing through the unit will forever be zero from that point on. Leaky ReLUs attempt to solve this: the function will not be zero but will instead have a small negative slope.

لایه‌ی pooling:

pooled feture map را با حرکت دادن یک max pooling با اندازه‌ی 2×2 بر روی feture map، بدست می‌آوریم.



- لایه Flatten:

این لایه برای استخراج ویژگی صورت می‌گیرد.

- لایه‌های dense:

این لایه‌ها برای تفسیر ویژگی‌های استخراج شده به مدل اضافه می‌شوند.

برای loss function از تابع categorical cross-entropy استفاده کردیم زیرا این تابع برای مسائل چند کلاسه مانند مسئله‌ی ما، خوب عمل می‌کند.

خروجی یک مساله‌ی طبقه‌بندی چند کلاسه، یک لایه با ۱۰ نود است. ولی دراینجا خروجی، احتمال توزیع یک عکس به هر یک از ۱۰ کلاس است. برای تغییر آن می‌توان از یک تابع softmax activation در لایه dense بین feature extraction و لایه‌ی خروجی استفاده کرد. که در اینجا این لایه ۱۰۰ نود دارد.

مدل طراحی شده به شرح زیر است:

```python
def baseline_cnn_model():
    model = Sequential()

    model.add(Conv2D(32, kernel_size=(3, 3),activation='linear',padding='same',input_shape=(28,28,1)))
    model.add(LeakyReLU(alpha=0.1))
    model.add(MaxPooling2D((2, 2),padding='same'))

    model.add(Flatten())

    model.add(Dense(128, activation='linear'))
    model.add(LeakyReLU(alpha=0.1))
    model.add(Dense(10, activation='softmax'))

    # compile model
    # opt = SGD(lr=0.01, momentum=0.9)
    # model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
    # return model

    model.compile(optimizer=keras.optimizers.Adam(), loss=keras.losses.categorical_crossentropy, metrics=['accuracy'])
    return model
```

کامپایل مدل به دو شیوه انجام پذیر است.

بخش سوم - ارزیابی

برای این که هر بار ۸۰ درصد داده‌ها را به داده‌ی آموزشی و ۲۰ درصد دیگر را به داده‌ی validation اختصاص دهیم کافی است تعداد فولدهای k-fold CV را ۵ در نظر بگیریم. در این صورت دیتاست به ۵ قسمت تقسیم می‌شود و هربار یک قسمت یعنی ۲۰ درصد را کنار می‌گذارد و از ۴ قسمت دیگر یعنی ۸۰ درصد باقی مانده استفاده می‌کند.

سپس مدل ساخته شده در مرحله‌ی قبل را بر روی داده‌ها فیت می‌کنیم. این عمل را در ۱۰ epoch انجام می‌دهیم.

اطلاعات فیت و دقت هربار اجرای این الگوریتم بر روی دیتاست را در متغیری با نام‌های histories و acc ذخیره می‌کنیم. تا در ادامه بتوانیم نمودارهای loss و accuracy را بر رو آن‌ها رسم کنیم.

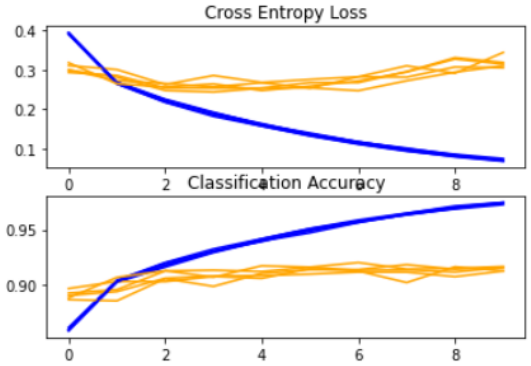
تغییرات برای مدل ذکر شده در مرحله‌ی قبل، به شرح زیر است:

```
====================evaluate_model============================
====================baseline_cnn_model=======================
> 91.258
> 91.667
> 91.275
> 91.583
> 91.583
====================baseline_cnn_model is finished===========
====================evaluate_model is finished===============
```

نمودارهای loss و accuracy:

این نمودار مشخص می‌کند، در پایان fold پنجم، روند نتایج بهبود پیدا کردند. زیرا خطای داده‌های آموزش کاهش و دقت آن در طبقه‌بندی، افزایش پیدا کرده است.

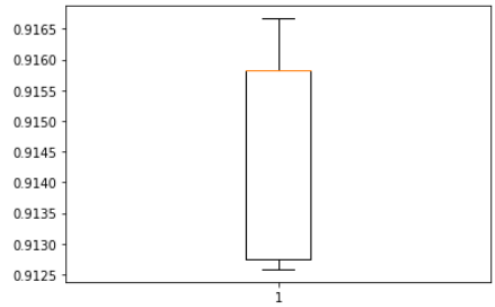
```
====================summarize_diagnostics====================
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6: I

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12:
  if sys.path[0] == '':
```




```
====================summarize_diagnostics is finished========
```

در ادامه نیز شاهد عملکرد مدل بر اساس یک boxplot هستید :همانطور که مشخص است، میانگین مدل مقدار ۹۱,۴۷۳ و انحراف از معیار آن مقدار ۰,۱۷۲ را به خود اختصاص داده است.

```
====================summarize_performance====================
Accuracy: mean=91.473 std=0.172, n=5
```




```
====================summarize_performance is finished====================
```

برای جلوگیری از تکرار هر بار مدل در ۱۰ epoch، مدل را در یک فایل final_model.h5 سیو می‌کنیم و هربار برای استفاده از آن این فایل را load می‌کنیم.

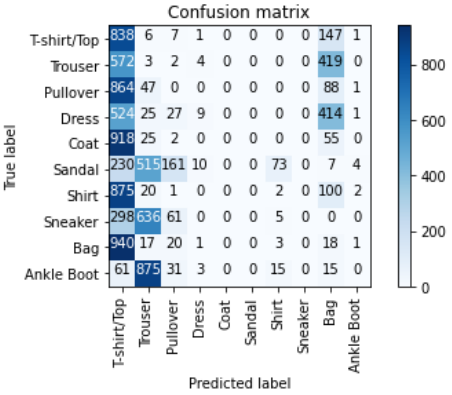
دقت مدل بر روی داده‌های تست به شرح زیر گزارش می‌شود:

```
evaluate model on test dataset:

> 91.140
```

Confusion matrix و Classification Report مدل در ادامه آمده است:

	precision	recall	f1-score	support
T-shirt/Top	0.20	0.00	0.00	1000
Trouser	0.05	0.15	0.07	1000
Pullover	0.00	0.00	0.00	1000
Dress	0.00	0.00	0.00	1000
Coat	0.00	0.00	0.00	1000
Sandal	0.00	0.00	0.00	1000
Shirt	0.15	0.26	0.19	1000
Sneaker	0.00	0.00	0.00	1000
Bag	0.00	0.00	0.00	1000
Ankle Boot	0.14	0.65	0.23	1000
accuracy			0.11	10000
macro avg	0.05	0.11	0.05	10000
weighted avg	0.05	0.11	0.05	10000

```
====Confusion Matrix and Classification Report is finished===
```



دقت در گزارش طبقه‌بندی، ۰,۱۱ گزارش شده است.

بخش چهارم – performance

بررسی افزایش عمق

برای افزایش عمق، می‌توان لایه‌های مختلفی را به مدل اضافه کرد. نکته‌ای که وجود دارد در این است که آیا با افزایش عمق و لایه، باید تعداد فیلترها را نیز افزایش داد؟ آیا انجام همزمان این دو مورد نتیجه‌ی بهتری را بوجود میاورد یا انجام تک تک آن‌ها به صورت جداگانه؟

برای دریافت پاسخ نهایی، پاسخ هر یک را در ادامه پیاده‌سازی می‌کنیم.

۱) ابتدا تنها لایه‌ها را افزایش می‌دهیم. یعنی تعداد فیلترها را عددی ثابت در نظر می‌گیریم و مقدار تمامی آن‌ها را ۳۲ می‌گذاریم:

```python
def baseline_cnn_model():
    model = Sequential()

    model.add(Conv2D(32, kernel_size=(3, 3),activation='linear',input_shape=(28,28,1),padding='same'))
    model.add(LeakyReLU(alpha=0.1))
    model.add(MaxPooling2D((2, 2),padding='same'))

    model.add(Conv2D(32, (3, 3), activation='linear',padding='same'))
    model.add(LeakyReLU(alpha=0.1))
    model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))

    model.add(Conv2D(32, (3, 3), activation='linear',padding='same'))
    model.add(LeakyReLU(alpha=0.1))
    model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))

    model.add(Flatten())

    model.add(Dense(100, activation='linear'))
    model.add(LeakyReLU(alpha=0.1))
    model.add(Dense(10, activation='softmax'))

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

    return model
```

```
====================evaluate_model============================
====================baseline_cnn_model========================
> 88.432
> 88.511
> 88.831
> 89.005
> 89.376
====================baseline_cnn_model is finished===========
====================evaluate_model is finished===============
```

همانطور که مشاهده می‌کنید، دقت از حدود ۹۱ به ۸۸ کاهش یافته است. پس درواقع افزایش تنهایی لایه‌ها برای بهتر کردن عملکرد مدل، کافی نیست.

۲) حال لایه‌ها را به همراه تعداد فیلترها افزایش می‌دهیم.

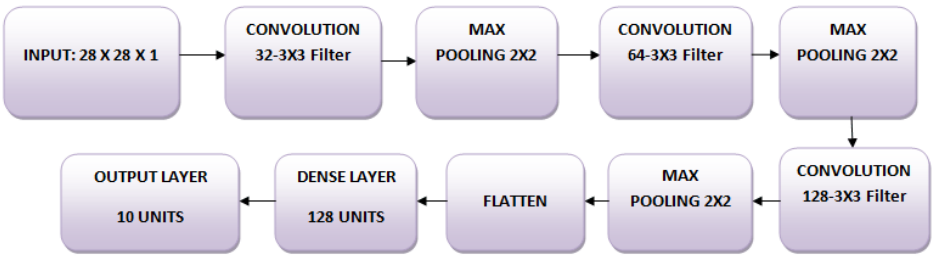
```
def baseline_cnn_model():
    model = Sequential()

    model.add(Conv2D(32, kernel_size=(3, 3),activation='linear',input_shape=(28,28,1),padding='same'))
    model.add(LeakyReLU(alpha=0.1))
    model.add(MaxPooling2D((2, 2),padding='same'))

    model.add(Conv2D(64, (3, 3), activation='linear',padding='same'))
    model.add(LeakyReLU(alpha=0.1))
    model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))

    model.add(Conv2D(128, (3, 3), activation='linear',padding='same'))
    model.add(LeakyReLU(alpha=0.1))
    model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))

    model.add(Flatten())

    model.add(Dense(128, activation='linear'))
    model.add(LeakyReLU(alpha=0.1))
    model.add(Dense(10, activation='softmax'))

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

    return model
```

```
====================evaluate_model==========================
====================baseline_cnn_model======================
> 92.183
> 91.858
> 91.883
> 92.050
> 92.425
====================baseline_cnn_model is finished===========
====================evaluate_model is finished===============
```

نتیجه نشان می‌دهد که دقت از ۹۱ به ۹۲ افزایش یافته است. درواقع زمانی افزایش لایه‌ها، عملکرد مدل را بهتر می‌کند که تعداد فیلترها را نیز برای شناسایی ویژگی‌ها افزایش دهیم. موضوع ذکر شده در مقاله‌ی زیر مورد بررسی واقع شده است. نتیجه‌ی آن بیان می‌کند که استخراج بیشتر feature نیز باعث بهتر شدن عملکرد مدل می‌گردد. این مقاله را می‌توانید از اینجا دانلود کنید.

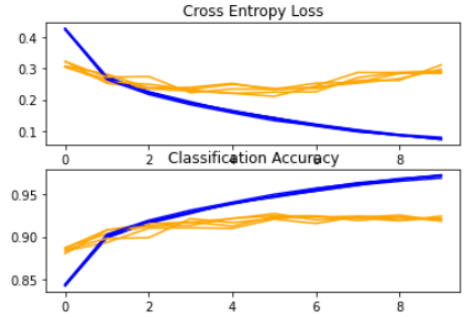
Snoek, J., Larochelle, H., & Adams, R. P. (2012). Practical Bayesian Optimization of Machine Learning Algorithms. In NIPS (pp. 2951–2959).

روند افزایش لایه در این مرحله به شرح زیر است:



بهتر شدن عملکرد مدل، نه تنها در نتایج accuracy قابل مشاهده است، بلکه در نمودار loss و accuracy نیز مشخص است:

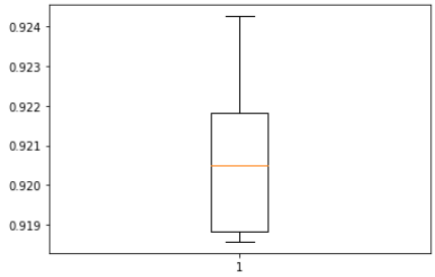
```
====================summarize_diagnostics====================
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6:

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:12:
  if sys.path[0] == '':
```




```
====================summarize_diagnostics is finished========
```

همچنین می‌توانید تغییرات box plot را نیز در ادامه مشاهده کنید:

```
====================summarize_performance====================
Accuracy: mean=92.080 std=0.209, n=5
```



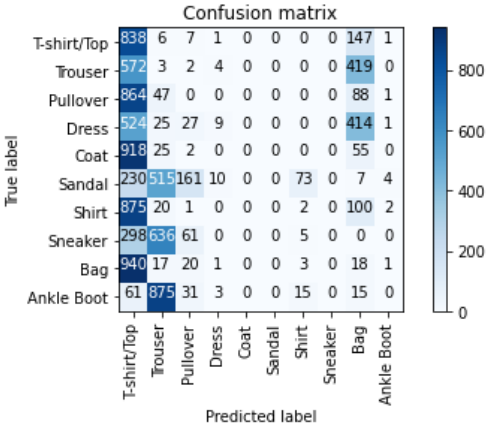
```
====================summarize_performance is finished====================
```

پس از ذخیره‌ی مدل و پیاده‌سازی آن بر روی داده‌های تست، نتیجه به شرح زیر گزارش می‌شود. این نتیجه در مقابل نتیجه‌ی مدل تک لایه با افزایش روبرو است. زیرا دقت مدل تک لایه بر روی داده‌های تست ۹۱,۱۴۰ بود.

```
evaluate model on test dataset:

> 91.320
```

Confusion matrix و Classification Report مدل در ادامه آمده است:

```
              precision    recall  f1-score   support

 T-shirt/Top       0.14      0.84      0.24      1000
     Trouser       0.00      0.00      0.00      1000
    Pullover       0.00      0.00      0.00      1000
       Dress       0.32      0.01      0.02      1000
        Coat       0.00      0.00      0.00      1000
      Sandal       0.00      0.00      0.00      1000
       Shirt       0.02      0.00      0.00      1000
     Sneaker       0.00      0.00      0.00      1000
         Bag       0.01      0.02      0.02      1000
  Ankle Boot       0.00      0.00      0.00      1000

    accuracy                           0.09     10000
   macro avg       0.05      0.09      0.03     10000
weighted avg       0.05      0.09      0.03     10000

====Confusion Matrix and Classification Report is finished===
```



بررسی افزودن dropout

معمولا لایه‌ی dropout زمانی به مدل اضافه می‌شود که بخواهیم با مشکل overfitting مقابله کنیم. عملکرد آن برای مقابله با این مشکل، بدین صورت است که در ابتدا، کسری از نورون‌ها را به صورت رندوم خاموش می‌کند و مانع از اجرای آن‌ها در زمان آموزش می‌شود. با این‌کار، وابستگی به داده‌های آموزش را کم می‌کند و مانع می‌شود که شبکه برخی از داده‌های آموزشی را حفظ کند. تعداد و مقدار کسر معرفی شده توسط یک hyperparameter مشخص می‌شود که می‌تواند مطابق با آن تنظیم یا tune شود.

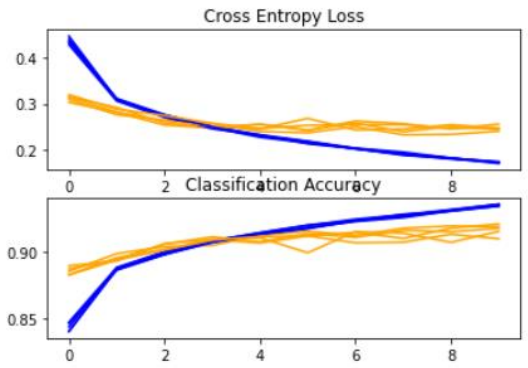
ابتدا dropout را به مدلی اضافه می‌کنیم که دارا یک لایه‌ی کانولوشن و یک لایه‌ی پولینگ است.

```
def baseline_cnn_model():
    model = Sequential()

    model.add(Conv2D(32, kernel_size=(3, 3),activation='linear',padding='same',input_shape=(28,28,1)))
    model.add(LeakyReLU(alpha=0.1))
    model.add(MaxPooling2D((2, 2),padding='same'))
    model.add(Dropout(0.25))

    model.add(Flatten())

    model.add(Dense(128, activation='linear'))
    model.add(LeakyReLU(alpha=0.1))
    model.add(Dropout(0.3))
    model.add(Dense(10, activation='softmax'))


    model.compile(optimizer=keras.optimizers.Adam(), loss=keras.losses.categorical_crossentropy, metrics=['accuracy'])
    return model
```

```
====================evaluate_model==========================
====================baseline_cnn_model======================
> 91.025
> 91.933
> 91.817
> 91.583
> 92.142
====================baseline_cnn_model is finished===========
====================evaluate_model is finished===============
```

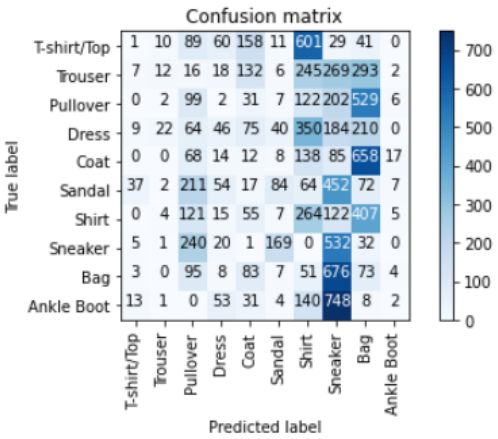
نمودار loss و accuracy و دقت آن بر روی داده‌های تست نیز به شرح زیر است:

```
evaluate model on test dataset:

> 91.140
```



در ادامه نیز می‌توانید شاهد نتایج ماتریس آشفتگی و گزارش طبقه بندی باشید:

	precision	recall	f1-score	support
T-shirt/Top	0.01	0.00	0.00	1000
Trouser	0.22	0.01	0.02	1000
Pullover	0.10	0.10	0.10	1000
Dress	0.16	0.05	0.07	1000
Coat	0.02	0.01	0.02	1000
Sandal	0.24	0.08	0.13	1000
Shirt	0.13	0.26	0.18	1000
Sneaker	0.16	0.53	0.25	1000
Bag	0.03	0.07	0.04	1000
Ankle Boot	0.05	0.00	0.00	1000
accuracy			0.11	10000
macro avg	0.11	0.11	0.08	10000
weighted avg	0.11	0.11	0.08	10000



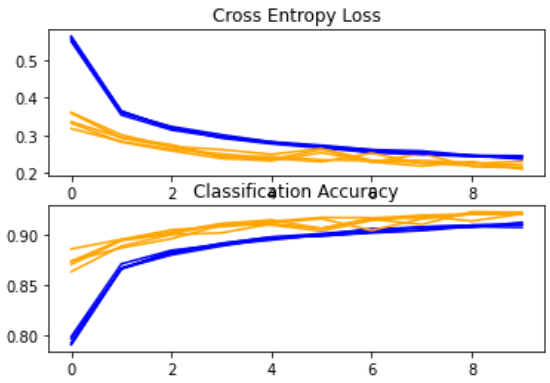
سپس dropout را زمانی اضافه می‌کنیم، که مدل دارای ۳ لایه کانولوشن و پولینگ باشد.

```python
def baseline_cnn_model():
    model = Sequential()

    model.add(Conv2D(32, kernel_size=(3, 3),activation='linear',padding='same',input_shape=(28,28,1)))
    model.add(LeakyReLU(alpha=0.1))
    model.add(MaxPooling2D((2, 2),padding='same'))
    model.add(Dropout(0.25))

    model.add(Conv2D(64, (3, 3), activation='linear',padding='same'))
    model.add(LeakyReLU(alpha=0.1))
    model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
    model.add(Dropout(0.25))

    model.add(Conv2D(128, (3, 3), activation='linear',padding='same'))
    model.add(LeakyReLU(alpha=0.1))
    model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
    model.add(Dropout(0.4))

    model.add(Flatten())

    model.add(Dense(128, activation='linear'))
    model.add(LeakyReLU(alpha=0.1))
    model.add(Dropout(0.3))
    model.add(Dense(10, activation='softmax'))


    model.compile(optimizer=keras.optimizers.Adam(), loss=keras.losses.categorical_crossentropy, metrics=['accuracy'])
    return model
```

```
====================evaluate_model==========================
====================baseline_cnn_model======================
> 92.017
> 92.175
> 92.008
> 92.050
> 92.100
====================baseline_cnn_model is finished==========
====================evaluate_model is finished==============
```

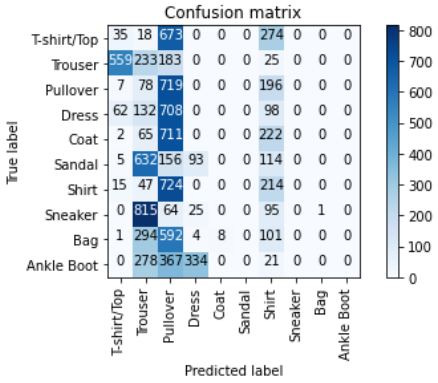
نمودار loss و accuracy و دقت آن بر روی داده‌های تست نیز به شرح زیر است:

```
evaluate model on test dataset:

> 91.450
```



در ادامه نیز می‌توانید شاهد نتایج ماتریس آشفتگی و گزارش طبقه بندی باشید:

	precision	recall	f1-score	support
T-shirt/Top	0.05	0.04	0.04	1000
Trouser	0.09	0.23	0.13	1000
Pullover	0.15	0.72	0.24	1000
Dress	0.00	0.00	0.00	1000
Coat	0.00	0.00	0.00	1000
Sandal	0.00	0.00	0.00	1000
Shirt	0.16	0.21	0.18	1000
Sneaker	0.00	0.00	0.00	1000
Bag	0.00	0.00	0.00	1000
Ankle Boot	0.00	0.00	0.00	1000
accuracy			0.12	10000
macro avg	0.04	0.12	0.06	10000
weighted avg	0.04	0.12	0.06	10000



همانطور که مشخص است، عملکرد مدل زمانی بهتر می‌شود که dropout را به مدلی اضافه کنیم که دارای لایه‌های بیشتر و فیلترهای بیشتری باشد. این بهبود هم در دقت بر روی داده‌های تست مشخص است و هم بر روی داده‌های آموزش.

بخش پنجم – Hyperparameter Tuning

یک مدل در یادگیری ماشین، دارای دو پارامتر است:

۱) trainable parameters: این پارامترها توسط الگوریتم در زمان آموزش، یاد گرفته می‌شوند. برای مثال، می‌توان وزن‌های یک شبکه عصبی را پارامترهای قابل آموزش نامید.

۲) hyperparameters: این نوع پارامترها باید قبل از اجرای پروسه، مشخص گردند. برای مثال، نرخ یادگیری یا تعداد واحدهای لایه‌ی dense، هایپرپارامتر هستند.

هایپرپارامترهایی که می‌توان در مدل ساخته شده تنظیم کرد، به شرح زیر است:

- نرخ dropout
- تعداد فیلترهای لایه‌های کانولوشن
- تعداد واحدهای پردازشی لایه‌ی dense
- تابع فعالیت

تابعی برای تعریف کرنل، سایز آن و معرفی لایه‌های به شرح زیر می‌سازیم:

```
def build_model(hp):
    model = keras.Sequential([
        keras.layers.Conv2D(
            filters=hp.Int('conv_1_filter', min_value = 32, max_value = 128, step = 16),
            kernel_size = hp.Choice('conv_1_kernel', values = [3,5]),
            activation = 'relu',
            input_shape = (28, 28, 1)
        ),

        keras.layers.Conv2D(
            filters=hp.Int('conv_2_filter', min_value = 32, max_value = 128, step = 16),
            kernel_size = hp.Choice('conv_2_kernel', values = [3,5]),
            activation = 'relu'
        ),

        keras.layers.Flatten(),
        keras.layers.Dense(
            units=hp.Int('dense_1_units', min_value = 32, max_value = 128, step = 16),
            activation = 'relu'
        ),

        keras.layers.Dense(10, activation = 'softmax')
    ])

    model.compile(optimizer=keras.optimizers.Adam(hp.Choice('learning_rate', values=[1e-2, 1e-3])),
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

    return model
```

سپس با استفاده از ماژول RandomSearch از کتابخانه‌ی kerastuner، بهترین hyperparameter رندوم را پیدا می‌کنیم.

برای این‌کار تعداد epochها را ۳ و تعداد trialها را ۱۰ در نظر می‌گیریم.(در ابتدا تعداد epoch را نیز ۱۰ در نظر گرفتم، منتها پس از ۸ ساعت استفاده از colab GPU، با خطای محدودیت در GPU مواجه شدم.)

نتایج این ۱۰ trial به شرح زیر گزارش شد:

:۱

```
#Random search to find best hyperparameter
tuner_search = RandomSearch(build_model,
                            objective = 'val_accuracy',
                            max_trials = 10, directory = 'output', project_name = 'Mnist Fashion')

tuner_search.search(train_images, train_labels, epochs=3,validation_split=0.1)
```

```
Search: Running Trial #1

Hyperparameter    |Value             |Best Value So Far
conv_1_filter     |96                |?
conv_1_kernel     |3                 |?
conv_2_filter     |64                |?
conv_2_kernel     |5                 |?
dense_1_units     |48                |?
learning_rate     |0.001             |?

Epoch 1/3
1688/1688 [==============================] - 41s 5ms/step - loss: 0.5436 - accuracy: 0.8046 - val_loss: 0.3235 - val_accuracy: 0.8803
Epoch 2/3
1688/1688 [==============================] - 8s 5ms/step - loss: 0.2638 - accuracy: 0.9018 - val_loss: 0.2656 - val_accuracy: 0.9037
Epoch 3/3
1688/1688 [==============================] - 8s 5ms/step - loss: 0.1995 - accuracy: 0.9261 - val_loss: 0.2555 - val_accuracy: 0.9060
```

:۲

```
Trial 2 Complete [00h 00m 23s]
val_accuracy: 0.8741666674613953

Best val_accuracy So Far: 0.906000018119812
Total elapsed time: 00h 01m 22s

Search: Running Trial #3

Hyperparameter    |Value             |Best Value So Far
conv_1_filter     |48                |96
conv_1_kernel     |3                 |3
conv_2_filter     |128               |64
conv_2_kernel     |3                 |5
dense_1_units     |80                |48
learning_rate     |0.001             |0.001
```

:۳

```
Trial 3 Complete [00h 00m 32s]
val_accuracy: 0.9156666398048401

Best val_accuracy So Far: 0.9156666398048401
Total elapsed time: 00h 01m 55s

Search: Running Trial #4

Hyperparameter    |Value             |Best Value So Far
conv_1_filter     |64                |48
conv_1_kernel     |3                 |3
conv_2_filter     |48                |128
conv_2_kernel     |5                 |3
dense_1_units     |96                |80
learning_rate     |0.001             |0.001
```

:۴

```
Trial 4 Complete [00h 00m 23s]
val_accuracy: 0.9146666526794434

Best val_accuracy So Far: 0.9156666398048401
Total elapsed time: 00h 02m 19s

Search: Running Trial #5

Hyperparameter    |Value             |Best Value So Far
conv_1_filter     |48                |48
conv_1_kernel     |5                 |3
conv_2_filter     |80                |128
conv_2_kernel     |5                 |3
dense_1_units     |112               |80
learning_rate     |0.001             |0.001
```

:۵

```
Trial 5 Complete [00h 00m 25s]
val_accuracy: 0.9083333611488342

Best val_accuracy So Far: 0.9156666398048401
Total elapsed time: 00h 02m 44s

Search: Running Trial #6

Hyperparameter    |Value             |Best Value So Far
conv_1_filter     |96                |48
conv_1_kernel     |3                 |3
conv_2_filter     |64                |128
conv_2_kernel     |3                 |3
dense_1_units     |80                |80
learning_rate     |0.01              |0.001
```

:۶

```
Trial 6 Complete [00h 00m 27s]
val_accuracy: 0.8668333292007446

Best val_accuracy So Far: 0.9156666398048401
Total elapsed time: 00h 03m 12s

Search: Running Trial #7

Hyperparameter    |Value             |Best Value So Far
conv_1_filter     |96                |48
conv_1_kernel     |5                 |3
conv_2_filter     |48                |128
conv_2_kernel     |5                 |3
dense_1_units     |112               |80
learning_rate     |0.01              |0.001
```

:۷

```
Trial 7 Complete [00h 00m 24s]
val_accuracy: 0.8566666841506958

Best val_accuracy So Far: 0.9156666398048401
Total elapsed time: 00h 03m 36s

Search: Running Trial #8

Hyperparameter    |Value             |Best Value So Far
conv_1_filter     |32                |48
conv_1_kernel     |3                 |3
conv_2_filter     |112               |128
conv_2_kernel     |3                 |3
dense_1_units     |64                |80
learning_rate     |0.001             |0.001
```

:۸

```
Trial 8 Complete [00h 00m 27s]
val_accuracy: 0.9143333435058594

Best val_accuracy So Far: 0.9156666398048401
Total elapsed time: 00h 04m 04s

Search: Running Trial #9

Hyperparameter    |Value             |Best Value So Far
conv_1_filter     |80                |48
conv_1_kernel     |5                 |3
conv_2_filter     |64                |128
conv_2_kernel     |5                 |3
dense_1_units     |64                |80
learning_rate     |0.001             |0.001
```

:۹

```
Trial 9 Complete [00h 00m 25s]
val_accuracy: 0.909500002861023

Best val_accuracy So Far: 0.9156666398048401
Total elapsed time: 00h 04m 29s

Search: Running Trial #10

Hyperparameter    |Value             |Best Value So Far
conv_1_filter     |128               |48
conv_1_kernel     |5                 |3
conv_2_filter     |112               |128
conv_2_kernel     |3                 |3
dense_1_units     |32                |80
learning_rate     |0.01              |0.001
```

و در نهایت بهترین پارامترها به صورت زیر گزارش شدند:

```
Trial 10 Complete [00h 00m 27s]
val_accuracy: 0.86599999666213399

Best val_accuracy So Far: 0.9156666398048401
Total elapsed time: 00h 04m 57s
INFO:tensorflow:Oracle triggered exit
```

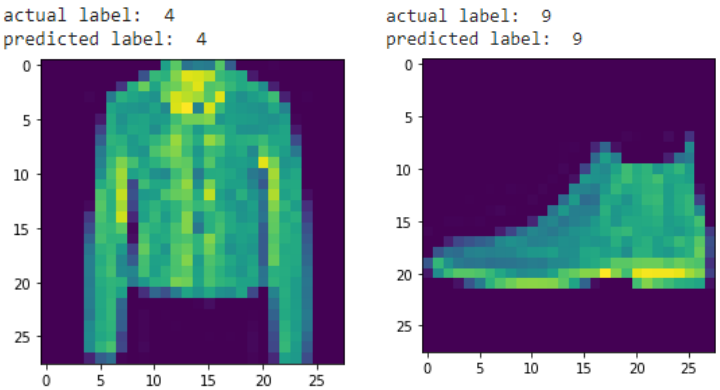
با استفاده از ماژول get_best_models نیز می‌توان بهترین hyperparameter را برای مدل دریافت کرد:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 48)        480
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 24, 24, 128)       55424
_________________________________________________________________
flatten (Flatten)            (None, 73728)             0
_________________________________________________________________
dense (Dense)                (None, 80)                5898320
_________________________________________________________________
dense_1 (Dense)              (None, 10)                810
=================================================================
Total params: 5,955,034
Trainable params: 5,955,034
Non-trainable params: 0
_________________________________________________________________
```

داده‌های آموزش را با مدلی که دارای hyperparameterهای بهتری است فیت می‌کنیم:

```
[19]  1 history = model.fit(train_images, train_labels, epochs=3, validation_split=0.1)

Epoch 1/3
1688/1688 [==============================] - 10s 6ms/step - loss: 0.1019 - accuracy: 0.9613 - val_loss: 0.2884 - val_accuracy: 0.9175
Epoch 2/3
1688/1688 [==============================] - 10s 6ms/step - loss: 0.0665 - accuracy: 0.9754 - val_loss: 0.2831 - val_accuracy: 0.9167
Epoch 3/3
1688/1688 [==============================] - 10s 6ms/step - loss: 0.0427 - accuracy: 0.9844 - val_loss: 0.3282 - val_accuracy: 0.9115
```

سپس prediction را برای اطمینان از صحت مدل طراحی شده و الگوریتم، انجام می‌دهیم:



همانطور که مشخص است، در هر دو مورد پاسخ پیش‌بینی شده با پاسخ اصلی همسان است.

منابع

- https://medium.com/@dipti.rohan.pawar/improving-performance-of-convolutional-neural-network-2ecfe0207de7
- https://www.datacamp.com/community/tutorials/convolutional-neural-networks-python
- https://nayakpplaban.medium.com/automatic-hyperparameter-tuning-with-keras-tuner-and-tensorflow-2-0-be709cc3d259
- https://machinelearningmastery.com/how-to-develop-a-cnn-from-scratch-for-fashion-mnist-clothing-classification/#:~:text=The%20Fashion%2DMNIST%20dataset%20is,shirts%2C%20dresses%2C%20and%20more