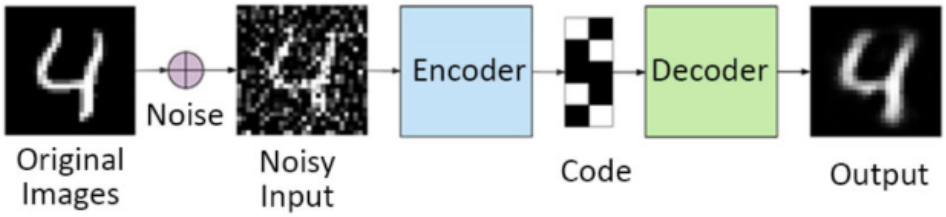
استفاده از خودرمزگذارها برای طبقه‌بندی در مجموعه داده Fashion-MNIST

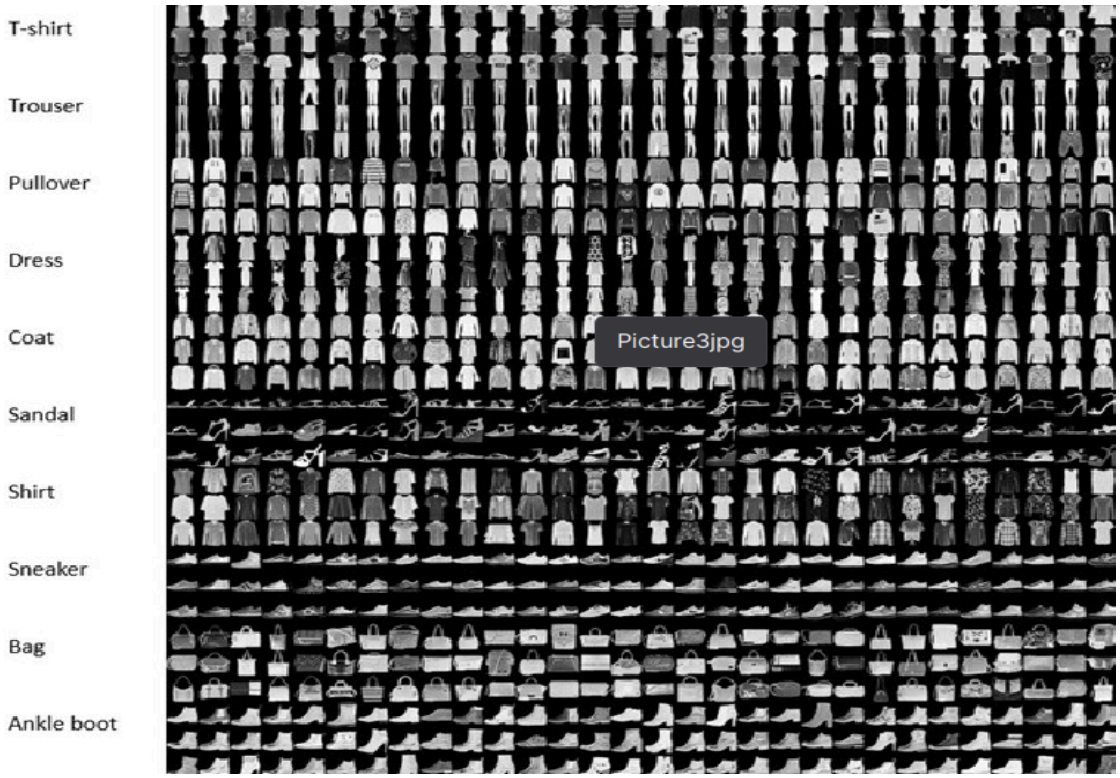
مقدمه

خودرمزگذارها شبکه‌های عصبی قدرتمندی هستند که برای یادگیری کدهای کارآمد از داده‌های ورودی طراحی شده‌اند. آنها از دو بخش اصلی تشکیل شده‌اند: یک رمزگذار که ورودی را به یک نمایش فشرده در فضای نهان تبدیل می‌کند و یک رمزگشا که ورودی را از این نمایش بازسازی می‌کند. در حالی که به طور سنتی برای وظایفی مانند کاهش ابعاد و یادگیری ویژگی‌ها استفاده می‌شوند، خودرمزگذارها را می‌توان برای وظایف طبقه‌بندی نیز تطبیق داد. این سند فرآیند استفاده از خودرمزگذارها برای طبقه‌بندی تصاویر از مجموعه داده Fashion-MNIST را توضیح می‌دهد و بررسی می‌کند که چگونه مراحل مختلف پیاده‌سازی بر عملکرد مدل تأثیر می‌گذارند.



مجموعه داده Fashion-MNIST

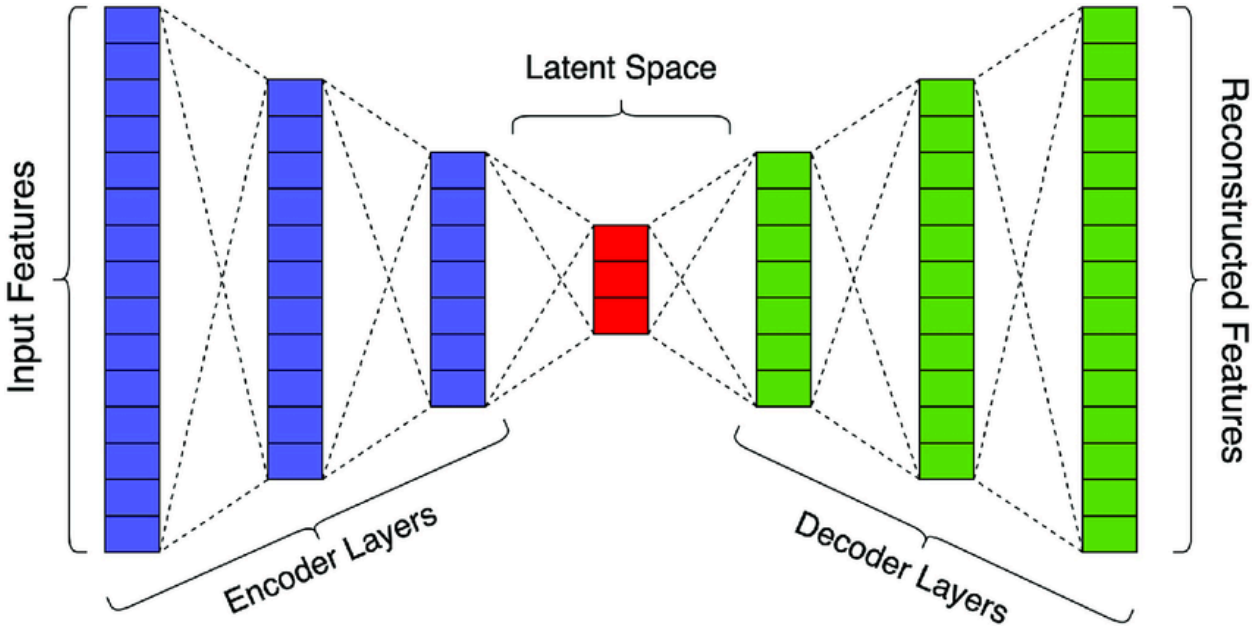
مجموعه داده Fashion-MNIST شامل تصاویر خاکستری ۲۸x۲۸ از ۷۰۰۰۰ محصول مد در ۱۰ دسته مختلف است. این مجموعه به ۶۰۰۰۰ تصویر آموزشی و ۱۰۰۰۰ تصویر آزمایشی تقسیم می‌شود.



خودرمزگذارها برای طبقه‌بندی

در زمینه Classification می‌توان از یک AE برای استخراج ویژگی‌های معنادار از داده‌های ورودی استفاده کرد. این Feature هل سپس به یک شبکه طبقه‌بندی جداگانه تغذیه می‌شوند تا وظیفه نهایی دسته‌بندی داده‌های ورودی را انجام دهد.

1) **Encoder** : تصویر ورودی را به یک نمایش پایین‌بعدی فشرده می‌کند.
2) **فضای Latent**: نمایش فشرده‌شده تصویر ورودی را نشان می‌دهد.
3) **Decoder**: تصویر ورودی را از نمایش نهان بازسازی می‌کند (فقط در مرحله آموزش استفاده می‌شود).
4) **Classifier**: از نمایش فضای نهان برای پیش‌بینی کلاس تصویر ورودی استفاده می‌کند.



5) **مزایای استفاده از Autoencoder ها**

- **کاهش ابعاد**: کمک می‌کند تا پیچیدگی داده‌ها کاهش یابد و تسک طبقه‌بندی ساده‌تر شود.
- **استخراج ویژگی‌ها (Feature Extraction)**: ویژگی‌های مقاومی را یاد می‌گیرد که می‌توانند عملکرد مدل طبقه‌بندی را بهبود بخشند.
- **کاهش نویز**: می‌تواند کمک کند تا نویز از داده‌ها حذف شود و منجر به نتایج بهتر طبقه‌بندی شود.

1. ساخت خودرمزگذار

Encoder تصویر ورودی را به یک فضای Latent فشرده می‌کند. این شامل چندین لایه Convolutional است که spatial dimensions را کاهش داده و عمق Feature Maps را افزایش می‌دهد.

```
def encoder(input_img):
    conv1 = Conv2D(32, (3, 3), activation='relu', padding='same')(input_img) #28 x 28 x 32
    conv1 = BatchNormalization()(conv1)
    conv1 = Conv2D(32, (3, 3), activation='relu', padding='same')(conv1)
    conv1 = BatchNormalization()(conv1)
    pool1 = MaxPooling2D(pool_size=(2, 2))(conv1) #14 x 14 x 32
    conv2 = Conv2D(64, (3, 3), activation='relu', padding='same')(pool1) #14 x 14 x 64
    conv2 = BatchNormalization()(conv2)
    conv2 = Conv2D(64, (3, 3), activation='relu', padding='same')(conv2)
    conv2 = BatchNormalization()(conv2)
    pool2 = MaxPooling2D(pool_size=(2, 2))(conv2) #7 x 7 x 64
    conv3 = Conv2D(128, (3, 3), activation='relu', padding='same')(pool2) #7 x 7 x 128 (small and thick)
    conv3 = BatchNormalization()(conv3)
    conv3 = Conv2D(128, (3, 3), activation='relu', padding='same')(conv3)
    conv3 = BatchNormalization()(conv3)
    conv4 = Conv2D(256, (3, 3), activation='relu', padding='same')(conv3) #7 x 7 x 256 (small and thick)
    conv4 = BatchNormalization()(conv4)
    conv4 = Conv2D(256, (3, 3), activation='relu', padding='same')(conv4)
    conv4 = BatchNormalization()(conv4)
    return conv4
```

Decoder تصویر ورودی را از Latent بازسازی می‌کند. این شامل لایه‌های Convolutional است که به تدریج feature maps را به اندازه input اصلی افزایش می‌دهد.

```
def decoder(conv4):
    conv5 = Conv2D(128, (3, 3), activation='relu', padding='same')(conv4) #7 x 7 x 128
    conv5 = BatchNormalization()(conv5)
    conv5 = Conv2D(128, (3, 3), activation='relu', padding='same')(conv5)
    conv5 = BatchNormalization()(conv5)
    conv6 = Conv2D(64, (3, 3), activation='relu', padding='same')(conv5) #7 x 7 x 64
    conv6 = BatchNormalization()(conv6)
    conv6 = Conv2D(64, (3, 3), activation='relu', padding='same')(conv6)
    conv6 = BatchNormalization()(conv6)
    up1 = UpSampling2D((2,2))(conv6) #14 x 14 x 64
    conv7 = Conv2D(32, (3, 3), activation='relu', padding='same')(up1) # 14 x 14 x 32
    conv7 = BatchNormalization()(conv7)
    conv7 = Conv2D(32, (3, 3), activation='relu', padding='same')(conv7)
    conv7 = BatchNormalization()(conv7)
    up2 = UpSampling2D((2,2))(conv7) # 28 x 28 x 32
    decoded = Conv2D(1, (3, 3), activation='sigmoid', padding='same')(up2) # 28 x 28 x 1
    return decoded
```

2. آموزش AE

خودرمزگذار را با استفاده از تصاویر آموزشی برای به حداقل رساندن خطای بازسازی آموزش دهید. این مرحله اطمینان می‌دهد که رمزگذار به طور مؤثری تصاویر ورودی را Compress می‌کند.

بررسی استفاده از GPU

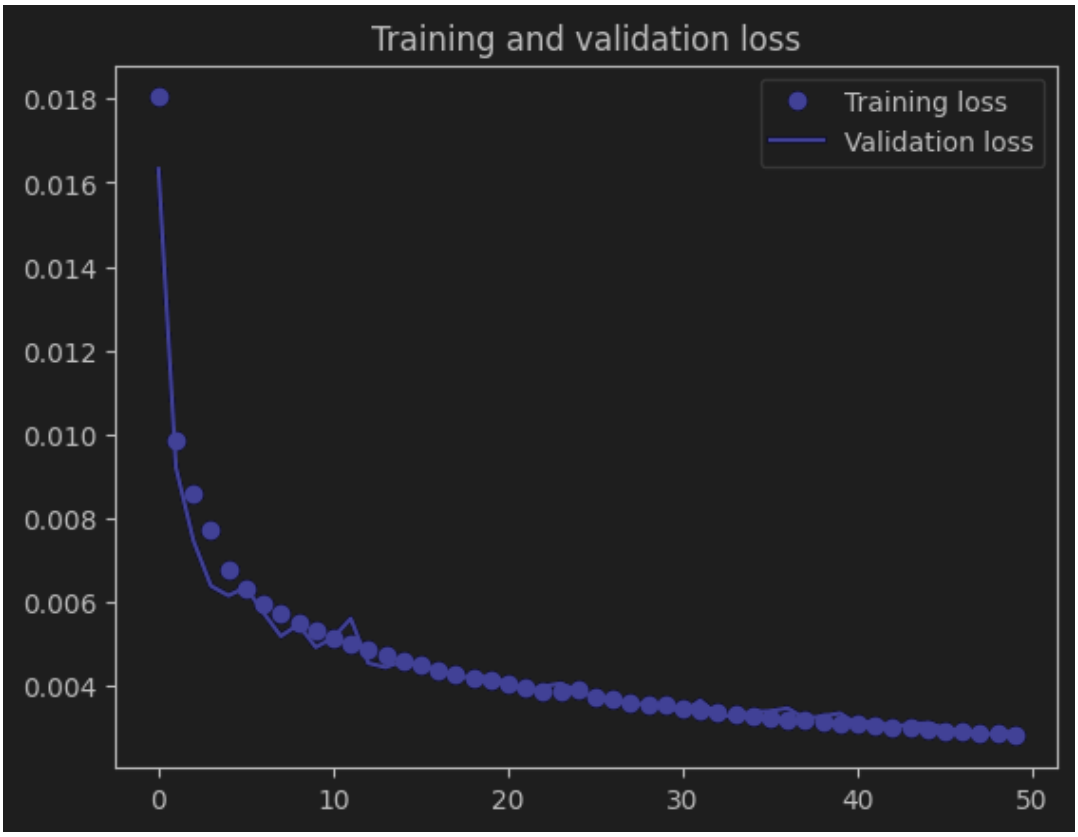
```
os.environ["CUDA_DEVICE_ORDER"] = "PCI_BUS_ID"
os.environ["CUDA_VISIBLE_DEVICES"]= "0"
```

آموزش مدل

```
autoencoder_train = autoencoder.fit(train_X, train_ground, batch_size=batch_size,epochs=epochs,verbose=1,validation_data=(valid_X,
 valid_ground))
```

```
Epoch 43/50
750/750 [==============================] - 12s 16ms/step - loss: 0.0030 - val_loss: 0.0030
Epoch 44/50
750/750 [==============================] - 12s 15ms/step - loss: 0.0030 - val_loss: 0.0031
Epoch 45/50
750/750 [==============================] - 11s 15ms/step - loss: 0.0030 - val_loss: 0.0031
Epoch 46/50
750/750 [==============================] - 12s 15ms/step - loss: 0.0029 - val_loss: 0.0030
Epoch 47/50
750/750 [==============================] - 12s 15ms/step - loss: 0.0029 - val_loss: 0.0029
```

نمایش عملکرد مدل



ذخیره مدل آموزش‌دیده

```
autoencoder.save_weights('autoencoder.h5')
```

تأثیر بر عملکرد:

- **یک خودرمزگذار خوب آموزش‌دیده پایه محکمی برای Feature Extraction فراهم می‌کند، که برای وظیفه Classification بعدی حیاتی است.**

3. Feature Extraction برای Classification

پس از آموزش خودرمزگذار، از بخش Encoder برای تبدیل تصاویر ورودی به Latent استفاده کنید. این ویژگی‌ها به عنوان Input برای شبکه Classifier استفاده می‌شوند.

تأثیر بر عملکرد:

- **ویژگی‌های استخراج‌شده انتظار می‌رود که تمایزی و مقاوم‌تر باشند، که منجر به دقت بهتر طبقه‌بندی می‌شود.**

4. ساخت و آموزش Classifier

تأثیر بر عملکرد:

- **آموزش فقط لایه‌های متراکم ابتدا به Classifier اجازه می‌دهد تا بدون تغییر ویژگی‌های پیش‌آموزش‌یافته سریع یاد بگیرد.**

تبدیل label ها به one-hot encoding

```
train_Y_one_hot = to_categorical(train_labels)
test_Y_one_hot = to_categorical(test_labels)
```

```
print('previous:', train_labels[0])
print('after:', train_Y_one_hot[0])
```

```
previous: 9
after: [0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
```

Train-Test Split & Validation Set Shapes

Train-Test Split

I'm going to use the same random state I used

```
train_X,valid_X,train_label,valid_label = train_test_split(train_data,train_Y_one_hot,test_size=0.2,random_state=13)
```

Training and Validation set shapes

```
train_X.shape,valid_X.shape,train_label.shape,valid_label.shape
```

```
((48000, 28, 28, 1), (12000, 28, 28, 1), (48000, 10), (12000, 10))
```

ساخت Fully Connected Layers

```
def fc(enco):
    flat = Flatten()(enco)
    den = Dense(128, activation='relu')(flat)
    out = Dense(num_classes, activation='softmax')(den)
    return out
```

```
encode = encoder(input_img)
full_model = Model(input_img,fc(encode))
```

```
for l1,l2 in zip(full_model.layers[:19],autoencoder.layers[0:19]):
    l1.set_weights(l2.get_weights())
```

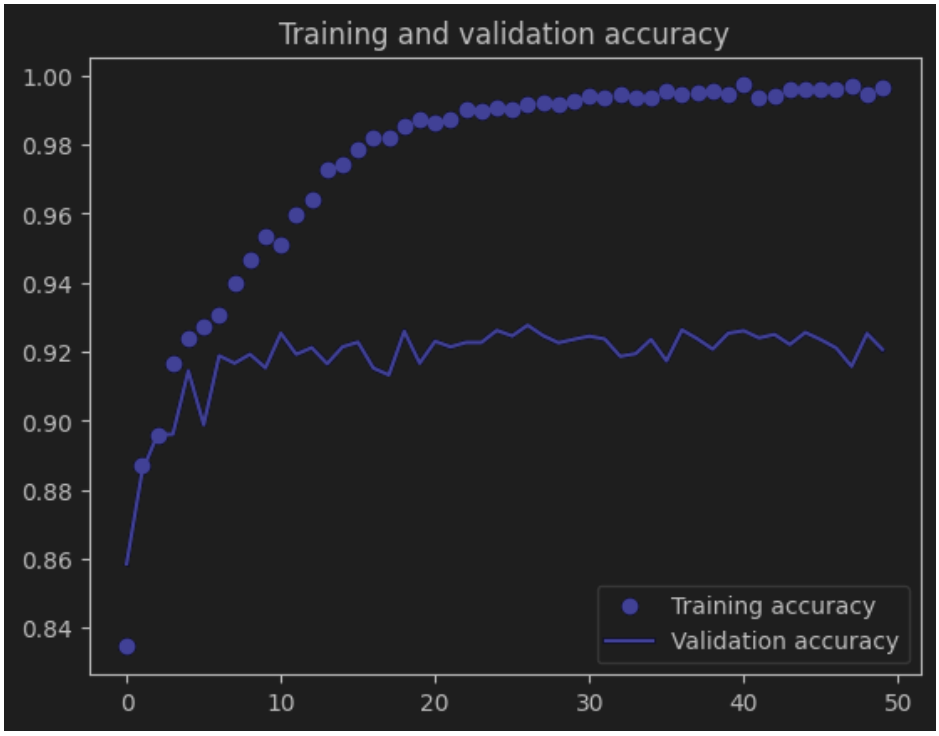
ترین کردن فقط بخش (Fully Connected) FC

```
for layer in full_model.layers[0:19]:
    layer.trainable = False
```

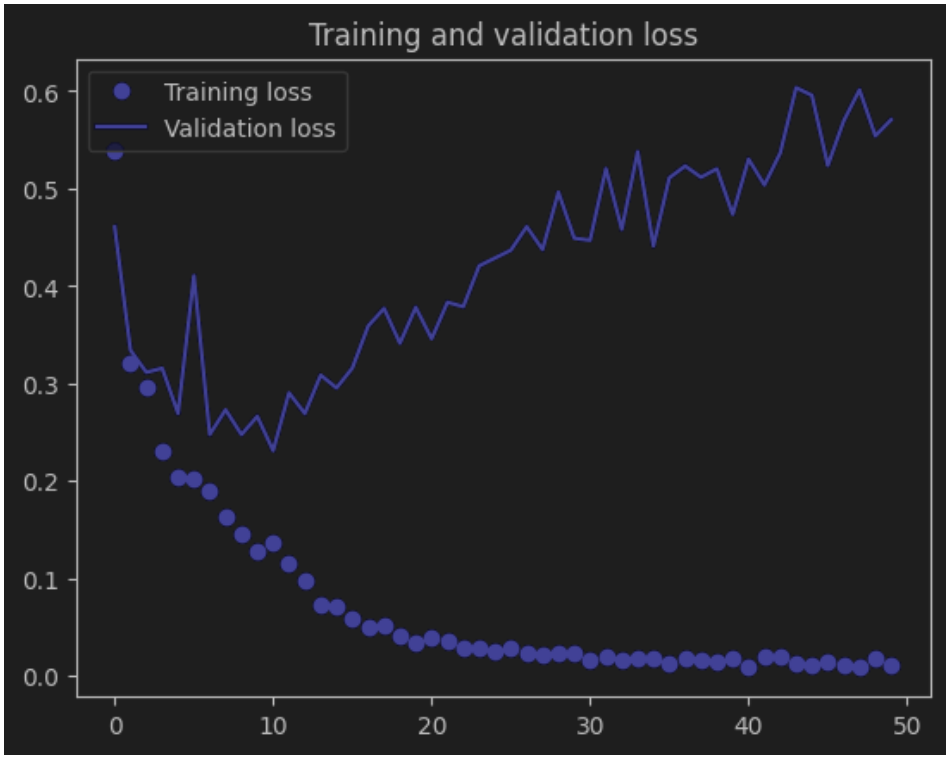
کامپایل مدل و بعد دیدن خلاصه لایه های مدل در صورت نیاز

و ترین کردن کل مدل با همان مقدار Random States

عملکرد مدل پلات اول



عملکرد مدل پلات دوم



لایه‌های Encoder را باز کرده و با آموزش کل شبکه از ابتدا به انتها تنظیم دقیق انجام دهید. این مرحله به رمزگذار اجازه می‌دهد تا وزن‌های خود را تنظیم کند تا عملکرد طبقه‌بندی را بیشتر بهبود بخشد.

تأثیر بر عملکرد:

- **تنظیم دقیق به دستیابی به دقت بالاتر کمک می‌کند زیرا بخش‌های رمزگذار و طبقه‌بندی‌کننده مدل بهتر هماهنگ می‌شوند.**

نتایج و بحث

معیارهای ارزیابی

مدل را با استفاده از معیارهای استاندارد طبقه‌بندی ارزیابی کنید:

- **دقت: نسبت نمونه‌های درست طبقه‌بندی‌شده.**
- **دقت، بازخوانی، نمره F1: درک دقیق‌تری از عملکرد مدل فراهم می‌کنند.**

```
test_eval = full_model.evaluate(test_data, test_Y_one_hot, verbose=0)
```

```
print('Test loss:', test_eval[0])
print('Test accuracy:', test_eval[1])
```

```
 Test loss: 0.5554893612861633
 Test accuracy: 0.9187999963760376
```

بهبودهای عملکرد

- **آموزش اولیه: عملکرد پایه را فراهم می‌کند.**
- **Feature Extraction: بهبود را به دلیل ویژگی‌های مقاوم یادگرفته‌شده توسط Encoder نشان می‌دهد.**
- **Fine-Tuning: با Optimize کل شبکه عملکرد را بیشتر بهبود می‌بخشد.**

Visualization

- **Confusion Matrix: برای Visualize عملکرد Classification در کلاس‌های مختلف.**
- **پیش‌بینی‌های Correct و Incorrect: برای تجزیه و تحلیل عملکرد خوب مدل و جایی که شکست می‌خورد.**

پیش بینی لیبل ها و نمایش نمونه درست ها و غلط ها:

```
predicted_classes = full_model.predict(test_data)
```

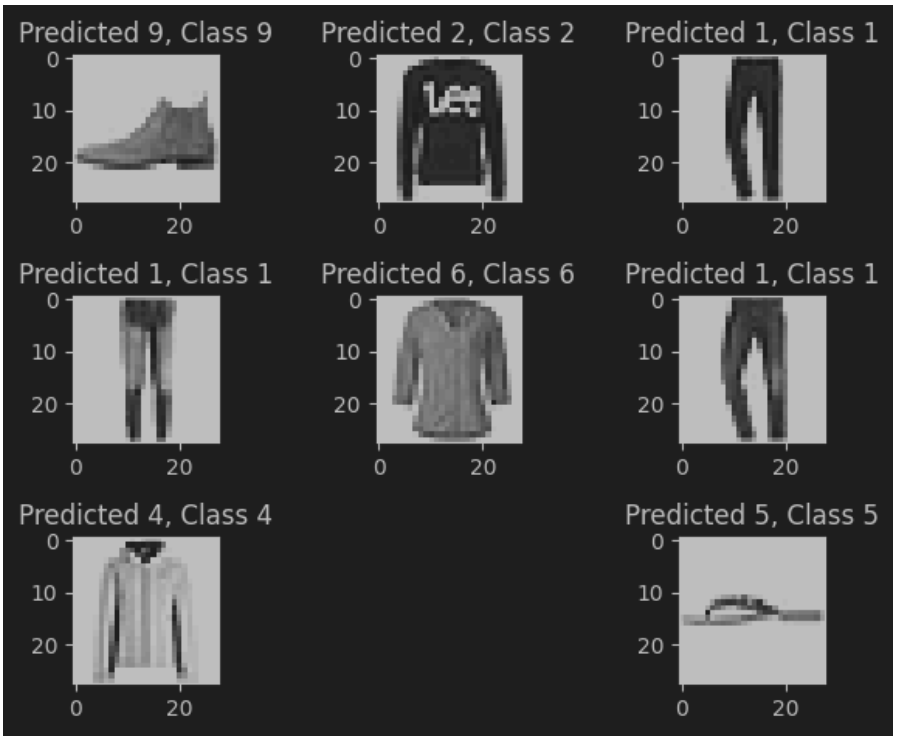
```
 313/313 [==============================] - 1s 3ms/step
```

```
predicted_classes = np.argmax(np.round(predicted_classes),axis=1)
```

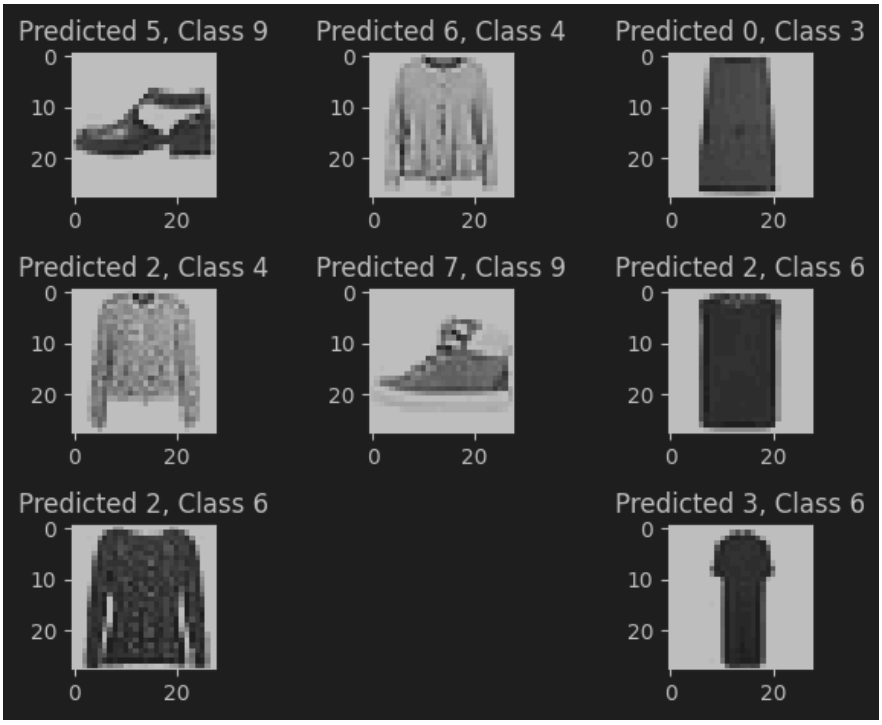
```
predicted_classes.shape, test_labels.shape
```

```
 ((10000,), (10000,))
```

Visualize Correct Labels Predicted



Visualize Incorrect Labels Predicted



گزارش طبقه بندی

	precision	recall	f1-score	support
Class 0	0.86	0.87	0.87	1000
Class 1	0.99	0.98	0.99	1000
Class 2	0.86	0.89	0.88	1000
Class 3	0.91	0.93	0.92	1000
Class 4	0.88	0.84	0.86	1000
Class 5	0.98	0.99	0.99	1000
Class 6	0.78	0.76	0.77	1000
Class 7	0.97	0.97	0.97	1000
Class 8	0.98	0.98	0.98	1000
Class 9	0.97	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

پایان