Q1)

الف)

KerasTunerیک ابزار است که برای بهینه‌سازی هایپرپارامترها در مدل‌های شبکه عصبی با استفاده از کتابخانه Keras در زبان برنامه‌نویسی Python طراحی شده است. این ابزار به طور خاص برای انتخاب بهینه‌ترین مقادیر برای هایپرپارامترهای شبکه عصبی کاربرد دارد، که این موارد مثل تعداد لایه‌ها، تعداد نورون‌ها در هر لایه، نرخ یادگیری، و سایر پارامترهای مهم در فرآیند آموزش مدل هستند.

میتوان یک شبکه کانولوشنی ساخت و با استفاده از این ابزار، هایپرپارامتر ها را تیون کرد. این هایپرپارامترها میتواند شامل پارامترهای آموزشی مثل نرخ یادگیری و بتا و... باشد یا این که هایپرپارامترهای ساختاری باشد. مثلا میتوان مشخص کرد که از چند لایه کانولوشنی یا چند لایه dense استفاده کرد. همچنین میتوان تعداد فیلترها را تیون کرد.

KerasTuner ابزاری است که امکان انجام بهینه‌سازی هایپرپارامترها در شبکه‌های عصبی را فراهم می‌کند. این ابزار از تنظیم‌دهنده‌ها (Tuners)مختلفی برای انجام جستجوی بهینه‌سازی استفاده می‌کند. در زیر، تعدادی از تنظیم‌دهنده‌های موجود در KerasTuner توضیح داده شده‌اند:

RandomSearch:

تنظیم‌دهنده RandomSearch به صورت تصادفی هایپرپارامترها را جستجو می‌کند. این روش مناسب برای اولین تلاش در بهینه‌سازی است و به عنوان یک جستجوی تصادفی در فضای هایپرپارامترها عمل می‌کند.

Hyperband:

تنظیم‌دهنده Hyperband از الگوریتم Hyperband برای انجام جستجوی بهینه‌سازی استفاده می‌کند. این الگوریتم به صورت موازی تعداد زیادی از مدل‌های مختلف را آموزش می‌دهد و هایپرپارامترهای بهینه را پیدا می‌کند.

BayesianOptimization:

تنظیم‌دهنده BayesianOptimization از الگوریتم بهینه‌سازی بیزی برای جستجوی بهینه‌سازی استفاده می‌کند. این الگوریتم معمولاً به عنوان یک روش هوش مصنوعی برای بهینه‌سازی هایپرپارامترها مورد استفاده قرار می‌گیرد.

Sklearn:

تنظیم‌دهنده Sklearn به عنوان یک تنظیم‌دهنده مبتنی بر روش‌های بهینه‌سازی از کتابخانه scikit-learn استفاده می‌کند.

TuneGridSearch:

تنظیم‌دهنده TuneGridSearch بر اساس روش جستجوی گرید (Grid Search)، به ترکیب تمامی مقادیر ممکن هایپرپارامترها پرداخته و بهینه‌سازی انجام می‌دهد.

این تنظیم‌دهنده‌ها می‌توانند به صورت همروند (parallel) کار کنند و برخی از آن‌ها از استراتژی‌های بهینه‌سازی مختلف برای کاهش زمان جستجو و افزایش دقت استفاده می‌کنند. برای استفاده از هر تنظیم‌دهنده، باید یک شی از آن ایجاد و سپس آن را برای جستجوی بهینه‌سازی شبکه عصبی خود استفاده کنید.

برای این آموزش، ابتدا از تیونر Hyperband استفاده میکنم چون تیونری با دقت مناسب است، اما در صورتی که این تیونر کند عمل میکرد و آموزش خیلی کند شده بود از Random Search استفاده میکنم.

پس از نوشتن کد و آموزش مدل، دیدم که به علت زیاد بودن تعداد متغیرها، تیونر Hyperband خیلی کند عمل میکند بنابراین از random search استفاده کردم.

ب)

مجموعه داده MNIST یکی از مجموعه‌های داده معروف در حوزه یادگیری ماشین و بینایی ماشین است. این مجموعه داده برای تشخیص ارقام دست‌نویس از 0 تا 9 استفاده می‌شود. مجموعه داده MNIST معمولاً به عنوان مقدمه‌ای برای مسائل دسته‌بندی تصویری و یادگیری عمیق (Deep Learning) به کار می‌رود. مشخصات این مجموعه داده به شرح زیر است:

تصاویر:

هر تصویر در این مجموعه داده یک تصویر دست‌نویس با ابعاد 28 28xپیکسل است.

تصاویر سیاه و سفید (گریس‌اسکیل) هستند.

مقادیر برچسب:

هر تصویر با یک عدد از 0 تا 9 برچسب‌گذاری شده است، که نمایانگر عدد دست‌نویس موجود در تصویر است.

مسئله دسته‌بندی در اینجا این است که به تصویرهای دست‌نویس یک برچسب از 0 تا 9 نسبت داده شود.

تعداد داده‌ها:

مجموعه داده MNIST شامل 60,000 تصویر برای آموزش (training set) و 10,000 تصویر برای آزمایش (test set) است.

این مجموعه داده از آنجا که ساده و کوچک است، به عنوان یک صفر به یادگیری ماشین برای مقدمه به مفاهیم مختلف از جمله شبکه‌های عصبی عمیق (Deep Neural Networks)، یادگیری تصویری، و دسته‌بندی معرفی می‌شود.

برای دسته بندی تصاویر این دیتاست، میتوانیم پارامترهای شبکه کانولوشنی را به کمک کتابخانه Keras-Tuner، بهینه سازی کنیم. شبکه مورد نظر ما نه باید آنقدر ساده باشد که دقت کمی داشته باشد، نه آنقدر پیچیده که داده ها را حفظ کند. این تعادل را میتوانیم به کمک این کتابخانه به دست بیاوریم. پارامترهای قابل تیون کردن در بالاتر گفته شده.

لایه Dropout برای کاهش overfitting گزینه بسیار مناسبی است. این لایه خروجی برخی از نورون ها را صفر میکند که این کار باعث میشود که خروجی نهایی، حساس به یک نورون خاص نباشد و این به این معناست که مدل نمیتواند داده ها را حفظ کند.

لایه pooling هم برای استخراج نقاط و ویژگی های برجسته و همچنین برای کاهش ابعاد تصویر و افزایش میدان تاثیر فیلترها کاربرد مهمی دارد. با استفاده از این لایه میتوانیم ویژگی های مهم را ار ابعاد بزرگ تری از تصویر استخراج کنیم.

ج)

کد این سوال در فایل Q1.ipynb موجود است و در پایین شرح داده شده:

```
[13] import tensorflow as tf
     from tensorflow import keras
     from keras_tuner.tuners import RandomSearch
     from kerastuner.tuners import Hyperband
     from keras_tuner.engine.hyperparameters import HyperParameters
```

در ابتدا، کتابخانه keras-tuner را نصب و ایمپورت کردم.

سپس دیتاست را لود کردم و preprocessing های مورد نیاز مثل تغییر رینج اعداد به 0 تا 1 و کتگوریکال کردن لیبل‌ها را انجام دادم.

```
[3] mnist = tf.keras.datasets.mnist
    (x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
```

```
[4] x_train, x_test = x_train / 255.0, x_test / 255.0

    y_train = keras.utils.to_categorical(y_train, 10)
    y_test = keras.utils.to_categorical(y_test, 10)

    x_train = x_train.reshape((x_train.shape[0], 28, 28, 1))
    x_test = x_test.reshape((x_test.shape[0], 28, 28, 1))
```

سپس تابع build_cnn_model را نوشتم که ساختار اصلی مدل است. برای این مدل، طبق توضیحات صورت سوال، برای تعداد لایه ها و تعداد نورون ها و فیلترها و همچنین نرخ یادگیری، یک متغیر قابل تیون کردن در نظر گرفتم.

```
for i in range(hp.Int('num_conv_layers', min_value=1, max_value=5)):
  if i % 2 == 1:
    model.add(keras.layers.MaxPooling2D(pool_size=2, strides=2))
    inputsize = inputsize // 2

  model.add(keras.layers.Conv2D(
      filters=hp.Int('conv1_filters', min_value=1, max_value=256, step=16),
      kernel_size=3,
      activation='relu',
      input_shape=(inputsize, inputsize, 1)
  ))
  inputsize -= 1
```

در این حلقه، که تعداد آن از 1 تا 5 متغیر است، لایه های کانولوشنی ایجاد میشوند که خود لایه های کانولوشنی از 1 تا 256 فیلتر دارند. به ازای هر دو لایه کانولوشنی، یک لایه pooling هم ایجاد کردم.

```
model.add(keras.layers.Flatten())
```

سپس لایه flatten گذاشته ام تا پس از آن از لایه دنس استفاده کنم.

```
for i in range(hp.Int('num_dense_layers', min_value=1, max_value=5)):
    model.add(keras.layers.Dense(
        units=hp.Int(f'dense_{i}_units', min_value=32, max_value=256, step=32),
        activation='relu'
    ))
```

در این حلقه هم مانند حلقه قبلی، 1 تا 5 لایه های دنس با 32 تا 256 نورون اضافه میشوند.

در نهایت لایه خروجی با 10 نورون به علت ده کلاسه بودن مسئله گذاشته ام و مدل را کامپایل کردم

```
model.add(keras.layers.Dense(10, activation='softmax'))
model.compile(optimizer=keras.optimizers.Adam(hp.Choice('learning_rate', values=[1e-3, 5e-4, 1e-4])),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

برای نرخ آموزش هم سه حالت 0.001 و 0.0005 و 0.0001 را در نظر گرفته ام.

```
tuner = RandomSearch(
    build_cnn_model,
    objective='val_accuracy',
    max_trials=10,
    directory='RandomSearch_tuner',
    project_name='RandomSearch_tuner'
)
```

سپس تیونر را ایجاد کردم و آموزش را شروع کردم. همانطور که بالاتر توضیح دادم ابتدا از Hyperband استفاده شده بود اما در نهایت به سراغ RandomSearch رفتم.

```
[32] best_model = tuner.get_best_models(num_models=1)[0]
```

```
best_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 225)       2250

 max_pooling2d (MaxPooling2  (None, 13, 13, 225)       0
 D)

 conv2d_1 (Conv2D)           (None, 11, 11, 225)       455850

 conv2d_2 (Conv2D)           (None, 9, 9, 225)         455850

 flatten (Flatten)           (None, 18225)             0

 dense (Dense)               (None, 64)                1166464

 dense_1 (Dense)             (None, 64)                4160

 dense_2 (Dense)             (None, 10)                650

=================================================================
Total params: 2085224 (7.95 MB)
Trainable params: 2085224 (7.95 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

در نهایت مدل را آموزش دادم و بهترین مدل را انتخاب کردم. نتیجه به این صورت بود. دقت این مدل 99 درصد بود.

در مورد اندازه کرنل در لایه های کانولوشنی، اندازه کرنل را 3 قرار دادم. چون از لایه pooling استفاده میکردم و میدان تاثیر به اندازه کافی بزرگ میشد، نیازی ندیدم که از کرنل های بزرگ تر استفاده کنم و نتیجه خوبی هم گرفتم.

در این ساختار، من از لایه dropout استفاده نکردم اما از لایه pooling استفاده کردم. استفاده از این لایه باعث میشود که میدان تاثیر افزایش پیدا کند و هنگام max گرفتن، تنها ویژگی های مهم استخراج میشود که باعث میشود که مدل براساس ویژگی های بی اهمیت پاسخ ندهد و نمونه ها را حفظ نکند. لایه Dropout هم با صفر کردن خروجی بعضی نورون ها حساسیت را به یک نورون خاص از بین میبرد اما من استفاده نکردم.

Q2)

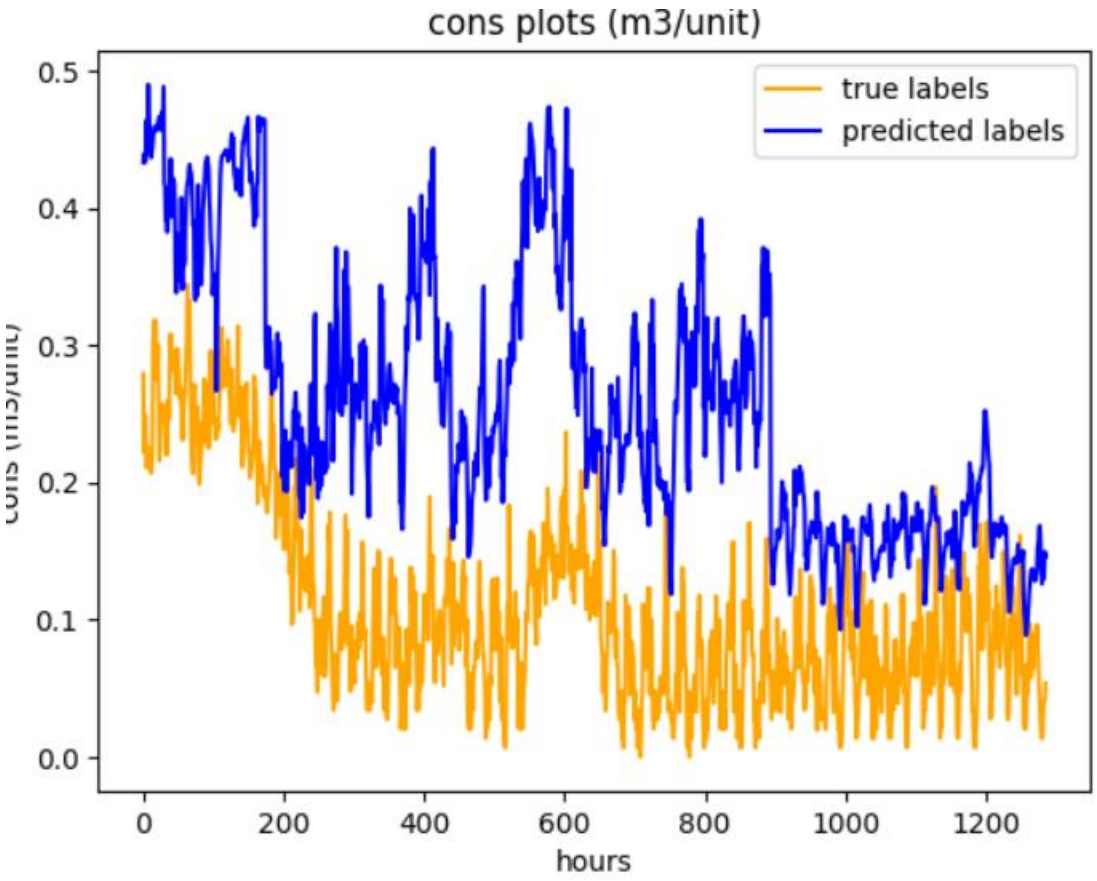
فایل نوتبوک کامل شده.

Q3)

دقت r2 منفی نشان دهنده این است که پیشبینی مدل خیلی ضعیف بوده و حتی از خروجی دادن میانگین مقادیر لیبل ها هم ضعیف تر بوده.

علت این مشکل این میتواند باشد که مقادیر تاریخ در داده های ترین کم تر از مقادیر تاریخ در داده های تست هستند.

از مقایسه لیبل ها با پیشبینی میبینیم که مدل توانسته کم و زیاد شدن لیبل ها را تقریبا پیشبینی کند، اما همه لیبل ها را با یک بایاس، بیشتر از مقدار واقعی پیشبینی کرده:



به طور کلی، این داده ها برای استفاده در LSMT آماده نشده اند. این مطلب از یک سایت برداشته شده:

The LSTM network expects the input data (X) to be provided with a specific array structure in the form of *[samples, time steps, features]*.

Currently, the data is in the form of [*samples, features*], and you are framing the problem as one time step for each sample. You can transform the prepared train and test input data into the expected structure using **numpy.reshape()** as follows:

این درحالی است که داده های ما به این صورت نیستند. بنابراین میتوانیم ابتدا با reshape کردن داده ها آنها را به فرم [samples, time steps, features] در بیاوریم، سپس sample ها را شافل کنیم تا تاریخ داده های تست همیشه بزرگ تر از داده های train نباشد. باید توجه کنیم که تنها sample ها را شافل کنیم و time step ها باید به همان صورت باقی بمانند.

همچنین مقادیر فیچرها اعداد بزرگی هستند برای همین برای برخی فیچرهای ورودی میتوان نرمالایزیشن انجام داد و مقادیر را به بین 0 تا 1 تغییر داد. این مورد هم میتواند در برطرف کردن خطای مدل موثر باشد.

Q4)

الف)

شبکه‌های همگشتی:(CNNs)

ساختار اصلی CNNs :برای استخراج ویژگی‌ها از داده‌های دو بعدی مانند تصاویر استفاده می‌شوند. این شبکه‌ها عمدتاً از لایه‌های کانولوشن (Convolutional layers)، لایه‌های ادغام (Pooling layers) و لایه‌های کاملاً متصل (Fully Connected layers) تشکیل شده‌اند.

کاربردها:

تشخیص الگو: مناسب برای تشخیص الگوها و ویژگی‌های مکرر در داده‌های تصویری.

تصویربرداری: برای وظایف مانند تصویربرداری و تشخیص اشیاء در تصاویر.

پردازش تصویر: در بسیاری از وظایف پردازش تصویر مانند تغییر اندازه، تصویربرداری، یادگیری ویژگی‌های تصویری و غیره استفاده می‌شوند.

شبکه‌های بازگشتی:(RNNs)

ساختار اصلی RNNs :برای مدل‌سازی داده‌های دنباله‌ای وابسته به زمان مانند متون یا سیگنال‌های صوتی استفاده می‌شوند. لایه‌های بازگشتی (Recurrent layers)در این شبکه‌ها مسئول حفظ و انتقال اطلاعات از یک زمان به زمان بعدی هستند.

کاربردها:

پردازش زبان طبیعی: برای وظایف مانند ترجمه ماشینی، تشخیص گفتار، و تولید متن.

تحلیل سیگنال‌های زمانی: در حوزه‌هایی مانند پردازش سیگنال‌های صوتی و زمانی.

پیش‌بینی داده‌های دنباله‌ای: مثل پیش‌بینی میزان فروش یک محصول در زمان آتی.

نکته مهم: در بسیاری از وظایف، استفاده از یک ترکیب از CNNs و RNNs (معروف به شبکه‌های بازگشتی-همگشتی یا Recurrent Convolutional Networks) بهترین نتایج را به دنبال دارد، زیرا این ترکیب می‌تواند ویژگی‌های مکانی و زمانی را همزمان در نظر بگیرد و برای وظایف پیچیده‌تری مناسب باشد.

ب)

از نظر تعداد پارامتر، شبکه های کانولوشنی تعداد پارامت های کم تری دارند چون در شبکه های کانولوشنی به جای این که برای هر ورودی یک وزن در نظر گرفته شود، یک فیتر روی تمام عکس حرکت میکند و تعداد وزن ها هم به اندازه تعداد خانه های فیلتر است. اما شبکه های بازگشتی تعداد پارامترهای بیشتری دارند.

از نظر قابلیت موازی سازی، شبکه های کانولوشنی قابلیت موازی سازی خیلی بهتری دارند. شبکه های بازگشتی به علت خاصیت زمانی و مرحله ای که دارند، لازم است که به ترتیب اجرا شوند برای همین قابلیت موازی سازی کمی دارند.

Q5)

پاسخ در فایل q5.pdf در فایل زیپ است.

Q6)

الف)

مورد دوم و سوم صحیح است. نرمال سازی زمان پردازش یک دسته را کم نمیکند زیرا یک محاسبه اضافه است. اما با نرمال کردن توزیع داده ها و همچنین جلوگیری از افزایش زیاد وزن ها، آموزش را با دقت بهتری جلو میبرد و همگرایی را شتاب میدهد.

ب)

کد تکمیل شده در فایل زیپ موجود است.

ج)

مقدار واریانس یک دسته داده ممکن است به صورت اتفاقی 0 شود. برای این که در فرمول نرمال سازی، با ارور تقسیم بر 0 مواجه نشویم، به مقدار V یک عدد بسیار کوچک مثل eps اضافه میکنیم که تاثیری در فرایند آموزش ندارد اما باعث میشود که با ارور تقسیم بر صفر مواجه نشویم.

د)

در صورتی که اندازه دسته در نرمال سازی دسته ای خیلی کوچک باشد، متغیرها و فرایند نرمال سازی همگرا نمیشوند و در واقع عملیات نرمال سازی هر داده را یک طور تغییر میدهد. در واقع این نرمال سازی باعث ایجاد نویز در داده ها میشود. اما اگر اندازه دسته دقیقا 1 باشد، مقدار میانگین با خود داده برابر میشود و تمام مقادیر 0 میشوند و آموزش مختل میشود.

ه)

با فرض این که تنها یک BN داریم که روی خروجی لایه FC و قبل از اکتیویشن فانکشن آن اعمال میشود.

FC parameters = (10 + 1) * 20 = 220

BN parameters = 2 * D = 2 * 20 = 40

Total = 220 + 40 = 260 parameters.

Q7)

کد سوال در فایل Q7.ipynb موجود است. شرح کد:

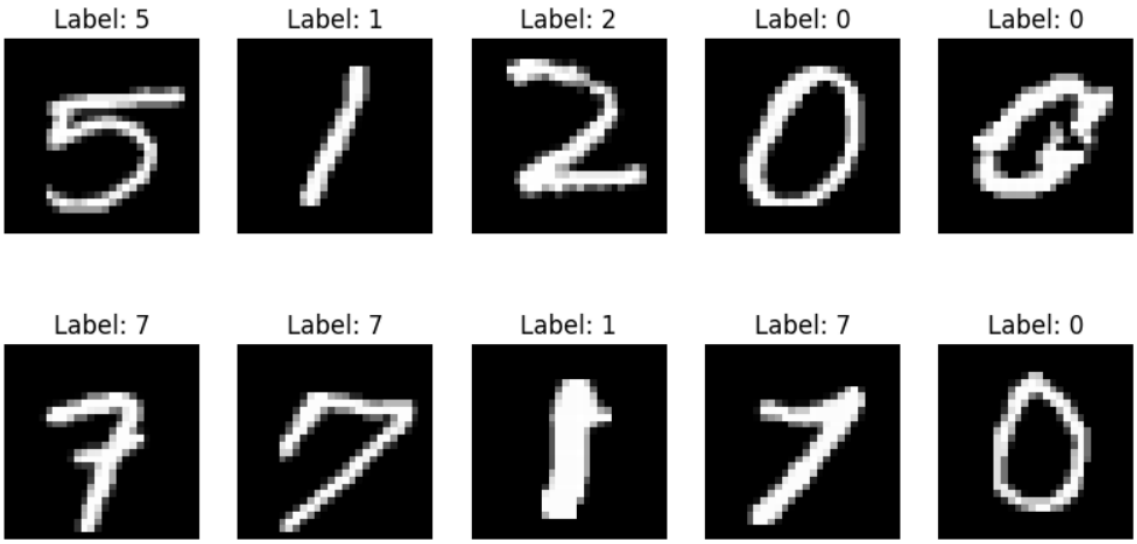
ابتدا دیتاست را لود کرده و ابعاد را همانطور که خواسته شده بود نشان دادم:

```
indices = np.arange(x_train.shape[0])
np.random.shuffle(indices)

x_train = x_train[indices]
y_train = y_train[indices]

print(x_train.shape)
print(y_train.shape)
```

```
(60000, 28, 28)
(60000,)
```

برخی از نمونه ها را چاپ کردم:

```
plt.figure(figsize=(10, 5))
for i in range(10):
    plt.subplot(2, 5, i + 1)
    plt.imshow(x_train[i], cmap='gray')
    plt.title(f"Label: {y_train[i]}")
    plt.axis('off')

plt.show()
```



سپس preprocessing را روی دیتاست انجام دادم. یعنی بازه اعداد را به 0 تا 1 تغییر دادم و لیبل ها را کگوریکال کردم. همچنین ابعاد را از 28*28 به 28*28*1 تغییر دادم که در آموزش مدل مشکل ایجاد نشود.

```
x_train = x_train.astype("float32") / 255
x_test = x_test.astype("float32") / 255
x_train = np.expand_dims(x_train, -1)
x_test = np.expand_dims(x_test, -1)
y_train = tf.keras.utils.to_categorical(y_train, num_classes=10)
y_test = tf.keras.utils.to_categorical(y_test, num_classes=10)
```

سپس مدل را همانطور که خواسته شده بود ایجاد کردم:

```
model = Sequential()
model.add(Input(shape=(28, 28, 1)))
model.add(Conv2D(32, (3, 3), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(10, activation='softmax'))

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 32)        320

 max_pooling2d (MaxPooling2  (None, 14, 14, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 14, 14, 64)        18496

 max_pooling2d_1 (MaxPoolin  (None, 7, 7, 64)          0
 g2D)

 conv2d_2 (Conv2D)           (None, 7, 7, 64)          36928

 max_pooling2d_2 (MaxPoolin  (None, 3, 3, 64)          0
 g2D)

 flatten (Flatten)           (None, 576)               0

 dense (Dense)               (None, 128)               73856

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 130890 (511.29 KB)
Trainable params: 130890 (511.29 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

سپس مدل را آموزش دادم:

```
[10] loss = 'categorical_crossentropy'
     optimizer = 'adam'
     batch_size = 64
     epochs = 15
```

```
model.compile(loss=loss, optimizer=optimizer, metrics=["accuracy"])
```

```
model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_data=(x_test, y_test))
```

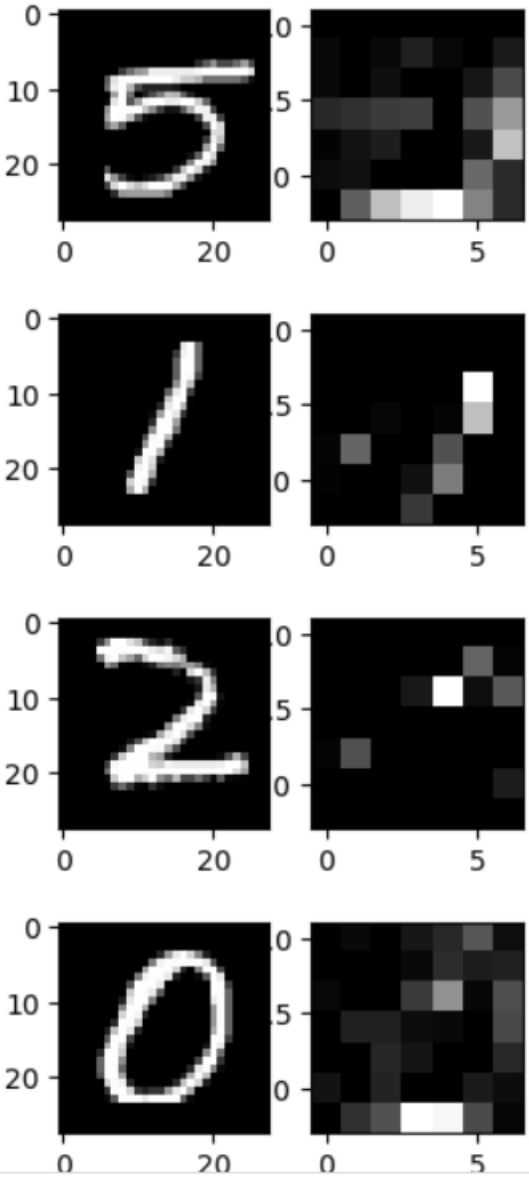
دقت مدل به 99 درصد رسید.

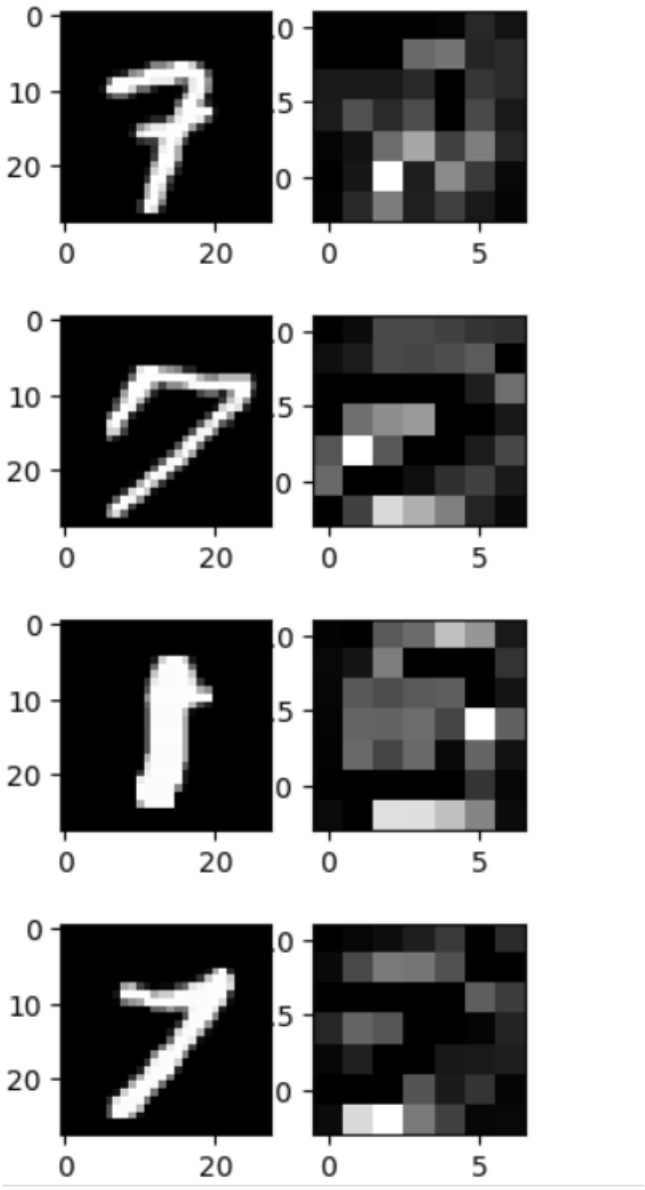
در این قسمت تابع grad-cam را تعریف کردم که با دریافت مدل و لایه، گرادیان خروجی لایه را نسبت به loss محاسبه میکند:

```
3] def get_gradcam(model, img_array, class_index, last_conv_layer_name):
       grad_model = models.Model([model.inputs], [model.get_layer(last_conv_layer_name).output, model.output])

       with tf.GradientTape() as tape:
           conv_output, predictions = grad_model(np.array([img_array]))
           loss = predictions[:, class_index]

       output_grads = tape.gradient(loss, conv_output)[0]
       weights = tf.reduce_mean(output_grads, axis=(0, 1))
       cam = tf.reduce_sum(tf.multiply(weights, conv_output), axis=-1)
       cam = np.maximum(cam, 0)
       cam /= np.max(cam)

       return cam
```

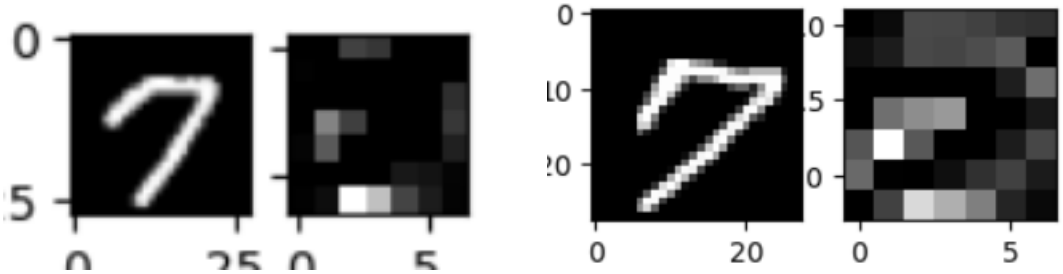
سپس با استفاده از گرادیان مدل، heatmap را محاسبه میکند.

در نهایت، به کمک این تابع، grad-cam را برای ده نمونه اول مدل حساب کردم و چاپ کردم. نتایج به این صورت بود:

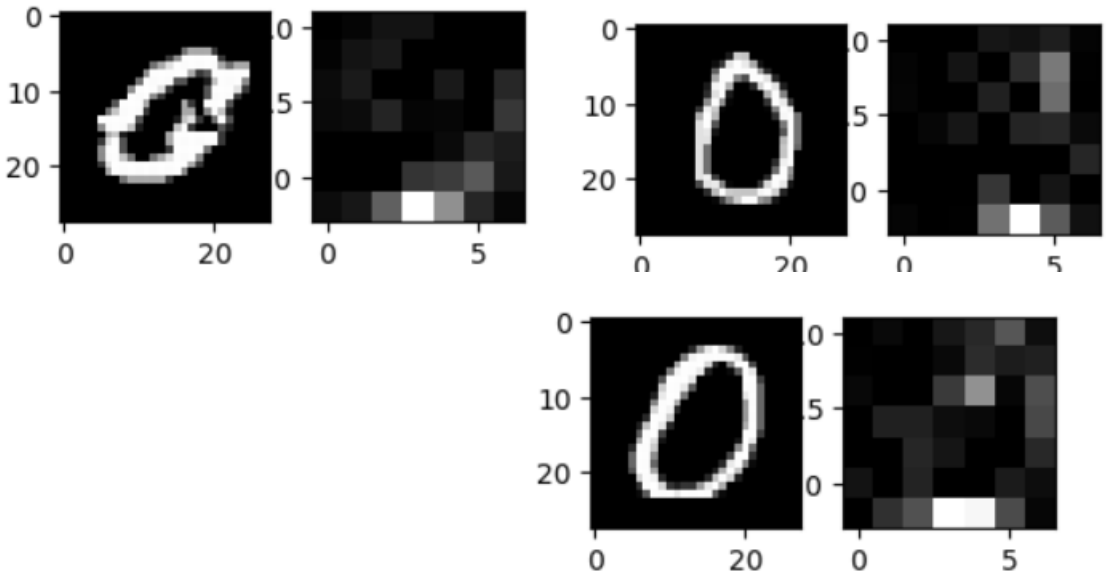

0
10
20
0
20
0
5
0
5
0
10
20
0
20
0
5
0
5
0
10
20
0
20
0
5
0
5
0
10
20
0
20
0
5



از خروجی ها میتوانیم بفهمیم که مدل، تشخیص هر کلاس را با توجه به کدام قسمت های تصویر انجام میدهد. مثلا برای کلاس های 7، اکثر نقشه ها چنین شکلی دارند:



یا مثلا برای کلاس 0 این نمونه ها مشابه همدیگر هستند:



که نشان میدهد مدل از کدام قسمت تصویر عدد 0 را تشخیص میدهد.

به کمک این نقشه ها میتوان برخی ایرادات را در مدل برطرف کرد. مثلا در تصاویری که مدل پیشبینی درستی انجام نمیدهد، به نقشه آن نگاه کرد و متوجه شد که علت پیشبینی اشتباه چیست. مثلا برای یک مدل پیشبینی نژاد حیوانات، مشخص شده بود که تشخیص مدل از حیوان گرگ، از برف پشت سر آن در تصویر است نه خود گرگ. به این صورت میتوان مدل را دیباگ گرد و بهبود داد.

در مورد MNIST میتوان تشخیص داد که مدل از کدام قسمت عدد آن را تشخیص داده. مثلا برای عدد 0 به نظر میرسد که به قوس پایین صفر اهمیت میدهد.

یا برای عدد 7 میتوان فهمید که به قسمت پایین سمت چپ تصویر اهمیت بیشتری میدهد که احتمالا به خاطر این است که عدد 7 در این قسمت با سایر اعداد متفاوت است.