به نام خالق رنگین کمان

گزارش تمرین سری 4 - ستاره باباجانی – 99521109

سوال 1:

الف قسمت اول) KerasTuner یک ابزار برای تنظیم Hyperparameter است که کمک می‌کند مدل‌های Keras را بهبود دهیم.

این ابزار به طور خودکار مقادیر hyperparameter بهتری را برای ما پیدا کرده تا عملکرد مدل را بهینه کند.(همان طور که میدانیم تنظیم Hyperparameter یکی از چالش‌های اصلی در آموزش مدل‌های عمیق است که این ابزار بسیار کمک کننده است.)

الف قسمت دوم) تنظیم hyperparameters در CNNs امری زمان‌بر و دشوار است. این hyperparameters ها میتوانند تعداد لایه‌ها، تعداد نورون‌ها در هر لایه، نرخ یادگیری، اندازه فیلترها و... باشند. این امر نیازمند تجربه و زمان زیاد است. برای تخمین بهترین مقدار ابر پارامتر چند مرحله وجود دارد که در ادامه ذکر کردم:

1. فضای Hyperparameter : hyperparameters ای که می‌خواهیم بهینه‌سازی کنیم. میتواند تعداد لایه‌ها، تعداد نورون‌ها در هر لایه، نرخ یادگیری، و... باشد.

2. فضای جستجو برای هر hyperparameter (مثلاً از 32 تا 256 برای تعداد نورون‌ها).
3. مدل CNN: تعریف یک تابع برای ساخت مدل CNN با hyperparameters دلخواه.
4. استفاده از KerasTuner: ساخت یک تابع برای جستجوی hyperparameters با استفاده از KerasTuner(مانند Hyperband).
5. تنظیم معیاری که می‌خواهیم بهینه‌سازی کنید (مثلاً دقت دسته‌بندی).
6. آموزش بهینه‌یابی: اجرای فرآیند بهینه‌سازی با فراخوانی تابع جستجوی hyperparameter.
7. انتخاب بهترین hyperparameters: دریافت بهترین hyperparameters مدل طبق نتایج جستجو.
8. ساخت مدل با بهترین hyperparameters و آموزش آن.
9. ارزیابی مدل نهایی: ارزیابی عملکرد مدل نهایی روی داده‌های آزمون.

الف قسمت سوم) در KerasTuner، tuner یک شیء از یک کلاس تنظیم‌کننده است که به منظور بهینه‌سازی hyperparameter برای یک مدل مشخص استفاده می‌شود.

هر Tuner یک الگوریتم خاص برای جستجوی hyperparameter دارد.

چند Tuner معروف در KerasTuner شامل موارد زیر میشوند:

1. RandomSearch: یک الگوریتم جستجو تصادفی است. این الگوریتم hyperparameters را به صورت تصادفی انتخاب می‌کند و برای هر ترکیب، مدل را آموزش می‌دهد. این الگوریتم برای برای جستجوی اولیه و اکتشاف فضای hyperparameter مناسب است.
2. BayesianOptimization: یک الگوریتم بهینه‌سازی است که از اطلاعات گذشته برای بهبود جستجوی آینده استفاده می‌کند. این الگوریتم مدل احتمالی برای تابع هدف می‌سازد و بهینه‌ترین نقطه را پیش‌بینی می‌کند.برای جستجوی هوشمند hyperparameter با توجه به اطلاعات گذشته مناسب است.
3. Hyperband: یک الگوریتم جستجوی همگرا است که همزمان چندین مدل را جستجو می‌کند. این الگوریتم مراحل جستجو را به گروه‌های مختلف تقسیم کرده و برای هر گروه، تعداد مدل‌ها را افزایش می‌دهد. این الگوریتم برای جستجوی توسعه‌پذیر و بهینه‌سازی سریع hyperparameter مناسب است.

من در پیاده‌سازی از RandomSearch که در ابزار KerasTuner موجود است، استفاده کردم. علت استفاده آن این است که، الگوریتم Random Search یک روش بهینه‌سازی برای انتخاب پارامترها در مدل‌های یادگیری ماشین است. این الگوریتم برای جستجو در فضای

پارامترها به صورت تصادفی انجام می‌شود، به جای اینکه از روش‌های سنتی و مکرر مانند جستجوی حریصانه (Grid Search) استفاده کند.

ب قسمت اول) مجموعه داده MNIST یکی از مجموعه‌های داده معروف در زمینه یادگیری عمیق و دسته‌بندی تصویر می‌باشد.(60,000 تصویر آموزش و 10,000 تصویر آزمون)

این مجموعه داده شامل تصاویر ارقام دست‌نویس از 0 تا 9 است که به صورت دسته‌بندی شده‌اند همچنین هر تصویر در آن به ابعاد 28*28 پیکسل تبدیل شده است.

ب قسمت دوم) برای استفاده از یک شبکه CNN بهینه‌سازی شده با KerasTuner برای دسته‌بندی تصاویر مجموعه داده MNIST، مراحل زیر باید طی شود:

1. مدلCNN : ایجاد یک مدل CNN با لایه‌های کانولوشن، لایه‌های حاشیه‌نویسی، لایه‌های پردازش تمام متصل و لایه‌های خروجی.
2. فضایHyperparameter : مشخص کردن hyperparametersکه می‌خواهیم بهینه‌سازی کنیم(مثل تعداد لایه‌ها، تعداد نورون‌ها، نرخ یادگیری و... تعیین فضای جستجو برای هرhyperparameter).

3. انتخاب الگوریتم جستجو: انتخاب یک الگوریتم جستجو از میان الگوریتم‌های KerasTuner (مثل Hyperband یا Random Search).
4. تنظیم Tuner : ساخت یک شیء از کلاس تنظیم‌کننده (Tuner) با استفاده از مدل، الگوریتم جستجو، معیار بهینه‌سازی، تعداد مداخلات (trials)و...
5. اجرای جستجو: اجرای فرآیند بهینه‌سازی hyperparameter با استفاده از تنظیم‌کننده .
6. جستجو بر روی داده‌های آموزش با تعداد ایپوک های مشخص.
7. دریافت hyperparameters بهترین مدل.
8. ساخت مدل با hyperparameters بهینه‌یافته و آموزش.
9. ارزیابی مدل نهایی: ارزیابی عملکرد مدل نهایی بر روی داده‌های آزمون.

ب قسمت سوم) استفاده از لایه‌های Pooling و Dropout در شبکه‌های عصبی عمیق می‌تواند بهبود عملکرد و کارایی مدل را تسریع دهد و از مشکلاتی مانند overfitting جلوگیری کند.

- Pooling: یک فرآیند کاهش ابعاد است که در لایه‌های مختلف شبکه‌های عصبی عمیق استفاده می‌شود. انواع مختلفی از آن وجود دارد که مرسوم ترین آن Max Pooling است. در آن، برای هر ناحیه (مثل

یک فیلتر) از ورودی، مقدار بیشینه گرفته می‌شود و به عنوان خروجی استفاده می‌شود.

اهمیت استفاده از Pooling :

1. کاهش ابعاد: استفاده از آن باعث کاهش ابعاد فضایی تصاویر می‌شود و این امر به سرعت آموزش و حافظه مصرفی کمتر کمک می‌کند.
2. استخراج ویژگی‌ها: با کاهش ابعاد، اطلاعات مهم‌تری از تصویر برای مدل قابل دسترس می‌شوند.

- Dropout: یک تکنیک در شبکه‌های عصبی است که در آن، با احتمال خاصی(نرخ Dropout)، برخی از نورون‌ها به طور تصادفی خاموش می‌شوند که این اقدام باعث می‌شود که هر نورون بر اساس مدل‌های مختلف آموزش ببیند و به این ترتیب، overfitting جلوگیری میشود.

اهمیت استفاده از Dropout :

1. جلوگیری از برازش زیاد: به مدل کمک می‌کند در مواجهه با داده‌های جدید و غیردیده شده، عملکرد بهتری داشته باشد.
2. تنوع در آموزش: با غیرفعال‌سازی تصادفی برخی از نورون‌ها در هر مرحله از آموزش، مدل با داده‌های مختلف آشنا می‌شود و تنوع در آموزش بهبود عملکرد مدل را افزایش می یابد.

ج قسمت اول) حال برای طراحی مدل خواسته شده مراحل زیر را طی میکنیم:

- صدا زدن کتابخانه های مورد نیاز:

Needed Libraries

```
[1]  1 !pip install keras-tuner
```

```
Collecting keras-tuner
  Downloading keras_tuner-1.4.6-py3-none-any.whl (128 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 128.9/128.9 kB 1.2 MB/
Requirement already satisfied: keras in /usr/local/lib/python3.10/di
Requirement already satisfied: packaging in /usr/local/lib/python3.1
Requirement already satisfied: requests in /usr/local/lib/python3.10
Collecting kt-legacy (from keras-tuner)
  Downloading kt_legacy-1.0.5-py3-none-any.whl (9.6 kB)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/loca
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/
Installing collected packages: kt-legacy, keras-tuner
Successfully installed keras-tuner-1.4.6 kt-legacy-1.0.5
```

```
 1 import tensorflow as tf
 2 import keras_tuner
 3 import numpy as np
 4 from tensorflow.keras import layers, models
 5 from tensorflow.keras.datasets import mnist
 6 from tensorflow.keras.utils import to_categorical
 7 from kerastuner.tuners import Hyperband
 8 from kerastuner.engine.hyperparameters import HyperParameters
```

```
<ipython-input-2-a25c4756a4d2>:7: DeprecationWarning: `import kerast
  from kerastuner.tuners import Hyperband
```

- ایجاد دیتاست:

Loading Dataset

```
 1 (x, y), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
 2
 3 x_train = x[:-10000]
 4 x_val = x[-10000:]
 5 y_train = y[:-10000]
 6 y_val = y[-10000:]
 7
 8 x_train = np.expand_dims(x_train, -1).astype("float32") / 255.0
 9 x_val = np.expand_dims(x_val, -1).astype("float32") / 255.0
10 x_test = np.expand_dims(x_test, -1).astype("float32") / 255.0
11
12 num_classes = 10
13 y_train = tf.keras.utils.to_categorical(y_train, num_classes)
14 y_val = tf.keras.utils.to_categorical(y_val, num_classes)
15 y_test = tf.keras.utils.to_categorical(y_test, num_classes)
```

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

- طراحی مدل: طبق شرایطی که در سوال ذکر شد، خواهیم داشت:

```python
def build_model(hp):
    model = tf.keras.Sequential() # my model is a sequential one

    # start tuning the number of layers
    for i in range(hp.Int("cnv_number_of_layers", 1, 5)):
        model.add(
            layers.Conv2D(
                # start tuning the number of filters
                filters = hp.Int(f"filters{i}", min_value=32, max_value=256, step=32),
                kernel_size = (3,3),
                activation = 'relu',
                padding = 'same'
            )
        )
        model.add(layers.MaxPooling2D(pool_size=(2, 2)))

    model.add(layers.Flatten())
    for i in range(hp.Int("number_of_layers", 1, 5)):
        model.add(
            layers.Dense(
                # start tuning the number of units
                units = hp.Int(f"units_{i}", min_value=32, max_value=256, step=32),
                activation = hp.Choice("activation", ["relu", "tanh"]),
            )
        )
    if hp.Boolean("dropout"):
        model.add(layers.Dropout(rate=0.25))
    model.add(layers.Dense(10, activation="softmax"))
    learning_rate = hp.Float("lr", min_value=1e-4, max_value=1e-2, sampling="log")
    model.compile(
        optimizer=tf.keras.optimizers.Adam(learning_rate=learning_rate), # using Adam optimizer
        loss = "categorical_crossentropy",
        metrics = ["accuracy"],
    )
    return model
```

- تعریف متغیر tuning:

Tunner Hyperparemeter

```
[5]  1 tuner = keras_tuner.RandomSearch(
     2     hypermodel=build_model,
     3     objective = "val_accuracy",
     4     max_trials =3,
     5     executions_per_trial = 2,
     6     overwrite = True,
     7     directory = "Q1",
     8     project_name = "Q1",
     9 )
```

- شروع سرچ برای بهترین ابر پارامتر:

Start using tunner

```
[6]  1 tuner.search(x_train, y_train, epochs = 3, validation_data=(x_val, y_val))
```

```
Trial 3 Complete [00h 27m 44s]
val_accuracy: 0.9828999936580658

Best val_accuracy So Far: 0.9875500202178955
Total elapsed time: 01h 02m 43s
```

- بهترین مدل:

Best 2 models

```
1 # best 2 models
2 models = tuner.get_best_models(num_models = 2)
3 best_model = models[0]
4
5 # Build model
6 best_model.build(input_shape = x_train.shape)
7 best_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (50000, 28, 28, 32)       320

 max_pooling2d (MaxPooling2  (50000, 14, 14, 32)       0
 D)

 conv2d_1 (Conv2D)           (50000, 14, 14, 256)      73984

 max_pooling2d_1 (MaxPoolin  (50000, 7, 7, 256)        0
 g2D)

 flatten (Flatten)           (50000, 12544)            0

 dense (Dense)               (50000, 96)               1204320

 dense_1 (Dense)             (50000, 224)              21728

 dense_2 (Dense)             (50000, 192)              43200

 dense_3 (Dense)             (50000, 32)               6176

 dropout (Dropout)           (50000, 32)               0

 dense_4 (Dense)             (50000, 10)               330

=================================================================
Total params: 1350058 (5.15 MB)
Trainable params: 1350058 (5.15 MB)
Non-trainable params: 0 (0.00 Byte)
```

- آموزش بر روی بهترین مدل:

Training on the best model

```
[8]   1 best_model.fit(x_train, y_train, epochs = 10, validation_split=0.2)

    Epoch 1/10
    1250/1250 [==============================] - 144s 114ms/step - loss: 0.0352 - accuracy: 0.9902 - val_loss: 0.0305 - val_accuracy: 0.9907
    Epoch 2/10
    1250/1250 [==============================] - 137s 110ms/step - loss: 0.0259 - accuracy: 0.9930 - val_loss: 0.0517 - val_accuracy: 0.9872
    Epoch 3/10
    1250/1250 [==============================] - 135s 108ms/step - loss: 0.0222 - accuracy: 0.9947 - val_loss: 0.0373 - val_accuracy: 0.9905
    Epoch 4/10
    1250/1250 [==============================] - 137s 110ms/step - loss: 0.0173 - accuracy: 0.9956 - val_loss: 0.0352 - val_accuracy: 0.9915
    Epoch 5/10
    1250/1250 [==============================] - 139s 111ms/step - loss: 0.0181 - accuracy: 0.9951 - val_loss: 0.0566 - val_accuracy: 0.9884
    Epoch 6/10
    1250/1250 [==============================] - 137s 110ms/step - loss: 0.0146 - accuracy: 0.9964 - val_loss: 0.0417 - val_accuracy: 0.9915
    Epoch 7/10
    1250/1250 [==============================] - 137s 109ms/step - loss: 0.0129 - accuracy: 0.9966 - val_loss: 0.0400 - val_accuracy: 0.9929
    Epoch 8/10
    1250/1250 [==============================] - 143s 114ms/step - loss: 0.0080 - accuracy: 0.9981 - val_loss: 0.0432 - val_accuracy: 0.9907
    Epoch 9/10
    1250/1250 [==============================] - 139s 111ms/step - loss: 0.0128 - accuracy: 0.9970 - val_loss: 0.0597 - val_accuracy: 0.9915
    Epoch 10/10
    1250/1250 [==============================] - 137s 110ms/step - loss: 0.0094 - accuracy: 0.9977 - val_loss: 0.0603 - val_accuracy: 0.9905
    <keras.src.callbacks.History at 0x7bd0204f9300>
```

- تست بر روی مدل آموزش دیده و شرح نتایج: همان طور که مشاهده میشود دقت مدل روی داده های تست، حدود 99 درصد شد.

Result

```
# evaluation
test_loss, test_accuracy = best_model.evaluate(x_test, y_test)

# test accuracy
print(f'Test accuracy is: {test_accuracy}')
```

```
313/313 [==============================] - 9s 27ms/step - loss: 0.0622 - accuracy: 0.9900
Test accuracy is: 0.9900000095367432
```

ج قسمت دوم) اندازه فیلترها در لایه‌های Convolutional با استفاده از kernel_size که به آن مقدار (3و3) داده شد، مشخص شده است یعنی اندازه هر فیلتر 3 در 3 است. این اندازه متداول است و برای بسیاری از مسائل پردازش تصویر، به خوبی کار می‌کند.

ج قسمت سوم) میدانیم که Dropout یک تکنیک منحصر به فرد در شبکه‌های عصبی است که با حذف تصادفی برخی از نورون‌ها در هر مرحله از آموزش، از برازش زیاد (overfitting) جلوگیری می‌کند. این تکنیک به مدل کمک می‌کند تا اطلاعات بیشتری را از تمرین بگیرد و بهتر به داده‌های جدید و ناشناخته عمل کند.

همچنین لایه‌های Pooling که از طریق میانگین‌گیری یا حداکثرگیری اطلاعات را کاهش می‌دهند، می‌توانند به کاهش ابعاد و تعداد پارامترهای شبکه کمک کنند و در نتیجه از overfitting جلوگیری کنند. این لایه‌ها به مدل کمک می‌کنند تا اطلاعات مهم‌تری از تصویر استخراج شوند و از ابعاد بالای تصاویر جلوگیری شود.

سوال 2: مراحل تکمیل کردن فایل داده شده را توضیح میدهیم:

- دانلود و صدا زدن کتابخانه های مورد نیاز:

Import libraries

First we import the libraries to use them later

```
[58]  1 ! pip install visualkeras
      2 ! pip install gdown tensorflow
```

```
1 !import os
2 import gdown
3 import cv2
4 import time
5 import math
6 import glob
7 import seaborn
8 import visualkeras
9 import numpy as np
10 import pandas as pd
11 from PIL import Image
12 import tensorflow as tf
13 from PIL import ImageFont
14 from tensorflow import keras
15 from google.colab import files
16 import matplotlib.pyplot as plt
17 from prettytable import PrettyTable
18 from keras.callbacks import ModelCheckpoint
19 from tensorflow.keras.utils import to_categorical
20 from sklearn.model_selection import train_test_split
21 from tensorflow.keras.preprocessing.image import ImageDataGenerator
22 from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
23 from sklearn.metrics import confusion_matrix , accuracy_score, classification_report
```

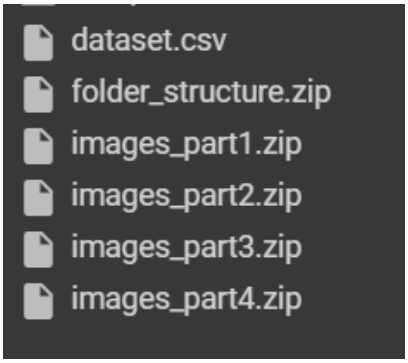
- دانلود دیتاست از url داده شده:

in this section we download the dataset from:
https://figshare.com/articles/dataset/GRAZPEDWRI-DX/14825193

```
[70]  1 # Download the first file
      2 output_file = 'dataset.csv'
      3 gdown.download(f'https://figshare.com/ndownloader/files/35026432', output_file, quiet=False)
```

Downloading

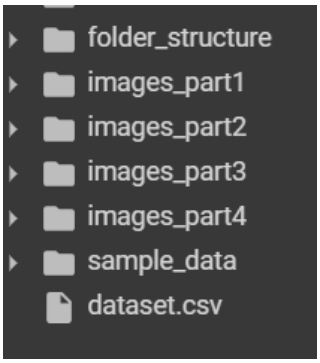
```
[76]  1 # Download the sixth file
      2 output_file = 'images_part4.zip'
      3 gdown.download(f'https://figshare.com/ndownloader/files/34268891', output_file, quiet=False)

     Downloading...
     From: https://figshare.com/ndownloader/files/34268891
     To: /content/images_part4.zip
     100%|██████████| 4.21G/4.21G [02:28<00:00, 28.4MB/s]
     'images_part4.zip'
```



- Unzip کردن فایل های داده شده و حذف فایل های زیپ (بخاطر محدودیت دیسک):

```
1 !unzip folder_structure.zip -d folder_structure
2 !unzip images_part1.zip -d images_part1
3 !unzip images_part2.zip -d images_part2
4 !unzip images_part3.zip -d images_part3
5 !unzip images_part4.zip -d images_part4
```

```
1 os.remove('folder_structure.zip')
2 os.remove('images_part1.zip')
3 os.remove('images_part2.zip')
4 os.remove('images_part3.zip')
5 os.remove('images_part4.zip')
```



- خواندن فایل CSV: طبق چیزهای خواسته شده، تغییرات روی آن دو ستون داده شد و سپس بقیه ستون ها را حذف کردم.

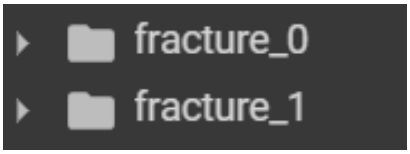
```
# Read the CSV file
file_path = 'dataset.csv'
data = pd.read_csv(file_path)

# Replace 'NaN' values in 'fracture_visible' column with '0'
data['fracture_visible'].fillna('0', inplace=True)

# Add 'png' extension to 'filestem'
data['filestem'] = data['filestem'].apply(lambda x: x+'.png')

# Remove other columns except filestem and fracture_visible
data = data[['filestem', 'fracture_visible']]

# show the shape of the dataframe
print("dataframe shape is: ", data.shape)
data.head(20)
```

خروجی دیتا به صورت زیر شده است:

```
dataframe shape is:  (20327, 2)
```

- برای واضح شدن کار، هر عکس را طبق لیبلی که داشت به یک فولدر جدا منتقل کردم:

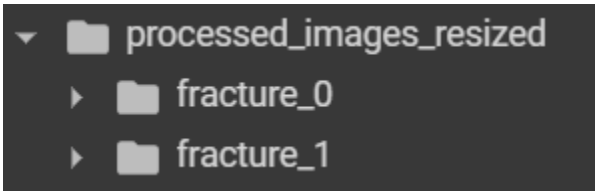
```python
import shutil

# Paths for directories
source_directory = 'dataset'
output_directory_0 = 'fracture_0'  # Directory for images labeled as '0'
output_directory_1 = 'fracture_1'  # Directory for images labeled as '1'

# Create output directories if they don't exist
os.makedirs(output_directory_0, exist_ok=True)
os.makedirs(output_directory_1, exist_ok=True)

# Iterate through each row in the DataFrame
for idx, row in data.iterrows():
    filestem = row['filestem']
    fracture_visible = row['fracture_visible']
    print(fracture_visible)
    source_file = os.path.join(source_directory, filestem)

    if fracture_visible == 1.0:
        destination_directory = output_directory_1
    else:
        destination_directory = output_directory_0

    destination_file = os.path.join(destination_directory, filestem)
    shutil.copy(source_file, destination_file)
    print(f"Moved {filestem} to {destination_directory}")
```



- تغییر سایز عکس ها و تبدیل آنها به RGB:

```python
size = 224
image_dir1 = 'fracture_0'
image_dir2 = 'fracture_1'
output_dir = 'processed_images_resized'

# Create the output directories if they don't exist
output_dir_0 = os.path.join(output_dir, 'fracture_0')
output_dir_1 = os.path.join(output_dir, 'fracture_1')
os.makedirs(output_dir_0, exist_ok=True)
os.makedirs(output_dir_1, exist_ok=True)

def resize_and_save(image_dir, output_directory):
    for filename in os.listdir(image_dir):
        file_path = os.path.join(image_dir, filename)
        if filename.endswith(('jpg', 'png')):
            try:
                # Open the image using PIL
                img = Image.open(file_path)

                # Convert grayscale image to RGB
                if img.mode != 'RGB':
                    img = img.convert('RGB')

                # Resize the image to (224, 224)
                img = img.resize((size, size))
```

```python

                # Convert grayscale image to RGB
                if img.mode != 'RGB':
                    img = img.convert('RGB')

                # Resize the image to (224, 224)
                img = img.resize((size, size))

                # Save the processed image to the corresponding output directory
                output_path = os.path.join(output_directory, filename)
                img.save(output_path)

                print(f"Processed and saved image: {filename}")
            except Exception as e:
                print(f"Error processing image: {filename} - {e}")

# Process and save images from each directory into separate output directories
resize_and_save(image_dir1, output_dir_0)
resize_and_save(image_dir2, output_dir_1)

```



- ساختن data loader:

```
# Creating dataloader
datagen = ImageDataGenerator(rescale=1./255, validation_split=0.2)

# Converting "fracture_visible" column values to strings
data["fracture_visible"] = data["fracture_visible"].astype(str) # for handling error in class_mode="binary"
```

```
train_generator = datagen.flow_from_dataframe(
    dataframe=data,
    directory="processed_images_resized",
    x_col="filestem",
    y_col="fracture_visible",
    subset="training",
    batch_size=64,
    seed=42,
    shuffle=True,
    class_mode="categorical",
    target_size=(size,size))

val_generator = datagen.flow_from_dataframe(
    dataframe=data,
    directory="processed_images_resized",
    x_col="filestem",
    y_col="fracture_visible",
    subset="validation",
    batch_size=64,
    seed=42,
    shuffle=True,
    class_mode="categorical",
    target_size=(size,size))
```

سوال 3: در این سوال R2 میزان نسبت واریانس در متغیر وابسته که از متغیرهای مستقل پیش‌بینی می‌شود را اندازه‌گیری می‌کند. یک R2 منفی نشان‌دهنده این است که مدل تنوع داده‌ها را در نظر نمیگیرد و عملکرد آن ضعیف است.

LSTM ها یا شبکه‌های حافظه کوتاه‌مدت بلندمدت، برای نگهداری و استفاده از اطلاعات در توالی‌های طولانی‌تر با حسابداری وابستگی‌های بلندمدت در داده‌ها طراحی شده‌اند. با این حال، اگر یک LSTM را تنها با ورودی فعلی در هر مرحله زمانی بدون هیچ زمینه تاریخی ارائه دهیم مشکلات زیر پیش می آید:

1. از دست دادن اطلاعات متوالی: میدانیم LSTM ها برای ثبت الگوهای متوالی در داده ها ساخته شده اند. با دادن تنها یک ورودی در هر زمان بدون در نظر گرفتن ورودی های گذشته، شبکه زمینه متوالی حیاتی برای درک الگوها یا وابستگی ها را در طول زمان از دست می دهد.
2. ناتوانی در یادگیری وابستگی های طولانی مدت: LSTM ها در گرفتن وابستگی های دوربرد با حفظ حافظه ورودی های گذشته از طریق حالت سلولی خود برتری می یابند. هنگامی که فقط ورودی‌های فعلی را میدهیم، LSTM نمی‌تواند از حافظه بلندمدت خود به طور موثر استفاده کند، که مانع از

توانایی آن در یادگیری و پیش‌بینی بر اساس دنباله‌های توسعه‌یافته می‌شود.

3. مشکل در گرفتن الگوها: قدرتLSTM ها در توانایی آنها برای یادگیری از دنباله ها نهفته است. بدون داده های تاریخی، شبکه ممکن است برای شناسایی و گرفتن الگوها یا روندهای پیچیده ای که در چندین مرحله زمانی آشکار می شوند، دچار مشکل شود.
4. قابلیت پیش بینی محدود: پیش بینی نتایج آینده معمولاً بر درک الگوهای گذشته متکی است. اگر LSTM به داده های تاریخی دسترسی نداشته باشد، قابلیت پیش بینی آن برای مراحل زمانی آینده محدود یا نادرست خواهد بود.

پیشنهاد: برای استفاده موثر از قدرت LSTM در داده‌های سری زمانی، ارائه ورودی‌های متوالی که شامل اطلاعات گذشته است، ضروری است زیرا به شبکه اجازه می‌دهد تا وابستگی‌های زمانی اساسی در داده‌ها را بیاموزد و مدل‌سازی کند.(یعنی استفاده از پنجره های زمانی به مدل)

سوال 4: الف)

- شبکه های عصبی کانولوشن:(CNN)

CNNها عمدتاً برای تشخیص تصویر، وظایف بینایی کامپیوتری و تجزیه و تحلیل متوالی داده ها استفاده می شوند. آنها در یادگیری سلسله مراتب فضایی ویژگی ها از طریق لایه های کانولوشنی برتر هستند. آن‌ها از فیلترها برای ادغام در داده‌های ورودی، استخراج ویژگی‌ها و حفظ روابط فضایی استفاده می‌کنند. CNN ها به دلیل اشتراک گذاری پارامترها و توانایی گرفتن الگوهای محلی در پردازش داده های شبکه مانند، کارآمد هستند.

- برنامه های کاربردی:

1. طبقه بندی تصویر: تشخیص اشیاء در تصاویر.
2. تشخیص شی: شناسایی و مکان یابی اشیاء درون یک تصویر.
3. بخش بندی: تقسیم یک تصویر به بخش ها برای تجزیه و تحلیل دقیق.
4. تحلیل ویدئو: ردیابی اشیا و درک حرکت در فیلم ها.

- مسائل پیش رو با استفاده از آنها:

1. حافظه متوالی محدود CNNها: فاقد حافظه ذاتی هستند و با پردازش متوالی داده ها مشکل دارند.
2. پیچیدگی در ورودی های با اندازه متغیر: مدیریت اندازه ورودی های متغیر می تواند چالش برانگیز باشد.

- شبکه های عصبی بازگشتی:(RNN)

RNNها برای داده های متوالی طراحی شده اند و می توانند ورودی های با طول متغیر را پردازش کنند و در عین حال حافظه را در طول زمان حفظ کنند. آنها از حلقه های بازخورد برای تداوم اطلاعات استفاده می کنند و آنها را برای کارهایی که شامل وابستگی های متوالی و داده های زمانی هستند، استفاده میکنند. RNNها می توانند زمینه را در توالی های طولانی تر در مقایسه با شبکه های پیشخور حفظ کنند.

- برنامه های کاربردی:

1. پردازش زبان طبیعی: ترجمه زبان، تولید متن، تجزیه و تحلیل احساسات.
2. تحلیل سری زمانی: پیش بینی قیمت سهام، پیش بینی آب و هوا.
3. تشخیص گفتار: تبدیل زبان گفتاری به متن.

4. تحلیل ویدئو: تشخیص اقدام و زیرنویس ویدئو.

- مسائل پیش رو با استفاده از آنها:

1. مشکل گرادیان ناپدید شدن/انفجار: RNNها به دلیل مشکل گرادیان ناپدید/انفجار ممکن است مشکلاتی در یادگیری از اطلاعات گذشته دور داشته باشند.
2. مشکل در گرفتن وابستگی های بلندمدت: علیرغم داشتن حافظه، RNNها برای جذب موثر وابستگی های دوربرد تلاش می کنند.

-CNNها در یادگیری فضایی، عالی هستند: برای کارهایی که شامل داده‌های شبکه‌ای هستند، در جایی که روابط فضایی مهم هستند، مانند پردازش تصویر، ایده‌آل هستند.

-RNNها برای داده های متوالی بهتر هستند: آنها برای کارهایی که نیاز به حافظه یا روابط زمانی دارند، مانند پردازش زبان طبیعی و تجزیه و تحلیل سری های زمانی، مناسب هستند.

-مدل‌های ترکیبی مانند شبکه‌های عصبی کانولوشنال-عودکننده (CRNN) نقاط قوت هر دو معماری را ترکیب می‌کنند. به عنوان مثال، CRNNها در کارهایی مانند نوشتن شرح تصاویر استفاده می شوند که در آن CNN ها

ویژگی های تصویر را استخراج می کنند، و RNN ها شرح های توصیفی تولید می کنند و از نقاط قوت هر دو معماری استفاده می کنند.

ب)

- تعداد پارامترها:

-CNNها معمولاً به دلیل وجود لایه های کانولوشنال و لایه های کاملاً متصل دارای تعداد زیادی پارامتر هستند. تعداد پارامترها در CNNها به عواملی مانند اندازه ورودی، عمق شبکه و تعداد فیلترها در هر لایه بستگی دارد.

-RNNها معمولاً پارامترهای کمتری در مقایسه با CNN ها دارند. پارامترها در RNN ها عمدتاً با اتصالات مکرر مرتبط هستند و تعداد پارامترها تحت تأثیر معماری شبکه مانند تعداد واحدهای بازگشتی (سلول ها) و اندازه حالت پنهان است.

- امکان موازی سازی:

-CNNها: عملیات کانولوشن در CNN ها را می توان تا حد قابل توجهی موازی کرد، به خصوص در کانال های مختلف و مکان های فضایی درون یک تصویر. این موازی سازی به دلیل ماهیت مستقل کانولوشن ها در بخش های مختلف ورودی است که امکان پردازش کارآمد در چندین هسته یا GPU را فراهم می کند.

-RNNها: به دلیل ماهیت متوالی خود چالش هایی را در موازی سازی ایجاد می کنند. RNNها توالی ها را مرحله به مرحله پردازش می کنند، جایی که هر مرحله به خروجی مرحله قبل بستگی دارد و محاسبات موازی را محدود می کند. با این حال، در یک توالی واحد، عملیات خاصی را می توان تا حدی موازی کرد، مانند عملیات عنصری یا پردازش دسته ای در توالی های مختلف.

سوال 5: الف) فرمول های استفاده شده برای محاسبه خروجی و تعداد پارامتر ها به شرح زیر است:

1. Conv:

$$output = \frac{inputSize - kernelSize + 2 * padding}{stride} + 1$$

$$NumParameters = (FilterSize * FilterSize * InputDepth + 1) * NumFilters$$

2. Dilated-Conv:

$$output = \frac{inputSize + 2 * padding - delationrate(kernel_{size} - 1) - 1}{stride} + 1$$

$$NumParameters = (FilterSize * FilterSize * InputDepth + 1) * NumFilters$$

3. Maxpool:

$$output = \frac{inputSize - poolingSize}{stride} + 1$$

$$NumParameters = 0$$

Layers	Activation volume dimensions	Number of parameters
input	(256, 256, 3)	0
Conv3-64	(256, 256, 64)	64 × (3 × 3 × 3 + 1)
Dilated-Conv5-32	(248, 248, 32)	32 × (5 × 5 × 64 + 1)
Maxpool2	(124, 124, 32)	0
Conv3-128	(124, 124, 128)	128 × (3 × 3 × 32 + 1)
Dilated-Conv5-64	(108, 108, 64)	64 × (5 × 5 × 128 + 1)
Maxpool2	(54, 54, 64)	0
Conv3-256	(54, 54, 256)	256 × (3 × 3 × 64 + 1)
Dilated-Conv5-128	(22, 22, 128)	128 × (5 × 5 × 256 + 1)
Maxpool2	(11, 11, 128)	0

ب) میدانیم:

$$output = \frac{inputSize - kernelSize + 2 * padding}{stride} + 1$$

حال اگر فرض کنیم که stride = 1 , input = output، خواهیم داشت:

$$padding = \frac{kernelSize - 1}{2}$$

سوال 6: الف)

- نادرست: نرمال سازی دسته ای تکنیکی است که در شبکه های عصبی عمیق برای بهبود ثبات و سرعت تمرین استفاده می شود. ورودی‌های هر

لایه را در یک دسته کوچک نرمال می‌کند تا میانگین و واریانس واحد صفر داشته باشد، که به کاهش مشکل تغییر متغیر داخلی کمک می‌کند.

همچنین به طور خاص سرعت پردازش یک دسته را افزایش نمی دهد. در عوض، با پایدارتر و سریع‌تر کردن آن به کل فرآیند تمرین کمک می‌کند. این کار را با اجازه دادن به استفاده از نرخ های یادگیری بالاتر انجام می دهد، که می تواند همگرایی را تسریع کند. علاوه بر این، نرمال سازی دسته ای اثر منظم سازی دارد.

نرمال سازی دسته ای به طور مستقیم تعداد به روز رسانی ها را کنترل نمی کند. بلکه با ارائه گرادیان های پایدارتر به شبکه کمک می کند تا سریعتر همگرا شود. تعداد به‌روزرسانی‌ها تحت تأثیر عواملی مانند نرخ یادگیری، اندازه دسته و معماری کلی شبکه عصبی است.

- درست: نرمال‌سازی دسته ای در شبکه‌های عصبی به این صورت عمل می‌کند که خروجی لایه را نرمال می‌کند یعنی مراحل مختلفی از خروجی لایه را به گونه‌ای تبدیل می‌کند که توزیع آن‌ها نرمال(با میانگین صفر و انحراف معیار یک) باشد.

این کار باعث می‌شود که توزیع مقادیر خروجی لایه در هر دسته داده نرمال شده و یکنواخت‌تر باشد. این موضوع می‌تواند کمک کننده باشد تا مشکلات مربوط به scale and shift ای که در آموزش شبکه پیدا میشوند، بهبود یابد و فرآیند آموزش سریعتر شود.

- نادرست: نرمال‌سازی دسته‌ای از شبکه در طول آموزش، برای هر دسته داده، مقادیر ویژگی‌های ورودی را نرمال میکند(میانگین صفر و واریانس یک).

  این به کاهش مشکل گرادیان گرفتگی کمک می‌کند و به طور مستقیم وزن‌ها را تغییر نمی‌دهد و هدف اصلی آن اصلاح توزیع ویژگی‌های لایه‌هاست.

ب) با استفاده از توابع واریانس و میانگین کتابخانه numpy قسمت های خواسته شده را تکمیل کردم:

```
if mode == 'train':
    # TODO: Mean of Z across first dimension
    mu = np.mean(Z, axis=0)

    # TODO: Variance of Z across first dimension
    var = np.var(Z, axis=0)

    # Take moving average for cache_dict['mu']
    cache_dict['mu'] = beta_avg * cache_dict['mu'] + (1-beta_avg) * mu

    # Take moving average for cache_dict['var']
    cache_dict['var'] = beta_avg * cache_dict['var'] + (1-beta_avg) * var

elif mode == 'test':
    # TODO: Load moving average of mu
    mu = cache_dict['mu']

    # TODO: Load moving average of var
    var =cache_dict['var']
```

سپس طبق فرمولی که در جزوه بود و به شرح زیر است، z_norm را تعریف کردم:

$$y = \gamma \frac{x - \mu}{\sigma} + \beta$$

```
# TODO: Apply z_norm transformation
Z_norm = (Z - mu) / np.sqrt(var + eps)

# TODO: Apply gamma and beta transformation to get Z tiled
out = gamma * Z_norm + beta
```

ج) در فرمول نرمال‌سازی دسته‌ای، ϵ یک پارامتر آزاد است که به جلوگیری از تقسیم بر صفر در صورتی که انحراف معیار برابر با صفر باشد، می‌پردازد.

د) مشکلات استفاده از نرمال سازی دسته ای با اندازه یک به شرح زیر است:

- عدم بهره‌مندی از میانگین دسته‌ای: یکی از مزایای نرمال‌سازی دسته‌ای، استفاده از میانگین و انحراف معیار دسته است که اطلاعات مهمی را از دسته‌های مختلف در طول آموزش انتقال می‌دهد که اگر اندازه دسته یک باشد، این موضوع سلب میشود.
- Training Instability: استفاده از دسته با اندازه یک می‌تواند باعث ناپایداری در آموزش شبکه شود زیرا گرادیان‌ها در این حالت تنها بر اساس یک نمونه محاسبه می‌شوند.
- ناپایداری در تخمین میانگین و انحراف معیار: در حین آموزش با دسته‌های بزرگتر، تخمین میانگین و انحراف معیار مستقل از تغییرات کوچک در دسته‌ها انجام می‌شود. اما در دسته با اندازه یک، تخمین این مقادیر ممکن است به دلیل نوسانات زیاد مقادیر نمونه‌ها ناپایدار باشد.

- سختی بهینه‌سازی: بهینه‌سازها استفاده از دسته با اندازه یک، ممکن است چون گرادیان‌ها تنها بر اساس یک نمونه محاسبه میشوند، در بهینه‌سازی کارایی کمی داشته باشند.

ه) فرمول برای محاسبه به شرح زیر است:

- وزن ها = تعداد ورودی ها * تعداد نورون های لایه
- بایاس = تعداد نورون ها

خواهیم داشت:

1. بدون اضافه کردن batch normalization:

10*20 + 20 = 220

2. با اضافه کردن batch normalization، میدانیم هر خروجی 2 پارمتر دیگر نیز خواهد داشت پس:

220 + 2 * 20 = 260

پس تعداد پارامتر های قابل آموزش در کل، 260 است.

سوال 7: مراحل خواسته شده به شرح زیر است:

- صدا زدن کتابخانه های مورد نیاز:

Needed Libraries

```
[13]  1 import numpy as np
      2 import matplotlib.pyplot as plt
      3 from keras.models import load_model
      4 from keras.datasets import mnist
      5 from keras import backend as K
      6 from sklearn.utils import shuffle
      7 from keras.utils import to_categorical
      8 from keras.models import Sequential
      9 from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
     10 import tensorflow as tf
     11 from skimage.transform import resize
```

- لود کردن دیتا از MNIST:

Loading Dataset

```
[14]  1 (x_train, y_train), (x_test, y_test) = mnist.load_data()
```

- پیش پردازش(شافل کردن دیتا):

Preprocessing

```
 1 # shuffling out data
 2 train_data = list(zip(x_train, y_train))
 3 shuffled_data = shuffle(train_data)
 4
 5 # Unziping the shuffled data
 6 x_train, y_train = zip(*shuffled_data)
 7
 8 # Converting to NumPy arrays
 9 x_train = np.array(x_train)
10 y_train = np.array(y_train)
11
12 print("Dimension of x_train is:", x_train.shape)
13 print("Dimension of y_train is:", y_train.shape)
14 print("Dimension of x_test is:", x_test.shape)
15 print("Dimension of y_test is:", y_test.shape)
```

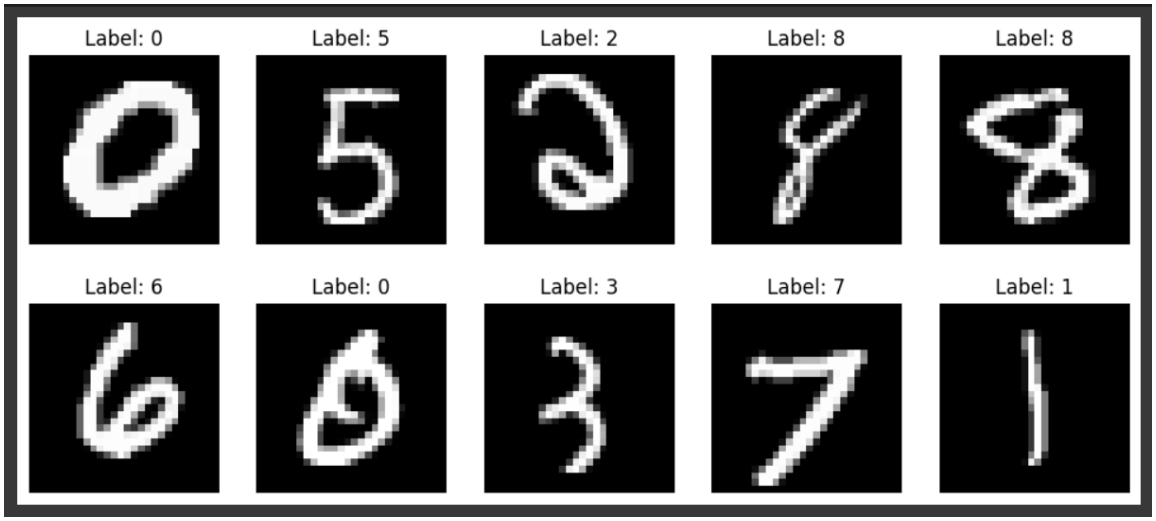
برای درک بهتر از اندازه داده ها، آن ها را پرینت کردم که به شرح زیر هستند:

```
Dimension of x_train is: (60000, 28, 28)
Dimension of y_train is: (60000,)
Dimension of x_test is: (10000, 28, 28)
Dimension of y_test is: (10000,)
```

- رسم 10 عکس از مجموعه داده های آموزشی:

Ploting some of training data

```
1 plt.figure(figsize=(12, 5))
2 for i in range(10):
3     plt.subplot(2, 5, i + 1)
4     plt.imshow(x_train[i].squeeze(), cmap='gray')
5     plt.title(f"Label: {y_train[i]}")
6     plt.axis('off')
7
8 plt.show()
```



- تغییر مقادیر پیکسل ها و دادن برچسب categorical:

Normalizing and Categoricalization

```
[17]  1 x_train = x_train / 255.0
      2 x_test = x_test / 255.0
      3
      4 num_classes = 10  # we have 10 classes
      5 y_train = to_categorical(y_train, num_classes)
      6 y_test = to_categorical(y_test, num_classes)
```

- طراحی مدل خواسته شده:

Implementing the model

```
[18]  1 model = Sequential()
      2
      3 # layer 1
      4 model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', padding='same', input_shape=(28, 28, 1)))
      5 model.add(MaxPooling2D(pool_size=(2, 2)))
      6
      7 # layer 2
      8 model.add(Conv2D(64, kernel_size=(3, 3), activation='relu', padding='same'))
      9 model.add(MaxPooling2D(pool_size=(2, 2)))
     10
     11 # layer 3
     12 model.add(Conv2D(64, kernel_size=(3, 3), activation='relu', padding='same', name='last_conv_layer'))
     13 model.add(MaxPooling2D(pool_size=(2, 2)))
     14
     15 # Flatten layer
     16 model.add(Flatten())
     17
     18 # FC layer 1
     19 model.add(Dense(128, activation='relu'))
     20
     21 # Output layer
     22 model.add(Dense(10, activation='softmax'))
```

- کامپایل کردن مدل طراحی شده:

Compiling

```
[19]  1 model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

- آموزش مدل و ارزیابی:

Training and Testing

```
[20]  1 history = model.fit(x_train, y_train, epochs=15, batch_size=64, validation_data=(x_test, y_test))
```

```
Epoch 1/15
938/938 [==============================] - 147s 154ms/step - loss: 0.1597 - accuracy: 0.9503 - val_loss: 0.0640 - val_accuracy: 0.9789
Epoch 2/15
938/938 [==============================] - 88s 94ms/step - loss: 0.0452 - accuracy: 0.9860 - val_loss: 0.0357 - val_accuracy: 0.9887
Epoch 3/15
938/938 [==============================] - 87s 92ms/step - loss: 0.0320 - accuracy: 0.9898 - val_loss: 0.0450 - val_accuracy: 0.9863
Epoch 4/15
938/938 [==============================] - 88s 94ms/step - loss: 0.0254 - accuracy: 0.9920 - val_loss: 0.0308 - val_accuracy: 0.9901
Epoch 5/15
938/938 [==============================] - 88s 94ms/step - loss: 0.0202 - accuracy: 0.9933 - val_loss: 0.0254 - val_accuracy: 0.9916
Epoch 6/15
938/938 [==============================] - 88s 94ms/step - loss: 0.0170 - accuracy: 0.9944 - val_loss: 0.0236 - val_accuracy: 0.9917
Epoch 7/15
938/938 [==============================] - 92s 98ms/step - loss: 0.0146 - accuracy: 0.9953 - val_loss: 0.0266 - val_accuracy: 0.9933
Epoch 8/15
938/938 [==============================] - 89s 94ms/step - loss: 0.0126 - accuracy: 0.9957 - val_loss: 0.0326 - val_accuracy: 0.9916
Epoch 9/15
938/938 [==============================] - 88s 94ms/step - loss: 0.0103 - accuracy: 0.9966 - val_loss: 0.0294 - val_accuracy: 0.9912
Epoch 10/15
938/938 [==============================] - 87s 93ms/step - loss: 0.0077 - accuracy: 0.9975 - val_loss: 0.0383 - val_accuracy: 0.9906
Epoch 11/15
938/938 [==============================] - 87s 92ms/step - loss: 0.0095 - accuracy: 0.9965 - val_loss: 0.0305 - val_accuracy: 0.9923
Epoch 12/15
938/938 [==============================] - 89s 95ms/step - loss: 0.0061 - accuracy: 0.9980 - val_loss: 0.0452 - val_accuracy: 0.9894
Epoch 13/15
938/938 [==============================] - 87s 92ms/step - loss: 0.0071 - accuracy: 0.9978 - val_loss: 0.0262 - val_accuracy: 0.9931
Epoch 14/15
938/938 [==============================] - 86s 92ms/step - loss: 0.0059 - accuracy: 0.9980 - val_loss: 0.0322 - val_accuracy: 0.9913
Epoch 15/15
938/938 [==============================] - 87s 93ms/step - loss: 0.0056 - accuracy: 0.9983 - val_loss: 0.0300 - val_accuracy: 0.9927
```

- الگوریتم Grad-Cam:

Grad-Cam

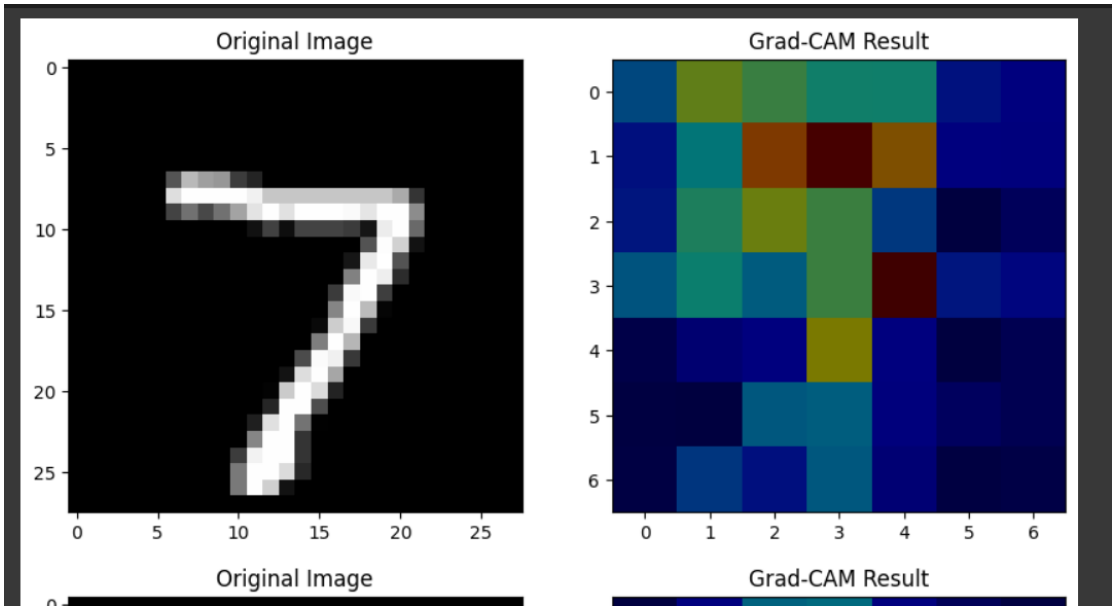
```
[21]  1 def grad_cam(img):
      2   last_conv_layer = model.get_layer('last_conv_layer')
      3   heatmap_model = tf.keras.models.Model(inputs=model.inputs, outputs=[last_conv_layer.output, model.output])
      4
      5
      6   with tf.GradientTape() as tape:
      7       conv_outputs, predictions = heatmap_model(x_test[:10].reshape(-1, 28, 28, 1))
      8       loss = predictions[:, tf.argmax(predictions[0])]
      9       grads = tape.gradient(loss, conv_outputs)
     10       gradients_pool = K.mean(grads, axis = (0, 1, 2))
     11   heatmap = tf.reduce_mean(tf.multiply(gradients_pool, conv_outputs), axis=-1)
     12   heatmap = np.maximum(heatmap, 0)
     13   heatmap /= np.max(heatmap)
     14   return heatmap
```

- نتیجه استفاده از الگوریتم:

Illustrating the result

```python
for i in range(10): # for 10 images
    img = x_test[i]
    plt.figure(figsize=(10, 5))

    # Original
    plt.subplot(1, 2, 1)
    plt.imshow(img.squeeze(), cmap='gray')
    plt.title('Original Image')

    # Grad-CAM
    plt.subplot(1, 2, 2)
    plt.imshow(img.squeeze(), cmap='gray')
    plt.imshow(grad_cam(img)[i], cmap='jet', alpha=0.5)
    plt.title('Grad-CAM Result')

    plt.show()
```

یک نمونه خروجی heat map آن به این صورت است:



Grad-CAM که مخفف Gradient-weighted Class Activation Mapping است، یک تکنیک تشخیص ویژگی برای توضیح این که مدل یک تصویر را با توجه به یک کلاس خاص چگونه تشخیص داده است، است.

- ناحیه‌های فعال سازی در این الگوریتم: این الگوریتم به ما اطلاع می‌دهد که مدل کدام ناحیه‌ها را برای تصمیم‌گیری در مورد یک کلاس خاص به حساب می‌آورد.(به اسم نواحی فعال سازی)در مسئله گفته شده، Grad-CAM نشان میدهد که مدل به چه بخش‌های خاصی از تصویر توجه دارد، تا با استفاده از آن، این ناحیه‌ها را به عنوان ویژگی‌های مهم برای تصمیم‌گیری در مورد این ارقام در نظر بگیریم.
- توجه به اهمیت ویژگی‌ها: با استفاده از این الگوریتم می‌توان اهمیت ویژگی‌های مختلف در تصویر را ارزیابی کرد.(برخی از نواحی با ویژگی‌های مهم‌تری برای تصمیم‌گیری مدل ممکن است تطابق داشته باشند).
- بررسی دقت مدل: اگر مدل در تصمیم‌گیری در مورد یک تصویر خاص دچار خطا شده باشد، ممکن است با نگاه به نواحی فعال سازی متوجه شویم که چرا مدل به اشتباه کرده است.

- تمرکز مدل: این الگوریتم می‌تواند کمک کند تا ما تمرکز مدل را درک کنیم و بفهمیم که آیا مدل به ویژگی‌های مهم و معنادار توجه دارد یا خیر.

همانطور که در تصاویر قابل رویت است لایه آخر به بخش‌های خاصی از تصاویر هر کلاس توجه بیشتری دارد. این به این معناست که الگوریتم به مدل کمک می‌کند تا به راحتی بر اساس heatmap، دسته‌بندی انجام دهد.

پایان