

## به نام خالق رنگین کمان

### ستاره باباجانی – گزارش تمرین سری 6

سوال 1:

(الف)

layer	Output shape	Number of parameters
Input	(512, 512, 3)	0
Conv2D(32, (9, 9), strides=2, padding='same')	(256, 256, 32)	7,808 ( $9*9*3*32 + 32$ )
MaxPooling2D((4, 4), strides=4)	(64, 64, 32)	0
Conv2D(64, (5, 5), strides=1)	(60, 60, 64)	51,264 ( $5*5*32*64 + 64$ )
AveragePooling2D((2, 2), strides=2)	(30, 30, 64)	0
Conv2D(128, (3, 3), strides=1, padding='valid')	(28, 28, 128)	73,856 ( $3*3*64*128 + 128$ )
Conv2D(128, (3, 3), strides=1, padding='same')	(28, 28, 128)	147,584 ( $3*3*128*128 + 128$ )
MaxPooling2D((2, 2), strides=2)	(14, 14, 128)	0
Conv2D(512, (3, 3), strides=1, padding='valid')	(12, 12, 512)	590,336 ( $3*3*128*512 + 512$ )
GlobalAveragePooling2D()	(512)	0
Dense(1024)	(1024)	525,312 ( $512*1024 + 1024$ )
Dense(10)	(10)	10,250 ( $1024*10 + 10$ )

(ب)

Layer	Multiplications	Additions
Conv2D(32, (9, 9), strides=2, padding='same')	$256*256*32*3*9*9$	$256*256*32*(3*81 - 1)$
MaxPooling2D((4, 4), strides=4)	0	$64*64*32*15$
Conv2D(64, (5, 5), strides=1)	$60*60*32*64*5*5$	$60*60*64*(32*25 - 1)$
AveragePooling2D((2, 2), strides=2)	0	$30*30*64*3$
Conv2D(128, (3, 3), strides=1, padding='valid')	$28*28*64*128*3*3$	$28*28*128*(64*9 - 1)$
Conv2D(128, (3, 3), strides=1, padding='same')	$28*28*128*128*3*3$	$28*28*128*(128*9 - 1)$
MaxPooling2D((2, 2), strides=2)	0	$14*14*128*3$
Conv2D(512, (3, 3), strides=1, padding='valid')	$12*12*128*512*3*3$	$12*12*512*(128*9 - 1)$
GlobalAveragePooling2D()	0	$512*(12*12 - 1)$
Dense(1024)	$512*1024$	$512*1024 + 1024$
Dense(10)	$1024*10$	$1024*10 + 10$

(ج) تعداد پارامترهای سه لایه آخر به شرح زیر خواهد شد:

**Flatten():** 0 parameters

**Dense(1024):**  $73,728 \times 1024 + 1024 = 75498496$  parameters

**Dense(10):**  $1024 \times 10 + 10 = 10250$  parameters

- Total count of parameters with Flatten: 76379594
- Total count of parameters with Flatten: 1406410

که نسبت آنها 54.31 میشود.

سوال 2: ابتدا مشتق  $L$  نسبت به  $X$  را باید بدست آوریم که برابر با  $2X - 10$  میشود.  
 پس مقدار گرادیان با در نظر گرفتن مقدار 20 برای  $X$ ، 30 میشود.  
 همانطور که میدانیم فرمول آپدیت  $X$  طبق GD به شرح زیر است:

$$X_{new} = X_{old} - \eta * dL/dX$$

حال به بررسی هر کدام از مقادیر  $X$  جدید داده شده میپردازیم:

1.  $X = 14 \Rightarrow 14 = 20 - \eta * 30 \Rightarrow \eta = 0.2$
2.  $X = 19.4 \Rightarrow 19.4 = 20 - \eta * 30 \Rightarrow \eta = 0.02$
3.  $X = 19.94 \Rightarrow 19.94 = 20 - \eta * 30 \Rightarrow \eta = 0.002$

همانطور که میدانیم هرچه نرخ یادگیری بیشتر باشد، مقدار ضرر سریعتر کم میشود.  
 پس اولین گزینه مربوط به نمودار c، دومی مربوط به b و سومی مربوط به a میباشد.

سوال 3: اگر برای همه ی بلوک ها پدینگ یکسان در نظر بگیریم، خواهیم داشت:

$$\text{Conv } 1 \times 1 \Rightarrow (12, 12, 64)$$

$$\text{Conv } 1 \times 1 \text{ and Conv } 3 \times 3 \Rightarrow (12, 12, 32)$$

$$\text{Conv } 1 \times 1 \text{ and Conv } 5 \times 5 \Rightarrow (12, 12, 128)$$

$$\text{max-pool } 3 \times 3 \text{ and Conv } 1 \times 1 \Rightarrow (12, 12, 64)$$

با concat کردن آنها، ابعاد خروجی (12, 12, 288) میشود.

اگر فیلتر بلوک کانولوشن انتخابی را تغییر دهیم، خروجی متفاوت نخواهد بود زیرا بعد آن بلوک، ما بلوک کانولوشن دیگری داریم که باعث میشود خروجی همان شود.

سوال 4: الف) برای تعیین اینکه وزن‌های شبکه چند بار در طول آموزش به‌روزرسانی می‌شوند، باید تعداد کل نمونه‌های آموزشی ( $t$ )، تعداد دوره‌ها ( $e$ ) و اندازه دسته ( $b$ ) را در نظر بگیریم.

تعداد دفعاتی که وزن شبکه به روز می‌شود را می‌توان به صورت زیر محاسبه کرد:

- تعداد دسته در هر دوره: این تعداد کل نمونه‌های آموزشی ( $t$ ) تقسیم بر اندازه دسته ( $b$ ) است:

$$\text{Number of batches per epoch} = \left\lceil \frac{t}{b} \right\rceil$$

- تعداد کل به روز رسانی‌ها: این تعداد دسته در هر دوره ضرب در تعداد دوره‌ها ( $e$ ) است:

$$\text{Total number of updates} = e \times \left\lceil \frac{t}{b} \right\rceil$$

بنابراین، فرمول محاسبه تعداد کل به روز رسانی وزن شبکه در طول آموزش به صورت زیر است:

$$\text{Total number of updates} = e \times \left\lceil \frac{t}{b} \right\rceil$$

ب) نمودار داده شده مربوط به mini-batch GD می‌باشد زیرا که در این روش برخلاف batch GD کل مجموعه داده به batch‌های کوچکتر تقسیم می‌شود و برای هر کدام از آن batch‌ها، آپدیت جدا داریم. (علت نویزی شدن loss‌ها هم همین است.)

ج) برای نمودار A، به دلیل تفاوت زیاد ضرر در training و validation (ضرر در validation خیلی بیشتر است)، overfitting رخ داده است که برای حل آن از داده‌افزایی و کاهش تعداد ویژگی‌های ورودی استفاده میکنیم تا پیچیدگی مدل را کمتر کنیم (با افزایش لایه‌های شبکه، مدل پیچیده‌تر میشود و نتیجه بدتر میشود). برای نمودار B نیز، ضرر validation و training شبیه به هم است و مدل تعمیم پذیری نسبتاً خوبی را از خود نشان داده است، پس با افزودن لایه‌های شبکه، مدل میتواند پیچیدگی‌های بیشتری را یاد بگیرد و خروجی بهتری داشته باشد. (در اینجا استفاده از داده‌افزایی و کاهش تعداد ویژگی‌های ورودی ممکن است فایده نداشته باشد).

سوال 5: الف) برای هر پیکسل غیر صفر باید همسایگان 8-connectivity آن را در نظر بگیریم. سپس، هر همسایه باید با پیکسل مرکزی مقایسه شود تا بررسی شود که بزرگتر یا مساوی آن است، که اگر بود 1 میشود و در غیراینصورت 0 میشود. سپس یک کد باینری از جهت عقربه‌های ساعت میسازیم:

(1, 1): 00000000

(1, 2): 00000001

(1, 3): 00000111

(2, 1): 01100100

(2, 2): 11000011

(2, 3): 10000001

(3, 1): 00000000

(3, 2): 11110001

(3, 3): 11000000

ب) اگر متغیر C بزرگتر از 0 باشد، چه در حالت ضرب و چه جمع، تغییری در کد ایجاد نمیشود.

ج)

- هیستوگرام A با تصویر سوم مطابقت دارد زیرا تصویر از تعداد زیادی سیاه و سفید تشکیل شده است و در خود هیستوگرام نیز توزیع نسبتاً یکنواخت‌تری وجود دارد.
  - هیستوگرام B با تصویر اول مطابقت دارد زیرا اگر بخواهیم تصویر را به LBP تبدیل کنیم، از بسیاری از پیکسل‌های سفید (سیاه‌های موجود در تصویر) و بسیاری از پیکسل‌های خاکستری تشکیل می‌شود.
  - هیستوگرام C با تصویر دوم مطابقت دارد زیرا تصویر LBP از تعداد زیادی پیکسل سفید و کمی سیاه یا خاکستری تشکیل شده است.
- سوال 6: الف) مراحل ساخت و آموزش CNN به شرح زیر است:
- صدا زدن کتابخانه‌های مورد نیاز و load کردن دیتا:

```
1 import tensorflow as tf
2 import tensorflow_datasets as tfds
3 from tensorflow.keras.preprocessing.image import ImageDataGenerator
4 from tensorflow.keras import layers
5 import numpy as np
6 from tensorflow.keras.callbacks import EarlyStopping
7
8 # Load the Dogs vs. Cats dataset
9 train_dataset, info = tfds.load('cats_vs_dogs', split='train[:80%]', with_info=True, as_supervised=True)
10 test_dataset = tfds.load('cats_vs_dogs', split='train[80%:]', with_info=False, as_supervised=True)
```

• Preprocessing:

```

1 # Set constants
2 IMG_SIZE = 150 # Resize images to 150x150
3 BATCH_SIZE = 32
4
5 def preprocess_image(image, label):
6     # Normalize the pixel values to the range [0, 1]
7     image = tf.cast(image, tf.float32) / 255.0
8     # Resize the image to IMG_SIZE x IMG_SIZE
9     image = tf.image.resize(image, [IMG_SIZE, IMG_SIZE])
10    return image, label
11
12 def prepare_dataset(dataset):
13    dataset = dataset.map(preprocess_image, num_parallel_calls=tf.data.experimental.AUTOTUNE)
14    dataset = dataset.shuffle(buffer_size=1000)
15    dataset = dataset.batch(BATCH_SIZE)
16    return dataset
17
18 # Prepare the datasets
19 train_dataset = prepare_dataset(train_dataset)
20 test_dataset = prepare_dataset(test_dataset)

```

• تعريف مدل:

```

5 def create_model():
6     model = models.Sequential([
7         layers.Conv2D(64, (3,3), padding='same', input_shape=(150, 150, 3), activation='relu'),
8         layers.MaxPooling2D(pool_size=(2,2)),
9         layers.Conv2D(64, (3, 3), padding='same'),
10        layers.MaxPooling2D(pool_size=(2,2)),
11        layers.Flatten(),
12        layers.Dense(64, activation='relu'),
13        layers.Dense(1, activation='sigmoid')
14    ])
15
16    # Compile the model with Adam optimizer and binary crossentropy loss
17    model.compile(optimizer='adam',
18                  loss='binary_crossentropy',
19                  metrics=['accuracy'])
20
21    return model

```

```

Model: "sequential_4"

```

Layer (type)	Output Shape	Param #
conv2d_14 (Conv2D)	(None, 150, 150, 64)	1792
max_pooling2d_14 (MaxPooling2D)	(None, 75, 75, 64)	0
conv2d_15 (Conv2D)	(None, 75, 75, 64)	36928
max_pooling2d_15 (MaxPooling2D)	(None, 37, 37, 64)	0
flatten_4 (Flatten)	(None, 87616)	0
dense_8 (Dense)	(None, 64)	5607488
dense_9 (Dense)	(None, 1)	65

```

=====
Total params: 5646273 (21.54 MB)
Trainable params: 5646273 (21.54 MB)
Non-trainable params: 0 (0.00 Byte)

```

- آموزش مدل: همانطور که مشاهده میشود از تکنیک early-stopping برای جلوگیری از overfitting استفاده شده است:

```
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

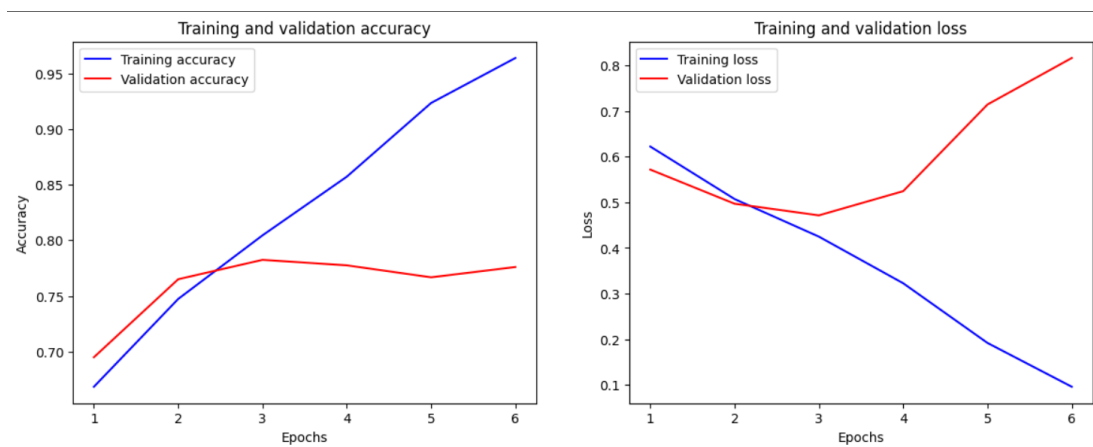
# Train the model with early stopping
history = model.fit(train_dataset,
                    epochs=10,
                    validation_data=test_dataset,
                    callbacks=[early_stopping])
```

```
Epoch 1/10
582/582 [=====] - 41s 64ms/step - loss: 0.6221 - accuracy: 0.6685 - val_loss: 0.5716 - val_accuracy: 0.6950
Epoch 2/10
582/582 [=====] - 39s 66ms/step - loss: 0.5071 - accuracy: 0.7473 - val_loss: 0.4968 - val_accuracy: 0.7650
Epoch 3/10
582/582 [=====] - 40s 67ms/step - loss: 0.4247 - accuracy: 0.8045 - val_loss: 0.4713 - val_accuracy: 0.7825
Epoch 4/10
582/582 [=====] - 39s 64ms/step - loss: 0.3225 - accuracy: 0.8572 - val_loss: 0.5243 - val_accuracy: 0.7775
Epoch 5/10
582/582 [=====] - 39s 65ms/step - loss: 0.1917 - accuracy: 0.9234 - val_loss: 0.7143 - val_accuracy: 0.7668
Epoch 6/10
582/582 [=====] - 42s 70ms/step - loss: 0.0957 - accuracy: 0.9638 - val_loss: 0.8162 - val_accuracy: 0.7760
```

که در اینجا مقدار ضرر و accuracy روی داده های تست به شرح زیر است:

**Test accuracy: 0.7825**

- رسم نمودار و نشان دادن خروجی‌ها:



ب) حال در این بخش از یک مدل pre-trained استفاده میکنیم. مراحل preprocessing عین بخش الف است.

- ابتدا چند نمونه نشان میدهیم:





- مدل از پیش آموزش داده شده را صدا میکنیم:

```
1 # Load the Inception-v3 model
2 inception_model = InceptionV3(weights='imagenet', include_top=False, input_shape=(299,299, 3))
3 print(inception_model.summary())
```

- پارامترهای آن را فریز کرده و سپس یک GAP و یک لایه Dense اضافه میکنیم:

```
4 from tensorflow.keras import layers, models
5 inception_model.trainable = False
6
7 model = models.Sequential([
8     inception_model,
9     layers.GlobalAveragePooling2D(),
10    layers.Dense(1)
11 ])
```

Model: "sequential\_3"

Layer (type)	Output Shape	Param #
inception_v3 (Functional)	(None, 8, 8, 2048)	21802784
global_average_pooling2d_3 (GlobalAveragePooling2D)	(None, 2048)	0
dense_3 (Dense)	(None, 1)	2049

=====  
Total params: 21804833 (83.18 MB)  
Trainable params: 2049 (8.00 KB)  
Non-trainable params: 21802784 (83.17 MB)  
=====  
None

- آموزش مدل:

```
Epoch 1/10  
582/582 [=====] - 137s 217ms/step - loss: 2.4986 - accuracy: 0.8136 - val_loss: 0.0969 - val_accuracy: 0.9862  
Epoch 2/10  
582/582 [=====] - 98s 166ms/step - loss: 0.1017 - accuracy: 0.9861 - val_loss: 0.0734 - val_accuracy: 0.9901  
Epoch 3/10  
582/582 [=====] - 99s 168ms/step - loss: 0.1553 - accuracy: 0.9815 - val_loss: 0.0898 - val_accuracy: 0.9882  
Epoch 4/10  
582/582 [=====] - 102s 172ms/step - loss: 0.0898 - accuracy: 0.9883 - val_loss: 0.0867 - val_accuracy: 0.9905  
Epoch 5/10  
582/582 [=====] - 101s 172ms/step - loss: 0.1491 - accuracy: 0.9837 - val_loss: 0.0831 - val_accuracy: 0.9912
```

- دقت روی داده تست:

Test accuracy: 0.9901

سوال 7: کد خواسته شده به شرح زیر است:

1. دانلود مجموعه داده و محدود کردن آن به ارقام 0 و 1 و 2:

```
1 # Load the MNIST dataset  
2 (x_train, y_train), (x_test, y_test) = mnist.load_data()  
3  
4 # Filter the dataset to only include digits 0, 1, and 2  
5 train_filter = np.where((y_train == 0) | (y_train == 1) | (y_train == 2))  
6 test_filter = np.where((y_test == 0) | (y_test == 1) | (y_test == 2))  
7  
8 x_train, y_train = x_train[train_filter], y_train[train_filter]  
9 x_test, y_test = x_test[test_filter], y_test[test_filter]
```

2. نرمال سازی مقادیر: همان طور که در کد ذکر شده، چون عکس های MNIST خودشان 28\*28 هستند، نیاز به resize کردن نیست.

```

1 # Normalize pixel values to the range [0, 1]
2 x_train = x_train / 255.0
3 x_test = x_test / 255.0

```

3. محاسبه Hu Moments: همان طور که میدانیم Hu Moments مجموعه‌ای از هفت عدد هستند که نسبت به تبدیل‌های تصویر مانند ترجمه، مقیاس‌بندی و چرخش ثابت هستند. آنها از لحظه‌های تصویر گرفته شده‌اند، که ویژگی‌های شکل اشیاء را در یک تصویر ثبت می‌کنند. Hu Moments به ویژه برای تجزیه و تحلیل شکل و وظایف تشخیص الگو مفید هستند زیرا آنها نمایش فشرده و کارآمدی از شکل را ارائه می‌دهند.

```

1 # Function to calculate Hu moments
2 def calculate_hu_moments(images):
3     hu_moments = []
4     for image in images:
5         # Convert image to binary
6         _, binary_image = cv2.threshold(image, 0.5, 1.0, cv2.THRESH_BINARY)
7         # Calculate moments
8         moments = cv2.moments(binary_image)
9         # Calculate Hu moments
10        hu = cv2.HuMoments(moments).flatten()
11        hu_moments.append(hu)
12    return np.array(hu_moments)

```

4. انتخاب مدل و آموزش بر روی دیتا: همان طور که میدانیم SVC یک مدل قوی و مورد استفاده در مسائل طبقه‌بندی است.

```

1 # Creating and training the SVM model
2 model = SVC(kernel='linear')
3 model.fit(hu_train, y_train)




```

5. خروجی: مقدار دقت بر روی داده تست به شرح زیر است:

	precision	recall	f1-score	support
0	0.65	0.98	0.78	980
1	0.98	0.96	0.97	1135
2	0.94	0.50	0.66	1032
accuracy			0.82	3147
macro avg	0.86	0.81	0.80	3147
weighted avg	0.86	0.82	0.81	3147

حال به نمایش برخی از عکس ها با لیبل واقعی و پیش بینی شده آنها

میپردازیم:

Label: 2	Label: 1	Label: 0	Label: 1	Label: 0	Label: 0	Label: 1	Label: 0	Label: 0	Label: 1
Pred: 2	Pred: 1	Pred: 0	Pred: 1	Pred: 0	Pred: 0	Pred: 1	Pred: 0	Pred: 0	Pred: 1
									

همچنین برای درک بهتر، چند نمونه داده اشتباه برچسب گذاری شده نیز

نمایش میدهیم:

True: 2	True: 2	True: 2	True: 2	True: 2	True: 2	True: 2	True: 2	True: 2	True: 2
Pred: 0	Pred: 0	Pred: 0	Pred: 0	Pred: 0	Pred: 0	Pred: 0	Pred: 0	Pred: 0	Pred: 0
