

a

دانشگاه علم و صنعت

تمرین ششم مبانی بینایی کامپیوتر

نام و نام خانوادگی:

فرناز خوش دوست آزاد

شماره دانشجویی:

99521253

نام استاد:

دکتر محمدرضا محمدی

۱- شبکه عصبی کانولوشنی زیر را در نظر بگیرید و به سوالات پاسخ دهید: (۲۵)

- ۱) `Input(shape=(512, 512, 3))`
- ۲) `Conv2D(32, (9, 9), strides=2, padding='same', activation='relu')`
- ۳) `MaxPooling2D((4, 4), strides=4)`
- ۴) `Conv2D(64, (5, 5), strides=1)`
- ۵) `AveragePooling2D((2, 2), strides=2)`
- ۶) `Conv2D(128, (3, 3), strides=1, padding='valid', activation='relu')`
- ۷) `Conv2D(128, (3, 3), strides=1, padding='same', activation='relu')`
- ۸) `MaxPooling2D((2, 2), strides=2)`
- ۹) `Conv2D(512, (3, 3), strides=1, padding='valid', activation='relu')`
- ۱۰) `GlobalAveragePooling2D()`
- ۱۱) `Dense(1024)`
- ۱۲) `Dense(10)`

الف) ابعاد خروجی و تعداد پارامترهای هر لایه را محاسبه کنید.

ب) تعداد اعمال ضرب و جمع در هر لایه که بر ورودی اعمال می شود را حساب کنید.

ج) اگر به جای لایه GAP از `flatten` استفاده شود تعداد پارامترهای شبکه چند برابر می شود؟

پاسخ:

الف)

برای هر عملگر بر روی ورودی تعداد پارامتر ها و ابعاد خروجی را با استفاده از فرمول های زیر که در اسلاید ها نیز موجود هستند، به دست می آوریم.

Convolutional Layer Output Dimensions:

For a convolutional layer:

For a convolutional layer:

$$H_{\text{out}} = \left\lfloor \frac{H_{\text{in}} + 2P - K}{S} \right\rfloor + 1$$
$$W_{\text{out}} = \left\lfloor \frac{W_{\text{in}} + 2P - K}{S} \right\rfloor + 1$$

Pooling Layer Output Dimensions

For a pooling layer:

$$H_{\text{out}} = \left\lfloor \frac{H_{\text{in}} - K}{S} \right\rfloor + 1$$
$$W_{\text{out}} = \left\lfloor \frac{W_{\text{in}} - K}{S} \right\rfloor + 1$$

Number of Parameters

For a convolutional layer:

$$\text{Params} = (K_H \times K_W \times C_{\text{in}} + 1) \times C_{\text{out}}$$

where K_H and K_W are the height and width of the kernel, C_{in} is the number of input channels, and C_{out} is the number of output channels.

For a dense (fully connected) layer:

$$\text{Params} = (C_{\text{in}} + 1) \times C_{\text{out}}$$

حال به محاسبات مورد نظر برای هر لایه می پردازیم.

1. Input Layer: (512, 512, 3)

2. Conv2D(32, (9, 9), strides=2, padding='same')

- Output size:

$$H_{\text{out}} = \left\lfloor \frac{512 + 2 \cdot 4 - 9}{2} \right\rfloor + 1 = 256$$
$$W_{\text{out}} = 256$$

- Parameters:

$$(9 \times 9 \times 3 + 1) \times 32 = 7808$$

لایه ی اول که ورودی است و لذا همان ابعاد ورودی را داریم و پارامتر ها نیز ثابت هستند. در لایه ی دوم چون 'padding = 'same' را داریم بدان معناست که ابعاد ورودی و خروجی یکی هستند، با این تفاوت که چون stride داریم، باید ابعاد ورودی را بر stride تقسیم کنیم.

3. MaxPooling2D((4, 4), strides=4)

- Output size:

$$H_{\text{out}} = \left\lfloor \frac{256-4}{4} \right\rfloor + 1 = 64$$

$$W_{\text{out}} = 64$$

- Parameters: 0 (pooling layers have no parameters)

چون عرض و طول محاسبات یکسانی دارند، از نوشتن آنها پرهیز کردم.

4. Conv2D(64, (5, 5), strides=1)

- Output size:

$$H_{\text{out}} = \left\lfloor \frac{64+0-5}{1} \right\rfloor + 1 = 60$$

$$W_{\text{out}} = 60$$

- Parameters:

$$(5 \times 5 \times 32 + 1) \times 64 = 51264$$

چون از قبل 32 تا لایه فیلتر داریم لذا باید برای تعداد پارامترها آن را ضربدر 32 کنیم و سپس به علاوه ی بایاس کنیم.

5. AveragePooling2D((2, 2), strides=2)

- Output size:

$$H_{\text{out}} = \left\lfloor \frac{60-2}{2} \right\rfloor + 1 = 30$$

$$W_{\text{out}} = 30$$

- Parameters: 0

مانند حالت قبل برخورد می کنیم.

6. Conv2D(128, (3, 3), strides=1, padding='valid')

- Output size:

$$H_{\text{out}} = \left\lfloor \frac{30+0-3}{1} \right\rfloor + 1 = 28$$

$$W_{\text{out}} = 28$$

- Parameters:

$$(3 \times 3 \times 64 + 1) \times 128 = 73792$$



حال می دانیم از لایه ی قبل 64 تا فیلتر داریم، لذا از آن در فرمول parameters استفاده می کنیم.

7. Conv2D(128, (3, 3), strides=1, padding='same')

- Output size:

$$H_{\text{out}} = 28$$

$$W_{\text{out}} = 28$$

- Parameters:

$$(3 \times 3 \times 128 + 1) \times 128 = 147584$$

مانند حالت های بالا برخورد می کنیم.

8. MaxPooling2D((2, 2), strides=2)

- Output size:

$$H_{\text{out}} = \left\lfloor \frac{28-2}{2} \right\rfloor + 1 = 14$$

$$W_{\text{out}} = 14$$

- Parameters: 0

9. Conv2D(512, (3, 3), strides=1, padding='valid')

- Output size:

$$H_{\text{out}} = \left\lfloor \frac{14+0-3}{1} \right\rfloor + 1 = 12$$

$$W_{\text{out}} = 12$$

- Parameters:

$$(3 \times 3 \times 128 + 1) \times 512 = 590336$$

10. GlobalAveragePooling2D()

- Output size:
 $1 \times 1 \times 512$
- Parameters: 0

11. Dense(1024)

- Parameters:
 $(512 + 1) \times 1024 = 524288$

12. Dense(10)

- Parameters:
 $(1024 + 1) \times 10 = 10250$

Summary

به اختصار جدول زیر محاسبات بالا را نشان می دهد.

Layer	Output Size	Number of Parameters
Input	(512, 512, 3)	0
Conv2D(32, (9, 9), strides=2, padding='same')	(256, 256, 32)	7808
MaxPooling2D((4, 4), strides=4)	(64, 64, 32)	0
Conv2D(64, (5, 5), strides=1)	(60, 60, 64)	51264
AveragePooling2D((2, 2), strides=2)	(30, 30, 64)	0
Conv2D(128, (3, 3), strides=1, padding='valid')	(28, 28, 128)	73792
Conv2D(128, (3, 3), strides=1, padding='same')	(28, 28, 128)	147584
MaxPooling2D((2, 2), strides=2)	(14, 14, 128)	0
Conv2D(512, (3, 3), strides=1, padding='valid')	(12, 12, 512)	590336
GlobalAveragePooling2D()	(512,)	0
Dense(1024)	(1024,)	524288
Dense(10)	(10,)	10250

Total number of parameters: (7808 + 51264 + 73792 + 147584 + 590336 + 524288 + 10250 = 1,405,322).



For a convolutional layer, the number of operations is determined by the following formula:

$$\text{Operations} = (K_H \times K_W \times C_{\text{in}}) \times (H_{\text{out}} \times W_{\text{out}} \times C_{\text{out}})$$

Here, each element in the output feature map requires $K_H \times K_W \times C_{\text{in}}$ multiplications (for the convolution) and $K_H \times K_W \times C_{\text{in}} - 1$ additions (to sum the products).

Dense Layer Operations

For a dense layer, the number of operations is:

$$\text{Operations} = (C_{\text{in}} \times C_{\text{out}})$$

Each neuron in the dense layer performs C_{in} multiplications and $C_{\text{in}} - 1$ additions.

Let's compute the operations for each layer in your model:

1. Conv2D(32, (9, 9), strides=2, padding='same')

- Input: (512, 512, 3)
- Output: (256, 256, 32)
- Operations:
$$\begin{aligned} &9 \times 9 \times 3 \times 256 \times 256 \times 32 \\ &= 9 \times 9 \times 3 \times 256^2 \times 32 \\ &= 9 \times 9 \times 3 \times 65536 \times 32 \\ &= 1528823808 \text{ multiplications} \end{aligned}$$
- Additions:
$$\begin{aligned} &(9 \times 9 \times 3 - 1) \times 256 \times 256 \times 32 \\ &= 2429884416 \text{ additions} \end{aligned}$$

2. Conv2D(64, (5, 5), strides=1)

- Input: (64, 64, 32)
- Output: (60, 60, 64)
- Operations:
$$5 \times 5 \times 32 \times 60 \times 60 \times 64$$
$$= 5 \times 5 \times 32 \times 3600 \times 64$$
$$= 184320000 \text{ multiplications}$$
- Additions:
$$(5 \times 5 \times 32 - 1) \times 60 \times 60 \times 64$$
$$= 177888000 \text{ additions}$$

3. Conv2D(128, (3, 3), strides=1, padding='valid')

- Input: (30, 30, 64)
- Output: (28, 28, 128)
- Operations:
$$3 \times 3 \times 64 \times 28 \times 28 \times 128$$
$$= 3 \times 3 \times 64 \times 784 \times 128$$
$$= 70860864 \text{ multiplications}$$
- Additions:
$$(3 \times 3 \times 64 - 1) \times 28 \times 28 \times 128$$
$$= 70041600 \text{ additions}$$

4. Conv2D(128, (3, 3), strides=1, padding='same')

- Input: (28, 28, 128)
- Output: (28, 28, 128)
- Operations:
$$3 \times 3 \times 128 \times 28 \times 28 \times 128$$
$$= 3 \times 3 \times 128 \times 784 \times 128$$
$$= 141721600 \text{ multiplications}$$
- Additions:
$$(3 \times 3 \times 128 - 1) \times 28 \times 28 \times 128$$
$$= 140810240 \text{ additions}$$

5. Conv2D(512, (3, 3), strides=1, padding='valid')

- Input: (14, 14, 128)
- Output: (12, 12, 512)
- Operations:
$$3 \times 3 \times 128 \times 12 \times 12 \times 512$$
$$= 3 \times 3 \times 128 \times 144 \times 512$$
$$= 30474240 \text{ multiplications}$$
- Additions:
$$(3 \times 3 \times 128 - 1) \times 12 \times 12 \times 512$$
$$= 30233088 \text{ additions}$$

6. Dense(1024)

- Input: 512
- Output: 1024
- Operations:
 512×1024
 $= 524288$ multiplications
- Additions:
 $(512 - 1) \times 1024$
 $= 523264$ additions

7. Dense(10)

- Input: 1024
- Output: 10
- Operations:
 1024×10
 $= 10240$ multiplications
- Additions:
 $(1024 - 1) \times 10$
 $= 10230$ additions

Layer	Output Size	Multiplications	Additions
Conv2D(32, (9, 9), strides=2, padding='same')	(256, 256, 32)	1,528,823,808	2,429,884,416
MaxPooling2D((4, 4), strides=4)	(64, 64, 32)	0	0
Conv2D(64, (5, 5), strides=1)	(60, 60, 64)	184,320,000	177,888,000
AveragePooling2D((2, 2), strides=2)	(30, 30, 64)	0	0
Conv2D(128, (3, 3), strides=1, padding='valid')	(28, 28, 128)	70,860,864	70,041,600
Conv2D(128, (3, 3), strides=1, padding='same')	(28, 28, 128)	141,721,600	140,810,240
MaxPooling2D((2, 2), strides=2)	(14, 14, 128)	0	0
Conv2D(512, (3, 3), strides=1, padding='valid')	(12, 12, 512)	30,474,240	30,233,088
GlobalAveragePooling2D()	(512,)	0	0
Dense(1024)	(1024,)	524,288	523,264
Dense(10)	(10,)	10,240	10,230

(ج)

اگر به جای لایه GAP از flatten استفاده شود، تعداد پارامترهای شبکه چند برابر می شود؟

1. GlobalAveragePooling2D:

- Input: (14, 14, 128)
- Output: (128,)

2. Flatten:

- Input: (14, 14, 128)
- Output: $(14 \times 14 \times 128) = (25088,)$

Effect on Dense Layers

Let's consider the Dense layers:

Dense Layer with GAP (1024 units)

- Input size: 128
- Parameters: $128 \times 1024 + 1024 = 131072 + 1024 = 132096$

Dense Layer with Flatten (1024 units)

- Input size: 25088
- Parameters: $25088 \times 1024 + 1024 = 25690112 + 1024 = 25691136$

Dense Layer (10 units) - Same for both scenarios

- Input size: 1024
- Parameters: $1024 \times 10 + 10 = 10240 + 10 = 10250$

1. With GlobalAveragePooling2D:

- Dense(1024): 132096 parameters
- Dense(10): 10250 parameters
- Total: $132096 + 10250 = 142346$

2. With Flatten:

- Dense(1024): 25691136 parameters
- Dense(10): 10250 parameters
- Total: $25691136 + 10250 = 25701386$

Ratio of Parameters

To find the ratio of the number of parameters with Flatten to the number of parameters with GlobalAveragePooling2D:

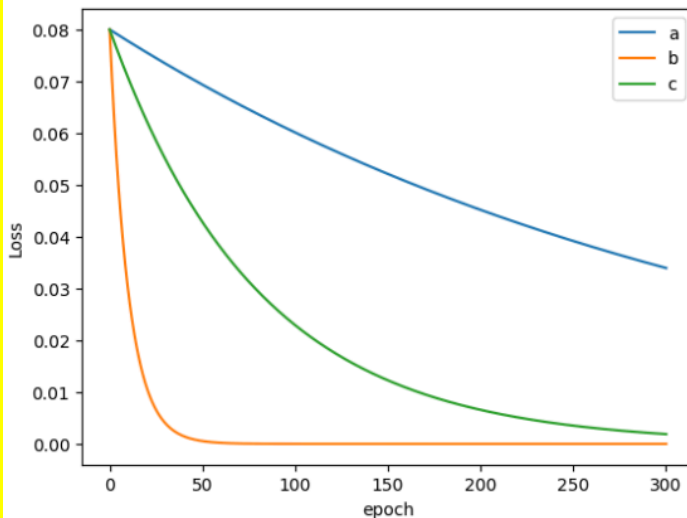
$$\text{Ratio} = \frac{25701386}{142346} \approx 180.6$$

(2)

۲- تابع ضرر زیر را با مقدار اولیه ۲۰ برای X در نظر بگیرید (۵)

$$L = x^2 - 10x + e^{\dots\dots\dots}$$

اگر با نرخ یادگیری متفاوت مقدار X را طی یک مرحله با الگوریتم گرادیان کاهشی بروز رسانی کنیم، طبق شکل زیر هر یک از مقادیر جدید X می تواند مربوط به کدام نمودار باشد؟ چرا؟



(۱) $X = 14$

(۲) $X = 19.4$

(۳) $X = 19.94$

پاسخ:

از آنجایی که باید مقدار تابع ضرر را مینیمم کنیم، لذا به سراغ مشتق می رویم، و آن را برابر با صفر قرار می دهیم تا مقدار مینیمم پیدا شود لذا داریم:

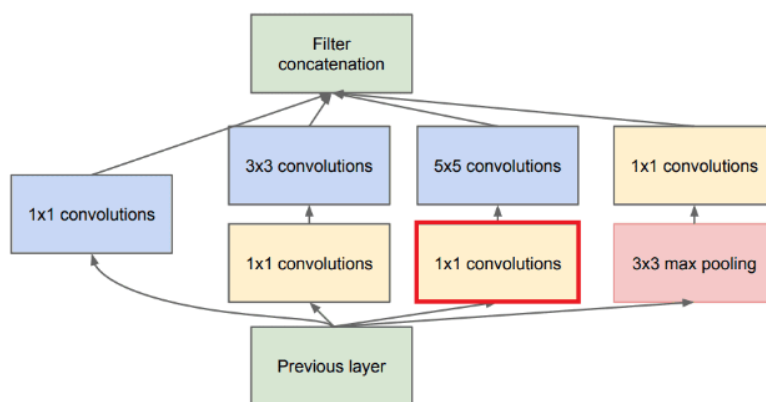
$$2x - 10 = 0 \rightarrow x = 5$$

چون از مقدار 20 شروع کرده ایم، لذا باید با مقدار a مناسب آن را به کمترین مقدارش یعنی 5 برسانیم، بنابراین از آنجایی که خط آبی با سرعت بسیار کمی به سمت کم شدن قدم بر میدارد لذا مقدار 19.94 را به آن assign می کنیم چون فقط 0.06 از آن کم شده است که حاکی از سرعت کم آن است و از آنجایی

که خط سبز با سرعت مناسبی در حال کم شدن است لذا مقدار $X = 19.4$ را برای آن assign می کنیم و از آنجایی که خط نارنجی با سرعت بسیار زیادی به سمت کم شدن می رود مقدار 14 برای آن مناسب است که از مقدار قبلی خود 6 واحد فاصله دارد.

(3)

۳- در ماژول inception زیر اگر ابعاد ورودی (۱۲،۱۲،۳۲) باشد و تعداد فیلتر کانولوشن های $۱*۱$ ، $۳*۳$ و $۵*۵$ به ترتیب ۶۴، ۳۲ و ۱۲۸ باشد، ابعاد خروجی را بدست آورید. اگر تعداد فیلترهای کانولوشن مشخص شده را به ۲۵۶ تغییر دهیم ابعاد خروجی را حساب کنید. (۵)



پاسخ:

در ابتدا در شکل می بینیم که سه لایه ی متوالی داریم و در آخر 4 لایه با هم convatenate می شوند. کد مربوط به آن در فایل آپلود شده موجود می باشد که می بینیم قابلیت concatenate کردن ندارد و به ارور خواهیم خورد که برای رفع این ارور باید از لایه هایی مثل globalMaxPooling یا globalAveragePooling استفاده کرد تا بتوان آنها را concatenate کرد، لذا داریم:

لایه ترتیبی 1:

:MaxPooling2D(pool_size=(3, 3))

ورودی: $(32, 12, 12)$

خروجی: $(32, 4, 4)$ (هر بعد فضایی به یک سوم کاهش می‌یابد)

:Conv2D(filters=64, kernel_size=(1, 1))

ورودی: $(32, 4, 4)$

خروجی: $(64, 4, 4)$ (تعداد فیلترها به 64 افزایش می‌یابد)

لایه ترتیبی 2:

:Conv2D(filters=64, kernel_size=(1, 1))

ورودی: $(32, 12, 12)$

خروجی: $(64, 12, 12)$ (تعداد فیلترها به 64 افزایش می‌یابد)

:Conv2D(filters=128, kernel_size=(5, 5))

ورودی: $(64, 12, 12)$

خروجی: $(128, 8, 8)$ (ابعاد فضایی به دلیل کرنل 5×5 کاهش می‌یابد)

لایه ترتیبی 3:

:Conv2D(filters=64, kernel_size=(1, 1))

ورودی: $(32, 12, 12)$

خروجی: $(64, 12, 12)$ (تعداد فیلترها به 64 افزایش می‌یابد)

:Conv2D(filters=32, kernel_size=(3, 3))

ورودی: $(64, 12, 12)$

خروجی: $(32, 10, 10)$ (ابعاد فضایی به دلیل کرنل 3×3 کاهش می‌یابد)

برای ترکیب خروجی‌ها با استفاده از **Concatenate**، باید خروجی‌ها ابعاد فضایی یکسانی داشته باشند. که در اینجا خروجی‌ها دارای ابعاد متفاوتی هستند و نمی‌توان آنها را مستقیماً ترکیب کرد. لذا از روشی که در ابتدا گفته شد، استفاده می‌کنیم و در صورتی که بخواهیم ابعاد خروجی‌ها را به درستی برای هر یک از لایه‌های ترتیبی و موازی محاسبه کنیم، این ابعاد به شکل زیر خواهند بود:

- لایه ترتیبی 1: (64, 4, 4)
- لایه ترتیبی 2: (128, 8, 8)
- لایه ترتیبی 3: (32, 10, 10)
- لایه موازی دیگر: (64, 12, 12)

در نهایت، این خروجی ها را با استفاده از GlobalMaxPooling خروجی های 64، 32، 128، 64 و ... را می گیریم و سپس خروجی های یک بعدی را با استفاده از Concatenate ترکیب می کنیم.

```
concatenated.shape : (None, 288)
Model: "model_2"
```

که کد کامل آن در در قسمت دوم فایل q3_hw6_cv.ipynb موجود می باشد. که پس از concatenate پاسخ آن (None, 288) نمایش داده شده است، لذا در لایه concatenate ابعاد خروجی از جمع لایه های بالا به دست می آید که از طرفی None را می توانیم $12 * 12$ نیز در نظر بگیریم. اما در کد زده شده همان None به ما برگردانده شد چون برای رفع ارور نیز از Dense استفاده می کنیم که در فایل q3_hw6_cv.ipynb نیز موجود می باشد و نتیجه ی آن مشخص است

حال در صورت تغییر تعداد فیلترها خواهیم داشت:

- لایه ترتیبی 1: (64, 4, 4)
- لایه ترتیبی 2: (128, 8, 8)
- لایه ترتیبی 3: (32, 10, 10)
- لایه موازی دیگر: (64, 12, 12)

همانطور که در بخش سوم فایل بالا دیده می شود ابعاد خروجی آن با اعمال GlobalMaxPooling یا GlobalAveragePooling هر کدام به صورت (None, 288) می شوند. زیرا پس از این کانولوشن، ما کانولوشن دیگری داریم که تعداد فیلترها را به سائز اولیه ی خود برمی گرداند.

```
concatenated.shape : (None, 288)
Model: "model_3"
```


لذا تغییری ایجاد نمی شود.

(4)

۴- به سوالات زیر پاسخ دهید: (۱۰)

الف) اگر t تعداد نمونه های آموزشی، e تعداد دوره های آموزشی (epoch) و b را $batch_size$ در نظر بگیریم چند بار وزن های شبکه به روز رسانی می شوند؟

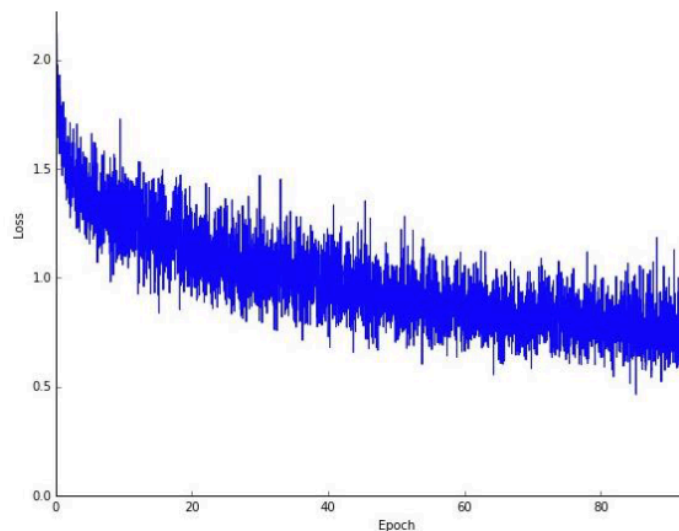
پاسخ:

در این صورت از فرمول های زیر استفاده می کنیم:

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

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

ب) به نظر شما نمودار زیر مربوط به Batch GD است یا mini-batch GD؟ توضیح دهید.



پاسخ:

در Batch Gradient Descent، کل مجموعه داده های آموزشی در یک دسته (batch) قرار داده شده و استفاده می شود. در هر epoch، یک به روز رسانی برای وزن های شبکه انجام می شود. این به روز رسانی با استفاده از تمام داده های آموزشی انجام می شود.

در **Mini-Batch Gradient Descent**، مجموعه داده‌های آموزشی به دسته‌های کوچکتر (**mini-batches**) تقسیم می‌شوند. هر دسته شامل تعداد کمی از نمونه‌های آموزشی است. در هر **epoch**، وزن‌های شبکه با استفاده از هر **mini-batch** به‌روزرسانی می‌شوند حال به مقایسه ی نمودارها می پردازیم.

مقایسه نمودارها:

1. Batch Gradient Descent:

- نمودار خطا در طول زمان بسیار هموار است.
- تغییرات هموار و پایدار است، زیرا به‌روزرسانی‌ها با استفاده از تمام داده‌ها انجام می‌شود.
- همگرایی کندتری دارد زیرا به‌روزرسانی‌ها کمتر انجام می‌شود اما دقیق‌تر است.

2. Mini-Batch Gradient Descent:

- نمودار خطا دارای نوسانات کوچکی است که به دلیل به‌روزرسانی‌های مکرر با **mini-batch**ها است.
- همگرایی سریع‌تری نسبت به **Batch GD** دارد، زیرا به‌روزرسانی‌های بیشتری در هر **epoch** انجام می‌شود.

- در نمودار آن تغییرات نسبتاً صاف اما با نوسانات کوچک است، زیرا به‌روزرسانی‌ها بر اساس

mini-batchها انجام می‌شود.

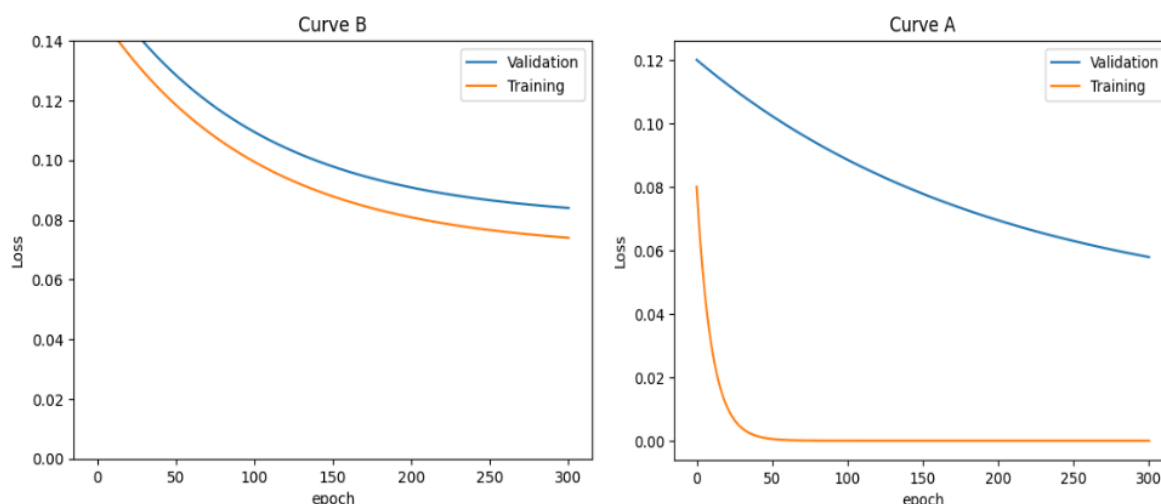
لذا با توجه ویژگی های بالا که نمودار هموار است اما نوسانات کوچک بسیار زیادی دارد، لذا این نمودار مربوط به **mini-batch** می باشد.

د) هر یک از اعمال زیر را برای کدام نمودار پیشنهاد می دهید؟ چرا؟

۱. داده افزایی

۲. افزایش لایه های شبکه

۳. کاهش تعداد ویژگی های ورودی



پاسخ:

Curve A → data augmentation, decrease input features

زیرا train با validation تفاوت بسیار زیادی دارد، لذا داده ها را زیاد می کنیم تا با تنوع بیشتری روبرو باشیم و مدل ما نمونه ها را حفظ نکند و انقدر تفاوت فاحش بین این دو نمونه نداشته باشیم. احتمالاً این مدل شما دچار بیش‌برازش (overfitting) شده است. این وضعیت به این معناست که مدل به خوبی داده‌های آموزش را یاد گرفته اما نمی‌تواند به خوبی روی داده‌های جدید (اعتبارسنجی یا تست) عمل کند. از کاهش ورودی ها نیز می توانیم برای جلوگیری از حفظ کردن داده ی train توسط مدل استفاده کنیم.

Curve B → Increasing Network Depth, data augmentation

افزایش تعداد لایه‌های شبکه به مدل اجازه می‌دهد تا ویژگی‌های پیچیده‌تر و سطح بالاتری را از داده‌ها استخراج کند. این تکنیک معمولاً برای مسائل پیچیده و بزرگتر مورد استفاده قرار می‌گیرد. که با این کار می توانیم loss این دو را به هم نزدیک کنیم.

این نمودار به این معناست که مدل شما به خوبی به داده‌های آموزش و داده‌های اعتبارسنجی آموزش داده شده است و مشکل بیش‌برازش یا کم‌برازش (underfitting) ندارد. با این حال، اگر هنوز به دقت مطلوب نرسیده‌اید، می‌توانید اقداماتی انجام دهید تا دقت مدل را بیشتر بهبود بخشید. در این شرایط، می‌توانید از تکنیک‌های زیر استفاده کنید:

(5)

۵- الف) کد LBP_u^1 را برای پیکسل‌های غیر صفر تصویر بدست آورید. (۱۵)

۰	۰	۰	۰	۰
۰	۲۵۰	۲۰۰	۵۰	۰
۰	۱۸۰	۱۰۰	۸۰	۰
۰	۲۰۰	۴۰	۷۰	۰
۰	۰	۰	۰	۰

پاسخ:

در ابتدا مربع‌های سه در سه را در نظر می‌گیریم و برای هر پیکسل در صورتی که بزرگتر یا مساوی مرکز باشد، یک و در غیر این صورت صفر می‌گذاریم:

و در از پیکسل بالا سمت راست شروع می‌کنیم و به صورت ساعتگرد 0 و 1 ها را می‌نویسیم که یک عدد 8 بیتی می‌شود، و در آخر آن را به بیت تبدیل می‌کنیم که برای محاسبه ی آن می‌توانیم کد مربوط به LBP را می‌زنیم که به صورت زیر است:

```

import numpy as np
def calculate_lbp(image):
    rows, cols = image.shape
    lbp_image = np.zeros((rows - 2, cols - 2), dtype=np.uint8)
    for i in range(1, rows - 1):
        for j in range(1, cols - 1):
            center = image[i, j]
            binary_string = ''
            binary_string += '1' if image[i-1, j-1] >= center else '0'
            binary_string += '1' if image[i-1, j] >= center else '0'
            binary_string += '1' if image[i-1, j+1] >= center else '0'
            binary_string += '1' if image[i, j+1] >= center else '0'
            binary_string += '1' if image[i+1, j+1] >= center else '0'
            binary_string += '1' if image[i+1, j] >= center else '0'
            binary_string += '1' if image[i+1, j-1] >= center else '0'
            binary_string += '1' if image[i, j-1] >= center else '0'
            lbp_image[i-1, j-1] = int(binary_string, 2)
    return lbp_image

image = np.array([
    [0, 0, 0, 0, 0],
    [0, 250, 200, 50, 0],
    [0, 180, 100, 80, 0],
    [0, 200, 40, 70, 0],
    [0, 0, 0, 0, 0]
], dtype=np.uint8)

lbp_image = calculate_lbp(image)
print(lbp_image)

```

که پاسخ آن به صورت زیر است:

```

[[ 0  1  7]
 [100 195 129]
 [ 0 241 192]]

```

که در کد ما همانطور که دیده می شود، بیت ها تبدیل به integer شده اند.

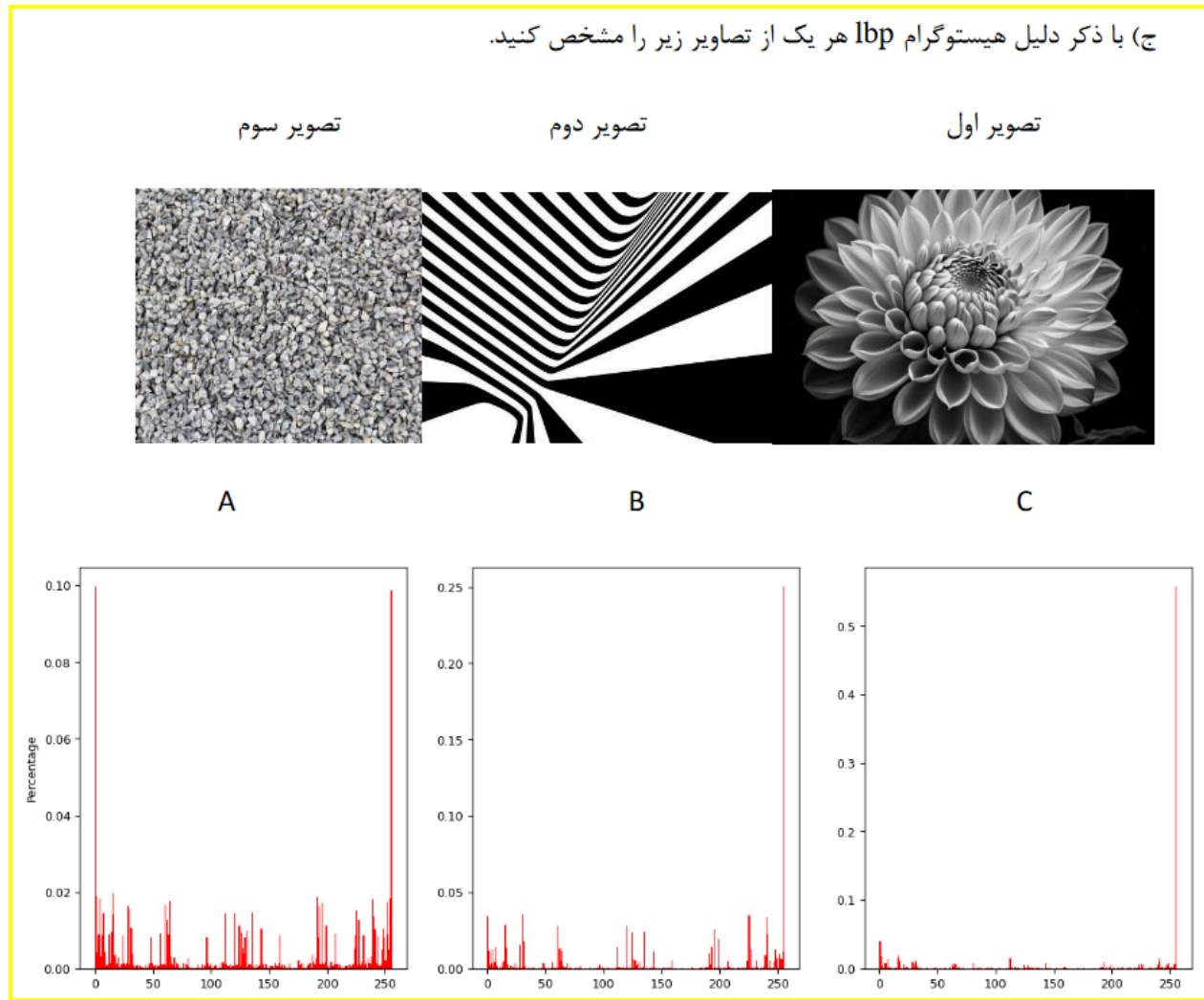
ب) اگر تمامی پیکسل های تصویر با عدد ثابت $C > 0$ جمع شوند کد چه تغییری می کند؟ اگر ضرب شوند چطور؟

پاسخ:

از آنجا که کوچکی و بزرگی اعداد در صورت جمع شدن با یک عدد ثابت عوض نمی شود لذا بر روی صفر و یک ها نیز تاثیری نمی گذارد، بنابراین هیچ تغییری نمی کند و در صورت ضرب شدن نیز از آنجا که $C > 0$ می باشد، لذا باز هم جهت علامت عوض نمی شود و همین جواب بالا را خواهیم داشت.

$$A > D \rightarrow A + b > D + b$$

$$A > D \rightarrow (k > 0) \rightarrow Ak > Dk$$



پاسخ:

می دانیم که LBP بافت یک تصویر را مورد ارزیابی قرار می دهد. در ابتدا می توانیم به سراغ تصویر دوم برویم که بیشتر تصویر را سفید یا مشکی یک دست تشکیل داده است. لذا بیشتر اطرافیان یک پیکسل مرکزی با آن برابر هستند، لذا مقادیر آنها همگی 1 می شود، پس مقدار 255 را خواهیم داشت، و از آنجایی که تنوع بافت در این عکس دیده نمی شود، بنابراین C نمودار مناسبی برای آن به شمار می رود.

در تصویر اول نیز از آنجا که بافت ها ساده هستند و بخش زیادی از تصویر را background مشکی گرفته است، لذا پراکندگی کمتری به نسبت تصویر سوم داریم و فراوانی مربوط به کد 255 در آن نیز نسبت به تصویر سوم بیشتر می باشد. لذا نمودار B مربوط به آن است.

در تصویر سوم نیز چون در تصویر پراکندگی بافت بسیار زیادی دیده می شود و بافت مشخص و تکرار شونده ای در آن دیده نمی شود لذا نمودار A نیز مربوط به آن است.

A → تصویر سوم

B → تصویر اول

C → تصویر دوم

(6)

۶- الف) به نوتبوک cats_vs_dogs مراجعه کنید. در این سوال قرار است یک شبکه CNN روی مجموعه داده cats_vs_dogs آموزش دهید. برای حل این سوال مجاز هستید از تمامی روش هایی که در کلاس تدریس شده است استفاده کنید. همچنین از تمامی روش هایی که می شناسید استفاده کنید تا حتی الامکان از بیش برآزش جلوگیری شود. در پایان دقت تست را گزارش کنید و همچنین نمودار loss و accuracy مربوط به train و validation را در حین آموزش رسم کنید. (۴۰)

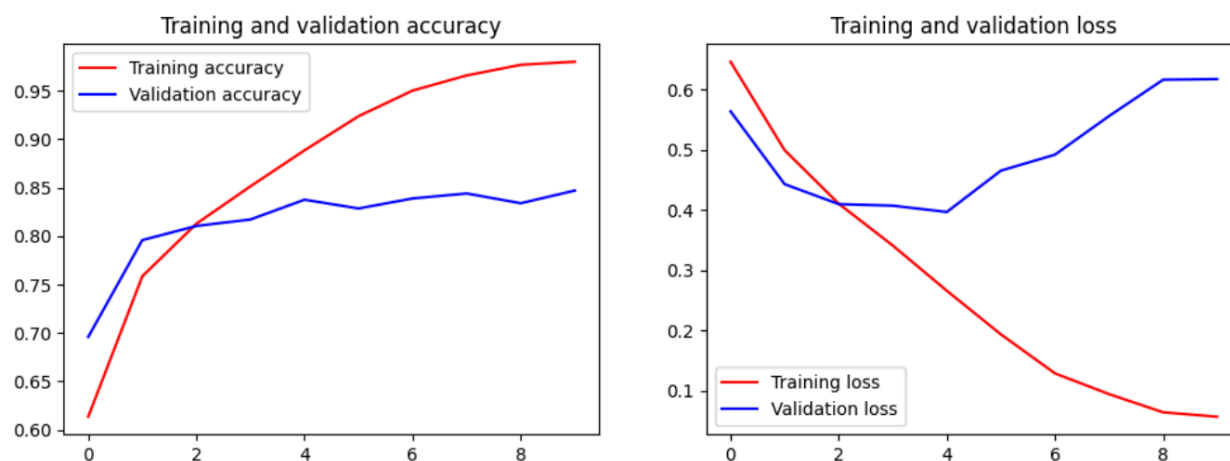
پاسخ:

شبکه ی CNN ای که برای این سوال در نظر گرفتم، به شرح زیر است و طرز عملکرد آنها دقیقاً مثل همان اسلایدهای سر کلاس می باشد. شبکه ی ما به شرح زیر می باشد:

```
#create your model
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(128, 128, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dropout(0.45), # Dropout to prevent overfitting
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

پس از استفاده از شبکه ی بالا و fit و predict کردن برای epoch 10، دقت آن را به دست می آوریم و سپس plot آن را رسم می کنیم. از آنجا که در شکل اول train_data، دقت خوبی دارند ولی test_data از دقت خیلی کمتری برخوردار است لذا می توانیم بگوییم که مدل ما train_data را حفظ کرده است و overfit رخ داده است که نتایج آن در زیر موجود است.

```
Epoch 1/10
582/582 [=====] - 48s 68ms/step - loss: 0.6457 - accuracy: 0.6135 - val_loss: 0.5633 - val_accuracy: 0.6958
Epoch 2/10
582/582 [=====] - 19s 33ms/step - loss: 0.4989 - accuracy: 0.7584 - val_loss: 0.4425 - val_accuracy: 0.7958
Epoch 3/10
582/582 [=====] - 18s 30ms/step - loss: 0.4101 - accuracy: 0.8127 - val_loss: 0.4094 - val_accuracy: 0.8104
Epoch 4/10
582/582 [=====] - 18s 30ms/step - loss: 0.3406 - accuracy: 0.8513 - val_loss: 0.4068 - val_accuracy: 0.8171
Epoch 5/10
582/582 [=====] - 18s 31ms/step - loss: 0.2655 - accuracy: 0.8884 - val_loss: 0.3963 - val_accuracy: 0.8375
Epoch 6/10
582/582 [=====] - 17s 29ms/step - loss: 0.1932 - accuracy: 0.9237 - val_loss: 0.4649 - val_accuracy: 0.8285
Epoch 7/10
582/582 [=====] - 23s 39ms/step - loss: 0.1284 - accuracy: 0.9501 - val_loss: 0.4914 - val_accuracy: 0.8388
Epoch 8/10
582/582 [=====] - 19s 32ms/step - loss: 0.0940 - accuracy: 0.9658 - val_loss: 0.5555 - val_accuracy: 0.8439
Epoch 9/10
582/582 [=====] - 17s 30ms/step - loss: 0.0638 - accuracy: 0.9767 - val_loss: 0.6158 - val_accuracy: 0.8338
Epoch 10/10
582/582 [=====] - 18s 30ms/step - loss: 0.0565 - accuracy: 0.9799 - val_loss: 0.6169 - val_accuracy: 0.8469
146/146 [=====] - 5s 34ms/step - loss: 0.6169 - accuracy: 0.8469
Test Accuracy: 0.8469
```



همانطور که دیده می شود دقت داده train 97 درصد ولی برای test_data، هشتاد و چهار درصد می باشد.

لذا در قسمت دوم نوتبوک از روش های دیگری استفاده کردم که در زیر گفته ام:

1. افزایش پیچیدگی مدل:

- اضافه کردن لایه های بیشتر و نورون های بیشتر به مدل.

- استفاده از لایه‌های کانولوشن (Convolutional Layers) که برای کار با تصاویر بسیار مناسب هستند.

2. داده‌افزایی:

- استفاده از تکنیک‌های داده‌افزایی مانند چرخش، تغییر مقیاس، انتقال و وارونگی برای افزایش تنوع داده‌های آموزش.

3. منظم‌سازی:

- استفاده از Dropout.










- استفاده از L2 regularization.

4. تنظیم ابرپارامترها:

- تنظیم تعداد اپوک‌ها (epochs) و اندازه دسته‌ها (batch size).

داده‌افزایی (ImageDataGenerator): با استفاده از تکنیک‌های داده‌افزایی، تنوع داده‌های آموزش افزایش یافته و به شبکه کمک می‌کند تا بهتر آموزش ببیند و از بیش‌برازش جلوگیری شود و تعداد epoch ها به 50 افزایش یافت که نتیجه‌ی آن را در زیر می‌بینیم:

```
99/99 [=====] - 0s 3ms/step - loss: 0.0034 - accuracy: 0.9987
Test Accuracy: 1.00
99/99 [=====] - 0s 2ms/step
```

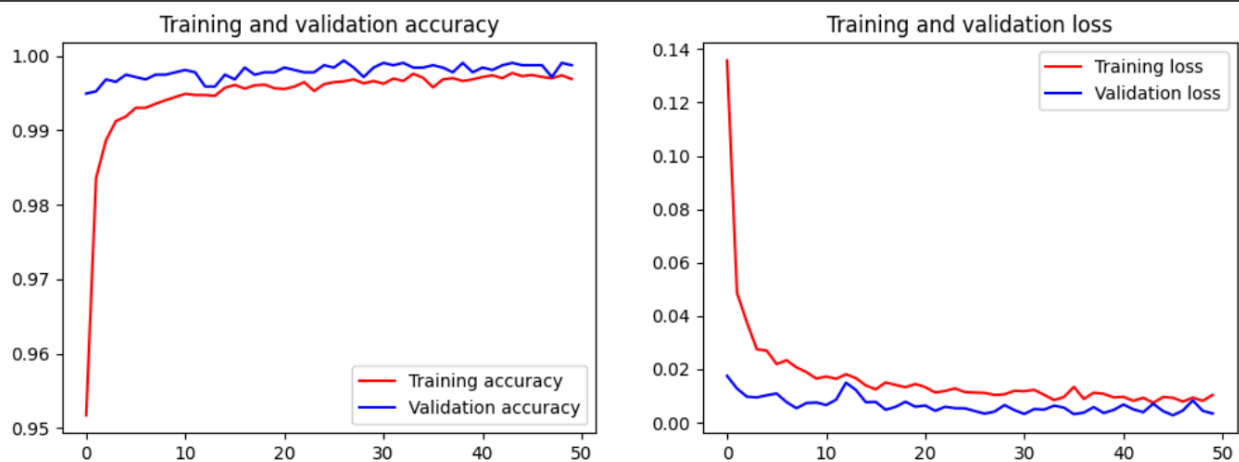
True: 2	True: 2	True: 1	True: 1	True: 2	True: 2	True: 0	True: 2	True: 1	True: 2
Pred: 2	Pred: 2	Pred: 1	Pred: 1	Pred: 2	Pred: 2	Pred: 0	Pred: 2	Pred: 1	Pred: 2
									

```
] 1
2 # Plot accuracy and loss
3 acc = history.history['accuracy']
4 val_acc = history.history['val_accuracy']
5 loss = history.history['loss']
6 val_loss = history.history['val_loss']
7 epochs = range(len(acc))
8 acc[-1]
```

0.9968802332878113

که می‌بینیم برای train دقت 1 و برای test دقت بالای 99 درصد را داریم:

و نتیجه ی آن به این صورت شد که دقت بسیار خوبی دارد:



ب) به نوتبوک Transfer_cats_vs_dogs مراجعه کنید. در این سوال به بخش قبل برمیگردیم اما این بار به جای آموزش یک شبکه از صفر، از یک مدل آماده InceptionV3 استفاده میکنید و تنها لایه آخر را متناسب با مسئله خود تغییر داده و آن را آموزش می دهید. نهایتاً دقت test را گزارش کنید.

پاسخ:

در ابتدا به preprocess داده هایی که از tfds خوانده ایم، می پردازیم که به صورت زیر است:

```
1 # Load the Dogs vs. Cats dataset
2 train_dataset, info = tfds.load('cats_vs_dogs', split='train[:80%]', with_info=True, as_supervised=True)
3 test_dataset = tfds.load('cats_vs_dogs', split='train[80%:]', as_supervised=True)
4
5 def preprocess(image, label):
6     image = tf.image.resize(image, [299, 299]) # InceptionV3 expects 299x299 images
7     image = tf.cast(image, tf.float32) / 255.0
8     return image, label
9
10 # Apply preprocessing
11 train_dataset = train_dataset.map(preprocess).cache().shuffle(1000).batch(32).prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
12 test_dataset = test_dataset.map(preprocess).batch(32).prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
```

که همانطور که دیده می شود در ابتدا همه ی تصاویر را resize می کنیم و سپس همه ی آنها را بین صفر و یک cast می کنیم و سپس train_dataset و test_dataset را با preprocess به دست می آوریم. بخشی از نمونه ها هم به شرح زیر است که در پایین دیده می شود.



حال با استفاده از کد زیر اسکوپ های مختلف کد را میزنیم و با استفاده از imagenet، می توانیم InceptionV3 را پر کنیم و همینطور که در کد زیر دیده می شود، هر layer را trainable می کنیم

```

22 # Load the Dogs vs. Cats dataset
23 train_dataset, info = tfds.load('cats_vs_dogs', split='train[:80%]', with_info=True, as_supervised=True)
24 test_dataset = tfds.load('cats_vs_dogs', split='train[80%:]', as_supervised=True)
25 print(len(train_dataset))
26 print(len(test_dataset))
27 def preprocess(image, label):
28     image = tf.image.resize(image, [299, 299]) # InceptionV3 expects 299x299 images
29     image = tf.cast(image, tf.float32) / 255.0
30     return image, label
31
32 # Apply preprocessing
33 train_dataset = train_dataset.map(preprocess).cache().shuffle(1000).batch(32).prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
34 test_dataset = test_dataset.map(preprocess).batch(32).prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
35
36 # Load the base model
37 base_model = InceptionV3(weights='imagenet', include_top=False, input_shape=(299, 299, 3))
38
39 # Set the layers of the base model to be trainable
40 for layer in base_model.layers:
41     layer.trainable = True
42
43 # Add custom layers on top of the base model
44 x = base_model.output
45 x = GlobalAveragePooling2D()(x)
46 x = Dense(1024, activation='relu')(x)
47 x = Dropout(0.5)(x) # Dropout to prevent overfitting
48 predictions = Dense(1, activation='sigmoid')(x)

```

در آخر نیز همانطور که در کد بالا دیده می شود با استفاده از لایه های مختلف GlobalAveragePooling2D، Dense، Dropout لایه های آخر را ویرایش می کنیم و با استفاده از activation های مختلف خروجی ها را به دست می آوریم.

همانطور که در نوتبوک دیده می شود نیز در اسکوپ های مختلف قسمت های خواسته شده ی سوال پر شده اند و در آخر plot مورد نظر کشیده شده است و دقت 86 درصد در آخر گزارش شده است که در نوتبوک نیز مشاهده می شود.

(7)

۷- در این سوال قصد داریم به طبقه بندی تصاویر ارقام دستنویس مختلف از مجموعه داده MNIST با استفاده از توصیف گرهای شکل (Hu moments) و یک مدل یادگیری ماشین بپردازیم: (۲۰)

- ابتدا مجموعه داده MNIST را دانلود کنید. سپس یک زیرمجموعه از داده ها را که فقط شامل ارقام ۰، ۱ و ۲ است استخراج کنید.
- تصاویر را با نرمال سازی مقادیر پیکسل ها و در صورت لزوم تغییر اندازه آن ها پیش پردازش کنید.
- Hu moments را برای هر تصویر محاسبه کنید. Hu moments توصیف گرهای شکلی هستند که نسبت به تغییرات تصویر مانند انتقال، مقیاس و چرخش نامتغیرند.
- یک مدل یادگیری ماشین برای classification انتخاب کنید. و با داده آن را آموزش دهید. در انتها برخی از تصاویر و برجسب های پیش بینی شده آن ها را نمایش دهید.

پاسخ:

برای انجام این پروژه، باید مراحل زیر را طی کنیم:

دانلود مجموعه داده MNIST و استخراج ارقام 0، 1 و 2.

1. پیش پردازش تصاویر (نرمال سازی مقادیر پیکسل ها و در صورت نیاز تغییر اندازه آنها).
2. محاسبه Hu moments برای هر تصویر.
3. انتخاب و آموزش مدل یادگیری ماشین برای طبقه بندی.
4. نمایش برخی از تصاویر و برجسب های پیش بینی شده.

بیاید این مراحل را به ترتیب انجام دهیم:

مرحله 1: دانلود و استخراج داده‌ها

ابتدا مجموعه داده MNIST را دانلود می‌کنیم و فقط ارقام 0، 1 و 2 را استخراج می‌کنیم که چند نمونه‌ی اول آن نیز در نوتبوک با استفاده از plt نمایش داده شده‌اند.

مرحله 2: پیش‌پردازش تصاویر

نرمال‌سازی مقادیر پیکسل‌ها به مقادیر بین 0 و 1 که برای این کار می‌دانیم که 0 تا 255 را به 0 تا 1 تبدیل کنیم و سائز همه‌ی تصاویر را یکی کنیم تا برای process کردن به چالش برنخوریم.

```
1 # MNIST داتلود مجموعه داده
2 (X_train_full, y_train_full), (X_test_full, y_test_full) = mnist.load_data()
3
4 # انتخاب فقط ارقام 0، 1 و 2
5 mask_train = np.isin(y_train_full, [0, 1, 2])
6 mask_test = np.isin(y_test_full, [0, 1, 2])
7
8 X_train = X_train_full[mask_train]
9 y_train = y_train_full[mask_train]
10 X_test = X_test_full[mask_test]
11 y_test = y_test_full[mask_test]
12
13 # نرمال‌سازی مقادیر پیکسل‌ها به محدوده [0, 1]
14 X_train = X_train.astype('float32') / 255.0
15 X_test = X_test.astype('float32') / 255.0
16
17 # نمایش چند نمونه از تصاویر
18 fig, axes = plt.subplots(1, 10, figsize=(10, 1))
19 for i in range(10):
20     ax = axes[i]
21     ax.imshow(X_train[i], cmap='gray')
22     ax.axis('off')
23 plt.show()
```

کد مربوط به پارت اول و دوم اینجا موجود است



در اینجا نیز چند نمونه فقط از دیتاهای 0 و 1 و 2 می‌بینیم.

مرحله 3: محاسبه Hu moments

برای محاسبه Hu moments، ابتدا باید تصاویر را به فرمت مناسب تبدیل کنیم و برای محاسبه hu نیز از cv2 استفاده می کنیم که این الگوریتم را در خود دارد.

کد آن در زیر موجود است:

```
1 def extract_hu_moments(images):
2     hu_moments = []
3     for image in images:
4         # تبدیل تصویر به باینری
5         _, binary_image = cv2.threshold(image, 0.5, 1, cv2.THRESH_BINARY_INV)
6         # OpenCV برای استفاده از uint8 تبدیل تصویر به نوع
7         binary_image = (binary_image * 255).astype(np.uint8)
8         # محاسبه لحظات Hu
9         moments = cv2.moments(binary_image)
10        hu = cv2.HuMoments(moments).flatten()
11        hu_moments.append(hu)
12    return np.array(hu_moments)
13
14 X_train_hu = extract_hu_moments(X_train)
15 X_test_hu = extract_hu_moments(X_test)
16
17 # نرمال سازی ویژگی ها
18 scaler = StandardScaler()
19 X_train_hu = scaler.fit_transform(X_train_hu)
20 X_test_hu = scaler.transform(X_test_hu)
```

حال به مرحله ی بعدی می رسیم:

مرحله 4: انتخاب و آموزش مدل یادگیری ماشین

ما از یک مدل طبقه بندی ساده مانند RandomForestClassifier استفاده می کنیم که کتابخانه ی آماده ی آن نیز در پایتون وجود دارد و دیتاهای خود را به train و test تقسیم می کنیم و سپس آن را fit و predict می کنیم و نتیجه ی accuracy آن نیز پایینتر موجود است.

```

1  # تقسیم داده‌ها به مجموعه‌های آموزش و آزمایش
2  X_train_hu, X_val_hu, y_train, y_val = train_test_split(X_train_hu, y_train, test_size=0.2, random_state=42)
3
4  # آموزش مدل جنگل‌های تصادفی
5  clf = RandomForestClassifier(n_estimators=100, random_state=42)
6  clf.fit(X_train_hu, y_train)
7
8  # پیش‌بینی برجسب‌ها
9  y_val_pred = clf.predict(X_val_hu)
10 y_test_pred = clf.predict(X_test_hu)
11
12 # محاسبه دقت
13 val_accuracy = accuracy_score(y_val, y_val_pred)
14 test_accuracy = accuracy_score(y_test, y_test_pred)
15 print(f'Validation Accuracy: {val_accuracy:.2f}')
16 print(f'Test Accuracy: {test_accuracy:.2f}')

```

نتیجه آن نیز به صورت زیر است:

```

15  print(f'Validation Accuracy: {val_accuracy:.2f}')
16  print(f'Test Accuracy: {test_accuracy:.2f}')
17
Validation Accuracy: 0.86
Test Accuracy: 0.85

```

مرحله 5: نمایش تصاویر و برجسب‌های پیش‌بینی شده

در این قسمت آخر نیز باید برخی از تصاویر و برجسب‌های پیش‌بینی شده آنها را نمایش می‌دهیم که به صورت زیر می‌باشد:

و نتیجه‌ی آن نیز به صورت زیر است:

```

1  # نمایش برخی از تصاویر و برچسبهای پیشبینی شده
2  def plot_images(images, labels_true, labels_pred, num_images=10):
3      plt.figure(figsize=(10, 10))
4      for i in range(num_images):
5          plt.subplot(1, num_images, i + 1)
6          image = images[i]
7          plt.imshow(image, cmap='gray')
8          plt.title(f'True: {labels_true[i]}\nPred: {labels_pred[i]}')
9          plt.axis('off')
10     plt.show()
11
12     # انتخاب چند تصویر برای نمایش
13     num_images = 10
14     indices = np.random.choice(range(len(X_test_hu)), num_images, replace=False)
15     plot_images(X_test[indices], y_test[indices], y_test_pred[indices])
16

```



که برای این ده نمونه تنها یک نمونه اشتباه می باشد و accuracy ما بالاتر از 85 درصد است. در ابتدا از Svm استفاده کردم و برای حالت های مختلف از gridCV استفاده کردم اما بسیار طول کشید تا نتیجه ی حالت های مختلف را ببینم و برای حالت های مختلف زیر 65 Accuracy تا 80 را به دست آوردم که بهترین آن مربوط به 'kernel = 'rbf' بود. اما به دلیل accuracy کم تصمیم گرفتم از random forest استفاده کنم. اما متوجه شدم با شبکه های عصبی بسیار راحت تر می توان accuracy و دقت را بالا برد که داریم:











```

24  # one-hot encoding تبدیل برچسبها به قالب
25  y_train = to_categorical(y_train, 3)
26  y_test = to_categorical(y_test, 3)
27  model = Sequential([
28      Flatten(input_shape=(28, 28)), # D یک بردار 1 تبدیل تصاویر 2
29      Dense(128, activation='relu'), # لایه مخفی با 128 نورون و فعال سازی ReLU
30      Dense(64, activation='relu'), # لایه مخفی با 64 نورون و فعال سازی ReLU
31      Dense(3, activation='softmax') # softmax لایه خروجی با 3 نورون و فعال سازی
32  ])
33  model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
34  history = model.fit(X_train, y_train, epochs=10, validation_split=0.2, batch_size=32)
35  test_loss, test_acc = model.evaluate(X_test, y_test)
36  print(f'Test Accuracy: {test_acc:.2f}')

```

و با توجه به شبکه ی عصبی بالا، دقت به شرح زیر است که بالای 99 درصد می باشد:


```
Epoch 8/10
466/466 [=====] - 2s 5ms/step - loss: 0.0035 - accuracy: 0.9989 - val_loss: 0.0
Epoch 9/10
466/466 [=====] - 2s 5ms/step - loss: 0.0028 - accuracy: 0.9990 - val_loss: 0.0
Epoch 10/10
466/466 [=====] - 3s 7ms/step - loss: 0.0063 - accuracy: 0.9981 - val_loss: 0.0
99/99 [=====] - 0s 4ms/step - loss: 0.0215 - accuracy: 0.9933
Test Accuracy: 0.99
99/99 [=====] - 1s 4ms/step
```

True: 1	True: 0	True: 2	True: 2	True: 2	True: 2	True: 1	True: 1	True: 0	True: 2
Pred: 1	Pred: 0	Pred: 2	Pred: 2	Pred: 2	Pred: 2	Pred: 1	Pred: 1	Pred: 0	Pred: 2
									

که این قسمت در بخش آخر نوتبوک مربوط به mnist نشان داده شده است.