سوال ۱

بخش الف

لایهٔ اول، یعنی لایهٔ Input تنها برای مشخص کردن ساختار ورودی بوده و تاثیری در فرآیند تمرین دادن و پیش‌بینی کردن نهایی ندارد پس در جدول زیر آورده نشده است.

برای محاسبهٔ ابعاد قبل و بعد از هر یک از لایه‌های کانولوشنی و pooling از رابطهٔ زیر استفاده شده است:

$$W_2 = \frac{W_1 - F + 2P}{S} + 1$$
$$H_2 = \frac{H_1 - F + 2P}{S} + 1$$
$$D_2 = D_1$$

برای لایه‌های FC نیز، خروجی به صورت یک بردار با اندازهٔ برابر با تعداد نورون‌های آن لایه است.

برای محاسبهٔ ابعاد تعداد پارامترها در لایه‌های کانولوشنی نیز از رابطهٔ زیر استفاده می‌کنیم:

$$\#Parameters = W_1 \times H_1 \times D_2 + D_2$$

برای لایهٔ FC نیز داریم:

$$\#Parameters = D_{in} \times D_{out}$$

لایهٔ $Pooling$ نیز فاقد پارامتر است.

لایه	ابعاد ورودی	ابعاد خروجی	تعداد پارامترها
Conv2D (32, (9, 9), strides=2, padding='same', activation='relu')	512, 512, 3	256, 256, 32	$9 \times 9 \times 3 \times 32 + 32 = 7808$
MaxPooling2D((4, 4), strides=4)	256, 256, 32	64, 64, 32	0
Conv2D(64, (5, 5), strides=1)	64, 64, 32	60, 60, 64	$5 \times 5 \times 32 \times 64 + 64 = 51264$
AveragePooling2D((2, 2), strides=2)	60, 60, 64	30, 30, 64	0
Conv2D(128, (3, 3), strides=1, padding='valid', activation='relu')	30, 30, 64	28, 28, 128	$3 \times 3 \times 64 \times 128 + 128 = 73856$
Conv2D(128, (3, 3), strides=1, padding='same', activation='relu')	28, 28, 128	28, 28, 128	$3 \times 3 \times 128 \times 128 + 128 = 147584$
MaxPooling2D((2, 2), strides=2)	28, 28, 128	14, 14, 128	0
Conv2D(512, (3, 3), strides=1, padding='valid', activation='relu')	14, 14, 128	12, 12, 512	$3 \times 3 \times 128 \times 512 + 512 = 590336$
GlobalAveragePooling2D()	12, 12, 512	512	0
Dense(1024)	512	1024	$512 \times 1024 + 1024 = 525312$
Dense(10)	1024	10	$1024 \times 10 + 10 = 10250$

بخش ب

در ابتدا رابطهٔ محاسبهٔ تعداد عملیات‌ها برای لایهٔ Covolution را بدست می‌آوریم. تعداد حاصل ضرب‌ها به ازای هر پیکسل برابر است با:

$$K \times K \times C_{in}$$

همچنین نیاز است که بعد از ضرب هر وزن در ورودی مربوطه‌اش، یک جمع انجام دهیم. پس تعداد حاصل جمع‌ها برابر است با:

$$K \times K \times C_{in} - 1$$

منهای یک، به دلیل آن است که یک واحد تعداد عملیات از تعداد کل عبارات کمتر است.

حال از آنجا که تعداد N فیلتر داشته و همچنین اینکه نیاز است که عملیات بالا را به ازای تمام پیکسل‌های موجود در خروجی انجام داد، تعداد کل عملیات جمع و ضرب به صورت زیر می‌شود:

$$Total\ Multiplications = N \times H_{out} \times W_{out} \times (K \times K \times C_{in})$$

$$Total\ Additions = N \times H_{out} \times W_{out} \times (K \times K \times C_{in} - 1)$$

البته با توجه با استفاده از بایاس، برای حاصل جمع داریم:

$$Total\ Additions = N \times H_{out} \times W_{out} \times (K \times K \times C_{in})$$

پس روابط یکسان برای هر دو بدست می‌آید.

در لایهٔ *MaxPooling* از آنجا که مقایسه انجام می‌گیرد، و فرض می‌کنیم که مقایسه فاقد جمع و ضرب است، هیچ عملیات جمع و ضربی صورت نمی‌گیرد.

در لایهٔ *AveragePooling* با اندازهٔ کرنل K، $K^2 - 1$ عملیات جمع به ازای هر پیکسل صورت می‌گیرد. همچنین نیاز است که به ازای هر پیکسل در خروجی این کار را انجام دهیم؛ یعنی $W_1 \times H_1$. همچنین از آنجا که D_1 کانال ورودی داریم و بر روی تمام این کانال‌ها عملیات جمع را انجام می‌دهیم، پس داریم:

$$D_1 \times W_1 \times H_1 \times (K^2 - 1)$$

برای لایهٔ *GlobalAveragePooling* نیز همینکار را انجام می‌دهیم تنها در مقیاس تمام ابعاد ورودی و به ازای هر کانال. یعنی داریم:

$$D_1 \times (W_1 \times H_1 - 1)$$

برای لایهٔ *Dense* نیز تعداد هر عملیات از طریق رابطهٔ زیر بدست می‌آید (برای جمع بدلیل وجود بایاس برابر با ضرب شده است.)

$$D_1 \times D_2$$

تعداد حاصل ضرب‌ها	تعداد حاصل جمع‌ها	ابعاد خروجی	ابعاد ورودی	لایه

Conv2D (32, (9, 9), strides=2, padding='same', activation='relu')	512, 512, 3	256, 256, 32	$256 \times 256 \times 32 \times 9 \times 9 \times 3 = 507510784$	$256 \times 256 \times 32 \times 9 \times 9 \times 3 = 507510784$
MaxPooling2D((4, 4), strides=4)	256, 256, 32	64, 64, 32	0	0
Conv2D(64, (5, 5), strides=1)	64, 64, 32	60, 60, 64	$5 \times 5 \times 32 \times 64 \times 60 \times 60 = 18432000$	$5 \times 5 \times 32 \times 64 \times 60 \times 60 = 18432000$
AveragePooling2D((2, 2), strides=2)	60, 60, 64	30, 30, 64	$(2 \times 2 - 1) \times 30 \times 30 \times 64 = 172800$	$1 \times 30 \times 30 \times 64 = 57600$
Conv2D(128, (3, 3), strides=1, padding='valid', activation='relu')	30, 30, 64	28, 28, 128	$3 \times 3 \times 64 \times 28 \times 28 \times 128 = 57527808$	$3 \times 3 \times 64 \times 28 \times 28 \times 128 = 57527808$
Conv2D(128, (3, 3), strides=1, padding='same', activation='relu')	28, 28, 128	28, 28, 128	$3 \times 3 \times 128 \times 28 \times 28 \times 128 = 115055616$	$3 \times 3 \times 128 \times 28 \times 28 \times 128 = 115055616$
MaxPooling2D((2, 2), strides=2)	28, 28, 128	14, 14, 128	0	0
Conv2D(512, (3, 3), strides=1, padding='valid', activation='relu')	14, 14, 128	12, 12, 512	$3 \times 3 \times 128 \times 12 \times 12 \times 512 = 84869376$	$3 \times 3 \times 128 \times 12 \times 12 \times 512 = 84869376$
GlobalAveragePooling2D()	12, 12, 512	512	$512 \times (12 \times 12 - 1) = 73216$	$1 \times 512 = 512$
Dense(1024)	512	1024	$1024 \times 512 = 524288$	$1024 \times 512 = 524288$
Dense(10)	1024	10	$1024 \times 10 = 10240$	$1024 \times 10 = 10240$

برای لایه‌های مربوط به میانگین‌گیری، هم حاصل جمع استفاده شده است و هم تقسیم (که نوعی حاصل ضرب است). همچنین برای لایه‌های ماکس‌گیری اینطور فرض شده است که مقایسه، جدا از جمع کردن و ضرب کردن است.

بخش ج

در حال حاضر تعداد کل پارامترها برابر با 1406410 است. حال اگر از لایهٔ Flatten استفاده کنیم، در تعداد پارامترهای لایهٔ یکی مانده به آخری، تغییر بوجود خواهد آمد. به طوری که داریم:

لایه	ابعاد ورودی	ابعاد خروجی	تعداد پارامترها
Flatten()	12, 12, 512	73728	0
Dense(1024)	73728	1024	$73728 \times 1024 + 1024 = 75498496$
Dense(10)	1024	10	$1024 \times 10 + 10 = 10250$

در چنین حالتی تعداد کل پارامترها برابر خواهد شد با 76379594. بنابراین خواهیم داشت:

$$Increase\ Factor = \frac{76379594}{1406410} = 53.68$$

سوال ۲

با توجه به اینکه مقدار اولیهٔ داده شده برای x برابر با ۲۰ بوده، و نیز تابع ضرر نیز داده شده است، پس داریم:

$$L = x^2 - 10x + e^{0.0001} \rightarrow \frac{dL}{dx} = 2x - 10 \xrightarrow[x=20]{} 30 = \frac{dl}{dx}(x = 20)$$

بنابراین مقدار ثانویهٔ x از طریق رابطهٔ زیر بدست می‌آید:

$$x_2 = x - \alpha \times \frac{dl}{dx} \xrightarrow{x=20} x_2 = 20 - \alpha \times 30$$

با توجه به گزینه‌های داده شده، مقدار نرخ آموزش را برای هر یک بدست می‌آوریم:

$$1)\ x = 14 \rightarrow 14 = 20 - \alpha \times 30 \rightarrow \alpha = 0.2$$

$$2)\ x = 19.4 \rightarrow 19.4 = 20 - \alpha \times 30 \rightarrow \alpha = 0.02$$

$$3)\ x = 19.4 \rightarrow 19.94 = 20 - \alpha \times 30 \rightarrow \alpha = 0.002$$

نرخ آموزش بر روی آموزش مدل به این صورت تاثیر می‌گذارد که هر چه بزرگ‌تر باشد، میزان گام‌های برداشته شده جهت بهینه‌سازی را بزرگ‌تر می‌کند. با توجه به نمودارهای داده شده، نمودار نارنجی‌رنگ بیشترین میزان نرخ آموزش را دارد. چرا که در ابتدا خیلی سریع پیش رفته و شیب زیادی را تجربه کرده، و سپس متوقف شده است. این موضوع با توجه با این نکته است که تابع ضرر داده شده تنها دارای یک نقطهٔ مینیمم بوده و مینیمم محلی ندارد. از طرفی با توجه به اختلاف بین دو طرف نمودار تابع ضرر مقدار آلفای بزرگ (0.2) در اینجا باعث ایجاد Overshoot نشده است که در نمودار مورد نظر صدق می‌کند. بنابراین نمودار نارنجی مربوط به نرخ آموزش شمارهٔ ۳ است.

نرخ آموزش کوچک نیز باعث می‌شود که سرعت حرکت پارامتر به سمت مقدار بهینه کند شود؛ طوری که گرچه به سمت نقطهٔ بهینه حرکت می‌کند اما مدت زمان آموزش طولانی می‌شود. با توجه به اینکه تابع ضرر داده شده تنها یک مینیمم دارد، پس پارامتر در مسیر رسیدن به مینیمم اصلی، در مینیمم‌های محلی گیر نخواهد کرد. در نتیجه کاهش میزان نرخ آموزش تنها باعث کندی روند آموزش می‌شود. این موضوع در نمودار آبی رنگ قابل مشاهده است که به کندی دارد به سمت نقطهٔ بهینه پیش می‌رود. پس این نمودار مربوط به نرخ آموزش شمارهٔ ۱ است.

از طرفی نرخ آموزش متوسط نه به کندی پیش می‌رود و نه خیلی سریع. این موضوع در نمودار Loss سبز رنگ قابل مشاهده است که مقدار Loss دارد در حد متوسطی تغییر می‌کند. بنابراین این نمودار مربوط به نرخ آموزش شمارهٔ ۲ است.

نرخ آموزش	نمودار
سوم	آبی
دوم	سبز
اول	نارنجی

سوال ۳

در ماژول Inception، در هر یک از مراحل میانی، با استفاده از padding از تغییر طول و عرض تصویر ورودی جلوگیری می‌کنیم. چرا که در مرحلهٔ نهایی نیاز است که خروجی بدست آمده از هر مسیر را با یکدیگر Concatenate کنیم. لذا تنها تفاوت در تعداد کانال‌ها امکان‌پذیر است و طول و عرض خروجی‌ها بایستی مشابه به ورودی‌ها باقی بماند. با توجه به توضیحات داده شده، برای لایه‌های میانی خواهیم داشت:

- مسیر $conv(1 \times 1)$:

$$(12, 12, 32) \rightarrow (12, 12, 64)$$

- مسیر $conv(3 \times 3)$:

$$(12, 12, 32) \rightarrow (12, 12, 64) \rightarrow (12, 12, 32)$$

- مسیر $conv(5 \times 5)$:

$$(12, 12, 32) \rightarrow (12, 12, 64) \rightarrow (12, 12, 128)$$

- مسیر $maxpool$:

$$(12, 12, 32) \rightarrow (12, 12, 32) \rightarrow (12, 12, 64)$$

در نتیجه، برای بدست آوردن خروجی نهایی، خروجی هر مسیر را با یکدیگر Concatenate می‌کنیم.

$$(12, 12, 64 + 32 + 128 + 64) = (12, 12, 288)$$

با تغییر تعداد کرنل‌های استفاده شده در فیلتر مورد نظر، با توجه به اینکه یک فیلتر ۵ در ۵ بعد از این فیلتر قرار گرفته است، تغییری در تعداد کانال‌های خروجی قرار گرفته در مسیر این لایه بوجود نخواهد آمد، و خروجی این مسیر ابعاد *(12, 12, 128)* خواهد شد.

سوال ۴

بخش الف)

در ابتدا توضیحات اولیه را می‌دهیم:

- در هر دوره (epoch)، کل داده‌های آموزشی یک بار از ابتدا تا انتها طی می‌شوند.
- برای رسیدن به هدف بالا، در هر دوره داده‌ها به دسته‌های کوچک‌تر (mini-batches) با اندازهٔ b تقسیم می‌شوند.
- تعداد دسته‌ها در هر دوره برابر است با $\left\lceil \frac{t}{b} \right\rceil$. علت گرفتن سقف آن است که ممکن است که دو عدد بر هم بخش پذیر نباشند و دستهٔ آخر تعداد اعضای کمتری نسبت به دسته‌های پیشین داشته باشد.

بنابراین تعداد کل به‌روزرسانی‌ها برابر است با:

$$e \times \left\lceil \frac{t}{b} \right\rceil$$

بخش ب)

با توجه به نوسان موجود در تغییر میزان تابع ضرر و نویزی بودن حرکت آن، می‌توان نتیجه گرفت که نمودار داده شده مربوط به نمودار mini-batch GD است. چرا که در این مدل بهینه‌سازی، در هر گام تنها یک بخشی از داده‌های آموزشی را در نظر می‌گیریم و نه تمام آن را. در نتیجه، بر اثر در نظر گرفتن این بخش از داده‌ها، باعث می‌شود که پارامترها در جهتی که برای این دسته از داده‌ها بهینه است حرکت کند و نه در مسیر گرادیان تمام داده‌های ورودی. به همین دلیل مقداری جهت‌دهی در مسیر تغییر داده بوجود خواهد آمد که در دور بعدی و با در نظر گرفتن دستهٔ بعد این جهت‌دهی تا حدودی اصلاح خواهد شد. در نهایت و بر اثر برهم‌نهی تمام این تغییرات با یکدیگر به مقدار بهینه دست خواهیم یافت. این در حالی است که با استفاده از Batch GD در هر گام گرادیان را به ازای تمام داده‌های آموزشی حساب می‌کنیم و در همان جهت حرکت می‌کنیم. بنابراین گامی که طی می‌کنیم در جهت بهبودی پارامترها به ازای تمام داده‌های ورودی است.

بخش ج)

نمودار اول:

- **تابع ضرر برای داده‌های آموزشی و اعتبارسنجی نزدیک به هم بوده و هر دو بالا هستند.**
- این وضعیت نشان‌دهنده این است که مدل به خوبی روی داده‌های آموزشی آموزش ندیده و همچنین بر روی داده‌های اعتبارسنجی نیز عملکرد خوبی ندارد. این حالت معمولاً بیانگر آن است که مدل به اندازهٔ کافی پیچیده نیست یا به اصطلاح کم‌برازش (Underfitting) دارد.

نمودار دوم:

- **ضرر داده آموزشی پایین آورده شده است و به صفر رسیده در حالی که اعتبارسنجی در حال کاهش یافتن است با شیبی کم و در آخرین گام به مقدار ۰.۰۶ رسیده است.**

- این وضعیت نشان‌دهنده این است که مدل به خوبی داده‌های آموزشی را یاد گرفته است (احتمالاً حتی بیش از حد خوب یاد گرفته که به آن بیش‌برازش (Overfitting) گفته می‌شود) اما عملکردش روی داده‌های اعتبارسنجی نتوانسته که به خوبی داده‌های آموزشی باشد. به عبارت دیگر، مدل توانایی تعمیم به داده‌های جدید را ندارد.

پیشنهاد اعمال برای نمودار اول:

۱. **افزایش لایه های شبکه:**

- **چرا؟** زیرا مدل احتمالاً به اندازه کافی پیچیده نیست و افزودن لایه‌های بیشتر یا نرون‌های بیشتر در هر لایه می‌تواند به افزایش ظرفیت مدل و بهبود عملکرد آن کمک کند.

۲. **داده افزایی:**

- **چرا؟** هرچند که داده‌افزایی بیشتر در زمانی به کار می‌آید که عملکرد مدل بر روی دادهٔ اعتبارسنجی بد بوده و بر روی دادهٔ آموزشی خوب باشد، اما با این‌حال استفاده از داده‌افزایی بی‌ضرر بوده و حتی ممکن است که باعث بهبود کیفیت مدل شود، به شرط اینکه به طرز مناسبی این عملیات انجام گیرد.

پیشنهاد اعمال برای نمودار دوم:

۱. **داده افزایی:**

- **چرا؟** داده افزایی می‌تواند به مدل کمک کند تا بهتر تعمیم یابد و از بیش‌برازش جلوگیری کند. با افزایش تنوع داده‌های آموزشی، مدل می‌تواند یاد بگیرد که به الگوهای بیشتری واکنش نشان دهد و بر روی داده‌های جدید عملکرد بهتری داشته باشد.

۲. **کاهش تعداد ویژگی های ورودی:**

- **چرا؟** اگر مدل دارای ویژگی‌های زیاد یا بی‌کیفیت است، کاهش تعداد ویژگی‌ها می‌تواند به ساده‌تر شدن مدل و بهبود عملکرد کمک کند. این کار می‌تواند به مدل کمک کند تا روی ویژگی‌های مهم‌تر تمرکز کند و به کاهش بیش‌برازش کمک کند.

سوال ۵

بخش الف:

در هر گام یک پنجرهٔ سه در سه را در نظر می‌گیریم و عنصر مرکزی را به عنوان آستانه در نظر گرفته و باقی پیکسل‌ها را با آن مقایسه می‌کنیم و در پایان با شروع از پیکسل بالا چپ، به صورت ساعتگرد حرکت می‌کنیم و کد باینری مربوط به پیکسل را بدست می‌آوریم.

برای LBP داده شده یک دایره به شعاع یک و تعداد پیکسل ۸ را در نظر می‌گیریم. داریم:

- پیکسل ۲۵۰:

0	0	0	کد باینری	کد دسیمال
0		0	00000000	0
0	0	0		

- پیکسل ۲۰۰:

0	0	0	کد باینری	کد دسیمال
1		0	00000001	1
0	0	0		

- پیکسل ۵۰:

0	0	0	کد باینری	کد دسیمال
1		0	00000111	7
1	1	0		

- پیکسل ۱۸۰:

0	1	1	کد باینری	کد دسیمال
0		0	01100100	100
0	1	0		

- پیکسل ۱۰۰:

1	1	0	کد باینری	کد دسیمال
1		0	11000011	195
1	0	0		

- پیکسل ۸۰:

1	0	0	کد باینری	کد دسیمال
1		0	10000001	129

		0	0	0

- پیکسل ۲۰۰:

کد دسیمال	کد باینری			
0	00000000	0		0
		0		0
		0	0	0

- پیکسل ۴۰:

کد دسیمال	کد باینری			
241	11110001	1	1	1
		1		1
		0	0	0

- پیکسل ۷۰:

کد دسیمال	کد باینری			
192	11000000	0	1	1
		0		0
		0	0	0

بخش ب: در صورت جمع کردن شدت روشنایی‌های مربوط به پیکسل‌ها با یک عدد ثابت، تغییری در کد LBP نهایی مربوط به پیکسل‌ها بوجود نمی‌آید چرا که در LBP کدهای نهایی با مقایسهٔ پیکسل‌های موجود در شعاع تحت بررسی با پیکسل مرکزی، بدست می‌آیند. بنابراین وقتی همگی تا یک اندازه شدت روشنایی‌شان بیشتر شود، تفاوتی در نتیجهٔ نهایی بوجود نخواهد آورد.

در اثر ضرب در یک عدد مثبت نیز توضیح داده، صدق می‌کند.

البته توضیحات داده شده، زمانی صحیح هستند که باعث اشباع پیکسل‌ها نشوند. یعنی چندین پیکسل با اندازهٔ متفاوت به مقدار ۲۵۵ نرسد. چرا که در این شرایط کد LBP بدست آمده تغییر خواهد کرد.

بخش ج: الگوریتم LBP بافت تصویر ورودی را مورد ارزیابی قرار می‌دهد و متناسب با آن یک کد را به ما تحویل می‌دهد. بنابراین می‌توان با استدلال بر روی بافت هر تصویر، آن را مورد ارزیابی قرار دهیم.

در ابتدا از تصویر با بافت ساده‌تر آغاز می‌کنیم. تصویر دوم، ساده‌ترین بافت را داشته طوری که تنها از خطوط سفید و سیاه تشکیل شده است. با توجه به اینکه بخش زیادی از این تصویر را نواحی flat که بدون تغییر هستند فرا گرفته است و همچنین گوناگونی بافت در این تصویر کم است، می‌توان گفت، نموداری که مقدار بالاتر منتسب به کد گفته شده بالاتر است، مربوط به این تصویر است. از آنجا که در چنین ناحیه‌ای پیکسل مرکزی با تمام پیکسل‌های اطرافش برابر است، پس کد مورد نظر عدد ۲۵۵ می‌باشد. لذا نمودار C مربوط به تصویر دوم است.

در تصویر اول، نسبت به تصویر سوم بافت‌ها ساده‌تر هستند و تا حدودی تکرار شونده. همچنین با توجه به اینکه بخشی از پیش‌زمینهٔ تصویر، به رنگ یکنواخت مشکی در آمده است، پس می‌توان گفت که این تصویر پراکندی کمتری در مقادیر داشته و نیز فراوانی مربوط به کد ۲۵۵ در آن، نسبت به تصویر سوم بیشتر است. بنابراین نمودار B مربوط به این تصویر است.

در تصویر سوم، تا حدودی می‌توان گفت سنگ‌های موجود در تصویر، یک تصویر شبه نویزی را به وجود آورده‌اند و بافت مشخصی را تشکیل نداده‌اند و لذا در نمودار هیستوگرام آن، پراکندگی بیشتری را شاهد خواهیم بود. بنابراین نمودار A مربوط به این تصویر است.

تصویر	نمودار
اول	B
دوم	C
سوم	A

سوال ۶

بخش الف

در ابتدا دیتا را به صورت زیر لود می‌کنیم:

```
# Load the Dogs vs. Cats dataset
train_dataset, info = tfds.load('cats_vs_dogs', split='train[:80%]', with_info=True, as_supervised=True)
test_dataset = tfds.load('cats_vs_dogs', split='train[80%:]', with_info=False, as_supervised=True)
```

علت فعال‌سازی as_supervised آن است که می‌خواهیم یک مدل برای دسته‌بندی ایجاد کنیم. در نتیجه نیاز است که داده‌ها به صورت تصویر و لیبل باشند. این عمل را برای هر دوی داده‌های آموزشی و ارزیابی انجام می‌دهیم.

در ادامه تعداد هر یک از داده‌ها را پرینت می‌کنیم:

```
# Observe the size of training and test data
len(train_dataset), len(test_dataset)
```

```
(18610, 4652)
```

حال در ادامه ساختار هر تصویر را چاپ می‌کنیم. با چاپ کردن آن‌ها متوجه می‌شویم که ساختار تصاویر با یکدیگر متفاوت بوده و نیاز است که تمام این داده‌ها را resize کنیم که داخل تابع preprocess قرار گرفته شده است.

```
# Observe the shape of data
for image, label in train_dataset.take(3):
    print(f"Image shape: {image.shape}, Label: {label.numpy()}")
```

```
Image shape: (262, 350, 3), Label: 1
Image shape: (409, 336, 3), Label: 1
Image shape: (493, 500, 3), Label: 1
```

در ادامه هایپرپارامترها را تنظیم می‌کنیم.

```
# Defining hyperparametrs and constants
IMG_SIZE = 128
BATCH_SIZE = 32
EPOCH_SIZE = 20
NUM_CLASSES = info.features['label'].num_classes
TEST_SIZE = len(test_dataset) // 2
VALIDATION_SIZE = len(test_dataset) - TEST_SIZE
TRAIN_SIZE = len(train_dataset)
```

همچنین نیاز است که هر یک از تصاویر را نیز نرمالایز کرد به این معنی که مقادیر مربوط به شدت هر پیکسل را به بازهٔ ۰ تا ۱ مپ بکنیم. این عمل را به همراه عمل ذکرشدهٔ پیشین داخل تاب preprocess قرار داده شده است.

```python
#preprocess the data
def preprocess(image, label):
    image = tf.image.resize(image, (IMG_SIZE, IMG_SIZE))
    image = image / 255.0  # Normalize to [0, 1] range
    return image, label

# Apply the preprocessing to the datasets
train_dataset = train_dataset.map(preprocess, num_parallel_calls=tf.data.experimental.AUTOTUNE)
test_dataset = test_dataset.map(preprocess, num_parallel_calls=tf.data.experimental.AUTOTUNE)
```

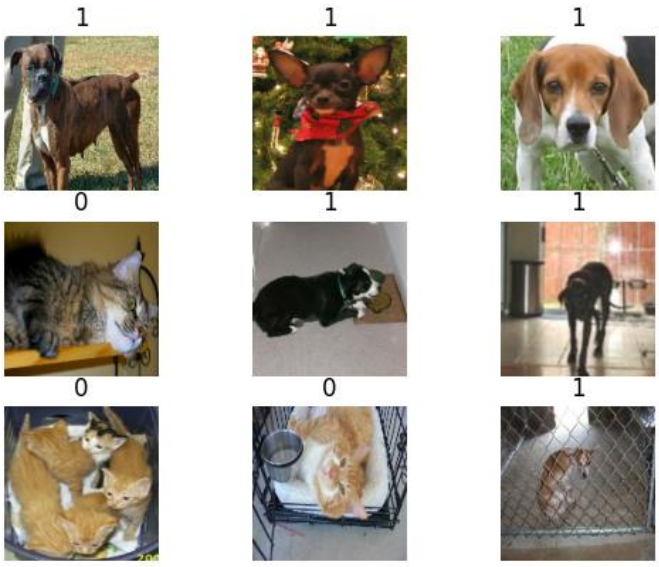
جهت اعمال تابع مورد نظر روی داده‌ها، نیاز است که به طور موثر انجام دهیم. این کار را به صورت بالا انجام می‌دهیم.

حال دوباره ساختار تصاویر را چاپ می‌کنیم تا تغییرات اعمال‌شده را مشاهده کنیم.

```python
    # Observe shape of data
    for image, label in train_dataset.take(3):
        print(f"After Preprocessing --- Image shape: {image.shape}, Label: {label.numpy()}")
```

```
After Preprocessing --- Image shape: (128, 128, 3), Label: 1
After Preprocessing --- Image shape: (128, 128, 3), Label: 1
After Preprocessing --- Image shape: (128, 128, 3), Label: 1
```

حال تعدادی از تصاویر را به نمایش می‌گذاریم.



حال نیاز است که داده‌های تست را به دو بخش تست و ارزیابی تبدیل کنیم. برای این کار از قطعه کد زیر استفاده می‌کنیم و به دو بخش مساوی تقسیم می‌کنیم:

```python
# Split the test dataset into validation and test datasets
validation_dataset = test_dataset.take(VALIDATION_SIZE)
test_dataset = test_dataset.skip(VALIDATION_SIZE)
```

در گام بعدی، در ابتدا دادهٔ آموزشی را شافل می‌کنیم و سپس آن را به چندین بچ تبدیل می‌کنیم. داده‌های تست و ارزیابی را نیز به چندین بچ تبدیل می‌کنیم.

```
# Shuffle and batch the datasets
train_dataset = train_dataset.shuffle(buffer_size=1000).batch(BATCH_SIZE).prefetch(tf.data.experimental.AUTOTUNE)
test_dataset = test_dataset.batch(BATCH_SIZE).prefetch(tf.data.experimental.AUTOTUNE)
validation_dataset = validation_dataset.batch(BATCH_SIZE).prefetch(tf.data.experimental.AUTOTUNE)
```

در گام بعدی، نیاز است که مدل را طراحی کنیم و آن را بر روی داده‌ها آموزش دهیم.

```
# Building the model
model = Sequential([
    layers.Input(shape=(IMG_SIZE, IMG_SIZE, 3)),
    layers.Conv2D(32, (3, 3), activation='relu'),
    layers.Conv2D(32, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.GlobalAveragePooling2D(),
    layers.Dense(64, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(1, activation='sigmoid')
])
```

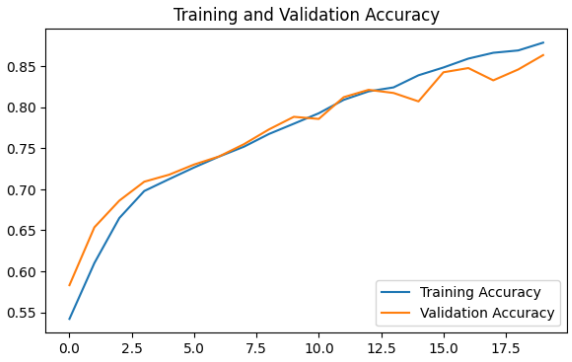
لایهٔ نهایی sigmoid بوده چرا که نیاز است که تنها صفر یا یک را پیش‌بینی کند که مشخص‌کنندهٔ گربه یا سگ بودن است.

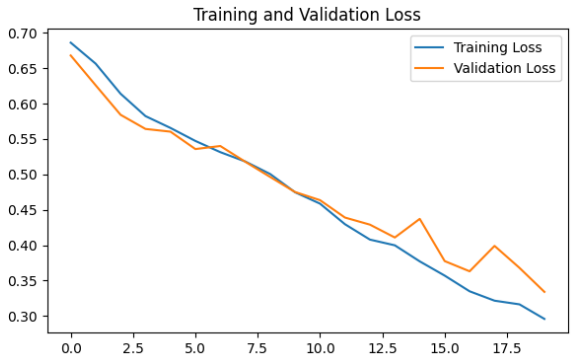
برای آموزش مدل دو callback تعریف می‌کنیم. یکی برای ذخیره کردن بهترین مدل و دیگری برای متوقف کردن زودهنگام مدل در اثر عدم تغییر بر روی val_loss و یا کاهش عملکرد مدل با توجه به این مورد. با توجه به اینکه دو لیبل داریم، نیاز است که تابع ضرر را برابر با binary_crossentropy قرار دهیم.

دقت مدل بر روی داده‌های تست برابر خواهد شد با:

```
# load model and evaluate on test dataset
model.load_weights("best_model.keras")
test_loss, test_acc = model.evaluate(test_dataset)
print(f"Test accuracy: {test_acc}")


73/73 [==============================] - 6s 43ms/step - loss: 0.3072 - accuracy: 0.8774
Test accuracy: 0.8773747682571411
```

در پایان بهترین مدل را ذخیره می‌کنیم و دوباره هم آن را لود می‌کنیم و بر روی داده‌های تست آن را ارزیابی می‌کنیم. در ادامه نمودار دقت مدل و نیز میزان ضرر بدست آمده برای آن را در داده‌های تست و آموزشی، رسم می‌کنیم.





بخش ب

برای بخش ب نیز مانند بخش الف پیش می‌رویم. بنابراین ابتدای کد را توضیح نمی‌دهم. تنها بخشی که تغییر دادم مربوط به سایز تصویر ورودی بود که از ۱۲۸ به ۲۲۴ تبدیل کردم.

برای آنکه مدل را ایجاد کنم، در ابتدا یک مدل InceptionV3 را با وزن‌های مربوط به ImageNet را لود می‌کنیم.

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

در ادامه یک مدل را درست می‌کنیم که مدل بیس در ابتدای آن قرار گیرد و لایهٔ بالایی آن برداشته شود. برای لایهٔ رویی نیاز است که در ابتدا از یک GAP استفاده کنیم و در ادامه با استفاده از یک Dropout شرایط را برای مدل پیچیده‌تر کنیم تا بتواند یادگیری بیشتری را بر روی داده‌های آموزشی انجام دهد.

همچنان در هنگام آموزش نیاز است که قابل آموزش بودن مدل بیس را برداریم. سپس مدل نهاییی را می‌سازیم.

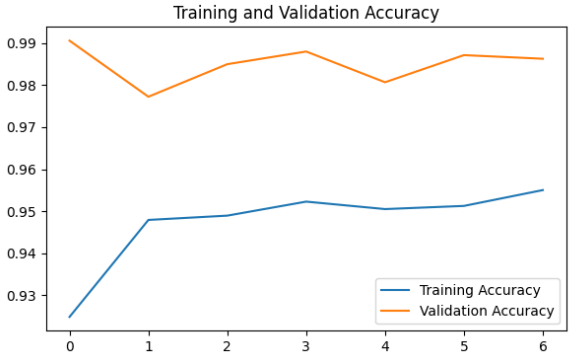
```
#create your model
#dont forget to freeze the pretrained part

# Freeze the layers of the base model
inception_model.trainable = False

# Define the final model
model = models.Sequential([
    inception_model,
    layers.GlobalAveragePooling2D(),
    layers.Dropout(0.3),
    layers.Dense(1, activation='sigmoid')  # For binary classification (cats vs dogs)
])
```

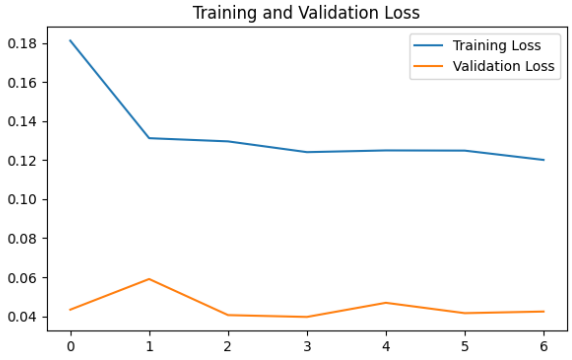
چک پوینت‌ها و سایر هایپرپارامترها مانند سابق تنظیم شده‌اند. در نهایت نتیجهٔ نهایی پس از تمرین دادن مدل و تست کردن بر روی داده‌های تست به صورت زیر خواهد شد.

```
# load model and evaluate on test dataset
# print(test_dataset)
model.load_weights("best_model.keras")
test_loss, test_acc = model.evaluate(test_dataset)
print(f"Test accuracy: {test_acc}")
```

```
73/73 [==============================] - 8s 73ms/step - loss: 0.0467 - accuracy: 0.9837
Test accuracy: 0.9836629629135132
```

نمودار مربوط به دقت مدل و نیز ضرر آن بر روی داده‌های آموزشی و نیز اعتبارسنجی به صورت زیر خواهد شد.





سوال ۷

در ابتدا داده‌ها را لود می‌کنیم و سپس آن را نرمال می‌کنیم.

```
# Load the dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Convert data to float32 and normalize
x_train, x_test = x_train.astype('float32') / 255.0, x_test.astype('float32') / 255.0
```

در گام بعدی نیاز است که داده‌های مربوط به لیبل‌های ۰ تا ۲ را جدا کنیم. برای این کار از تابع np.where استفاده می‌کنیم. سپس با استفاده از اندیس‌های بدست آمده داده‌های مورد نظر را استخراج می‌کنیم.

```
train_indeces = np.where((y_train == 0) | (y_train == 1) | (y_train == 2))
test_indeces = np.where((y_test == 0) | (y_test == 1) | (y_test == 2))

x_train, y_train = x_train[train_indeces], y_train[train_indeces]
x_test, y_test = x_test[test_indeces], y_test[test_indeces]
```

در ادامه مقدار hu moments را برای هر یک از تصاویر محاسبه می‌کنیم. بنابراین برای هر تصویر یک بردار هفت‌تایی خواهیم داشت که از momentهای تصویر بدست آمده‌اند. برای این منظور از تابع زیر استفاده می‌کنیم:

```
# Compute Hu moments for each image
def calculate_hu_moments(images):
    hu_moments = []
    for img in images:
        moments = cv2.moments(img)
        hu = cv2.HuMoments(moments).flatten()
        hu_moments.append(hu)
    return np.array(hu_moments)

hu_train = calculate_hu_moments(x_train)
hu_test = calculate_hu_moments(x_test)
```

در ادامه بردارهای بدست آمده را نرمال می‌کنیم. جهت نرمال کردن از StandardScalar استفاده می‌کنیم تا میانگین صفر شده و واریانس بردارها به ۱ تبدیل شود.

```
# Standardize features by removing the mean and scaling to unit variance
scaler = StandardScaler()
hu_train = scaler.fit_transform(hu_train)
hu_test = scaler.transform(hu_test)
```

در ادامه خروجی‌ها را categorical می‌کنیم و یک بردار one hot را به ازای هر یک از لیبل‌ها بدست می‌آوریم.

```
# Categorize the labels
y_train_categorical = to_categorical(y_train, 3)
y_test_categorical = to_categorical(y_test, 3)
```

حال مدل را طراحی می‌کنیم و آن را آموزش می‌دهیم.

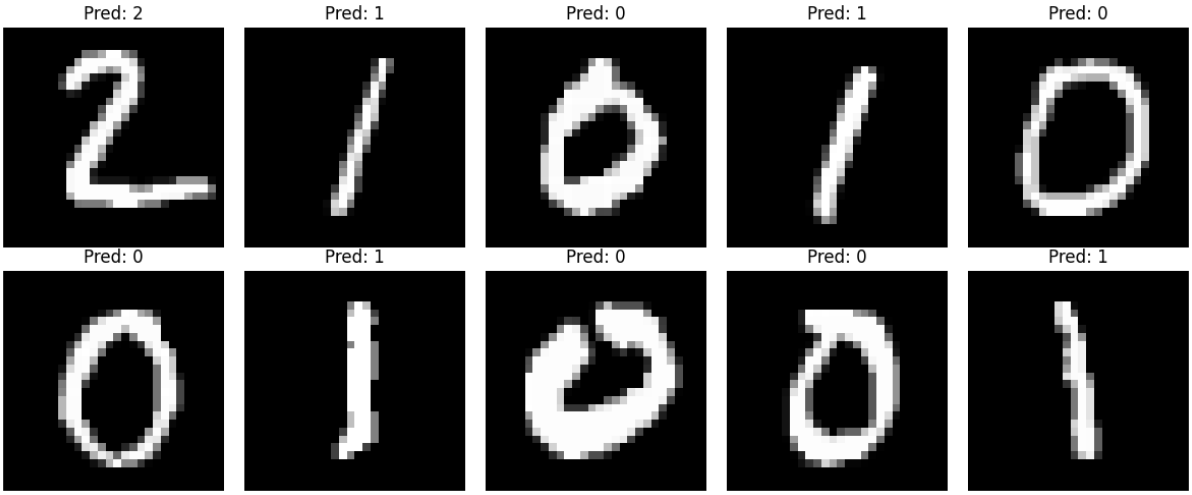
```
# Define the MLP model
model = Sequential([
    Input(shape=hu_train.shape[1]),
    Dense(128, activation='relu'),
    Dropout(0.2),
    Dense(64, activation='relu'),
    Dropout(0.2),
    Dense(3, activation='softmax')
])

model.summary()
```

سپس با استفاده از بهینه‌ساز Adam و با نرخ آموزش 0.001 مدل را آموزش می‌دهیم. همچنین از آنجا که سه خروجی داریم، نیاز است که از Cross-entropy استفاده کنیم. تعداد بچ‌ها را روی ۳۲ ست می‌کنیم و داده‌های آموزشی را به دو داده‌های آموزشی و نیز اعتبارسنجی تقسیم می‌کنیم و مدل را به اندازهٔ ۲۰ دور برای آموزش قرار می‌دهیم.

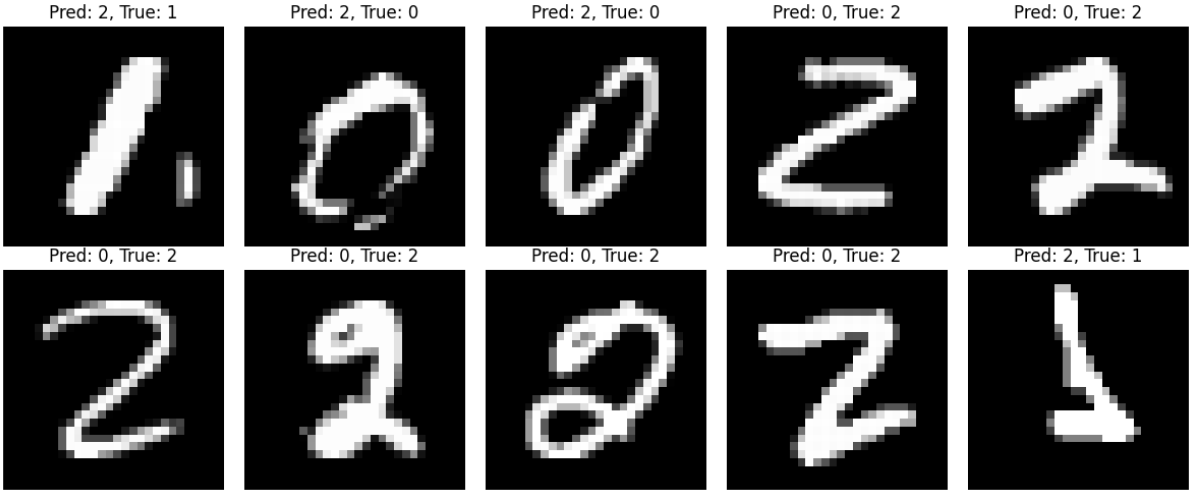
نتیجهٔ بدست آمده بر روی دادهٔ تست به صورت زیر خواهد شد:

```
    # Evaluate the model
    test_loss, test_accuracy = model.evaluate(hu_test, y_test_categorical)
    print(f'Accuracy on test data: {test_accuracy * 100:.2f}%')


99/99 [==============================] - 0s 4ms/step - loss: 0.1719 - accuracy: 0.9352
Accuracy on test data: 93.52%
```

تعدادی از تصاویر به همراه خروجی‌های پیش‌بینی شده به صورت زیر هستند:



همچنین تعدادی از مواردی که مدل به اشتباه پیش‌بینی کرده است نیز به صورت زیر هستند:



کد مربوط به رسم نمودار نهایی به صورت زیر است:

```
wrong_labels = np.where(y_test != predicted_labels)[0]
ten_wrong_labels = np.random.choice(wrong_labels, size=10, replace=False)

# Display some images with their predicted labels
fig, axes = plt.subplots(2, 5, figsize=(12, 5))
axes = axes.ravel()

for i in range(len(ten_wrong_labels)):
    axes[i].imshow(x_test[ten_wrong_labels[i]], cmap='gray')
    axes[i].set_title(f'Pred: {predicted_labels[ten_wrong_labels[i]]}, True: {y_test[ten_wrong_labels[i]]}')
    axes[i].axis('off')

plt.tight_layout()
plt.show()
```