به نام خدا

تمرین سری سوم

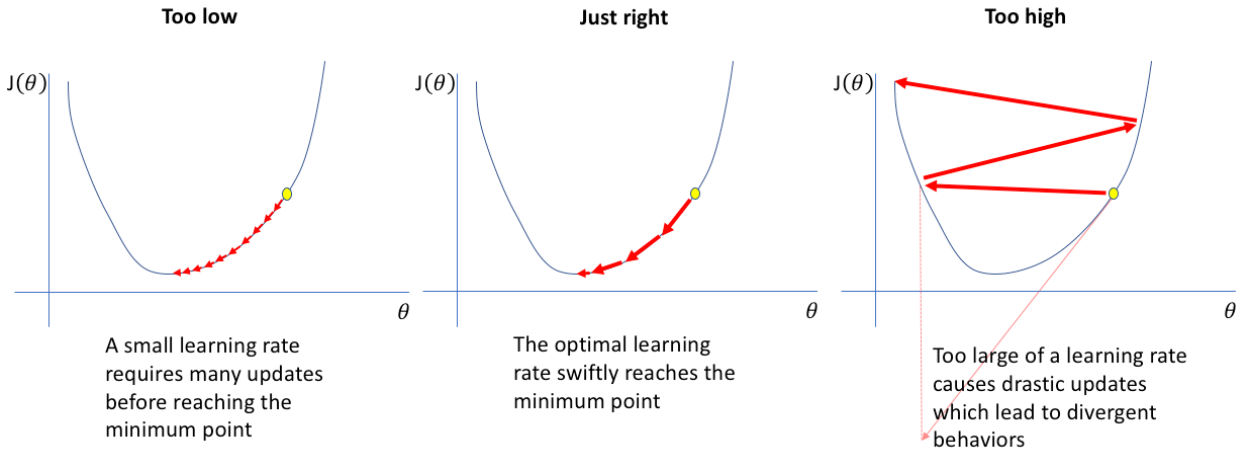
درس یادگیری عمیق

استاد مرضیه داودآبادی

سینا علی‌نژاد ۹۹۵۲۱۴۶۹

سوال ۱-

الف) نرخ یادگیری بالا باعث میشود مدل ما رفتار باثباتی نداشته باشد و حتی واگرا شود و مقدار ضرر آن در هر مرحله بیشتر و بیشتر شود.



طریقه تشخیص آن میتواند به کمک کشیدن نمودار تابع ضرر باشد. تصویر بالا، نمودار اول از سمت راست نشان دهنده مدلی است که از نرخ یادگیری بزرگ استفاده کرده است. همانطور که مشخص است، مقدار ضرر بیشتر میشود، البته ممکن است کلا از این قسمت از تابع ضرر خارج شود به نقاط دیگر برود و شاید بعضی مواقع مقدار ضرر نسبت به گام قبل کمتر هم بشود اما مشخصترین نکته درباره آن، همین stable نبودن و بالا و پایین شدن است. در این حالت، پس از چند epoch مقدار loss عدد بزرگی میشود و در پایتون مقدار nan میدهد. پس این هم میتواند در تشخیص این مشکل کمک کند.

ب) نرخ یادگیری پایین باعث میشود آموزش مدل بسیار کند شود و در نتیجه شاید در تعداد epoch های مشخص شده، همگرا نشود. همچنین احتمال گیر کردن در مینیمم محلی را بالا می‌برد. در تصویر قبل، عکس اول از سمت چپ نشان‌دهنده آموزش مدل با نرخ یادگیری پایین است.

اگر مدل خیلی آهسته همگرا میشود یا اصلا همگرا نمیشود، احتمالا مشکل از نرخ یادگیری پایین است.

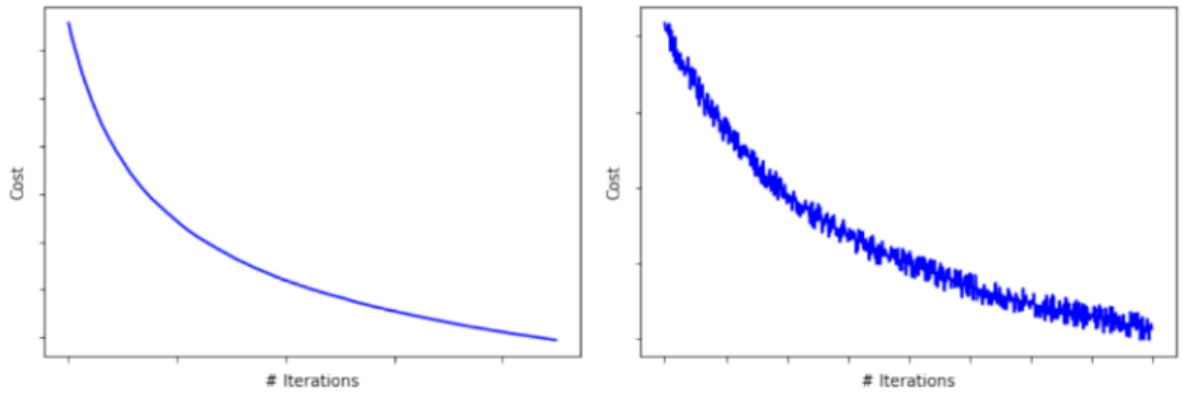
پ) Adam: روش‌های تطبیقی مانند Adam به‌عنوان SGD پیش‌تهویه‌کننده(preconditioner) دیده می‌شوند، که در آن پیش‌تهویه‌کننده به صورت آنلاین تخمین زده می‌شود. پیش‌تهویه‌کننده نویز گرادیان تصادفی را مجدداً مقیاس می‌دهد تا نزدیک نقاط ثابت همسانگرد باشد، که به فرار از نقاط زین کمک می‌کند. در مقایسه با SGD، روش‌های تطبیقی سریع‌تر از نقاط زینی فرار می‌کنند و می‌توانند در مجموع سریع‌تر به نقاط ثابت مرتبه دوم همگرا شوند. با این حال، آدام و سایر روش‌های تطبیقی به مقادیر خاصی از نرخ یادگیری بسیار حساس هستند و اگر نرخ یادگیری بیش از حد بالا باشد، می‌توانند به‌طور فاجعه‌باری همگرا نشوند.

SGD: Stochastic Gradient Descent (SGD) یک الگوریتم بهینه‌سازی است که در آن فراپارامترها باید از قبل تعریف شوند. SGD ممکن است به حداکثر محلی همگرا شود، ممکن است خودسرانه به آرامی از نقطه زینی فرار کند، و ممکن است مینیمم‌های تیز را نسبت به موارد مسطح ترجیح دهد. با این حال، SGD به اندازه آدام و سایر روش‌های تطبیقی به‌طور فاجعه‌باری شکست نمی‌خورد.

پرامپت من برای مورد پ:

What is a saddle point? Compare the two algorithms Adam and SGD in dealing with these points. Write the advantages and disadvantages of each.

ت)

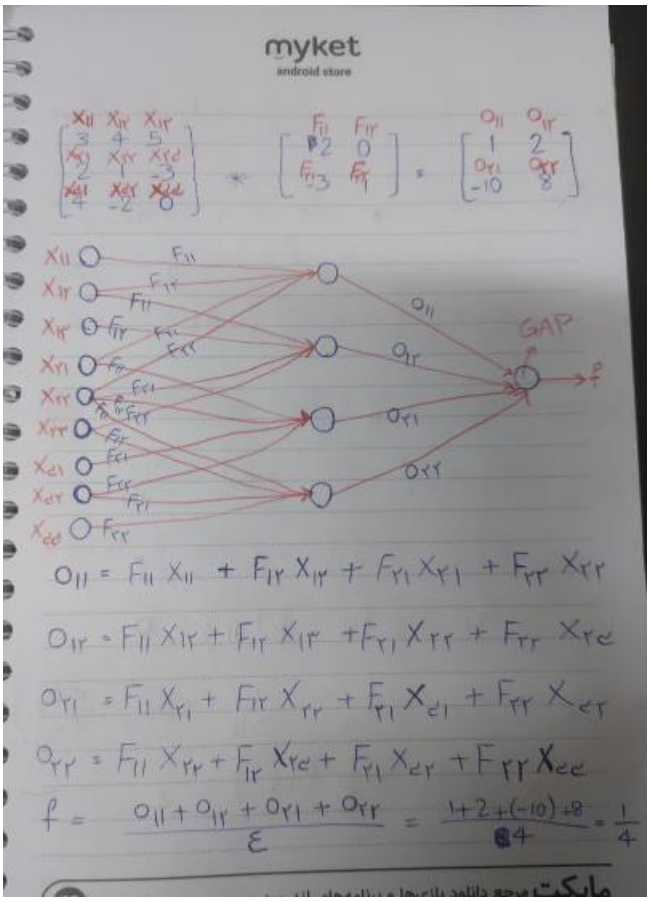


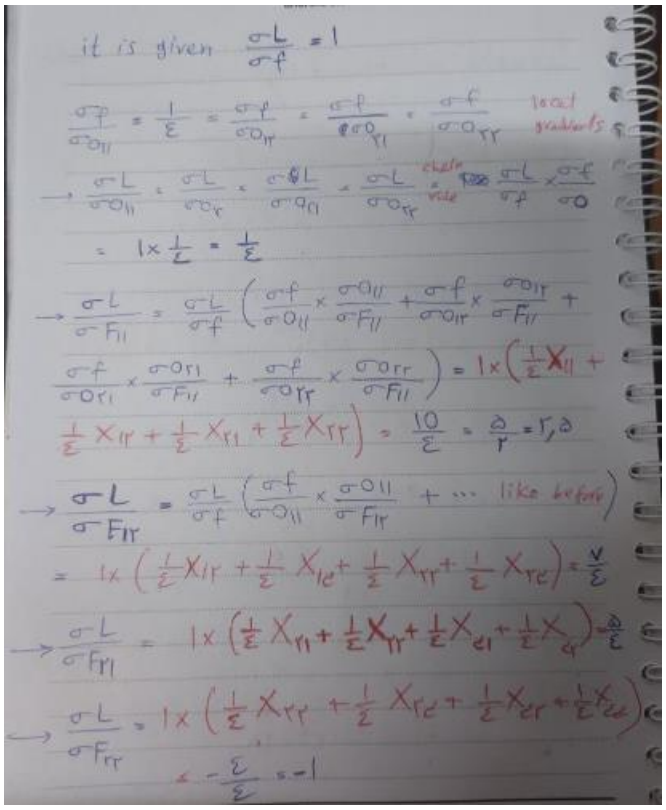
در شکل بالا، نمودار سمت چپ برای زمانی است که آپدیت پارامترها پس از معرفی تمام دیتاست صورت میگیرد ولی نمودار سمت راست برای زمانی است که دیتاست به یک سری mini-batch تقسیم میشود و آپدیت وزنها روی هر mini-batch به صورت جدا رخ می‌دهد. دلیل نویزی بودن نمودار راست این است که، دیتایی که در هر mini-batch قرار دارد، با batch های دیگر در همان epoch متفاوت است و در نتیجه تابع ضرر نیز برای

آن شکل متفاوتی با بقیه و همچنین با کل دیتاست دارد ولی میتوان گفت تقریبا مشابه است، و وقتی یک بار کل دیتاست را طی میکنیم، به طور میانگین به سمت مینیمم کل دیتاست حرکت میکند ولی به طور نویزی و با بالا و پایین شدن. اگر سایز هر دسته یا batch را کوچکتر کنیم، نویز نیز بیشتر میشود ولی به طور کلی و پس از طی تعداد مرحله کافی انتظار میرود همگرا شود.

اما برای شکل سمت چپ، از آنجا که کل دیتاست به یکباره به مدل داده میشود، بنابراین روی میانگین ضرر روی کل دیتاست، مقادیر وزنها را آپدیت میکند که این باعث میشود یک رفتار smoothتری داشته باشد و به شکلی که میبینیم همگرا شود.

سوال ۲-

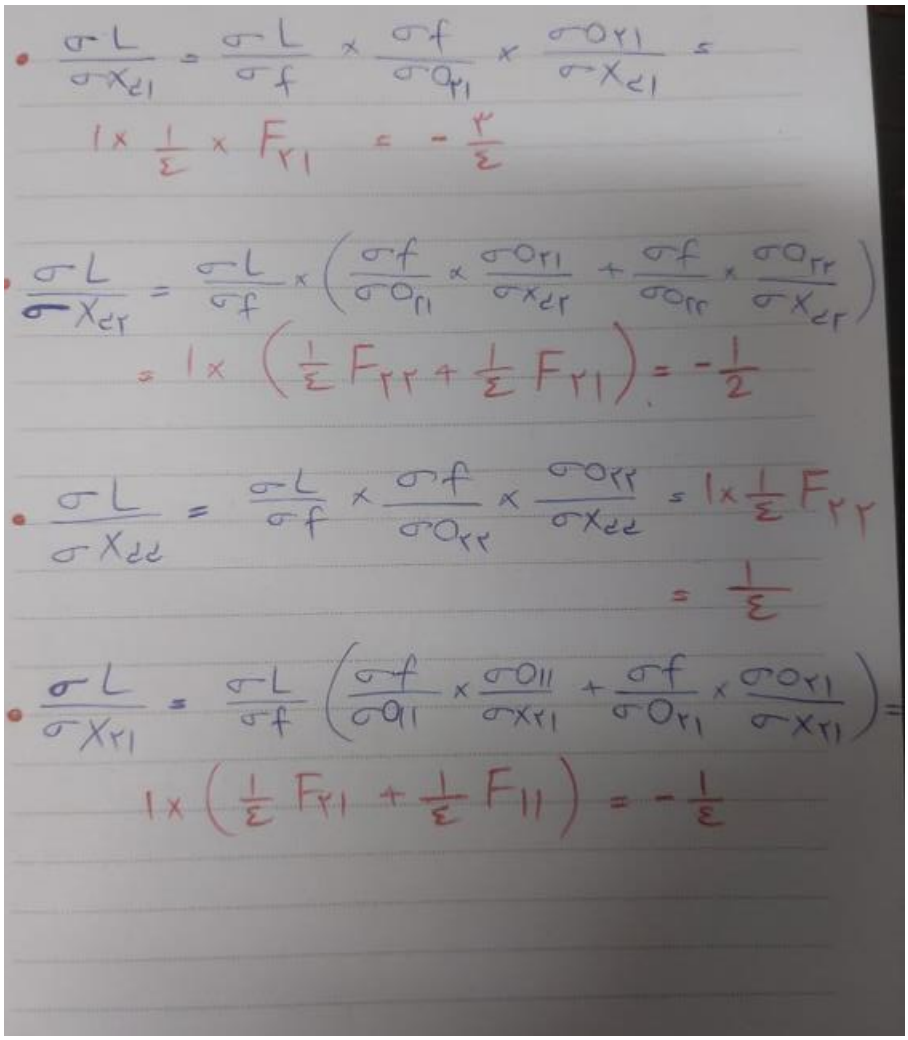




it is given $\frac{\partial L}{\partial f} = 1$

local gradients

chain rule



سوال ۳-

الف)

لایه اول پارامتری ندارد.

لایه دوم به تعداد ۱۶ عدد فیلتر یک بعدی با سایز ۳ داریم که در واقع فیلترهای ما میشوند 3*7 زیرا عمق ورودی ما ۷ است. پس تعداد پارامترها میشود:

16*3*7 = 336

همچنین هر فیلتر یک بایاس هم دارد پس:

336 + 16 = 352

ابعاد خروجی پس از این لایه میشود: 16 * 498 به دلیل padding=valid که دیفالت این لایه در کراس است.

لایه max pooling پارامتری ندارد ولی ابعاد خروجی را نصف میکند زیرا پارامتر stride به طور دیفالت برابر با ۲ است. پس ابعاد خروجی میشود: 16 * 249

لایه کانولوشنی دوم:

32 * 5 * 16 + 32 = 2592

ابعاد خروجی: 32 * 245 به دلیل padding=valid

لایه max pooling دوم:

پارامتر ندارد و ابعاد خروجی برابراست با: 32 * 122

لایه کانولوشنی سوم:

تعداد پارامترها:

64 * 5 * 32 + 64 = 10304

ابعاد خروجی: 64 * 118

لایه max pooling سوم:

پارامتر ندارد، ابعاد خروجی: 64 * 59

لایه flatten:

پارامتر ندارد، ابعاد خروجی: 59*64 = 3776

لایه کاملا خطی اول:

تعداد پارامتر: 128 * 3776 + 128(biases) = 483456

ابعاد خروجی: 128

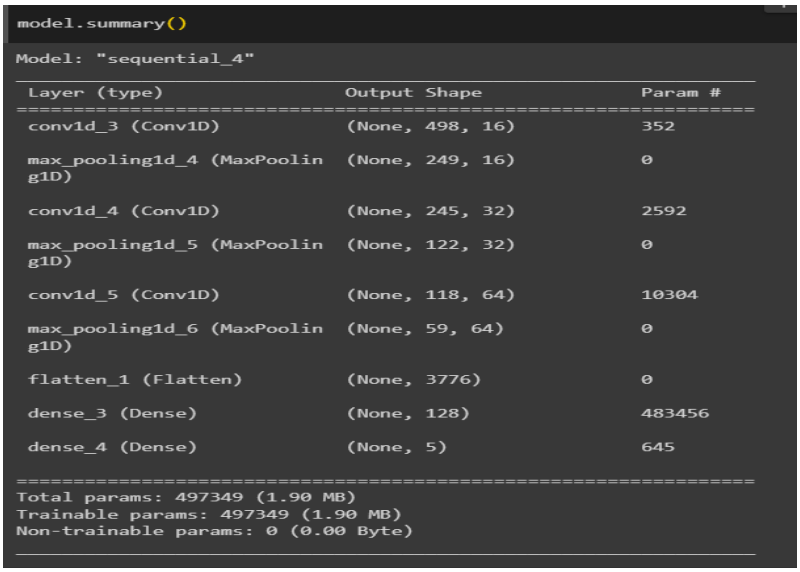
لایه کاملا خطی دوم:

تعداد پارامتر: 5 * 128 + 5 = 645

ابعاد خروجی: 5

نکته: البته قبل از همه ابعاد خروجی، سایز batch هم می آید ولی چون نمیدانستم، چیزی نذاشتم.

خروجی model.summary در کراس برای این مدل:



```
model.summary()

Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d_3 (Conv1D)           (None, 498, 16)           352

 max_pooling1d_4 (MaxPoolin  (None, 249, 16)           0
 g1D)

 conv1d_4 (Conv1D)           (None, 245, 32)           2592

 max_pooling1d_5 (MaxPoolin  (None, 122, 32)           0
 g1D)

 conv1d_5 (Conv1D)           (None, 118, 64)           10304

 max_pooling1d_6 (MaxPoolin  (None, 59, 64)            0
 g1D)

 flatten_1 (Flatten)         (None, 3776)              0

 dense_3 (Dense)             (None, 128)               483456

 dense_4 (Dense)             (None, 5)                 645

=================================================================
Total params: 497349 (1.90 MB)
Trainable params: 497349 (1.90 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

ب) «Conv2D» و «Conv3D» هر دو لایه‌های کانولوشنی هستند که در شبکه‌های عصبی کانولوشن (CNN) استفاده می‌شوند، اما آنها بر روی انواع مختلفی از داده‌های ورودی کار می‌کنند و برای برنامه‌های مختلف استفاده می‌شوند.

:Conv2D

- «Conv2D» برای کانولوشن داده های ۲ بعدی استفاده می شود.

- بیشتر برای داده های تصویری که الگوهای فضایی مهم هستند استفاده می شود.

- به عنوان مثال، در وظایف تشخیص تصویر، که در آن شما نیاز به شناسایی الگوها در ابعاد فضایی (ارتفاع و عرض تصویر) دارید، معمولاً از Conv2D استفاده می شود.

:Conv3D

- از طرف دیگر، Conv3D برای کانولوشن داده های ۳ بعدی استفاده می شود.

- اغلب با داده های حجمی استفاده می شود، جایی که بعد سوم (مانند عمق یا زمان) مهم می شود.

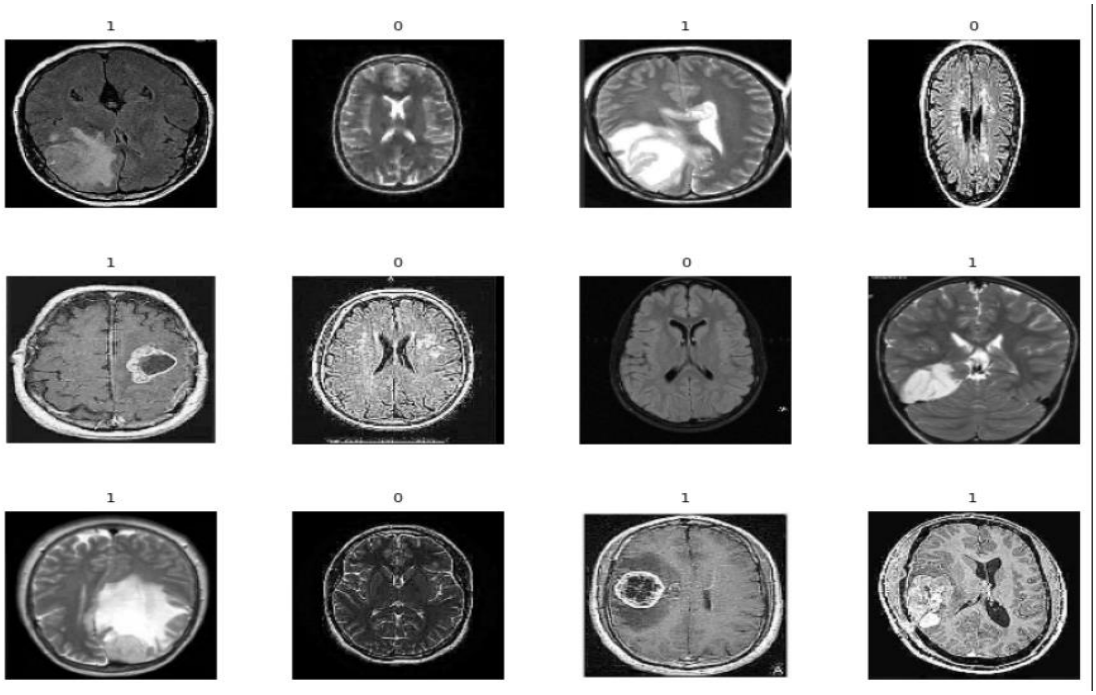
- به عنوان مثال، در کارهای تشخیص ویدیو، که در آن شما نیاز به شناسایی الگوها نه تنها در ابعاد مکانی (ارتفاع و عرض فریم ها) بلکه در بعد زمانی (در چندین فریم) دارید، معمولاً از «Conv3D» استفاده می شود.

- یکی دیگر از کاربردهای Conv3D در تصویربرداری پزشکی است که در آن تصاویر حجمی (مانند سی تی اسکن یا اسکن MRI) علاوه بر ارتفاع و عرض، دارای بعد عمق نیز هستند.

به طور خلاصه، تفاوت اصلی بین «Conv2D» و «Conv3D» در نوع داده‌ای است که برای آن استفاده می‌شود - «Conv2D» برای داده‌های دو بعدی (مانند تصاویر) و «Conv3D» برای داده‌های سه بعدی (مانند ویدیوها یا تصاویر حجمی). انتخاب بین آنها به ساختار و ماهیت داده های ورودی شما بستگی دارد. بعد اضافی در "Conv3D" به آن اجازه می دهد تا الگوها را در طول بعد عمق یا زمان ثبت کند، که می تواند در برنامه های خاص بسیار مفید باشد. با این حال، مدل‌های «Conv3D» به دلیل افزایش پیچیدگی، می‌توانند از نظر محاسباتی فشرده‌تر و سخت‌تر از مدل‌های «Conv2D» آموزش داده شوند.

سوال ۴-

نمایش برخی از عکسهای دیتاست:



برای مدل این سوال، از لایه های کانولوشنی استفاده کردم زیرا برای داده عکسی به دلیل در نظر گرفتن همسایگی، مناسب میباشد.

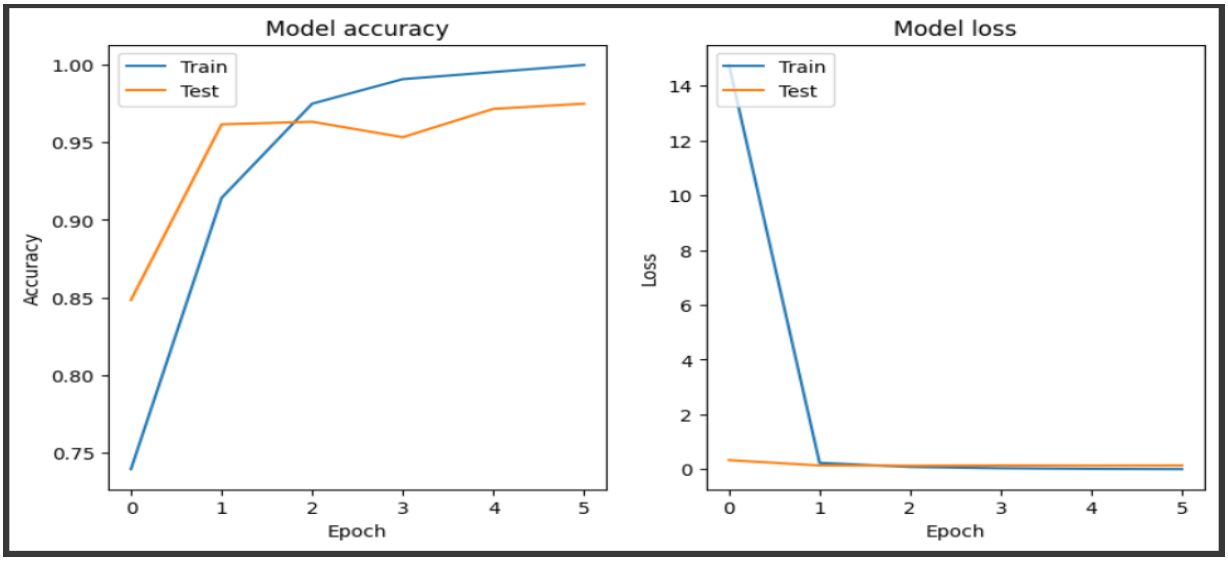
همچنین از لایه های max pooling استفاده کردم تا ابعاد را کاهش دهیم تا به overfitting نخوریم. در آخر دو لایه کاملا خطی داریم که آخرین آن از تابع سیگموید برای binary classification استفاده میکند. برای تابع ضرر نیز از binary_crossentropy استفاده کردم که مناسب دسته بندی باینری میباشد.

```
# Initialize the model
model = Sequential()

# Add layers
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(128, 128, 1)))
model.add(MaxPool2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))

model.add(Flatten())

model.add(Dense(units=64, activation='relu'))
model.add(Dense(units=1, activation='sigmoid'))

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Summary of the model
model.summary()
```

ساختار مدل:

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_14 (Conv2D)          (None, 126, 126, 32)      320

 max_pooling2d_14 (MaxPooli  (None, 63, 63, 32)        0
 ng2D)

 conv2d_15 (Conv2D)          (None, 61, 61, 64)        18496

 max_pooling2d_15 (MaxPooli  (None, 30, 30, 64)        0
 ng2D)

 flatten_7 (Flatten)         (None, 57600)             0

 dense_14 (Dense)            (None, 64)                3686464

 dense_15 (Dense)            (None, 1)                 65

=================================================================
Total params: 3705345 (14.13 MB)
Trainable params: 3705345 (14.13 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

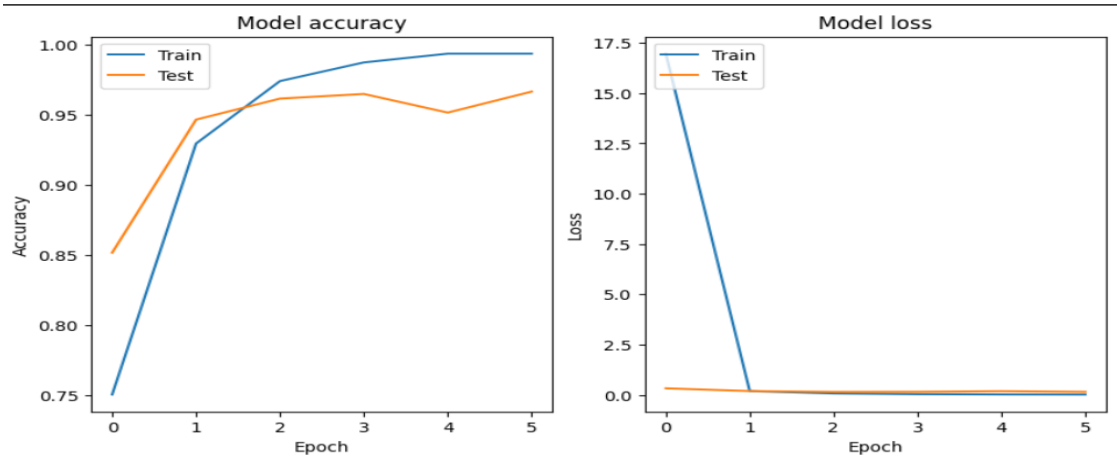
خروجی مدل روی دیتاست:




```
# Assuming `test_dataset` is your test dataset
test_loss, test_accuracy = model.evaluate(val_data)

print(f'Test loss: {test_loss}')
print(f'Test accuracy: {test_accuracy}')
```

```
19/19 [==============================] - 1s 43ms/step - loss: 0.1281 - accuracy: 0.9633
Test loss: 0.12808942794799805
Test accuracy: 0.9633333086967468
```

خروجی مدل فانکشنال:



```
# Assuming `test_dataset` is your test dataset
test_loss, test_accuracy = func_model.evaluate(val_data)

print(f'Test loss: {test_loss}')
print(f'Test accuracy: {test_accuracy}')
```

```
19/19 [==============================] - 1s 64ms/step - loss: 0.1504 - accuracy: 0.9617
Test loss: 0.15037794411182404
Test accuracy: 0.9616666436195374
```

جزئیات کد به صورت کامنت در کد قرار داده شده است.

سوال ۵-

برای مثال فرض کنید که مسئله دسته بندی گربه و سگ را داریم. در این نوع مسائل لایه های کانولوشن به خوبی کار میکنند، لایه های اولیه ویژگیهای ساده تر مثل لبه ها و گوشه ها و رنگها را پیدا میکنند، لایه های انتها تر ویژگیهای پیچیده تر مثل گوش و چشم و ... را تشخیص میدهد. پس برای این کاربرد مناسب است.

برای کاربردهایی که به خوبی با استفاده لایه های همگشتی جواب نمیدهند، به عنوان مثال، در محلی‌سازی شی یا object localization، یک CNN ممکن است به دلیل فقدان اطلاعات موقعیتی یا positional information، برای پیش‌بینی دقیق جعبه مرزی شیء مشکل داشته باشد. این می تواند منجر به precision و recall کمتری شود. به طور مشابه، در کارهایی که شی مورد نظر فقط در یک ناحیه کوچک از تصویر وجود دارد، یک CNN ممکن است به دلیل فرض ثابت بودن، برای شناسایی شیء مشکل داشته باشد که منجر به دقت پایین‌تر می‌شود. برای مثال زمانی که در face detection میدانیم که چهره فرد در مرکز تصویر قرار دارد، بهتر است ویژگیهای فیلترها را بین همه ناحیه ها share نکنیم.

مزایای این نوع لایه:

- Parameter sharing: یک فیلتر که یک ویژگی خاص را استخراج میکند، بین تمام ناحیه های تصویر share میشود که این باعث کاهش تعداد پارامترها و جلوگیری از overfitting میشود.
- Local connectivity: هر نورون به یک ناحیه کوچک از تصویر وصل است، این یعنی پیکسلهای تصویر با همسایه های خود مرتبطتر هستند تا با نقاط دورتر تصویر.
- Translation invariance: وقتی یک ویژگی برای یک بخش از تصویر یاد گرفته شد، برای بخشهای دیگر هم میتواند مورد استفاده قرار بگیرد و این باعث میشود با جابجایی آن شی در تصویر، همچنان مدل ما آن را تشخیص دهد.

معایب:

- Lack of positional information: در حالی که Translation invariance برای کارهایی مانند طبقه بندی تصویر مفید است، اما برای کارهایی که به اطلاعات مکانی دقیق نیاز دارند، مانند محلی سازی و تقسیم بندی اشیا، می تواند یک نقطه ضعف باشد. در این کارها موقعیت دقیق جسم در تصویر مهم است.
- Assumption of stationarity: این نوع لایه ها فرض میکنند، ویژگی های یاد گرفته شده در یک نقطه، برای نقاط دیگر نیز قابل تعمیم است. این همواره درست نیست، مخصوصا برای تصاویر بزرگ و تصاویری که شی هدف ما در یک ناحیه کوچک قرار دارد.
- Difficulty in Capturing Long-Range Dependencies: به دلیل اتصال محلی، گرفتن وابستگی های دوربرد نیاز به لایه های زیادی از کانولوشن دارد که می تواند آموزش شبکه را دشوار کند و مستعد بیش از حد برازش شود.

جواب بینگ به این سوال:

Convolutional Neural Networks (CNNs) are widely used in image classification due to their unique properties. Here are some advantages and a practical example:

Advantages:

1. **Parameter Sharing:** A feature detector (a filter) that's useful in one part of the image is probably useful in another part of the image. This drastically reduces the number of parameters in the model, making the network less prone to overfitting and more computationally efficient.
2. **Local Connectivity:** Each neuron in the first convolutional layer is connected only to a small region of the input image, called the receptive field. This reflects the fact that pixels in an image are strongly correlated with their neighbors, but much less so with distant pixels.
3. **Translation Invariance:** Once a feature is learned at a certain location in the image, the CNN can recognize that feature at a different location. This is very useful in image classification where the position of the object can vary.

Practical Example: Consider an image classification task where the goal is to distinguish between images of cats and dogs. The unique properties of CNNs allow the model to learn local features such as edges and textures in the early layers, and

more complex features such as shapes (e.g., ears, eyes) in the deeper layers. These features are then used to classify the image. The parameter sharing and local connectivity properties make the model computationally efficient, while the translation invariance property allows the model to recognize a cat or a dog regardless of its position in the image.

However, there are also situations where the properties of convolutional layers can create challenges:

Disadvantages:

1. **Lack of Positional Information:** While translation invariance is useful for tasks like image classification, it can be a disadvantage for tasks that require precise spatial information, such as object localization and segmentation. In these tasks, the exact position of the object in the image is important.
2. **Assumption of Stationarity:** CNNs assume that the statistics of one part of the image are the same as any other part. This might not always be true, especially for larger images or images where the object of interest is only present in a small region.
3. **Difficulty in Capturing Long-Range Dependencies:** Due to local connectivity, capturing long-range dependencies requires many layers of convolutions, which can make the network difficult to train and prone to overfitting.

Effect on Efficiency: These disadvantages can lead to sub-optimal performance in certain tasks. For example, in object localization, a CNN might struggle to accurately predict the bounding box of the object due to the lack of positional information. This could result in lower precision and recall scores. Similarly, in tasks where the object of interest is only present in a small region of the image, a CNN might struggle to detect the object due to the assumption of stationarity, leading to lower accuracy. Finally, the difficulty in capturing long-range dependencies can make the model computationally expensive and prone to overfitting, especially for larger images. This could result in longer training times and lower generalization performance.

To overcome these challenges, modifications and additions to the standard CNN architecture are often used. For example, Fully Convolutional Networks (FCNs) for segmentation tasks, or the addition of attention mechanisms to help capture long-range dependencies.

سوال ۶-

الف) هدف استفاده از فیلترهای 1*1 در شبکه های عصبی کانولوشن معمولا کاهش ابعاد است. این نوع فیلتر به کاهش تعداد نقشه های ویژگی کمک می کند و در عین حال ویژگی های مهم را حفظ می کند. این به ویژه در شبکه های عصبی کانولوشن عمیق که در آن تعداد نقشه های ویژگی اغلب با عمق شبکه افزایش می یابد مفید است. استفاده از فیلترهای 1*1 می تواند از افزایش چشمگیر تعداد پارامترها و محاسبات مورد نیاز هنگام استفاده از فیلترهای بزرگتر جلوگیری کند. در واقع این نوع فیلتر به نوعی channel-wise pooling یا feature map pooling انجام میدهد.

ب) پس از اعمال یکی از این فیلترها، یک نقشه ویژگی جدید خواهیم داشت که در واقع تمام نقشه‌های ویژگی ورودی را به یک نقشه ویژگی جدید که حاصل ترکیب خطی آنها و سپس یک تابع فعالسازی روی حاصل است، تبدیل میکند. حال میتوانیم ۱۰ تا از این نوع فیلتر داشته باشیم و تعداد نقشه‌های ویژگی ورودی ۱۶ تا باشد. در این صورت ۱۰ ترکیب مختلف از این ۱۶ نقشه ویژگی خواهیم داشت.

پ) نقشه ویژگی از یک فیلتر 1*1 با تصویر اصلی یا فیلترهای دیگر با اندازه های مختلف متفاوت است، زیرا بدون تغییر resolution، کانال‌های نقشه ویژگی‌های ورودی را تغییر می دهد. این کار یک تبدیل خطی از نقشه‌های ویژگی ورودی را انجام می دهد که هیچ پیکسل مجاوری را در ورودی شامل نمی‌شود. بنابراین، میتوان آن را یک عملیات کانولوشن تلقی نکرد.

ت) از این فیلترها در GoogleNet و ResNet استفاده شده است و نقش مهمی در موفقیت آنها داشته است.

ث) ممکن است شرایطی وجود داشته باشد که استفاده از فیلترهای 1*1 مفید نباشد. به عنوان مثال، اگر هدف کاهش ابعاد نباشد یا اگر پیچیدگی محاسباتی موضوع مهمی برای ما نباشد، استفاده از فیلترهای 1*1 ممکن است مزایای قابل توجهی ارائه نکند. همچنین، از آنجایی که فیلترهای 1*1 هیچ پیکسل مجاوری را در ورودی شامل نمی‌شوند، ممکن است برای کارهایی که نیاز به بدست آوردن spatial relationship در ورودی دارند مثل تشخیص چهره، مناسب نباشند. پس بطور خلاصه، استفاده زیاد از این نوع فیلتر ممکن است مدل ما را خیلی ساده کند و مدل دچار underfit شود.

ج) طبق شکل زیر مشخص است که وقتی از فیلتر 3*3 استفاده کردیم، رزولوشن تصویر به دلیل padding = valid تغییر کرد ولی در لایه بعد چنین اتفاقی نیفتاد. همچنین تعداد کانالها یا نقشه های ویژگی از ۱۲۸ به ۳۲ کاهش پیدا کرد که این همان کاربرد اصلی فیلترهای 1*1 یعنی کاهش ابعاد است.

```
# Define the model
model = Sequential()
model.add(Conv2D(128, (3, 3), input_shape=(64, 64, 3)))
model.add(Conv2D(32, (1, 1)))

# Print the model summary
model.summary()

# Create an arbitrary input
input_data = np.random.rand(1, 64, 64, 3)

# Pass the input through the model
output_data = model.predict(input_data)

# Print the input and output size
print(f'Input size: {input_data.shape}')
print(f'Output size: {output_data.shape}')
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_1 (Conv2D)           (None, 62, 62, 128)       3584

 conv2d_2 (Conv2D)           (None, 62, 62, 32)        4128

=================================================================
Total params: 7712 (30.12 KB)
Trainable params: 7712 (30.12 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
1/1 [==============================] - 0s 71ms/step
Input size: (1, 64, 64, 3)
Output size: (1, 62, 62, 32)
```

سوال ۷-

- ماژول inception برای اولین بار در GoogleNet معرفی شد و در واقع در جواب این سوال بود که چه سایز کرنلی بهترین نتایج را برای لایه های کانولوشنی میدهد. در این مدل، inception module به شکل چندین لایه کانولوشنی موازی با سایز کرنل مختلف معرفی شد که در نهایت خروجی همه این قسمتهای موازی در بُعد آخر یعنی channel با هم concat میشود. سایز این کرنل ها معمولا

1*1,3*3,5*5 میباشد. هدف این ماژول این است که مدل ما به طور همزمان بتواند ویژگیهای مختلفی را یاد بگیرد، برای مثال ویژگی‌هایی که همسایگی بزرگتری را میطلبند یا آنهایی که همسایگی کوچکتری را میطلبند.

- پارامتر گام یا همان stride در شبکه های کانولوشنی تعیین میکند که فیلتر با چه گامی روی ورودی بلغزد و در واقع برای کاهش ابعاد و در نتیجه کاهش پارامترها و کاهش پیچیدگی مدل میشود و برای جلوگیری از overfitting استفاده میشود و معمولا به همراه لایه max pooling استفاده میشود. هرچه مقدار stride بزرگتر باشد، ابعاد خروجی کوچکتر خواهد بود. البته هرچه مقدار stride بزرگتر شود، مقدار دیتای از دست رفته بیشتر میشود که خوب نیست. برای مثال stride=(m,n) ورودی w*h را به خروجی w/m * h/n تبدیل میکند.

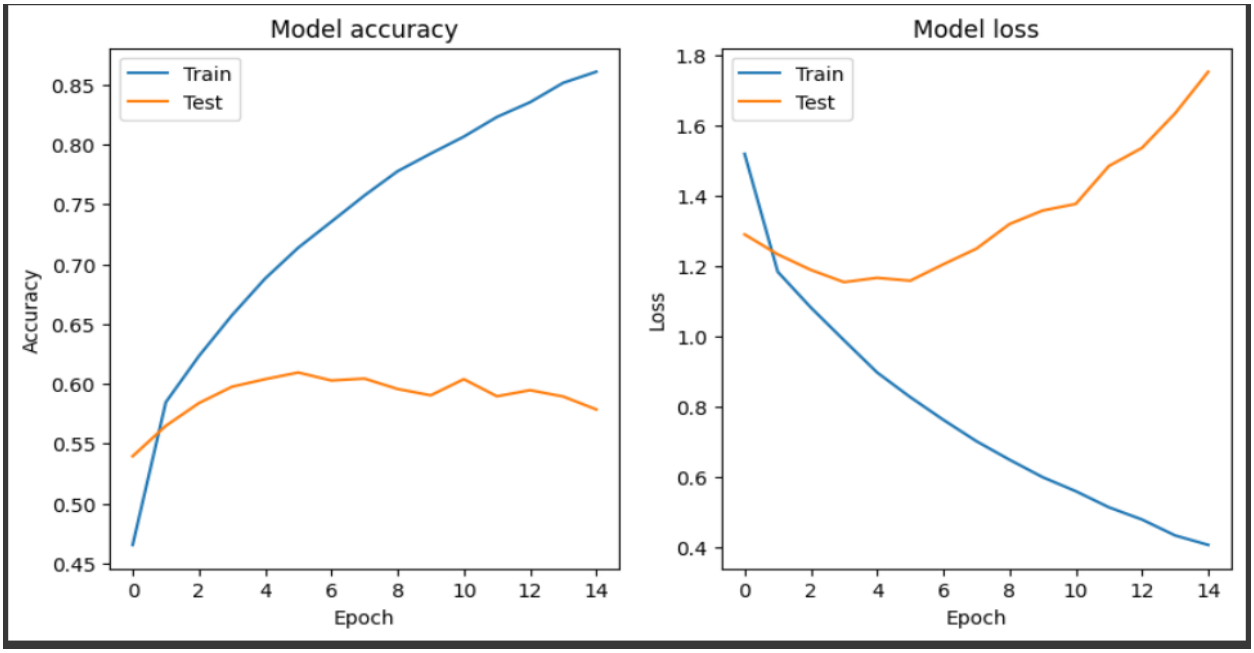
- لایه های کانولوشنی کار اصلی استخراج ویژگی را در شبکه های عصبی کانولوشنی انجام میدهند، این لایه ها یک سری فیلتر روی ورودی اعمال میکنند که هر کدام از این فیلترها یک ویژگی خاص را شناسایی میکنند، لایه های پایینتر ویژگیهای ساده‌تر مثل لبه ها و رنگها را شناسایی میکنند و لایه های بالاتر ویژگیهای سطح بالاتر را با استفاده از ترکیب ویژگیهای سطح پایین شناسایی میکنند مثل وجود گوش و چشم و ... در تشخیص انسان.

```
input_img = Input(shape=(32, 32, 3))

tower_1 = Conv2D(16, (1, 1), padding='same', activation='relu')(input_img)
tower_1 = Conv2D(8, (3, 3), padding='same', activation='relu')(tower_1)

tower_2 = Conv2D(16, (1, 1), padding='same', activation='relu')(input_img)
tower_2 = Conv2D(8, (5, 5), padding='same', activation='relu')(tower_2)

tower_3 = MaxPool2D((3, 3), strides=(1, 1), padding='same')(input_img)
tower_3 = Conv2D(16, (1, 1), padding='same', activation='relu')(tower_3)

output = concatenate([tower_1, tower_2, tower_3], axis=-1)

output = Flatten()(output)
out    = Dense(10, activation='softmax')(output)

# Create the model
model = Model(inputs = input_img, outputs = out)
```

در اینجا یک inception module داریم که از سه قسمت یا سه برج تشکیل شده. اولین برج شامل لایه های کانولوشنی 1*1 و 3*3 است. برج دوم از سایز کرنل 1*1 و 5*5 استفاده کرده است و در نهایت برج سوم از

یک لایه max pooling و یک لایه کانولوشنی 1*1 استفاده تشکیل شده است و در نهایت خروجی این سه برج در بُعد آخر یعنی channel با هم concat میشوند. بعد از آن یک لایه flatten گذاشتم که بتوانم آن را به یک لایه کاملا خطی برای دسته بندی نهایی ببرم.

خروجی مدل روی تابع آموزش و ارزیابی:



دقت نهایی بر روی داده آموزش:

```
# testing model on train dataset
train_loss, train_accuracy = model.evaluate(x_train, y_train)

print(f'Train loss: {train_loss}')
print(f'Train accuracy: {train_accuracy}')
```

```
1563/1563 [==============================] - 6s 4ms/step - loss: 0.3132 - accuracy: 0.8965
Train loss: 0.31323131918907166
Train accuracy: 0.8965200185775757
```

برای کد این سوال از سایت زیر استفاده کردم:

https://becominghuman.ai/understanding-and-coding-inception-module-in-keras-eb56e9056b4b

جزئیات کد به صورت کامنت در کد قرار داده شده است.