بسم الله الرحمن الرحیم

محمد عرفان زارع زردینی

۹۸۴۱۱۴۳۲

تمرین سری سوم درس یادگیری عمیق

مدرس درس: استاد داوود آبادی

پاییز 1402

1)منابع:

https://www.quora.com/In-an-artificial-neural-network-algorithm-what-happens-if-my-learning-rate-is-wrong-too-high-or-too-low

https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/#:~:text=A%20learning%20rate%20that%20is,carefully%20selecting%20the%20learning%20rate.

https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/

chatgpt(prompt: high and low learning rate and problem? How to detect these?)

https://towardsdatascience.com/batch-mini-batch-stochastic-gradient-descent-7a62ecba642a

الف)

نرخ یادگیری بالا سبب مشکلاتی همچون :1)واگرایی 2)overshooting یا 3 overfitting)عدم ثبات شود.

وقتی نرخ یادگیری خیلی زیاد است، سبب می شود بهینه سازی از حداقل نقطه در تابع ضرر بیشتر شود و به جای همگرایی، واگرایی رخ می دهد. همچنین نرخ بالای یادگیری، سبب ناپایداری در مدل و نوسانی شدن پارامترهای مدل شود.

راه حل: برای تشخیص آن می توان منحنی ضرر را رسم نماییم و در صورت مشاهده نوسان های زیاد و عدم کاهش آن ، به مشکل پی ببریم.همچنین عملکرد مدلمان را روی داده های اعتبارسنجی بررسی کنیم و اگر خطای آن شروع به افزایش نماید یا رفتاری نامنظم داشته باشد، از علائم یادگیری بسیاربالا است. همچنین وجود مقادیرNaN در مقادیر خروجی می تواند نشان دهنده بزرگ شدن بسیارگرادیان شود و از علائم نرخ یادگیری بالا است

ب)

نرخ یادگیری پایین سبب مشکلاتی همچون : همگرایی باسرعت پایین 2) گیر اقتادن در مینیمم محلی 3) به دام اقتادن در نقاط زینی، شود. نرخ یادگیری پایین سبب طول کشیدن همگرا شدن می شود و نیاز باشد برای رسییدن به سطح قابل قبولی از عملکرد ، به تعداد دوره های بیشتری نیاز باشد.

راه حل: تحلیل منحنی ضرر که در آن هرگاه منحنی بسیار آهسته کم شود ، یا زودتر و بدون بهبود زیاد شود از علائم نرخ یادگیری پایین است. زمان آموزش وهمگرایی آن نیز اگر زیاد طولانی شود، و بعد از تعداد قابل توجهی دوره، به نتیجه مطلوب نرسد، از دیگر علائم هست. همچنین بررسیی داده های اعتبارسنجی نیز موثر است و درصورت بالا ماندن خطا، وعدم کاهش آن در طی دوره ها، نمایانگر این مشکل است. برای رفع مشکل میتوان نرخ یادگیری را زیاد نمود و از برخی بهینه ساز ها که براساس پارامتر های مختلف ، نرخ را تنظیم می کنند همچون adam ، استفاده نمود.

پ)

نقطه زینی، یک نقطه بحرانی در بهینه سازی یک تابع ، بویژه در فضاهای با ابعاد بالا است . در این نقطه گرادیان صفر است. در فضای سه بعدی اگر به نمودار نگاه کنیم مانند زین می ماند(در بعضی بخش ها به سمت بالا و در برخی امتداد ها به سمت پایین شیب دارد.) این نقاط در بهینه سازی مشکل ساز اند، زیرا میتوانند برخی از الگوزیتم های بهینه ساز همچون sgd را دچار مشکل کرده و مانع از همگرایی آن ها شوند.

در مواجهه با نقاط زینی، روش آدام، بدان کمک مب نماید و در مقایسه با sgd ، این نقاط را به شکل موثر تری کنترل و ببرسی می نماید. البته هردو آن ها می توانند در برخورد با نقاط زینی چالش هایی مواجهه شوند. با این حال adamبه علت ماهیت تطبیقی آن، عملکرد بهتری از خود نشان می دهد و انتخاب میان این دو به ویژگی های مسئله بهینه سازی و منابع محاسباتی موجود بستگی دارد.

پیرامون مقایسه کلی دو الگوریتم ذکر شده داریم:

SGD	Adam	
1)نیاز به حافظه کمتر 2))ساده بودن 3)تعمیم	1)نرخ یادگیری تطبیقی 2)ادغام مومنتوم 3)کنترل داده های پراکنده و مشکل زا	مزایا
1)همگرایی کند تر 2)حساس به نرخ یادگیری 3)در نقاط زینی گیر می افتد	1) حفظ حالات اضافه و نیاز به حافظه بیشتر 2) به هایپر پارامتر ها حساس است 3) همگرایی به کمینه های به صورت تیز	معایب

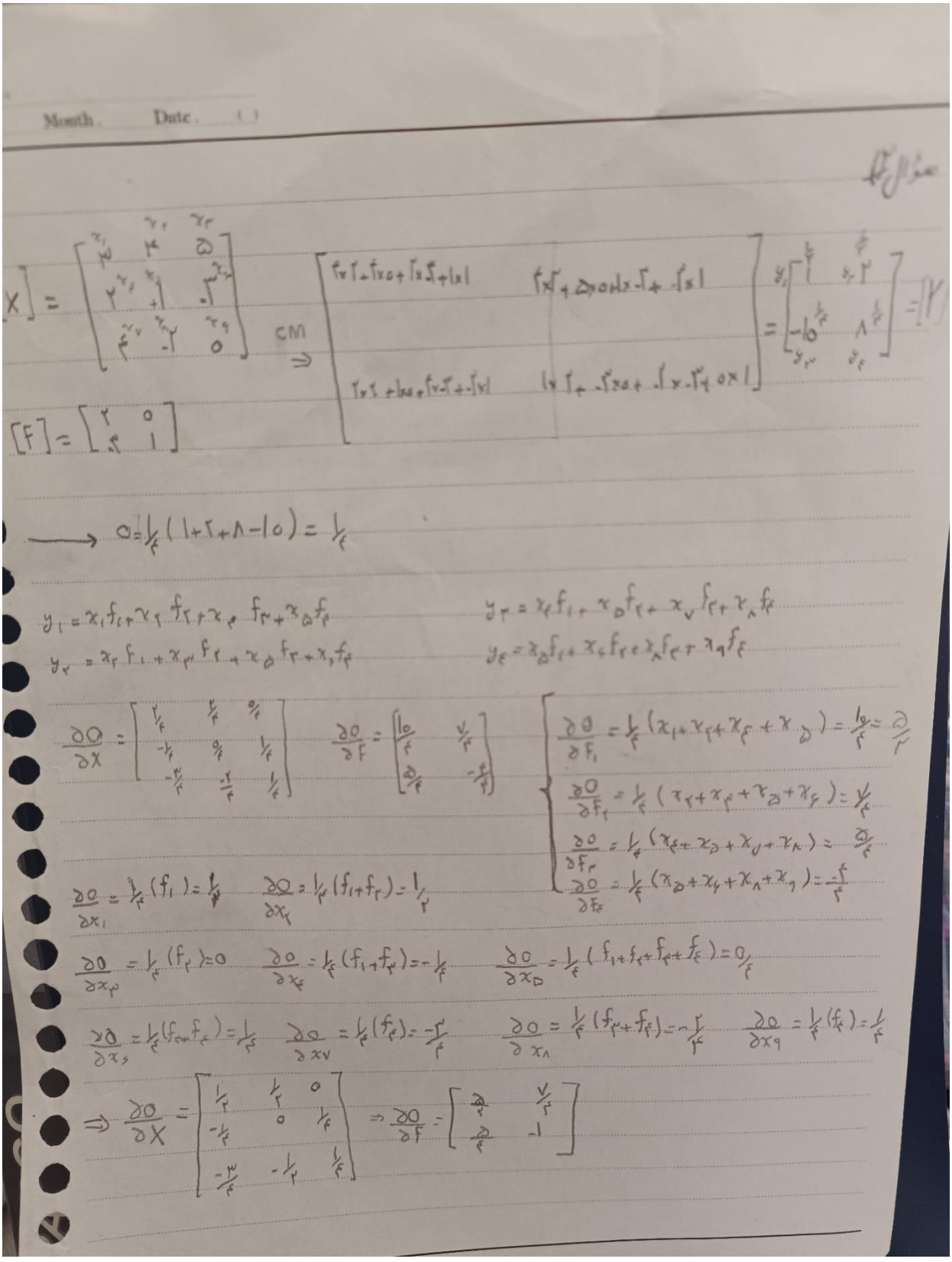
در تکمیل نکات بالا، آدام نرخ یادگیری را برای هر پارامتر به شکل جداگانه و براساس گرادیان گذشته تنظیم می کند و با تطبیق با ویژگی ها و گرادیان های مختلف، امکان همگرایی سریعتر را فراهم میکند .همچنین نیاز به تنظیم هایپرپارامتر دارد که با تنظیم نادرست، سبب همگرایی کمتر می شود. همگرایی به کمینه های به صورت تیز نیز سبب می شود به شکل بالقوه بر تعمیم داده های دیده نشده تاثیر بگذارد.

پیرامون sgd نیز اجرای نسبتا ساده و سهل طرحی دارد و به علت همگرایی به سمت مینمم های مسطح تر به جای موارد تیز ، بهتر تعمیم یابد. به علت نرخ یادگیری ثابت برای همه پارامتر ها، همگرایی کندی ممکن است داشته باشد.

ت)

نمودار سمت چپ برای حالت batch gradient descent و نمودار سمت راست برای mini_batch gradient descent می باشد. در الگوریتم سمت چپ، پارامترهای مدل را بوسیله همه مجموعه داده ها در هر تکرار بروز می کند، پس هزینه به آرامی و با هر تکرار کم می شود و نمودار پیوسته و بودن نوسان کاهش یافته است ، از نشانه های این الگوریتم است . در حالی که در شکل سمت چپ، از زیرمومعه داده های کوچک یا مینی دسته های مجموعه داده برای به روزرسانی پارامترها سود می برد. این رویکرد تصادفی بودن را به علت استفاده از زیرمجموعه ها نشان می دهد که سبب نوساناتی در نمودار هزینه نسبت به تکرار ها می شود. پس نمودار سمت راست که هزینه با نوسانات کاهش یافته از نشانه های این الگوریتم است.دلیل نوسانات موجود این است که هر مینی بچ، تخمینی از گرادیان های واقعی ارائه می نماید و باعث تغییرات در کاهش هزینه در هر تکرار می شود.

سوال 2)



سوال3)

الف)

ما لایه به لایه وخط به خط محاسبات را انجام میدهیم.

لایه ورودی:

این لایه پارامتر قابل آموزشی ندارد.

لایه کانولوشن اول:

تعداد پارامترها:

$(3*7+1)*16=352$

لایه اول ماکسیمم پولینگ:

این لایه پارامتری ندارد.

لایه دوم کانولوشن:

تعدادپارامترها:

$(5*16+1)*32=2592$

لایه دوم ماکسیمم پولینگ:

این لایه هم پارامتر ندارد

لایه سوم کانولوشن:

تعداد پارامترها:

$(5*32+1)*64=10304$

لایه سوم ماکسیمم پولینگ:

پارامتر ندارد.

لایه flatten:

پارامتر ندارد.

لایه اول dense:

ورودی این لایه از روی خروجی لایه های بعد از flatten است. پس داریم:

$$(59*64+1)*128=483456$$

لایه dense دوم:

تعداد پارامتر ها:

$$(128+1)*5=645$$

در نتیجه تعداد پارامترهای شبکه برابر است با:

$$497349=645+483456+352+2592+10304$$

ب)

کانولوشن 2 بعدی:برروی داده های فضای دوبعدی مانند تصاویرکار می نماید و اطلاعات فضای دوبعدی همچون عرض و ارتفاع را پردازش می نماید. کرنل ها وفیلتر ها به شکل دوبعدی بوده و در ابعاد فضای دوبعدی لغزانده می شوند . در گرفتن ویژگی های دورن تصویر همچون لبه ها ،بافت ها،اشکال و الگوها موثر اند. کاربرد آن بیشتر در بینایی ماشین مانند طبقه بندی تصاویر، تشخیص اشیا، تقسیم بندی و انتقال شیوه کاربرد دارد.

کانولوشن 3 بعدی:داده های فضای سه بعدی را مدیریت نموده که شامل عمق یا زمان به همراه عناصر مکانی هستند.

از فیلترهای کانولوشن سه بعدی شامل عرض، ارتفاع و عمق(زمان) ، سود می برد. همچنین امکان درک الگوهای مکانی و تغییرات زمانی در توالی یا حجم داده ها را فراهم می کند.

کاربرد کانولوشن 3 بعدی:در کارهای مربوط به داده های مکانی-زمانی همچون تجزیه و تحلیل ویدئو، تشخیص عمل، تصویربرداری سه بعدی و پیش بینی دینامیک در فضای سه بعدی در واحد زمان کاربرد دارد.

سوال 4)

ابتدا دیتا ست مورد نیاز را از لینک داده شده دانلود نموده و از حالت فشرده خارج می کنیم. تصاویر موجود در دایرکتوری را خوانده و 20 درصد داده هارا به تست اختصاص میدهیم، سایز تصاویر را256*256 اختصاص می دهیم.

```
gdown.download('https://drive.google.com/uc?id=1SCpVEdJ6_YOAcy2iW05ENlMh-OCcFz3P', 'dataset.zip', quiet=False)
with zipfile.ZipFile("dataset.zip", 'r') as zips:
    zips.extractall("dataset")
```

```
train_dt, test_dt = keras.preprocessing.image_dataset_from_directory("dataset",validation_split=0.2,subset="both",
seed=42,image_size=(256, 256),batch_size=32,)
```

```
Found 3000 files belonging to 2 classes.
Using 2400 files for training.
Using 600 files for validation.
```

سپس دیتاست را با لیبل های زده شده شان نمایش می دهیم و برای اینکار به شکل رندوم تعدادی از تصاویر را نمایش می دهیم.

```
plt.figure(figsize=(15,15))
for img, lbl in train_dt.take(5):
    for i in range(16):
        ax = plt.subplot(4, 4, i + 1)
        plt.imshow(img[i].numpy().astype("uint8"))
        plt.title(int(lbl[i]))
        plt.axis("off")
```

حال باید مدلی را طراحی کرده و به شکل sequential,functional پیاده سازی نماییم. ابتدا تصویر را اسکیل می کنیم تا رده رنگ ها درر بازه 0تا 1 قرار گیرند. پدینگ را برای مدلمان یکسان در نظر گرفته وتابع فعالساز را relu در نظر میگیریم. کانولوشن2 بعدی های در نظر گرفته شده شامل 3 لایه است و بعد از هر لایه

کانولوشن، نرمال سازی بچ و لایه maxpool دوبعدی زده شده. سپس لایه flatten زده شده و یک لایه dense میزنیم و سپس dropout میزنیم تا از بیش برازش جلوگیری کنیم. در نهایت هم باید لایه dense و تابع فعالساز سیگموید لایه اخر خروجی را مشخص می کنیم.در نهایت هم مدل را با بهینه ساز آدام و تابع ضرر binary crossentropy کامپایل کرده و مدل را آموزش می دهیم. در نهایت مدل راا روی داده های تست بررسی می کنیم.

حال نمودار دقت و خطا را نمایش میدهیم. برای هردو فرمت ، ساختار آن در تصاویر زیر موجود است.

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

history = model.fit(train_dt, epochs=15)
```

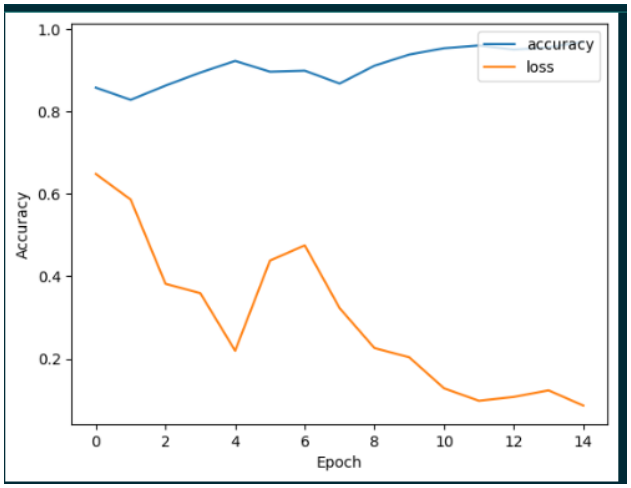
```
Epoch 1/15
75/75 [==============================] - 10s 91ms/step - loss: 0.6486 - accuracy: 0.8583
Epoch 2/15
75/75 [==============================] - 8s 98ms/step - loss: 0.5864 - accuracy: 0.8288
Epoch 3/15
75/75 [==============================] - 7s 96ms/step - loss: 0.3819 - accuracy: 0.8633
Epoch 4/15
75/75 [==============================] - 7s 90ms/step - loss: 0.3590 - accuracy: 0.8950
Epoch 5/15
75/75 [==============================] - 8s 97ms/step - loss: 0.2192 - accuracy: 0.9233
Epoch 6/15
75/75 [==============================] - 7s 90ms/step - loss: 0.4382 - accuracy: 0.8971
Epoch 7/15
75/75 [==============================] - 7s 95ms/step - loss: 0.4752 - accuracy: 0.8996
Epoch 8/15
75/75 [==============================] - 7s 90ms/step - loss: 0.3227 - accuracy: 0.8683
Epoch 9/15
75/75 [==============================] - 8s 96ms/step - loss: 0.2258 - accuracy: 0.9117
Epoch 10/15
75/75 [==============================] - 7s 94ms/step - loss: 0.2033 - accuracy: 0.9388
Epoch 11/15
75/75 [==============================] - 7s 90ms/step - loss: 0.1278 - accuracy: 0.9542
Epoch 12/15
75/75 [==============================] - 7s 95ms/step - loss: 0.0977 - accuracy: 0.9608
Epoch 13/15
...
Epoch 14/15
75/75 [==============================] - 8s 98ms/step - loss: 0.1230 - accuracy: 0.9563
Epoch 15/15
75/75 [==============================] - 7s 91ms/step - loss: 0.0858 - accuracy: 0.9696
```

```
model = Sequential([
    Rescaling(1./255, input_shape=(256, 256, 3)),
    Conv2D(32, 3, padding='same', activation='relu'),
    BatchNormalization(),
    MaxPool2D(),
    Conv2D(64, 3, padding='same', activation='relu'),
    BatchNormalization(),
    MaxPool2D(),
    Conv2D(128, 3, padding='same', activation='relu'),
    BatchNormalization(),
    MaxPool2D(),
    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])
```

Test the model

```
model.evaluate(test_dt)
```

```
19/19 [==============================] - 1s 45ms/step - loss: 0.0691 - accuracy: 0.9817
```



Functional api

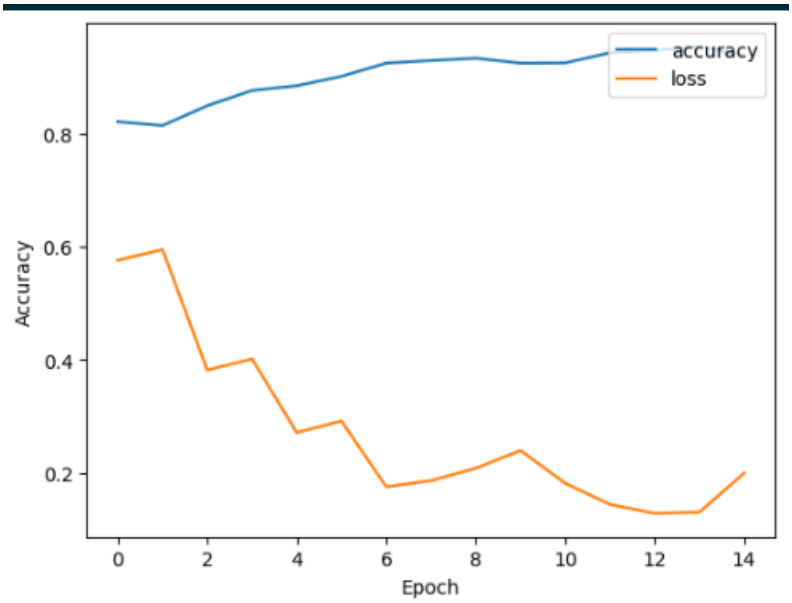
```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
history = model.fit(train_dt, epochs=15)
```

```
Epoch 1/15
75/75 [==============================] - 11s 94ms/step - loss: 0.5761 - accuracy: 0.8213
Epoch 2/15
75/75 [==============================] - 7s 94ms/step - loss: 0.5950 - accuracy: 0.8146
Epoch 3/15
75/75 [==============================] - 7s 89ms/step - loss: 0.3817 - accuracy: 0.8496
Epoch 4/15
75/75 [==============================] - 7s 96ms/step - loss: 0.4014 - accuracy: 0.8767
Epoch 5/15
75/75 [==============================] - 7s 90ms/step - loss: 0.2710 - accuracy: 0.8850
Epoch 6/15
75/75 [==============================] - 8s 103ms/step - loss: 0.2915 - accuracy: 0.9017
Epoch 7/15
75/75 [==============================] - 7s 93ms/step - loss: 0.1751 - accuracy: 0.9250
Epoch 8/15
75/75 [==============================] - 7s 92ms/step - loss: 0.1860 - accuracy: 0.9296
Epoch 9/15
75/75 [==============================] - 8s 97ms/step - loss: 0.2079 - accuracy: 0.9337
Epoch 10/15
75/75 [==============================] - 7s 90ms/step - loss: 0.2395 - accuracy: 0.9250
Epoch 11/15
75/75 [==============================] - 8s 97ms/step - loss: 0.1813 - accuracy: 0.9254
Epoch 12/15
75/75 [==============================] - 7s 90ms/step - loss: 0.1439 - accuracy: 0.9433
Epoch 13/15
...
Epoch 14/15
75/75 [==============================] - 7s 89ms/step - loss: 0.1301 - accuracy: 0.9546
Epoch 15/15
75/75 [==============================] - 8s 101ms/step - loss: 0.1993 - accuracy: 0.9388
```

```
inputs = Input(shape=(256, 256, 3))
x = Rescaling(1./255)(inputs)
x = Conv2D(32, 3, padding='same', activation='relu')(x)
x = BatchNormalization()(x)
x = MaxPool2D()(x)
x = Conv2D(64, 3, padding='same', activation='relu')(x)
x = BatchNormalization()(x)
x = MaxPool2D()(x)
x = Conv2D(128, 3, padding='same', activation='relu')(x)
x = BatchNormalization()(x)
x = MaxPool2D()(x)
x = Flatten()(x)
x = Dense(256, activation='relu')(x)
x = Dropout(0.5)(x)
outputs = Dense(1, activation='sigmoid')(x)
model = Model(inputs=inputs, outputs=outputs)
```

```
model.evaluate(test_dt)
```

```
19/19 [==============================] - 1s 47ms/step - loss: 0.1979 - accuracy: 0.9333
```



سوال 5)

منبع:

Chatgpt(prompt: give me practical example that show advantages of

convolutional laayers. Also say an example that show disadvantages of convolutional layers, also said performances of them)

جواب:

مثالی برای نمایش مزیت لایه های کانولوشنی:

<<استخراج ویژگی در cnn برای تشخیص اشیا>>

ویژگی مورد بررسی: آموزش ویژگی ها به صورت سلسله مراتبی.

استفاده از یک مدل cnn از قبل آموزش دیده مانند vgg یا resnet برای انتقال یادگیری روی مجموعه داده های تصویربرداریشده پزشکی برای بیماری هایی همچون ذات الریه.

در این مثال، لایه های کانولوشنی در مدل از پیش آموزش دیده مان، نمایش سلسله مراتبی را در مجموعه داده ها در مقیاس بزرگ مانند imagenet یاد گرفته اند. وقتی این این لایه ها روی دامنه های متفاوت اعمال شوند ، این لایه ها به عنوان استخراج کننده ویژگی های قدرتمند عمل می کنند. لای های پایین تر ویژگی های اساسی همچون لبه ها و باغت هارو میگیرند و لایه های عمیق تر الگو های پیچیده تری را بررسی می نمایند.استفاده از این ویژگی های سلسله مراتبی اموخته شده، به شکل قابل توجهی نیاز به آموزش را از ابتدا کاهش میدهده و به یک دامنه جدید تطبیق داده شده و تعمیم می یابد و داده های نسبتا کم، به دقت بالایی نمی رسد.

مثالی برای نمایش چالش لایه های کانولوشنی:

<< تقسیم بندی معنایی با cnn در رانندگی خودمختار>>

ویژگی مورد بررسی:Loss of Global Context

در این مثال به پیاده سازی یک مدل تقسیم بندی معنایی بر اساس cnn برای وظایف ماشین های خودران اشاره می شود. در تقسیم بندی، cnn هابرچسبی به هر پیکسل تخصیص داده و مشخص می کنند که پیکسل به کدام کلاس تعلق دارد.(جاده ، انسان، خودرو و...) . به دلیل میدان های دریافتی محلی لایه کانولوشن، مدل ممکن است فاقد درک وسیع تری از صحنه شوند. به شکل مصال ممکنه یک عابر در نزدیکی خط عابرپیاده ، به عنوان بخشی از جاده در نظر گرفته وطبقه بندی شود که فاقد global context است که انسان درنظر گرفته است. زمینه های محلی که توسط لایه های کانولوشنی آموخته می شود، ممکن است که سبب طبقه بندی نادرست و خطا در درک محیط های پیچیده شود روی ایمنی و قابلیت اطمینان به خودرو خودران اثر گذارد.

به شکل کلی این دو مثال به ترتیب نمایش دهنده آن اند که چگونه توانایی یادگیری ویژگی های سلسله مراتبی لایه های کانولوشنی به وظایفی همچون یادگیری انتقالی کمک نموده و هماهنگ سازی با حوزه های جدید را ممکن سازد، و دیگری هم محدودیت ذاتی local context، می تواند در سناریوهای مربوط به Global Context، برای پیش بینی های سریع و مهم چالش ایجاد نماید.

سوال6)

الف)

فیلتر 1*1 برای کاهش ابعاد و تبدیل وِیژگی در شبکه های cnn استفاده میشوند . از علائم آن کاهش ابعاد می شود که تعداد کانال ها و ویژگی های را در یک لایه کاهش می دهند و سبب کاهش پیچیدگی محاسباتی و استفاده از حافظه می شود. همچنین علیرغم سایز کوچک آن سبب اعمال تبدیلات غیرخطی شده و سبب آموزش روابط پیچیده ویژگی ها می شود. همچنین با استفاده از ترکیب وزنی از ویژگی های لایه قبلی ، به حفظ اطلاعات حیاتی کمک نموده ، ولی جزئیات کمتر مرتبط را کنار گذاشته و ignore می نامید و به جریان کارآمد اطلاعات و انتخاب ویژگی کمک می نماید.

ب)

نقشه ویژگی حاصل از این فیلتر ، ترکیبی خطی از ویژگی های ورودی در آن موقعیت مکانی را نشان می دهد که شامل یک نسخه تغییر یافته از ویژگی های ورودی که در آن هر عنصر یک مجموع وزنی از مقادیر ویژگی ورودی در آن مکان است، با وزن هایی که در طول آموزش آموخته می شود.

پ)

نقشه ویژگی به دست آمده از این فیلتر، از تصویر اصلی متمایز بوده ، چون اطلاعات انتزاعی سطح بالاتری را که از ترکیب خطی ویژگی ها میابد را رمز میکند. فیلترهایی با ابعاد فضایی بزرگتر ، الگوها و روابط فضایی را در میدان های مورد نظر ثبت نموده و اطلاعات مکانی را حفظ می نمایند، درحالی که فیلتر 1*1 بیشتر برای تبدیل کانال و کاهش ابعاد بدون گسترش فضایی تمرکز می کنند.

ت)

برای این مورد می توان به شبکه های Inception Network،Residual Networks (ResNets) و Squeeze-and-Excitation Networks (SENet) اشاره نمود. درInception Network ماژول های آغازین، شاخه های موازی فیلترهای با اندازه های مختلف، از جمله پیچیدگی های x11 را در خود جای می

دهند تا اطلاعات را در مقیاس های مختلف به طور موثر نگه میدارند. در ResNets بلوک های باقیمانده از پیچیدگی های 1x1 برای تنظیم تعداد کانال ها و کنترل پیچیدگی مدل استفاده می کند.

ث)

بله. در شبکه های کوچک و سلسله مراتب ویژگی های محدود. در معماری های اده تر یا شبکه های کم عمق، تاثیر پیچیدگی های 1*1 ممکن است حداقل یا غیرضروری باشد، چون ممکن است مدل پیچیدگی یا عمق کافی برای بهره مندی از تغییرات کانال را نداشته باشد. همچنین وقتی مجموعه داده کوچک است یا فاقد ویژگی های سلسله مراتبی است، فیلترهای 1*1ممکن است عملکرد را به طور قابل توجهی افزایش ندهند، زیرا اثربخشی آنها به توانایی شبکه برای یادگیری نمایش های معنادار در لایه ها بستگی دارد.

ج)

کد ساده نوشته شده در فایل پیوست در کنار داک، بدین شکل است که یک تصویر دلخواه 4*4 با 3 کانال ایجاد و مدل cnnساده ای را با لایه کانولوشنال 1*1 تعریف میکنیم. سپس ورودی تصادفی را تولید و آن را به مدل می دهیم تا خروجی محاسبه شود .

همانطور که از نتایج خروجی کد ومشاهده معماری مدل مشخص است، شکل تصویر ورودی (1، 4، 4، 3) و نمایش دهنده تصویر با سایز 4*4 و 3کانال رنگی است. همچنین شکل خروجی پس از کانولوشن 1*1 می شود: (1، 4، 4، 3) که نمایش می دهد که خرویجی همان ابعاد و تغداد کانال های ورودی را حفظ می کند. در واقع در این کد، ابعاد فضای مان را تغییر نمی دهد و تنها اطلاعات کانال را با حفظ اندازه نقشه های ویژگی تغییر می دهد.

```
import tensorflow as tf

input_image = tf.random.normal((1, 4, 4, 3))
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(filters=3, kernel_size=1, strides=1, padding='valid', activation='linear', input_shape=(4, 4, 3))
])
```

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_1 (Conv2D)           (None, 4, 4, 3)           12

=================================================================
Total params: 12 (48.00 Byte)
Trainable params: 12 (48.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
    output = model.predict(input_image)

    print(f"Input image shape: {input_image.shape}")
    print(f"Output shape after 1x1 convolution: {output.shape}")
```

```
1/1 [==============================] - 0s 41ms/step
Input image shape: (1, 4, 4, 3)
Output shape after 1x1 convolution: (1, 4, 4, 3)
```

سوال7)

ماژول inception یک جز مهم در شبکه های عصبی کانولوشن می باشد که برای طبقه بندی تصاویر استفاده می شود. این موارد برای نگهداری داده ها و پردازش اطلاعات در مقیاس های مختلف یا اندازه های فیلد ها در لایه های یکسان است. این معماری به یادگیری همزمان ویژگی های محلی با کانولوشن با سایز کوچکتر، و الگوریتم های جهانی با سایز کانولوشن بزرگتر کمک می نماید. در پیاده سازی، ماژول مان از برنچ های موازی لایه های کانولوشن با سایز های متفاوت ، همراه maxpooling تشکیل شده.این موارد به طور همزمان و موازی روی ورودی های لایه قبلی مار نموده و خروجی های خود را قبل از ارسال به لایه بعدی بهم متصل می نماید.این ساختار سبب می شود تا ویژگی های متنوعی را در مقیاس های مختلف نگه داری نماید.

همچنین با توجه به تاثیر گام ها در لایه های کانولوشنی بر روی نقشه برداری ویژگی ها ، تغییر پارامتر گام بر ابعاد نقشه های ویژگی اثر می گذارد. ابعاد گام بزرگتر، سبب نقشه ویژگی کوچکتر(ابعاد فضایی کوچکتر) شود. زیرا پیکسل های بیشتری را در کانولوشن، رد می نماید. برعکس، گام کوچکتر، وضوح فضایی بالاتری را در نقشه ویژگی ها حفظ می کند.

در ماژول نوشته شده ، پیچیدگی 1*1 برای کاهش ابعاد و کارایی محاسباتی با استفاده از یک هسته کوچکتر برای انحراف روی ورودی کمک می کند. پیچیدگی 3*3 الگوها یا ویژگی های با اندازه متوسط را در داده های ورودی ثبت می کند. همچنین پیچیدگی 5*5 برای الگوها با ویژگی های بزرگتر است. و همچنین لایه maxpooling ، ابعاد فضا را کاهش داده و ویژگی های غالب را به تصویر می کشد. برای تقویت این ماژول از نرمال سازی بچ برای سرعت بخشیدن به روند آموزش و بهبود نتیجه آن می شود.

```python
def module_inception(prev_layer):
    conv1x1 = Conv2D(16, (1, 1), padding='same', activation='relu')(prev_layer)
    conv1x1 = BatchNormalization()(conv1x1)

    conv3x3_reduce = Conv2D(16, (1, 1), padding='same', activation='relu')(prev_layer)
    conv3x3_reduce = BatchNormalization()(conv3x3_reduce)
    conv3x3 = Conv2D(32, (3, 3), padding='same', activation='relu')(conv3x3_reduce)
    conv3x3 = BatchNormalization()(conv3x3)

    conv5x5_reduce = Conv2D(16, (1, 1), padding='same', activation='relu')(prev_layer)
    conv5x5_reduce = BatchNormalization()(conv5x5_reduce)
    conv5x5 = Conv2D(32, (5, 5), padding='same', activation='relu')(conv5x5_reduce)
    conv5x5 = BatchNormalization()(conv5x5)

    maxpool = MaxPooling2D((3, 3), strides=(1, 1), padding='same')(prev_layer)
    pool_proj = Conv2D(16, (1, 1), padding='same', activation='relu')(maxpool)
    pool_proj = BatchNormalization()(pool_proj)

    return Concatenate(axis=-1)([conv1x1, conv3x3, conv5x5, pool_proj])


inp_layer = Input(shape=(32, 32, 3))
x = Conv2D(64, (3, 3), padding='same', activation='relu')(inp_layer)
x = module_inception(x)
x = MaxPooling2D((2, 2))(x)
x = Flatten()(x)
x = Dense(128, activation='relu')(x)
outp_layer = Dense(10, activation='softmax')(x)
```

```python
inp_layer = Input(shape=(32, 32, 3))
x = Conv2D(64, (3, 3), padding='same', activation='relu')(inp_layer)
x = module_inception(x)
x = MaxPooling2D((2, 2))(x)
x = Flatten()(x)
x = Dense(128, activation='relu')(x)
outp_layer = Dense(10, activation='softmax')(x)
```

```python
model = Model(inputs=inp_layer, outputs=outp_layer)
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history = model.fit(train_dt, train_lbls, epochs=10, batch_size=64, validation_data=(test_dt, test_lbls))
```

```
Epoch 1/10
782/782 [==============================] - 27s 16ms/step - loss: 1.5255 - accuracy: 0.4734 - val_loss: 1.7727 - val_accuracy: 0.4105
Epoch 2/10
782/782 [==============================] - 12s 15ms/step - loss: 1.0025 - accuracy: 0.6416 - val_loss: 1.0390 - val_accuracy: 0.6335
Epoch 3/10
782/782 [==============================] - 11s 15ms/step - loss: 0.8178 - accuracy: 0.7150 - val_loss: 1.0190 - val_accuracy: 0.6480
Epoch 4/10
782/782 [==============================] - 11s 14ms/step - loss: 0.6810 - accuracy: 0.7648 - val_loss: 1.0161 - val_accuracy: 0.6620
Epoch 5/10
782/782 [==============================] - 11s 14ms/step - loss: 0.5522 - accuracy: 0.8096 - val_loss: 1.0818 - val_accuracy: 0.6592
Epoch 6/10
782/782 [==============================] - 11s 14ms/step - loss: 0.4367 - accuracy: 0.8500 - val_loss: 1.1698 - val_accuracy: 0.6593
Epoch 7/10
782/782 [==============================] - 11s 14ms/step - loss: 0.3435 - accuracy: 0.8814 - val_loss: 1.2326 - val_accuracy: 0.6720
Epoch 8/10
782/782 [==============================] - 11s 14ms/step - loss: 0.2668 - accuracy: 0.9086 - val_loss: 1.3745 - val_accuracy: 0.6726
Epoch 9/10
782/782 [==============================] - 11s 14ms/step - loss: 0.2151 - accuracy: 0.9270 - val_loss: 1.5719 - val_accuracy: 0.6595
Epoch 10/10
782/782 [==============================] - 11s 14ms/step - loss: 0.1830 - accuracy: 0.9381 - val_loss: 1.7352 - val_accuracy: 0.6583
```