

## به نام خالق رنگین کمان

ستاره باباجانی – 99521109

سوال 1:

الف) استفاده از نرخ یادگیری بسیار بالا در آموزش یک شبکه عصبی می تواند به چندین موضوع منجر شود:

1. واگرایی: یک نرخ یادگیری بسیار بالا ممکن است مانع از همگرایی مدل شود و منجر به واگرایی شود و روند آموزش را بی اثر کند.
  2. ناپایداری: نرخ یادگیری بالا می تواند باعث بی ثباتی در فرایند یادگیری شود. به جای هموار کردن حرکت به سمت حداقل ، بهینه سازی ممکن است به طرز بدی نوسان کند یا رفتار نامنظم نشان دهد.
  3. Overshooting the Minimum: الگوریتم ممکن است از حداقل مورد نیاز برای دستیابی پرش کند و نقطه بهینه در ضرر را از دست دهد.
  4. تعمیم دهی ضعیف: نرخ یادگیری بالا می تواند باعث شود که مدل خیلی سریع همگرا شود و احتمالاً منجر به تعمیم ضعیف در داده های دیده نشده، شود. این مدل ممکن است در مجموعه تمرینات عملکرد خوبی داشته باشد اما در مورد اعتبار سنجی یا آزمون ضعیف باشد.
- این مشکلات را می توان با مشاهده رفتار فرایند آموزش و ارزیابی عملکرد مدل تشخیص داد:

1. Loss function: نظارت بر آموزش و از دست دادن اعتبار سنجی. اگر ضرر بعد از هر دوره کاهش یا افزایش نمی یابد ، این نشانگر یادگیری ناپایدار است که به طور بالقوه ناشی از نرخ یادگیری بالا است.
2. رفتار نامنظم: اگر روند آموزش پرش های نامنظم یا نوسانات در میزان ضرر را نشان دهد، این نشانگر نرخ یادگیری بالا است.

3. معیارهای اعتبار سنجی: عملکرد مدل را در یک مجموعه اعتبار سنجی باید ارزیابی کرد. اگر عملکرد مدل بدتر شود یا بهبود نیافته باشد، ممکن است ناشی از نرخ یادگیری بالا باشد که باعث می شود در حین بهینه سازی، پارامترهای بهینه مدل از دست داده شود.

برای پرداختن به این مشکلات، تنظیم مناسب نرخ یادگیری (با تنظیم دستی آن یا استفاده از تکنیک هایی مانند الگوریتم های نرخ یادگیری تطبیقی) مهم است. اعتبار سنجی متقابل نیز می تواند در تعیین نرخ یادگیری بهینه برای مدل و مجموعه داده خاص کمک کننده باشد.

ب) استفاده از نرخ یادگیری بسیار پایین در آموزش یک شبکه عصبی می تواند به موارد زیر منجر شود:

1. همگرایی آهسته: با نرخ یادگیری بسیار پایین، مدل برای همگرایی به حداقل ضرر، بیشتر طول می کشد. این می تواند به طور قابل توجهی زمان مورد نیاز برای آموزش را افزایش دهد، به خصوص برای مدل های پیچیده یا مجموعه داده های بزرگ.

2. Stuck in Local Minima or Plateaus: نرخ یادگیری بسیار پایین ممکن است باعث شود الگوریتم بهینه سازی در مینیم محلی یا فلات محلی گیر بیفتد طوری که فرار از این مناطق برای مدل دشوار می باشد.

3. Limited Exploration of the Parameter Space: نرخ یادگیری بسیار پایین، اکتشاف فضای پارامتر را محدود می کند، به طور بالقوه مانع از یافتن راه حل های بهتر بهتر از مقداردهی اولیه می شود.

این مشکلات را می توان از طریق مشاهدات مختلف در طی فرایند آموزش تشخیص داد:

1. کاهش آهسته در ضرر: باید به ضرر در حین آموزش و اعتبار سنجی نظارت کرد. اگر ضرر در بسیاری از دوره ها بدون پیشرفت قابل توجهی،

بسیار آهسته کاهش یابد ، ممکن است نشان دهد که میزان یادگیری خیلی کم است.

2. زمان آموزش طولانی: آموزش به ویژه با توجه به پیچیدگی مدل و مجموعه داده ، مدت زمان طولانی غیر منطقی طول می کشد. اگر مدل علائم همگرایی را در یک بازه زمانی معقول نشان ندهد ، نرخ یادگیری ممکن است یک عامل کمک کننده باشد.

3. Suboptimal Performance: با وجود آموزش برای مدت زمان قابل توجهی ، عملکرد مدل زیر انتظارات باقی مانده می ماند. این می تواند نتیجه ای باشد که مدل به دلیل کم بودن یادگیری در یک منطقه suboptimal به دام می افتد.

پ) نقطه زینی در بهینه سازی به نقطه ای در فضای پارامتر اطلاق می شود که شیب تابع صفر است، اما برخلاف حداقل یا حداکثر، انحنا تابع در همه جهات یکسان نیست. در یک جهت، تابع به سمت بالا منحنی می شود، در حالی که در جهت دیگر، به سمت پایین منحنی می شود، شبیه به شکل یک زین. در نقاط زینی، الگوریتم های بهینه سازی مبتنی بر گرادیان ممکن است گیر کنند زیرا ممکن است به دلیل گرادیان صفر، نقطه را به عنوان حداقل تفسیر کنند و همگرایی را کاهش دهند.

• Adam:

1. برخورد با نقاط زین: آدام به دلیل مکانیسم سرعت یادگیری تطبیقی و momentum، در مسیریابی نقاط زینی بهتر از SGD عمل می کند. ماهیت تطبیقی به آن کمک می کند تا بر مشکل همگرایی کند در اطراف نقاط زینی، که SGD سنتی ممکن است با آن مواجه شود، غلبه کند.

2. مزایا:

- نرخ های یادگیری تطبیقی: نرخ های یادگیری را برای هر پارامتر به صورت جداگانه بر اساس گرادیان های گذشته آنها تنظیم می کند، که منجر به همگرایی سریع تری می شود.

- Momentum: از اطلاعات گرادیان های گذشته برای تسریع همگرایی، به ویژه در مناطق با انحنای بالا استفاده می کند.
  - Robustness: آدام نسبت به SGD وانیلی حساسیت کمتری نسبت به ابرپارامترهایی مانند نرخ یادگیری دارد.
3. معایب:

- استفاده از حافظه: Adam میانگین های متحرک در حال کاهش گرادیان های مجذور گذشته و گرادیان های گذشته را برای هر پارامتر جمع آوری می کند، که می تواند نیاز به حافظه را افزایش دهد.
- پیچیدگی: Adam شامل ابرپارامترهای بیشتری برای تنظیم در مقایسه با SGD ساده است که ممکن است برای یافتن تنظیمات مناسب به تلاش بیشتری نیاز داشته باشد.
- عملکرد در برخی مشکلات: اگرچه به طور کلی موثر است، Adam ممکن است همیشه در انواع خاصی از مشکلات یا معماری ها بهتر از SGD عمل نکند.

## • SGD:

1. برخورد با نقاط زین: SGD سنتی به دلیل نرخ یادگیری ثابت و فقدان ویژگی های تطبیقی ممکن است در نقاط زینی بیشتر با مشکل مواجه شود. ممکن است در مجاورت نقاط زینی گیر کند و همگرایی را کندتر کند.
2. مزایا:

- سادگی: SGD نسبتاً ساده است و در مقایسه با الگوریتم های بهینه سازی پیچیده تر مانند Adam، تنظیمات ابرپارامترهای کمتری دارد.
- سهولت در تفسیر: درک و تفسیر آن در مقایسه با برخی از الگوریتم های بهینه سازی تطبیقی آسان تر است.

- استفاده از حافظه کمتر: SGD به حافظه کمتری نیاز دارد زیرا گرادین های گذشته و عبارت مربع آنها را ذخیره نمی کند.

### 3. معایب:

- حساسیت به میزان یادگیری: عملکرد SGD به شدت به انتخاب نرخ یادگیری مناسب بستگی دارد. اگر خیلی زیاد باشد، ممکن است واگرا شود، و اگر خیلی کم باشد، ممکن است به آرامی همگرا شود یا گیر کند.
- همگرایی کندتر: در مقایسه با بهینه سازهای پیچیده تر مانند Adam، SGD ممکن است کندتر همگرا شود، به خصوص در سطوح پیچیده با نقاط زینی زیاد یا مناطق با انحنا بالا.
- گیر کردن در نقاط حداقل یا زین محلی: SGD ممکن است در مینیم های محلی یا نقاط زین به دام بیفتد، همگرایی را کاهش دهد یا از فرار جلوگیری کند.

(ت)

- نمودار ضرر با نویز (mini-batch gradient decent): نمودار سمت راست نزول گرادین دسته ای کوچک (mini-batch) را نشان می دهد، جایی که به روز رسانی ها بر اساس یک زیر مجموعه تصادفی کوچک از داده های آموزشی در هر تکرار است.
- نمودار نویزدار است زیرا استفاده از mini-batch های مختلف باعث ایجاد تنوع می شود و باعث نوسانات در منحنی ضرر می شود.
- نمودار تلفات هموار (batch gradient decent): نمودار سمت چپ نشان دهنده نزول گرادین دسته ای است، جایی که کل مجموعه داده آموزشی برای هر به روز رسانی استفاده می شود.

منحنی ضرر صاف است زیرا کل مجموعه داده را در نظر می گیرد و در هر تکرار یک به روز رسانی پایدار و قطعی ارائه می دهد.

سوال 2:

سوال ۷: ابتدا مراحل Forward pass را انجام داده و خروجی فیلتر GAP را محاسبه کنید.

$$X = \begin{bmatrix} 3 & 4 & 5 \\ 2 & 1 & -3 \\ 4 & -4 & 0 \end{bmatrix} \times F = \begin{bmatrix} 2 & 0 \\ -3 & 1 \end{bmatrix} = Y = \begin{bmatrix} 1 & 2 \\ -10 & 1 \end{bmatrix}$$

حال در این خروجی، GAP را محاسبه می‌کنیم. ابتدا باید ببینیم که این فیلتر چگونه عمل می‌کند.

$$GAP(Y) = \frac{1}{M \times N} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} Y[i, j]$$

$$\Rightarrow GAP(Y) = \frac{1}{2 \times 2} \times (1 + 2 - 10 + 1) = \frac{1}{2} = 0.5$$

این عملیات در این مرحله انجام می‌گیرد. در مرحله بعد، back propagation را انجام می‌دهیم.

① جهت محاسبه گرادیان، ابتدا باید ببینیم که چگونه می‌توانیم این را محاسبه کنیم.

$$\frac{\partial L}{\partial Y} = \frac{\partial L}{\partial GAP(Y)} = 1$$

② 
$$Y_{11} = X_{11} F_{11} + X_{12} F_{12} + X_{21} F_{21} + X_{22} F_{22}$$

$$\frac{\partial Y_{11}}{\partial F_{11}} = X_{11}, \frac{\partial Y_{11}}{\partial F_{12}} = X_{12}, \frac{\partial Y_{11}}{\partial F_{21}} = X_{21}, \frac{\partial Y_{11}}{\partial F_{22}} = X_{22}$$

این عملیات را برای تمام index ها انجام می‌دهیم.

II) chain rule 
$$\frac{\partial L}{\partial F} = \frac{\partial Y}{\partial F} \times \frac{\partial L}{\partial Y} \times \frac{\partial Y}{\partial F}$$

$$\frac{\partial L}{\partial F} = \begin{bmatrix} \frac{\partial Y}{\partial F_{11}} & \frac{\partial Y}{\partial F_{12}} \\ \frac{\partial Y}{\partial F_{21}} & \frac{\partial Y}{\partial F_{22}} \end{bmatrix} = \begin{bmatrix} 3 & 4 \\ 2 & -1 \end{bmatrix}$$



- $Dense\ parameters = (inputsize + 1) \times neurons$

$Outputshape = neurons$

حال طبق فرمول های گفته شده، جدول خروجی و تعداد پارامتر را رسم میکنیم:

layers	parameters	output dimensions
Conv1D = (filters=16, kernel=3)	$16 * (3 * 7 + 1) = 352$	$(500 - 3 + 1, 16) = (498, 16)$
MaxPool1D	0	$((498-2) / 2 + 1, 16) = (249, 16)$
Conv1D = (filters=32, kernel=5)	$32 * (5 * 16 + 1) = 2592$	$(249 - 5 + 1, 32) = (245, 32)$
MaxPool1D	0	$((245-2) / 2 + 1, 32) = (122, 32)$
Conv1D = (filters=64, kernel=5)	$64 * (5 * 32 + 1) = 10304$	$(122 - 5 + 1, 64) = (118, 64)$
MaxPool1D	0	$((118-2) / 2 + 1, 64) = (59, 64)$
Flatten	0	$(59 * 64) = (3776,)$
Dense(neurons=128)	$(3776 + 1) * 128 = 483456$	(128)
Dense(neurons=5)	$(128 + 1) * 128 = 645$	(5)

(ب)

## • Conv2D:

- عملیات فضایی: روی داده های ورودی فضایی 2 بعدی مانند تصاویر یا شبکه ها عمل می کند.
- فیلترها/هسته ها: فیلترهای دوبعدی با ورودی دوبعدی ترکیب می شوند تا نقشه ویژگی خروجی دوبعدی تولید شود.
- برنامه ها: به طور گسترده در وظایف بینایی کامپیوتری مانند طبقه بندی تصویر، تشخیص اشیا و تقسیم بندی که در آن روابط فضایی به صورت دو بعدی است استفاده می شود.



## • Conv3D:

- عملیات مکانی و زمانی: بر روی داده های ورودی سه بعدی با در نظر گرفتن ابعاد مکانی و زمانی عمل می کند.
- فیلترها/هسته ها: فیلترهای سه بعدی با ورودی سه بعدی (داده های حجمی، ویدئوها) برای تولید نقشه ویژگی خروجی سه بعدی ترکیب می شوند.
- موارد استفاده از Conv3D: لایه های Conv3D در حوزه های مختلفی کاربرد پیدا می کنند که در آن داده ها نه تنها شامل اطلاعات مکانی بلکه زمانی نیز می شوند. برخی از برنامه های کاربردی قابل توجه عبارتند از:
  - تحلیل ویدئو: لایه های Conv3D برای کارهایی که شامل تجزیه و تحلیل داده های ویدئویی می شود، در جایی که اطلاعات مکانی و زمانی اهمیت دارند، مانند تشخیص کنش، طبقه بندی ویدئو، یا تقسیم بندی ویدئو، ارزشمند هستند.
  - طبقه بندی و تقسیم بندی ویدئو: طبقه بندی یا تقسیم بندی اشیا در داده های ویدئویی با در نظر گرفتن تکامل اشیا در طول زمان.
  - تصویربرداری پزشکی: تجزیه و تحلیل حجم های سه بعدی داده های پزشکی برای تشخیص بیماری، تقسیم بندی یا ردیابی تغییرات در طول زمان.
  - هواشناسی و مدل سازی آب و هوا: داده های آب و هوا اغلب شامل اطلاعات حجمی در طول زمان است. لایه های Conv3D را می توان برای تحلیل و مدل سازی الگوهای آب و هوا، شبیه سازی تغییرات آب و هوا و پیش بینی پدیده های جوی با در نظر گرفتن ماهیت سه بعدی داده ها به کار برد.
  - اکتشافات ژئوفیزیکی: در ژئوفیزیک، لایه های Conv3D به تجزیه و تحلیل داده های لرزه ای کمک می کنند و به شناسایی سازه های زیرسطحی کمک می کنند. این در اکتشاف نفت، مطالعات زیست محیطی و پیش بینی زلزله بسیار مهم است.

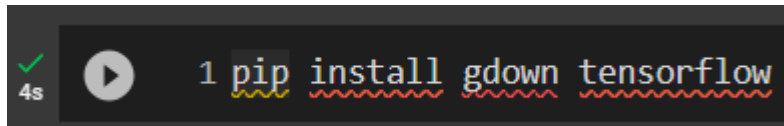
▪ ژست و تشخیص زبان اشاره: لایه‌های Conv3D برای تشخیص حرکات و زبان اشاره در توالی‌های ویدیویی استفاده می‌شوند. با در نظر گرفتن بعد زمانی، این شبکه‌ها می‌توانند الگوهای پویا مرتبط با ژست‌های مختلف را درک کنند.

سوال 4: برای دسته‌بندی دادگان تومور مغزی، مراحل زیر را طی می‌کنیم:

1. صدا زدن کتابخانه‌های مورد نیاز:

```
[2] 1 import tensorflow as tf
    2 from tensorflow import keras
    3 import matplotlib.pyplot as plt
    4 from tensorflow.keras.models import Sequential, Model
    5 from tensorflow.keras.layers import Dense, Conv2D, MaxPool2D, Activation, Input, Flatten, Rescaling
    6 from tensorflow.keras.optimizers import Adam
    7 from tensorflow.keras.callbacks import EarlyStopping
```

2. نصب کتابخانه مورد نیاز برای دانلود از گوگل درایو:



```
✓ 4s 1 pip install gdown tensorflow
```

3. دانلود فایل با استفاده از کتابخانه gdown و در نهایت آنزیمپ کردن فایل زیپ دانلود شده:

```
1 import gdown
2
3 # Define the Google Drive file ID and the output directory
4 file_id = '1SCpVEDJ6_YOAcY2iW05ENIMh-OCcFz3P'
5 output_file = 'dataset.zip'
6 gdown.download(f'https://drive.google.com/uc?id={file_id}', output_file, quiet=False)
7
8 #unzipping the downloaded file
9 !unzip dataset.zip -d dataset
```

4. حال در بخش بعدی خواسته شده تا از دایرکتوری ذخیره شده، فایل را بخوانیم و دیتای آموزشی و صحت‌سنجی و تست را ذخیره کنیم. حال ابتدا طبق خواسته سوال داده را خوانده و 20 درصد آن را مخصوص صحت‌سنجی و تست می‌کنیم:

```

1 (training_data, original_validation_data) = tf.keras.utils.image_dataset_from_directory(
2     '/content/dataset',
3     labels='inferred',
4     label_mode='int', # Labels are represented as integers
5     class_names=['no', 'yes'],
6     color_mode='grayscale',
7     batch_size=64,
8     image_size=(256, 256),
9     validation_split=0.2,
10    subset='both', # Use both training and validation subsets
11    seed=20
12 )

```

```

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

```

سپس دوباره داده های صحت سنجی (validation) و تست را با نسبت 0.2 جدا میکنیم:

```

13
14 #split the original validation data to two parts(for validation and test)
15 validation_size = int(0.2 * len(original_validation_data))
16 test_data = original_validation_data.take(validation_size)
17 validation_data = original_validation_data.skip(validation_size)

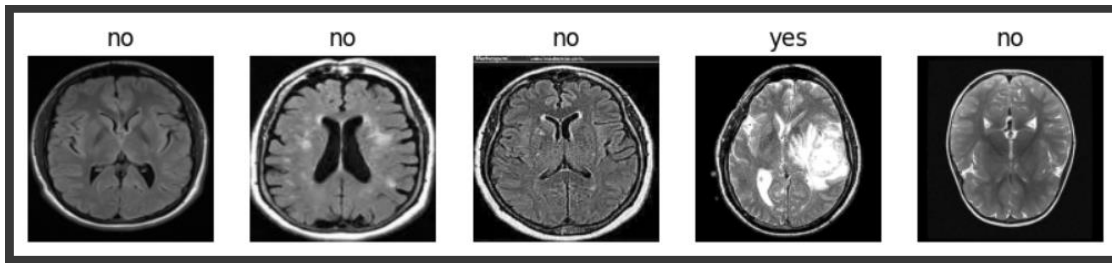
```

5. سپس، 5 نمونه از داده های آموزشی را رسم کردیم و لیبل آنها را برای درک بهتر نمایش دادیم:

```

1 num_samples = 5
2 class_names = ['no', 'yes'] # Class names used in the dataset
3
4 # Iterate through the training dataset to extract a few samples
5 for images, labels in training_data.take(1): # 1 batch
6     plt.figure(figsize=(10, 5))
7     for i in range(num_samples):
8         ax = plt.subplot(1, num_samples, i + 1)
9         plt.imshow(images[i].numpy().astype("uint8"), cmap='gray') # Display grayscale images
10        plt.title(class_names[labels[i]])
11        plt.axis('off')
12    plt.show()
13

```



6. حال مدل sequential ساده ای طراحی خواهیم کرد. این مدل شامل یک لایه کانولوشنی و پول است. سپس flat شده و در آخر برای جواب نهایی از دو لایه dense استفاده شده است:

```
1 seq_model = Sequential([
2     Conv2D(16, (3, 3), activation='relu', input_shape=(256, 256, 1)),
3     MaxPool2D((2, 2)),
4     Flatten(),
5     Dense(32, activation='relu'),
6     Dense(1, activation='sigmoid') # Assuming binary classification ('yes' or 'no')
7 ])
8
9 # Compile the model
10 seq_model.compile(optimizer='adam',
11                  loss='binary_crossentropy',
12                  metrics=['accuracy'])
```

چون یک دسته بندی دو کلاسه داریم، در لایه آخر تابع فعال سازی سیگموئید قرار داده شد. همچنین بهینه ساز آدام است و از binary\_crossentropy برای تابع ضرر استفاده شده است.

7. حال مدل را train میکنیم:

```
1 history = seq_model.fit(training_data, epochs=5, validation_data=validation_data)

Epoch 1/5
38/38 [=====] - 7s 110ms/step - loss: 346.7252 - accuracy: 0.6958 - val_loss: 33.1542 - val_accuracy: 0.8051
Epoch 2/5
38/38 [=====] - 4s 95ms/step - loss: 12.0633 - accuracy: 0.8742 - val_loss: 8.7120 - val_accuracy: 0.8729
Epoch 3/5
38/38 [=====] - 4s 92ms/step - loss: 1.6999 - accuracy: 0.9567 - val_loss: 2.9055 - val_accuracy: 0.9470
Epoch 4/5
38/38 [=====] - 3s 74ms/step - loss: 0.5304 - accuracy: 0.9825 - val_loss: 2.6210 - val_accuracy: 0.9492
Epoch 5/5
38/38 [=====] - 3s 75ms/step - loss: 0.4067 - accuracy: 0.9817 - val_loss: 1.7957 - val_accuracy: 0.9534
```

8. در مرحله بعد مدل آموزش دیده را تست میکنیم:

```
1 test_loss, test_accuracy = seq_model.evaluate(test_data)

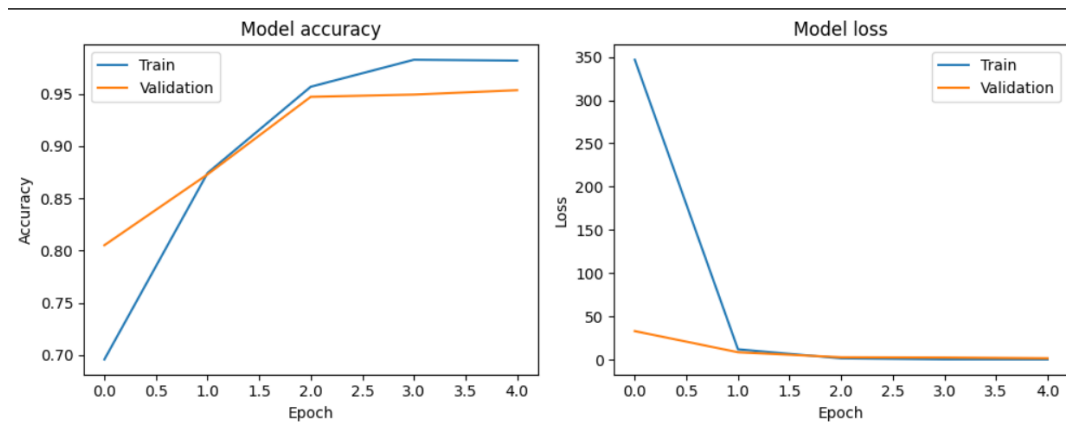
2/2 [=====] - 0s 67ms/step - loss: 2.7872 - accuracy: 0.9531
```

همان طور که مشاهده میشود دقت در آموزش بالای 98 درصد و در تست بالای 95 درصد بوده است که برای این مدل ساده بسیار خوب است.

```
Test Accuracy: 95.31%
Test Loss: 2.7872
```

9. حال ضرر و دقت را در حین آموزش و تست رسم میکنیم:

```
1 plt.figure(figsize=(10, 4))
2
3 # Plot training & validation accuracy values
4 plt.subplot(1, 2, 1)
5 plt.plot(history.history['accuracy'])
6 plt.plot(history.history['val_accuracy'])
7 plt.title('Model accuracy')
8 plt.xlabel('Epoch')
9 plt.ylabel('Accuracy')
10 plt.legend(['Train', 'Validation'], loc='upper left')
11
12 # Plot training & validation loss values
13 plt.subplot(1, 2, 2)
14 plt.plot(history.history['loss'])
15 plt.plot(history.history['val_loss'])
16 plt.title('Model loss')
17 plt.xlabel('Epoch')
18 plt.ylabel('Loss')
19 plt.legend(['Train', 'Validation'], loc='upper right')
20
21 plt.tight_layout()
22 plt.show()
23
24 print(f"Test Accuracy: {test_accuracy * 100:.2f}%")
25 print(f"Test Loss: {test_loss:.4f}")
```



همان طور که مشاهده میشود در دو حالت ضرر رو به کاهش و دقت و درستی رو به افزایش است.

10. حال تمامی مراحل را برای حالتی که مدل functional است انجام میدهیم:

```
1 input_shape = (256, 256, 1)
2 input_layer = Input(shape=input_shape)
3
4 x = Conv2D(16, (3, 3), activation='relu')(input_layer)
5 x = MaxPool2D((2, 2))(x)
6 x = Flatten()(x)
7 x = Dense(32, activation='relu')(x)
8 output_layer = Dense(1, activation='sigmoid')(x) # Assuming binary classification ('yes' or 'no')
9
10 func_model = Model(inputs=input_layer, outputs=output_layer)
11
12 func_model.compile(optimizer='adam',
13                    loss='binary_crossentropy',
14                    metrics=['accuracy'])
```

```
1 history = func_model.fit(training_data, epochs=5, validation_data=validation_data)

Epoch 1/5
38/38 [=====] - 8s 91ms/step - loss: 104.4830 - accuracy: 0.7325 - val_loss: 28.4824 - val_accuracy: 0.7818
Epoch 2/5
38/38 [=====] - 3s 72ms/step - loss: 5.4075 - accuracy: 0.9075 - val_loss: 3.5031 - val_accuracy: 0.9364
Epoch 3/5
38/38 [=====] - 5s 110ms/step - loss: 0.7278 - accuracy: 0.9729 - val_loss: 1.8890 - val_accuracy: 0.9597
Epoch 4/5
38/38 [=====] - 4s 75ms/step - loss: 0.3570 - accuracy: 0.9875 - val_loss: 1.7507 - val_accuracy: 0.9470
Epoch 5/5
38/38 [=====] - 4s 85ms/step - loss: 0.5698 - accuracy: 0.9729 - val_loss: 3.3692 - val_accuracy: 0.9174
```

```
1 test_loss, test_accuracy = func_model.evaluate(test_data)

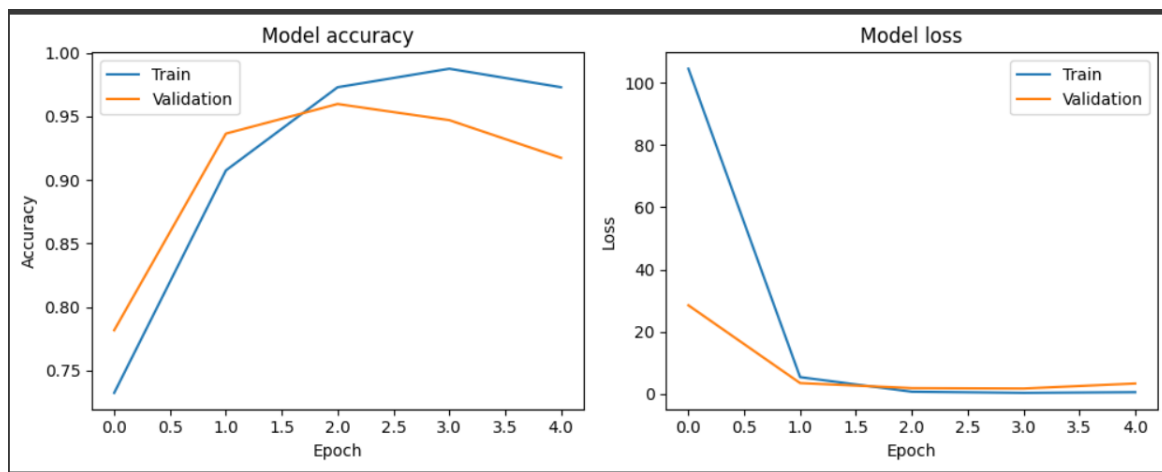
2/2 [=====] - 0s 123ms/step - loss: 4.1135 - accuracy: 0.9062
```

همان طور که مشاهده میشود دقت در آموزش به بالای 97 درصد و در تست به بالای 90 درصد رسیده است.

```
Test Accuracy: 90.62%
Test Loss: 4.1135
```

11. حال در نهایت نمودار ضرر و دقت با این مدل در حین آموزش و تست را رسم میکنیم:

```
1 plt.figure(figsize=(10, 4))
2
3 # Plot training & validation accuracy values
4 plt.subplot(1, 2, 1)
5 plt.plot(history.history['accuracy'])
6 plt.plot(history.history['val_accuracy'])
7 plt.title('Model accuracy')
8 plt.xlabel('Epoch')
9 plt.ylabel('Accuracy')
10 plt.legend(['Train', 'Validation'], loc='upper left')
11
12 # Plot training & validation loss values
13 plt.subplot(1, 2, 2)
14 plt.plot(history.history['loss'])
15 plt.plot(history.history['val_loss'])
16 plt.title('Model loss')
17 plt.xlabel('Epoch')
18 plt.ylabel('Loss')
19 plt.legend(['Train', 'Validation'], loc='upper right')
20
21 plt.tight_layout()
22 plt.show()
23
24 print(f"Test Accuracy: {test_accuracy * 100:.2f}%")
25 print(f"Test Loss: {test_loss:.4f}")
```



سوال 5: سناریو: فرض کنید مجموعه داده‌ای از اسکن‌های مغزی MRI داریم تا مشخص کنیم که آیا اسکن‌ها نشانه‌هایی از یک وضعیت پزشکی خاص دارند یا طبیعی هستند.

استفاده از لایه‌های کانولوشنال:

- استخراج ویژگی:

- تشخیص لبه: لایه‌های کانولوشن در ثبت الگوها و ویژگی‌های سلسله مراتبی در تصاویر عالی هستند. در اسکن MRI، بافت‌ها و ساختارهای مختلف دارای بافت و الگوهای خاصی هستند. لایه‌های کانولوشنال با توانایی تشخیص لبه‌ها، بافت‌ها و اشکال در مقیاس‌های مختلف می‌توانند ویژگی‌های مرتبط مانند ناهنجاری‌ها یا ساختارهای خاص در مغز را شناسایی کنند.

- نمایش سلسله مراتبی فضایی:

- برهم‌بندی لایه‌ها: عمق شبکه‌های عصبی کانولوشن به آنها اجازه می‌دهد تا بازنمایی‌های انتزاعی را در سطوح مختلف انتزاع بیاموزند. لایه‌های پایین‌تر ممکن است ویژگی‌های ساده‌ای مانند لبه‌ها را شناسایی کنند، در حالی که لایه‌های عمیق‌تر این ویژگی‌ها را برای تشخیص الگوها و ساختارهای پیچیده‌تر در اسکن‌های MRI،



مانند مناطقی که تحت تأثیر یک وضعیت پزشکی قرار گرفته‌اند، ترکیب می‌کنند.

- عدم تغییر ترجمه:

- اتصال محلی: لایه های کانولوشن از اتصال محلی برای دستیابی به ویژگی های تغییر ناپذیر ترجمه بهره برداری می کنند. در اسکن MRI، محل دقیق یک ناهنجاری خاص ممکن است از یک اسکن به اسکن دیگر متفاوت باشد. ویژگی اتصال محلی مدل را قادر می سازد تا ویژگی ها را مستقل از موقعیت مکانی دقیق آنها بیاموزد و به تشخیص ناهنجاری ها بدون توجه به موقعیت آنها در تصویر کمک می کند.

- فضای پارامتر کاهش یافته:

- اشتراک گذاری پارامتر: از طریق اشتراک وزن، لایه های کانولوشن تعداد پارامترها را در مقایسه با لایه های کاملاً متصل کاهش می دهند. این ویژگی در برنامه های تصویربرداری پزشکی که ممکن است مجموعه داده ها محدود باشد بسیار سودمند است. با پارامترهای کمتر، مدل کمتر مستعد بیش از حد برازش می شود و بهتر می تواند به داده های جدید و نادیده تعمیم دهد.

- یادگیری نمایش سلسله مراتبی:

- مدل های از قبل آموزش دیده: یادگیری انتقال، که شامل استفاده از شبکه های عصبی کانولوشن از قبل آموزش دیده در مجموعه داده های بزرگ مانند ImageNet است، اجازه می دهد تا از ویژگی های سلسله مراتبی آموخته شده برای لایه های اولیه استفاده کنید. تنظیم دقیق این شبکه ها روی مجموعه داده های کوچکتر تصویربرداری پزشکی به تعمیم بهتر و همگرایی سریعتر کمک می کند.

ویژگی های منحصر به فرد لایه های کانولوشن در این سناریوی طبقه بندی تصویر به مدل اجازه می دهد الگوهای پیچیده که نشان دهنده شرایط پزشکی هستند و ممکن است برای چشم انسان آشکار نباشد را شناسایی کنیم. همچنین

ویژگی های آموخته شده را به اسکن های مغزی جدید و نادیده تعمیم دهیم، که منجر به دقت تشخیصی بهتر حتی با مجموعه داده محدود می شود.

چالش ها:

مثال: پیچیدگی محاسباتی در تصویربرداری پزشکی سه بعدی

- افزایش هزینه محاسباتی: در تصویربرداری پزشکی سه بعدی (مانند MRI یا سی تی اسکن)، استفاده از لایه های Conv3D منجر به افزایش پیچیدگی محاسباتی می شود. ماهیت حجمی داده ها نیاز به پردازش سه بعدی دارد و لایه های Conv3D را در مقایسه با لایه های کانولوشنال دوبعدی از نظر محاسباتی فشرده می کند. این پیچیدگی چالش هایی را از نظر زمان آموزش مدل و نیاز به منابع ایجاد می کند.
- حساسیت به تغییرات ورودی: لایه های هم گشتی حساسیت زیادی به تغییرات کوچک در تصاویر دارند. این ممکن است باعث شود که مدل هایی که از این لایه ها استفاده می کنند، به نوعی نویز و اطلاعات غیرضروری در تصاویر حساس شوند.
- کمبود داده: مجموعه های تصویربرداری پزشکی اغلب به دلیل نگرانی های مربوط به حریم خصوصی و مشکل در به دست آوردن داده های برجسته دار محدود می شوند. ماهیت یادگیری سلسله مراتبی لایه های کانولوشن ممکن است منجر به تطبیق بیش از حد در مجموعه داده های کوچکتر شود، به ویژه در مواردی که داده ها برای ثبت تغییرات متنوع شرایط پزشکی کافی نیستند.
- قابلیت تفسیر: در حالی که لایه های کانولوشن در یادگیری بازنمایی عالی هستند، تفسیر و درک اینکه چگونه مدل به تصمیمات خود می رسد می تواند چالش برانگیز باشد. در کاربردهای حیاتی مانند تشخیص پزشکی، تفسیرپذیری تصمیمات مدل برای جلب اعتماد و درک محدودیت های آن بسیار مهم است.

## تأثیر بر کارایی مدل:

مزایای لایه های کانولوشن به طور قابل توجهی به کارایی و دقت مدل های طبقه بندی تصویر کمک می کند، به ویژه در سناریوهایی مانند تصویربرداری پزشکی. آنها استخراج ویژگی های معنی دار را امکان پذیر می سازند، تطبیق بیش از حد را کاهش می دهند و تعمیم قوی به داده های جدید را تسهیل می کنند.

با این حال، چالش هایی مانند افزایش پیچیدگی محاسباتی در تصویربرداری سه بعدی، کمبود داده های برچسب گذاری شده و مسائل تفسیرپذیری می تواند کارایی مدل ها را مختل کند. این چالش ها ممکن است بر مقیاس پذیری، تعمیم و توانایی مدیریت مؤثر داده های محدود مدل تأثیر بگذارد و بر عملکرد آن در برنامه های کاربردی دنیای واقعی تأثیر بگذارد. پرداختن به این چالش ها اغلب شامل تعادل بین پیچیدگی مدل، قابلیت تفسیر و کارایی محاسباتی است که نیازمند توجه دقیق در طراحی و استقرار شبکه های عصبی کانولوشن برای موارد استفاده خاص مانند تجزیه و تحلیل تصویر پزشکی است.

## سوال 6:

الف) فیلترهای  $1 \times 1$  در شبکه های عصبی هم گشتی به عنوان فیلترهای کانولوشن  $1 \times 1$  شناخته می شوند. این فیلترها کاربرد های متفاوتی از جمله قسمت های زیر را دارند:

- 1) کاهش تعداد کانال های ویژگی (feature channels): می توانند بهبودهای مختلفی در عملکرد مدل های عصبی ایجاد کنند.
- 2) کاهش تعداد نقشه های ویژگی (feature maps): اما همچنین ویژگی های مهم را حفظ می کنند
- 3) کاهش تعداد پارامترها: فیلترهای  $1 \times 1$  کمک می کنند تعداد پارامترهای مدل را کاهش یابد. این کاهش میزان پارامترها باعث کاهش زمان آموزش و حافظه مصرفی مدل می شود.

4) کاهش ابعاد: استفاده از فیلترهای  $1 \times 1$  می‌تواند به کاهش ابعاد فضای ویژگی کمک کند، بدون اینکه اطلاعات اساسی از دست بروند. این امر می‌تواند به مدل کمک کند تا به سرعت‌تر اطلاعات مهم را استخراج کند.

5) کنترل بُعد فضای ویژگی‌ها: فیلترهای  $1 \times 1$  به ما این امکان را می‌دهند که بُعد فضای ویژگی‌ها را کنترل کنیم. این ممکن است باعث بهبود انعطاف‌پذیری مدل شود و از بُعد اضافی در فضای ویژگی‌ها جلوگیری کند.

6) کاهش مصرف محاسبات: با کاهش تعداد کانال‌های ویژگی، محاسبات لازم برای پردازش تصاویر کاهش می‌یابد. این امر باعث سرعت بیشتر در آموزش و پیش‌بینی مدل می‌شود.

7) ترکیب اطلاعات: یکی از کارهای اصلی فیلتر  $1 \times 1$ ، ترکیب اطلاعات از کانال‌های ویژگی مختلف است. این فیلتر به ازای هر پیکسل، وزن‌های مختلف برای کانال‌ها اعمال کرده و اطلاعات را ترکیب می‌کند. این به معنای حفظ ویژگی‌های مهم و اطلاعات اساسی است.

8) اعمال غیرخطی: فیلتر  $1 \times 1$  با اعمال تابع غیرخطی (مثل ReLU) به مدل امکان غنی‌تری در تبدیل اطلاعات ویژگی فراهم می‌کند. این کار باعث می‌شود ویژگی‌ها به صورت غیرخطی ترکیب شوند و اطلاعات پراهمیت‌تری ارائه دهند.

ب) استفاده از یک فیلتر  $1 \times 1$  در یک شبکه عصبی کانولوشن (CNN) منجر به یک نقشه ویژگی تبدیل شده می‌شود.

اطلاعات رمزگذاری شده در نقشه ویژگی پس از پیچیدگی  $1 \times 1$ :

1) اطلاعات کانال: فیلتر  $1 \times 1$  به عنوان یک تبدیل خطی اعمال شده در کانال در سراسر نقشه ویژگی ورودی عمل می‌کند. در نتیجه، نقشه ویژگی حاصل، ابعاد فضایی ورودی را حفظ می‌کند اما تعداد کانال‌ها را تغییر می‌دهد.

هر کانال در نقشه ویژگی اطلاعات مربوط به ترکیبات خاصی از کانال‌های ورودی را رمزگذاری می‌کند.

نقشه ویژگی تبدیل شده ترکیبی خطی از کانال های ورودی را نشان می دهد، که در آن هر کانال در خروجی تعاملات وزنی کانال های ورودی را ثبت می کند.

(2) کاهش یا گسترش کانال ها: اگر تعداد کانال های خروجی در مقایسه با کانال های ورودی کاهش یابد، نقشه ویژگی تبدیل شده یک نمایش فشرده، خلاصه سازی و ترکیب اطلاعات از کانال های اصلی ارائه می کند. برعکس، اگر تعداد کانال های خروجی افزایش یابد، نقشه ویژگی ممکن است تعاملات پیچیده تری را بین کانال های ورودی ثبت کند که منجر به افزایش ظرفیت بازنمایی می شود.

(3) عملکرد غیرخطی از طریق فعال سازی: هنگامی که با یک تابع فعال سازی همراه می شود (به عنوان مثال، ReLU)، خروجی کانولوشن  $1*1$  غیر خطی می شود و تبدیلات غیرخطی را به تعاملات کانال ارائه می دهد.

این پیچیدگی و قدرت بیانی بیشتری را به اطلاعات کدگذاری شده در نقشه ویژگی ارائه می دهد و به شبکه اجازه می دهد تا روابط پیچیده تری را بین کانال های ورودی ثبت کند.

(4) ویژگی های آموخته شده و نمایش: نقشه ویژگی خروجی بعد از پیچیدگی  $1*1$  حاوی نمایش های آموخته شده است که ترکیبات مهمی از ویژگی های لایه قبلی را برجسته می کند.

این نمایش ها برای لایه های بعدی در شبکه برای استخراج ویژگی های سطح بالاتر و تصمیم گیری حیاتی هستند و به توانایی شبکه برای یادگیری و تعمیم کمک می کنند.

(پ) تصویر ورودی اصلی شامل اطلاعات فضایی مانند پیکسل ها، لبه ها و بافت ها می باشد بطوریکه هر پیکسل در تصویر نشانگر یک مکان خاص در فضا است. نقشه های ویژگی، ویژگی ها و الگوهای انتزاعی را از ورودی را در بر می گیرند. هر عنصر در یک نقشه ویژگی با فعال سازی یک فیلتر در یک مکان فضایی

خاص متناظر است. فیلترهای با ابعاد مختلف الگوهای مختلف فضایی را ثبت می‌کنند. فیلترهای بزرگتر دامنه دریافت (receptive field) گسترده‌تری دارند و می‌توانند global feature بیشتری را ثبت کنند، در حالی که فیلترهای کوچکتر بر روی الگوهای محلی تمرکز می‌کنند.

فیلترهای  $1 \times 1$  به عنوان نقطه‌ای عمل می‌کنند و در سطح پیکسل عمل می‌کنند. آن‌ها به تنهایی الگوهای فضایی مانند لبه‌ها یا بافت‌ها را ثبت نمی‌کنند، اما برای یادگیری ترکیب‌های مرتبط با کانال و تنظیم عمق نقشه ویژگی مفید هستند.

ت) استفاده از فیلترهای  $x11$  که به عنوان پیچیدگی‌های نقطه‌ای نیز شناخته می‌شوند، در معماری‌های مختلف شبکه عصبی کانولوشن (CNN) رایج است، به ویژه در مدل‌هایی که برای مدیریت کارآمد داده‌های با ابعاد بالا و در عین حال کاهش پیچیدگی محاسباتی طراحی شده‌اند. برخی از مدل‌های برجسته که در آنها فیلترهای  $x11$  به طور گسترده استفاده می‌شود عبارتند از:

#### • GoogLeNet:

- هدف: ماژول‌های آغازین در GoogLeNet، فیلترهای  $1 \times 1$  را در کنار فیلترهای بزرگتر (مانند  $3 \times 3$  و  $5 \times 5$ ) برای انجام کارآمد کاهش ابعاد ترکیب می‌کنند.
- استفاده: فیلترهای  $1 \times 1$  در لایه "گلوگاه" برای کاهش عمق (تعداد کانال‌ها) قبل از پیچیدگی‌های بزرگتر بعدی استفاده می‌شود که به کاهش بار محاسباتی و تسهیل جریان اطلاعات کمک می‌کند.

#### • MobileNets:

- هدف: MobileNets برای موبایل و دستگاه‌های لبه با منابع محاسباتی محدود طراحی شده است.
- استفاده: معماری به شدت از پیچیدگی‌های قابل تفکیک در عمق استفاده می‌کند، که از یک فیلتر در عمق  $(1 \times 1)$  و به دنبال آن یک فیلتر نقطه‌ای  $(1 \times 1)$  تشکیل شده‌اند تا به طور موثر پیچیدگی محاسباتی را کاهش دهند و در عین حال عملکرد را حفظ کنند.

## • ResNet:

- هدف: معماری های ResNet از اتصالات پرش برای رسیدگی به مشکل گرادیان ناپدید شده و تسهیل آموزش شبکه های بسیار عمیق استفاده می کنند.
- استفاده: فیلترهای  $1 \times 1$  اغلب در بلوک های گلوگاه برای کاهش ابعاد قبل و بعد از پیچش های بزرگتر در بلوک های باقی مانده استفاده می شوند و کارایی محاسباتی را بهبود می بخشند.

## • SqueezeNet:

- هدف: هدف SqueezeNet اندازه مدل کوچک بدون به خطر انداختن دقت است.
- استفاده: SqueezeNet به طور گسترده از فیلترهای  $1 \times 1$  در سراسر شبکه برای کاهش تعداد پارامترها و بار محاسباتی و در عین حال حفظ قدرت بیانی استفاده می کند.

## • Xception:

- هدف: Xception مفهوم پیچیدگی قابل تفکیک عمیق را به کمال بررسی می کند.
- استفاده: این مدل به طور گسترده از کانولوشن های قابل جداسازی در عمق استفاده می کند، که از فیلترهای  $1 \times 1$  بعد از کانولوشن های عمقی برای کاهش ابعاد استفاده می کند.

## • EfficientNet:

- هدف: EfficientNet بر دستیابی به عملکرد پیشرفته و در عین حال حفظ کارایی تمرکز دارد.
- استفاده: معماری از مقیاس بندی ترکیبی و کانولوشن های قابل تفکیک در عمق با فیلترهای  $1 \times 1$  استفاده می کند تا دقت و کارایی را در مقیاس های مختلف مدل متعادل کند.

ث) بله، سناریوهایی وجود دارد که در آن استفاده از فیلترهای  $1*1$  ممکن است سودمند یا مناسب نباشد. در اینجا چند موقعیت وجود دارد که استفاده از آنها ممکن است محدود یا کمتر سودمند باشد:

- داده های کم بعدی: هنگام برخورد با داده های بسیار کم ابعاد (مثلاً تصاویر یا ویژگی های بسیار کوچک)، استفاده از فیلترهای  $1*1$  ممکن است مزایای قابل توجهی ارائه نکند. سر بار معرفی شده توسط عملیات کانولوشن می تواند از مزایای تعامل کانال یا کاهش ابعاد بیشتر باشد.
- شبکه های کم عمق: در شبکه های کم عمق یا پیچیدگی محدود، معرفی کانولوشن های  $1*1$  ممکن است پیشرفت های قابل توجهی نداشته باشد. شبکه ممکن است لایه های کافی برای جذب و بهره برداری مؤثر از تعاملات کانالی که توسط این فیلترها تسهیل می شوند، نداشته باشد.
- حجم داده ناکافی: هنگامی که مجموعه داده از نظر اندازه محدود است یا فاقد تنوع است، افزودن فیلتر های  $1*1$  ممکن است کمک قابل توجهی به یادگیری تعاملات معنی دار کانال نداشته باشد. داده های ناکافی ممکن است توانایی شبکه در یادگیری روابط پیچیده بین کانال ها را مختل کند.
- سر بار محاسباتی: در سناریوهایی که منابع محاسباتی به شدت محدود می شوند (مثلاً دستگاه های لبه ای با قابلیت های پردازش بسیار محدود)، سر بار معرفی شده توسط کانولوشن های  $1*1$  ممکن است هزینه محاسباتی را توجیه نکند، به خصوص اگر مزایای آن از نظر عملکرد مدل باشد حداقل.
- تداخل در یادگیری: در برخی موارد، استفاده گسترده از فیلتر های  $1*1$  ممکن است با کاهش بیش از حد ابعاد، باعث از بین رفتن اطلاعات حیاتی یا مهار ظرفیت مدل برای گرفتن الگوهای پیچیده در فرآیند یادگیری تداخل کند.
- معماری شبکه خاص: برخی از معماری های تخصصی ممکن است به دلیل طراحی و اهداف خاص از فیلترهای  $1*1$  سود قابل توجهی نبرند. به عنوان مثال، اگر یک معماری به شدت به نوع دیگری از عملیات یا طرح



اتصال متکی باشد، ادغام کانولوشن های  $1*1$  ممکن است به خوبی با عملکرد مورد نظر آن هماهنگ نباشد.

(ج) حال یک مدل هم گشتی ساده طراحی میکنیم:

- کتابخانه مورد نیاز:

#### Libraries

```
[1] 1 import numpy as np
    2 import tensorflow as tf
```

- داده ورودی: بصورت رندوم ایجاد شده است:

#### Input Data

```
[12] 1 input_tensor = np.random.rand(1, 128, 128, 3) # Batch size 1, 128x128x3 input
```

- تعریف مدل:

#### Model

```
[13] 1 model = tf.keras.Sequential([
    2     tf.keras.layers.Conv2D(filters=1, kernel_size=1, strides=1, padding='valid', input_shape=(128, 128, 3))
    3 ])
```

- آموزش مدل:

#### Train

```
[14] 1 output_tensor = model.predict(input_tensor)
```

```
1/1 [=====] - 0s 41ms/step
```

- مقایسه: فیلتر  $1*1$  ابعاد ورودی را کم کرده است:

### Comparing

```
[15] 1 print("Input shape:", input_tensor.shape)
      2 print("Output shape:", output_tensor.shape)
```

```
Input shape: (1, 128, 128, 3)
Output shape: (1, 128, 128, 1)
```

سوال 7: برای طراحی یک شبکه عصبی هم گشتی که شامل یک ماژول inception برای طبقه بندی تصویر مجموعه داده cifar-10 است، این مراحل طی شد:

1) صدا زدن کتابخانه های مورد نیاز:

### Libraries

```
[13] 1 from tensorflow.keras import layers
      2 from tensorflow.keras.models import Model
      3 from tensorflow.keras.datasets import cifar10
      4 from tensorflow.keras.utils import to_categorical
      5 from tensorflow.keras.optimizers import Adam
```

2) لود کردن دیتا از cifar و پیش پردازش کردن آن: برای پیش پردازش داده های ورودی را تقسیم بر 255 کرده و برای داده های خروجی one-hot encoding تشکیل دادیم. در نهایت shape داده های ورودی آموزش و تست چاپ شد.

#### Data

```
1 (train_images, train_labels), (test_images, test_labels) = cifar10.load_data()
2 train_images = train_images.astype('float32') / 255.0
3 test_images = test_images.astype('float32') / 255.0
4
5 # Converting labels to one-hot encoding
6 train_labels = to_categorical(train_labels, 10)
7 test_labels = to_categorical(test_labels, 10)
8
9 print(train_images.shape)
10 print(test_images.shape)
```

```
(50000, 32, 32, 3)
(10000, 32, 32, 3)
```

3) طراحی مدل شخصی inception: ماژول Inception یک جزء محوری در معماری Inception است که برای ثبت ویژگی‌ها در مقیاس‌ها و پیچیدگی‌های چندگانه به طور موثر طراحی شده است. این شامل شاخه‌های کانولوشنال موازی است که با الحاق خروجی‌های آنها همراه می‌شود و شبکه را قادر می‌سازد تا ویژگی‌ها را در اندازه‌های مختلف میدان گیرنده به طور همزمان یاد بگیرد. ساختار ماژول:

- کانولوشن با اندازه فیلتر  $1 \times 1$ : یک لایه کانولوشن با اندازه فیلتر  $1 \times 1$  برای ضبط ویژگی‌ها و در عین حال حفظ اطلاعات مکانی استفاده می‌شود.
- کانولوشن  $1 \times 1$  و به دنبال آن کانولوشن  $3 \times 3$ : از کانولوشن  $1 \times 1$  برای کاهش ابعاد و به دنبال آن کانولوشن  $3 \times 3$  استفاده شده است. این ترکیب کمک می‌کند تا ویژگی‌ها را در یک میدان پذیرای کمی بزرگتر ثبت کنیم.
- کانولوشن  $1 \times 1$  و به دنبال آن کانولوشن  $5 \times 5$ : یک کانولوشن  $1 \times 1$  را برای کاهش ابعاد استفاده کرده و به دنبال آن یک کانولوشن  $5 \times 5$  را برای ثبت ویژگی‌ها در سطح بزرگتری از ورودی اعمال می‌کنیم.
- Max Pooling و سپس کانولوشن  $1 \times 1$

- خروجی(concatenation): خروجی‌های تمام عملیات را در امتداد محور کانال 1- به هم میرسانیم و یک تانسور خروجی حاوی مجموعه‌های متنوعی از ویژگی‌های استخراج شده از طریق مسیرهای کانولوشنی مختلف ایجاد میکنیم.

هدف ماژول inception:

- استخراج ویژگی در مقیاس های چندگانه: ماژول Inception از مسیرهای کانولوشنی موازی با اندازه فیلترها و عملیات ادغام متفاوت استفاده می کند و شبکه را قادر می سازد تا ویژگی ها را در مقیاس ها و پیچیدگی های مختلف در یک لایه ثبت کند.
- جمع اطلاعات کارآمد: با استفاده از چندین مسیر کانولوشن، شبکه اطلاعات را از زمینه های مختلف دریافتی جمع آوری می کند و به آن اجازه می دهد تا ویژگی های ریز دانه و سطح بالا را به طور موثر استخراج کند.
- راندمان پارامتر و کاهش محاسبات: طراحی ماژول تعداد پارامترها را در مقایسه با استفاده از هسته های بزرگتر در کل شبکه کاهش می دهد. این منجر به کارایی محاسباتی و در عین حال حفظ یا بهبود ظرفیت نمایشی می شود.
- آموزش ویژگی سلسله مراتبی: به عنوان بخشی از معماری Inception، ماژول یادگیری سلسله مراتبی ویژگی ها را با استخراج ویژگی ها در سطوح مختلف انتزاع در یک لایه تسهیل می کند و شبکه را قادر می سازد تا الگوهای پیچیده تری را بیاموزد.

در اصل، طراحی ماژول Inception با هدف ایجاد تعادل بین پیچیدگی محاسباتی، کارایی پارامترها و نمایش ویژگی‌ها، شبکه‌های عصبی عمیق را قادر می‌سازد تا نمایش‌های غنی و چند مقیاسی را از داده‌های ورودی بیاموزند.

### The Inception Model

```
[15] 1 def inception_module(inputs, filters):
2     # Convolution with 1x1 filter size
3     conv1x1 = layers.Conv2D(filters=filters[0], kernel_size=1, activation='relu', padding='same')(inputs)
4
5     # 1x1 convolution followed by 3x3 convolution
6     conv3x3_reduce = layers.Conv2D(filters=filters[1], kernel_size=1, activation='relu', padding='same')(inputs)
7     conv3x3 = layers.Conv2D(filters=filters[2], kernel_size=3, activation='relu', padding='same')(conv3x3_reduce)
8
9     # 1x1 convolution followed by 5x5 convolution
10    conv5x5_reduce = layers.Conv2D(filters=filters[3], kernel_size=1, activation='relu', padding='same')(inputs)
11    conv5x5 = layers.Conv2D(filters=filters[4], kernel_size=5, activation='relu', padding='same')(conv5x5_reduce)
12
13    # Max pooling followed by 1x1 convolution
14    maxpool = layers.MaxPooling2D(pool_size=3, strides=1, padding='same')(inputs)
15    pool_conv = layers.Conv2D(filters=filters[5], kernel_size=1, activation='relu', padding='same')(maxpool)
16
17    # Concatenating the outputs of all operations along the channel axis
18    output = layers.concatenate([conv1x1, conv3x3, conv5x5, pool_conv], axis=-1)
19    return output
```

گام در شبکه های عصبی کانولوشنال (CNN) یک ابرپارامتر است که اندازه گامی را که فیلتر کانولوشنال در حین لغزش/حرکت فضایی بر روی حجم ورودی انجام می دهد، تعیین می کند. افزایش مقدار گام، ابعاد فضایی نقشه های ویژگی را کاهش می دهد. گام بزرگتر به این معنی است که فیلتر هنگام حرکت در ورودی، پیکسل های بیشتری را رد می کند که منجر به حجم خروجی کمتری می شود. این منجر به یک میدان پذیرنده بزرگتر برای هر نورون در نقشه ویژگی می شود. میدان گیرنده ناحیه ای در فضای ورودی است که به محاسبه یک نورون خاص در خروجی کمک می کند. یک میدان پذیرنده بزرگتر می تواند برای ثبت الگوهای بزرگتر در ورودی مفید باشد. استفاده از یک گام بزرگتر از 1 شکلی از نمونه برداری پایین است. وضوح فضایی نقشه های ویژگی را کاهش می دهد، که می تواند در شرایطی که کاهش ابعاد فضایی مورد نظر است مفید باشد. تنظیم گام اغلب همراه با تنظیم سایر پارامترها مانند لایه صفر و اندازه فیلتر برای دستیابی به رفتار مطلوب در شبکه انجام می شود.

4) طراحی شبکه با استفاده از ماژول تعریف شده: در تابع `create_inception_model` که قبلاً برای طراحی یک شبکه عصبی مبتنی بر Inception استفاده شد، چندین ویژگی و عملیات کلیدی برای

تسهیل استخراج مؤثر ویژگی و جریان اطلاعات در شبکه به کار گرفته شده است. در اینجا به تفکیک اجزای مهم استفاده شده در آن تابع اشاره شده است:

- لایه های کانولوشنال Conv2D: لایه های کانولوشن برای استخراج ویژگی های اولیه استفاده می شود. لایه های «Conv2D» از فیلترهای قابل یادگیری برای جمع شدن در میان داده های ورودی استفاده می کنند و ویژگی های خاصی را ثبت می کنند.
- لایه های ادغام MaxPooling2D: حداکثر لایه های ادغام، نقشه های ویژگی را نمونه برداری می کنند، ویژگی های غالب را در مناطق خاص ثبت می کنند و ابعاد فضایی را کاهش می دهند.
- ماژول های inception: تابع 'inception\_module' برای ایجاد ماژول های Inception استفاده می شود. این ماژول ها شاخه های کانولوشنال موازی را برای ثبت ویژگی ها در مقیاس های چندگانه به طور همزمان ترکیب می کنند.
- نرمال سازی دسته ای Normalization Batch: لایه های نرمال سازی دسته ای، فعال سازی ها را عادی می کنند، یادگیری را تثبیت می کنند و توانایی شبکه برای استخراج ویژگی های مفید را افزایش می دهند.
- جمع آوری میانگین جهانی GlobalAveragePooling2D: ادغام میانگین جهانی با میانگین گیری نقشه های ویژگی در ابعاد فضایی، ابعاد فضایی را به یک بردار کاهش می دهد. این داده ها را برای طبقه بندی آماده می کند.
- لایه متراکم: یک لایه متراکم کاملاً متصل برای طبقه بندی استفاده می شود. طبقه بندی نهایی را بر اساس ویژگی های استخراج شده انجام می دهد.

اهمیت این مولفه ها:

- سلسله مراتب ویژگی ها: ترکیب لایه های کانولوشن، مازول های آغازین، و عملیات ادغام به گرفتن ویژگی های سلسله مراتبی در سطوح مختلف انتزاع کمک می کند.
- جریان اطلاعات: عادی سازی دسته ای و رد شدن از اتصالات (در صورت استفاده) به جریان روان تر اطلاعات در حین آموزش کمک می کند، به همگرایی کمک می کند و از بیش از حد مناسب جلوگیری می کند.
- کاهش فضایی: لایه های ادغام ابعاد فضایی نقشه های ویژگی را کاهش می دهند و روی اطلاعات مهم تمرکز می کنند و در عین حال پیچیدگی محاسباتی را کاهش می دهند.
- غیر خطی و طبقه بندی: توابع فعال سازی (ReLU) غیرخطی ها را معرفی می کنند و مدل را قادر به یادگیری روابط پیچیده می کنند، در حالی که لایه متراکم طبقه بندی نهایی را بر اساس ویژگی های استخراج شده انجام می دهد.

#### Architecture of the Network

```
[16] 1 def create_inception_model(input_shape, num_classes):
2     inputs = layers.Input(shape=input_shape)
3
4     # Convolutional layers for feature extraction
5     conv1 = layers.Conv2D(64, 7, activation='relu', padding='same', strides=2)(inputs)
6     pool1 = layers.MaxPooling2D(pool_size=3, strides=2, padding='same')(conv1)
7
8     # Additional Inception modules
9     inception1 = inception_module(pool1, filters=[64, 96, 128, 16, 32, 32])
10    inception2 = inception_module(inception1, filters=[128, 128, 192, 32, 96, 64])
11
12    # Batch normalization added after each Inception module
13    inception2 = layers.BatchNormalization()(inception2)
14
15    # Global average pooling and dense layers for classification
16    gap = layers.GlobalAveragePooling2D()(inception2)
17    outputs = layers.Dense(num_classes, activation='softmax')(gap)
18
19    # Create model
20    model = Model(inputs=inputs, outputs=outputs)
21    return model
```

## 5) ساخت و کامپایل مدل:

### Create and Compile the model

```
[17] 1 input_shape = (32, 32, 3)
      2 num_classes = 10 # Number of classes in CIFAR-10 dataset
      3
      4 # Create the model
      5 model = create_inception_model(input_shape, num_classes)
      6
      7 # Compile the model
      8 model.compile(optimizer='adam',
      9               loss='categorical_crossentropy',
     10               metrics=['accuracy'])
```

6) آموزش مدل: همان طور که میبینیم بعد حدود 10 اپیاک دقت مدل روی داده های آموزشی به بالای 80 درصد میرسد و در آخر 20 اپیاکف مدل روی داده های آموزشی دقتی حدود 95 درصد و روی داده های validation دقتی حدود 73 دارد.

### Training

```
[18] 1 history = model.fit(train_images, train_labels, batch_size=64, epochs=20, validation_data=(test_images, test_labels))
      2
      3 train_accuracy = history.history['accuracy'][-1]
      4 val_accuracy = history.history['val_accuracy'][-1]
      5
      6 print(f"Last training accuracy: {train_accuracy:.4f}")
      7 print(f"Last validation accuracy: {val_accuracy:.4f}")

Epoch 1/20
782/782 [=====] - 14s 12ms/step - loss: 1.4046 - accuracy: 0.4940 - val_loss: 1.3166 - val_accuracy: 0.5409
Epoch 2/20
782/782 [=====] - 8s 11ms/step - loss: 1.0563 - accuracy: 0.6267 - val_loss: 1.3374 - val_accuracy: 0.5519
Epoch 3/20
782/782 [=====] - 8s 10ms/step - loss: 0.9080 - accuracy: 0.6805 - val_loss: 1.2676 - val_accuracy: 0.5949
Epoch 4/20
782/782 [=====] - 9s 11ms/step - loss: 0.8077 - accuracy: 0.7171 - val_loss: 0.9722 - val_accuracy: 0.6644
```

```
Last training accuracy: 0.9509
Last validation accuracy: 0.7328
```

پایان