

# به نام خدا

تمرین سری چهارم

درس یادگیری عمیق

سینا علی نژاد ۹۹۵۲۱۴۶۹

الف) طراحی شبکه همگشتی با استفاده از KerasTuner:

- KerasTuner یک ابزار قدرتمند در TensorFlow است که به تنظیم هایپرپارامتر برای شبکه های عصبی، به ویژه برای مدل های Keras کمک می کند. این ابزار فرآیند جستجوی بهترین ترکیب از هایپرپارامترها را خودکار می کند و در زمان و تلاش شما صرفه جویی می کند. KerasTuner به جای آزمایش دستی ترکیب های مختلف، از الگوریتم های جستجوی مختلف استفاده می کند تا به طور مؤثر پیکربندی را پیدا کند که بهترین عملکرد را در داده های خاص شما دارد.
- جواب Bard:

KerasTuner can significantly optimize convolutional neural networks (CNNs) for image classification tasks. Here's how it works:

- Define the Model Architecture: You define the basic structure of your CNN using Keras layers like Conv2D, MaxPooling2D, Flatten, and Dense.
- Hyperparameter Search Space: You specify the range of values for hyperparameters like number of filters, kernel size, activation functions, optimizers, and learning rate.
- Search Algorithm and Tuner: You choose a search algorithm like RandomSearch or Bayesian Optimization and a tuner like Hyperband or RandomTuner.
- Training and Evaluation: KerasTuner trains multiple models with different hyperparameter combinations on your data and evaluates them based on a chosen metric (accuracy, F1 score, etc.).
- Best Model Selection: The tuner selects the model with the best performance metric and provides you with the optimal hyperparameter values.

که به طور خلاصه می گوید:

- ابتدا مدل را در Keras تعریف میکنیم.
- سپس برای هر هایپرپارامتر که میخواهیم بهترین مقدار آن پیدا شود، محدوده جستجو را مشخص میکنیم.
- سپس الگوریتم جستجو و همچنین یک تنظیم کننده (Tuner) را مشخص میکنیم.
- مدل را با استفاده از KerasTuner آموزش می دهیم و ارزیابی می کنیم.
- در نهایت، بهترین مدل از لحاظ performance توسط tuner انتخاب می شود.

- KerasTuner تونرهای مختلفی را ارائه می دهد که هر کدام نقاط قوت و ضعف خود را دارند. در اینجا چند انتخاب وجود دارد:

- Hyperband: به طور موثر فضای جستجوی بزرگ را با توقف زودهنگام (early stopping) کاوش می کند که برای مجموعه داده های بزرگ مناسب است.
- RandomSearch: ساده و سریع، برای مجموعه داده های کوچک یا کاوش اولیه مناسب است.
- Bayesian Optimization: قدرتمند اما از نظر محاسباتی گران است.

انتخاب من برای Tuner برای دیتاست MNIST، همان Hyperband است به دلایل زیر:

- Balanced Exploration: این دیتاست نسبت به دیتاست های دیگری مثل ImageNet کوچک بشمار می آید. این تونر، به طور موثری فضای جستجو را کاوش می کند بدون اینکه منابعی را بر روی پیکربندی هایی که امیدی به آنها نیست، هدر بدهد.
- Early stopping: آموزش مدلهایی که پیشرفت آنها نامحتمل است را متوقف می کند که باعث صرفه جویی در زمان و منابع می شود که این موضوع در دیتاست های کوچک مثل MNIST بسیار موثر است.
- Good performance: نشان داده شده است که Hyperband روی دیتاست MNIST خوب عمل می کند و مدلهایی با دقت بالا و قدرت تعمیم دهی بالا می دهد.

جواب Bard:

Pros of Hyperband for MNIST:

- Balanced exploration: MNIST is a relatively small dataset compared to others like ImageNet. Hyperband efficiently explores the search space without wasting resources on unpromising configurations.
- Early stopping: It stops training models that are unlikely to improve, saving you time and computational resources. This is particularly valuable for smaller datasets like MNIST.
- Good performance: Hyperband has been shown to perform well on MNIST, often finding high-performing models with good accuracy and generalization.

ب) استفاده از KerasTuner بر روی دیتاست MNIST:

- مجموعه داده MNIST یک معیار (benchmark) کلاسیک برای طبقه بندی تصاویر است که از ۷۰۰۰۰ تصویر خاکستری از ارقام دست نویس (۰-۹) با برچسب های مربوطه تشکیل شده است. هر تصویر ۲۸\*۲۸ پیکسل است که آن را به مجموعه ای نسبتاً کوچک و قابل مدیریت تبدیل می کند. MNIST به طور گسترده ای برای ارزیابی و مقایسه الگوریتم های مختلف یادگیری ماشین، از جمله شبکه های عصبی کانولوشنی (CNN) استفاده می شود.
  - KerasTuner یک چارچوب تنظیم فرایارامتر است که یافتن پیکربندی بهینه برای CNN شما را ساده می کند. در اینجا نحوه استفاده از آن با MNIST آمده است:
    - تعریف مدل: یک تابع ساخت مدل را تعریف کنید که فرایارامترها (به عنوان مثال، تعداد لایه ها، فیلترها در هر لایه، توابع فعال سازی) را به عنوان ورودی می گیرد و بر اساس آن یک CNN می سازد.
    - فضای جستجو: محدوده مقادیری را که می خواهید برای هر هایپرپارامتر بررسی کنید، مشخص کنید.
    - تنظیم ابرپارامترها: از بهینه سازی بیزی یا الگوریتم های دیگر برای جستجوی خودکار فضای تعریف شده و یافتن ترکیبی که منجر به بهترین عملکرد در مجموعه اعتبار سنجی می شود استفاده می کند.
    - آموزش و ارزشیابی: پس از شناسایی فرایارامترهای بهینه، مدل نهایی بر روی کل مجموعه آموزشی آموزش داده می شود و بر روی مجموعه آزمون ارزیابی می شود.
  - اهمیت لایه های Dropout و Pooling:
    - Dropout: این تکنیک به طور تصادفی نورون ها را در طول تمرین حذف می کند، از تطبیق بیش از حد جلوگیری می کند و شبکه را تشویق می کند تا ویژگی های قوی تری را بیاموزد. همچنین به عنوان یک تنظیم کننده عمل می کند و پیچیدگی کلی مدل را کاهش می دهد.
    - Pooling: این عملیات ابعاد فضایی نقشه های ویژگی را کاهش می دهد و به شبکه امکان می دهد ویژگی ها را در مقیاس های مختلف یاد بگیرد و روی تکه های تصویر کوچکتر پیش بینی کند. همچنین تعداد پارامترها را کاهش می دهد و کارایی محاسباتی را بهبود می بخشد.
- تاثیرات این لایه ها در عملکرد مدل:

کاهش Overfitting: Dropout به شبکه کمک می‌کند تا داده‌های دیده نشده را بهتر تعمیم دهد، که منجر به دقت بالاتر در مجموعه آزمایشی می‌شود.

یادگیری ویژگی‌های بهبود یافته: ادغام به شبکه اجازه می‌دهد تا ویژگی‌های عمومی و قوی‌تری را به تصویر بکشد که منجر به عملکرد بهتر در اشکال و تغییرات مختلف رقم می‌شود.

کارایی محاسباتی: ادغام تعداد پارامترها و عملیات را کاهش می‌دهد و باعث می‌شود مدل سریع‌تر آموزش و اجرا شود.

ج) پیاده‌سازی شبکه مورد نظر:

- پیاده‌سازی:

تابع build\_model:

```
def build_model(hp): # Defines a function that takes a HyperModel as input and returns a built Keras model.
    """
    This function builds a CNN architecture based on the hyperparameters provided by the HyperModel.
    """
    # Define input layer
    inputs = keras.Input(shape=(28, 28, 1)) # Creates an input layer for 28x28 grayscale images.

    # Convolutional layer block
    """
    This loop iterates a specified number of times (defined by the "conv_layers" hyperparameter)
    and builds a series of convolutional layers with various configurations.
    """
    x = inputs
    for _ in range(hp.Int("conv_layers", min_value=2, max_value=5)):
        x = keras.layers.Conv2D( # Applies a 2D Convolutional layer with:
            hp.Int("filters", min_value=32, max_value=256), # Number of filters (feature maps) specified by the "filters" hyperparameter.
            kernel_size=3, # Kernel size of 3x3 for extracting local features.
            activation="relu", # ReLU activation for non-linearity.
        )(x) # Applies the layer to the previous layer output (x).

        x = keras.layers.MaxPooling2D(pool_size=(2, 2))(x) # Applies Max Pooling with a 2x2 window for feature reduction.
        x = keras.layers.Dropout(hp.Float("dropout", min_value=0.1, max_value=0.5))(x) # Introduces dropout with a probability specified by the "dropout" hyperparameter for preventing overfitting.

    # Flatten and add dense layers
    x = keras.layers.Flatten()(x) # Flattens the output of the convolutional layers into a 1D vector.
    """
    This loop iterates a specified number of times (defined by the "dense_layers" hyperparameter)
    and builds a series of fully-connected layers with various configurations.
    """
    for _ in range(hp.Int("dense_layers", min_value=2, max_value=5)):
        x = keras.layers.Dense( # Applies a Dense layer with:
            hp.Int("neurons", min_value=32, max_value=256), # Number of neurons specified by the "neurons" hyperparameter.
            activation="relu", # ReLU activation for non-linearity.
        )(x)
        x = keras.layers.Dropout(hp.Float("dropout", min_value=0.1, max_value=0.5))(x) # Introduces dropout with a probability specified by the "dropout" hyperparameter for preventing overfitting.

    # Output layer
    outputs = keras.layers.Dense(10, activation="softmax")(x) # Creates a final Dense layer with 10 neurons (one for each digit) and softmax activation for classification.
    model = keras.Model(inputs=inputs, outputs=outputs) # Creates a Keras model with the defined input and output layers.
    model.compile( # Configures the model for training:
        optimizer=keras.optimizers.Adam(learning_rate=hp.Float("learning_rate", min_value=0.001, max_value=0.1)), # Uses Adam optimizer with a learning rate specified by the "learning_rate" hyperparameter.
        loss="sparse_categorical_crossentropy", # Defines the loss function for classification.
        metrics=["accuracy"], # Specifies "accuracy" as the metric to track during training.
    )
    return model # Returns the built and configured model.
```

این تابع مدل ما را بر اساس هایپرپارامترهایی که به آن می‌دهیم و این کار را با استفاده از keras tuner انجام می‌دهیم و فضای جستجو را با پارامترهای min value و max value برای پارامترهایی که می‌خواهیم مشخص می‌کنیم.

```
for _ in range(hp.Int("conv_layers", min_value=2, max_value=5)):
    x = keras.layers.Conv2D( # Applies a 2D Convolutional layer with:
        hp.Int("filters", min_value=32, max_value=256), # Number of filters (feature maps)
        kernel_size=3, # Kernel size of 3x3 for extracting local features.
        activation="relu", # ReLU activation for non-linearity.
    )(x) # Applies the layer to the previous layer output (x).

    x = keras.layers.MaxPooling2D(pool_size=(2, 2))(x) # Applies Max Pooling with a 2x2 wi
    x = keras.layers.Dropout(hp.Float("dropout", min_value=0.1, max_value=0.5))(x) # Intro
```

برای مثال در اینجا تعداد لایه های کانولوشنی توسط عددی که به تابع range داده میشود، مشخص میشود و ما این عدد را یک عدد صحیح بین ۲ تا ۵ قرار میدهم و این کار tuner خواهد بود که مقدار بهینه را پیدا کند. همچنین بعد از هر لایه کانولوشنی، یک لایه max pooling و بعد از آن لایه dropout داریم که مقدار keep probability را به tuner سپردیم، عددی بین 0.1 و 0.5.

```
for _ in range(hp.Int("dense_layers", min_value=2, max_value=5)):
    x = keras.layers.Dense( # Applies a Dense layer with:
        hp.Int("neurons", min_value=32, max_value=256), # Number of neuron
        activation="relu", # ReLU activation for non-linearity.
    )(x)
    x = keras.layers.Dropout(hp.Float("dropout", min_value=0.1, max_value=0.5))(x)

# Output layer
outputs = keras.layers.Dense(10, activation="softmax")(x) # Creates a final layer with 10 units
model = keras.Model(inputs=inputs, outputs=outputs) # Creates a Keras model
model.compile( # Configures the model for training:
    optimizer=keras.optimizers.Adam(learning_rate=hp.Float("learning_rate", min_value=1e-5, max_value=1e-1)),
    loss="sparse_categorical_crossentropy", # Defines the loss function
    metrics=["accuracy"], # Specifies "accuracy" as the metric to track
)
return model # Returns the built and configured model.
```

بعد از آن نوبت لایه های کاملاً متصل است. تعداد این لایه ها نیز بین ۲ تا ۵ تعیین میشود و در آخر یک لایه خطی نهایی برای دسته بندی نهایی میگذاریم.

```
# Define hypermodel for tuning
tuner = kt.Hyperband(build_model,
                     objective='val_accuracy',
                     max_epochs=10,
                     factor=3)

# Train and search for best hyperparameters
tuner.search(x_train, y_train, epochs=10, validation_split=0.2)

# Get best model and evaluate
best_model = tuner.get_best_models()[0]
```

در اینجا نوع tuner را مشخص کردیم. دلیل استفاده از hyperband را در قسمت های قبلی گفتیم. پارامتر اول همان تابعی است که مدل را میسازد و پارامتر دوم، معیار tuner برای یافتن بهترین مقادیر ابرپارامترها است، پارامتر سوم تعیین میکند که ماکزیمم تا ۱۰ epoch برای هر پیکربندی پیش برود و بهترین ها را انتخاب کند.

در نهایت جستجو توسط تابع search انجام میشود و بهترین مدل را توسط تابع get\_best\_models میگیریم.

```
Trial 22 Complete [00h 00m 00s]

Best val_accuracy So Far: 0.9460833072662354
Total elapsed time: 00h 08m 34s
```

ارزیابی بر روی داده تست:

```
best_model.evaluate(x_test, y_test)

313/313 [=====] - 2s 4ms/step - loss: 0.2209 - accuracy: 0.9466
[0.22088539600372314, 0.9466000199317932]
```

برای هر خط در کد کامنت گذاری دقیقتری شده است.

- اندازه فیلترها در لایه های کانولوشنی یک ابرپارامتر مهم است که می تواند بر عملکرد یک شبکه عصبی کانولوشن (CNN) تأثیر بگذارد. انتخاب اندازه فیلتر به مشکل موجود و ویژگی های داده های ورودی بستگی دارد.

محبوب ترین انتخاب برای اندازه فیلتر  $3 \times 3$  است. این به این دلیل است که اندازه فیلترهای کوچکتر هزینه های محاسباتی و اشتراک وزن را کاهش می دهد که در نهایت منجر به وزن کمتر برای back propagation می شود. اگر مقدار زیادی پیکسل برای شبکه برای تشخیص شی مورد نیاز باشد، ممکن است از اندازه فیلترهای بزرگتر، مانند  $11 \times 11$  یا  $9 \times 9$  استفاده شود. اگر قرار است تفاوت های موضعی بیشتر مورد توجه قرار گیرند، اندازه های فیلتر کوچک تر، مانند  $3 \times 3$  یا  $5 \times 5$  ترجیح داده می شوند.

به طور کلی، هیچ قانون سختی برای انتخاب اندازه فیلترها در لایه های کانولوشن وجود ندارد. بهتر است با اندازه های مختلف فیلتر آزمایش کنیم و ببینیم چه چیزی برای مشکل خاص ما بهتر است.

برای مثال در دیتاست MNIST تعداد کل پیکسل های تصویر  $28 \times 28$  است بنابراین استفاده از فیلتر با سایز بزرگ، ناگهان ابعاد تصویر را کاهش میدهد و نمیتواند خوب عمل کند برای همین از سایز  $3 \times 3$  استفاده کردم.

- ادغام و dropout دو تکنیکی هستند که می توانند برای جلوگیری از بیش برازش (overfitting) و افزایش دقت در شبکه های عصبی کانولوشن (CNN) استفاده شوند.

ادغام تکنیکی است که اندازه مکانی نقشه های ویژگی را با نمونه برداری از آنها کاهش می دهد. این می تواند به کاهش تعداد پارامترها در شبکه و جلوگیری از بیش برازش کمک کند. انواع مختلفی از لایه های ادغام وجود دارد، مانند ادغام ماکزیمم و ادغام متوسط، که می توانند در یک CNN استفاده شوند.

Dropout یک تکنیک منظم سازی است که به طور تصادفی برخی از نورون های شبکه را در طول آموزش حذف می کند. این می تواند با کاهش co-adaption نورون ها و وادار کردن شبکه به یادگیری ویژگی های قوی تر، به جلوگیری از بیش برازش کمک کند. Dropout را می توان با استفاده از یک لایه اضافی بر روی لایه های کانولوشن در یک CNN اعمال کرد، که کل برخی نقشه های ویژگی را از لایه کانولوشن حذف می کند.



به طور کلی، ادغام و dropout می توانند با هم برای بهبود عملکرد CNN استفاده شوند. ادغام می تواند به کاهش اندازه فضایی نقشه های ویژگی کمک کند، در حالی که dropout می تواند با کاهش co-adaption نوروها به جلوگیری از overfitting کمک کند. با این حال، توجه به این نکته مهم است که ترکیب بهینه ادغام و dropout به مشکل خاص در دست و ویژگی های داده های ورودی بستگی دارد.

سوال ۳-

طبق این [لینک](#)، دیتای ورودی که یک لایه LSTM از ما انتظار دارد، به فرم زیر است:

[samples, time steps, features]

بعد اول تعداد نمونه ها و بعد دوم تعداد استیت های قبلی برای پیش بینی استیت فعلی و بعد سوم تعداد فیچرهای هر کدام از استیت ها است که در مسئله پیش بینی گاز مصرفی، تعداد آن ۶ است. در این مسئله تعداد نمونه های آموزشی ۵۱۴۲ است و ابعاد دیتاست آموزشی به صورت زیر است:

```
X_train.shape  
(5142, 6)
```

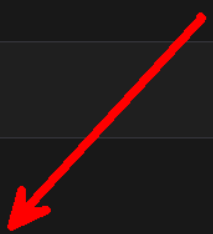
در واقع ما باید یک تابعی بنویسیم که این دیتاست را بگیرد و یک دیتاست مناسب شبکه بازگشتی ایجاد کند، ویژگی این دیتاست، وجود پنجره زمانی است، یعنی ما مصرف گاز در چند ساعت گذشته را داشته باشیم و با استفاده از آنها و ویژگی های زمان حاضر، پیش بینی جدید را انجام دهیم. کد این تابع به صورت زیر خواهد بود:

```
def create_proper_dataset_for_rnn(inputs, labels, look_back):  
    new_inputs = []  
    new_labels = []  
    for i in range(len(inputs) - look_back):  
        x = inputs[i:i+look_back, :]  
        y = labels[look_back]  
        new_inputs.append(x)  
        new_labels.append(y)  
    return np.array(new_inputs), np.array(new_labels)  
look_back = 12  
prepared_X_train, prepared_Y_train = create_proper_dataset_for_rnn(X_train, y_train, look_back)  
prepared_X_test, prepared_Y_test = create_proper_dataset_for_rnn(X_test, y_test, look_back)
```

که پارامتر look\_back اشاره به تعداد بازه زمانی قبلی دارد که برای پیش بینی مقدار زمان حاضر استفاده میشود. پس از اعمال این تابع، ابعاد دیتاست به صورت روبرو میشود: [samples, look\_back, num of features] که تعداد فیچرها در اینجا عدد ۶ است و بهترین مقدار look\_back باید با آزمون و خطا بدست


آید. برای مثال در اینجا مقدار آن را برابر با ۱۲ گذاشتم که یعنی مقادیر ۱۲ ساعت اخیر را در محاسبه خود دخالت دهد.

```
X_train.shape  
(5142, 6)  
  
# Create and compile the LSTM model  
model = Sequential()  
# First LSTM layer with Dropout regularisation  
model.add(LSTM(units=50, return_sequences=True, input_shape=(X_train.shape[1], 1)))  
# Second LSTM layer  
model.add(LSTM(units=50, return_sequences=True))  
# Third LSTM layer  
model.add(LSTM(units=50, return_sequences=True))  
# The model is ready for training
```



همچنین قسمت مشخص شده در بالا باید به شکل زیر تغییر کند:

```
# Create and compile the LSTM model  
model = Sequential()  
# First LSTM layer with Dropout regularisation  
model.add(LSTM(units=50, return_sequences=True, input_shape=(look_back, 6)))  
# Second LSTM layer  
model.add(LSTM(units=50, return_sequences=True))  
# Third LSTM layer  
model.add(LSTM(units=50, return_sequences=True))  
# The model is ready for training
```



که عدد ۶ همان تعداد ویژگی هاست.

همچنین best practice این است که دیتای ورودی به LSTM نرمالایز باشد که با تغییر زیر در کد امکان پذیر است:

```
scaler = MinMaxScaler(feature_range=(0,1))  
dataset = scaler.fit_transform(X_train)
```

سوال ۴-

الف) شبکه های عصبی کانولوشن (CNN) و شبکه های عصبی بازگشتی (RNN) دو نوع شبکه عصبی هستند که معمولاً در یادگیری عمیق استفاده می شوند. CNN ها برای پردازش داده ها با توپولوژی شبکه مانند

(grid)، مانند تصاویر، طراحی شده اند، در حالی که RNN ها برای پردازش داده های متوالی، مانند متن یا گفتار طراحی شده اند.

CNN ها برای طبقه بندی تصویر، تشخیص اشیا و image segmentation به خوبی کار می کنند زیرا می توانند الگوهای موجود در ساختار فضایی داده های ورودی را به صورت محلی تشخیص دهند. آنها همچنین از نظر محاسباتی کارآمد هستند زیرا از اشتراک پارامتر برای کاهش تعداد پارامترها در مدل استفاده می کنند.

RNN ها برای وظایف پردازش زبان طبیعی، مانند مدل سازی زبان، ترجمه ماشینی، و تشخیص گفتار به خوبی کار می کنند، زیرا می توانند مدل سازی وابستگی های زمانی در داده های ورودی را بیاموزند. آنها همچنین قادر به مدیریت توالی های ورودی با طول متغیر هستند که باعث انعطاف پذیری بیشتری نسبت به CNN ها می شود.

ب) CNN و RNN از نظر تعداد پارامترها و امکان موازی سازی با هم تفاوت دارند. CNN ها معمولاً پارامترهای کمتری نسبت به RNN دارند زیرا از اشتراک پارامتر برای کاهش تعداد پارامترها در مدل استفاده می کنند. این باعث می شود که CNN ها از نظر محاسباتی کارآمدتر و آموزش آسان تر از RNN ها باشند.

از سوی دیگر، موازی سازی RNN ها نسبت به CNN ها دشوارتر است، زیرا آنها به خروجی مرحله زمانی قبلی به عنوان ورودی به مرحله زمانی فعلی نیاز دارند. این امر پردازش چندین مرحله زمانی را به صورت موازی دشوار می کند. با این حال، پیشرفت های اخیر در سخت افزار و نرم افزار، موازی سازی RNN ها را تا حدودی ممکن کرده است.

به طور خلاصه، CNN ها برای وظایفی که شامل ساختار فضایی هستند، مناسب تر هستند، در حالی که RNN ها برای وظایفی که شامل وابستگی های زمانی هستند، مناسب تر هستند. CNN ها از نظر محاسباتی کارآمدتر هستند و آموزش آنها آسان تر از RNN ها است، اما RNN ها انعطاف پذیرتر هستند و می توانند توالی های ورودی با طول متغیر را مدیریت کنند.

پرامپت:

a) Give a brief definition of convolutional networks and recursive networks and say for what applications and problems each one works better and why. b) Compare these networks with each other regarding the number of parameters and the possibility of parallelization.

جواب :bing

a) Convolutional neural networks (CNNs) and recursive neural networks (RNNs) are two types of neural networks that are commonly used in deep learning. CNNs are designed to process data with a grid-like topology, such as images, while RNNs are designed to process sequential data, such as text or speech.

CNNs work well for image classification, object detection, and image segmentation tasks because they can learn to recognize patterns in the spatial structure of the input data. They are also computationally efficient because they use parameter sharing to reduce the number of parameters in the model.

RNNs work well for natural language processing tasks, such as language modeling, machine translation, and speech recognition, because they can learn to model the temporal dependencies in the input data. They are also capable of handling variable-length input sequences, which makes them more flexible than CNNs.

b) CNNs and RNNs differ in terms of the number of parameters and the possibility of parallelization. CNNs typically have fewer parameters than RNNs because they use parameter sharing to reduce the number of parameters in the model. This makes CNNs more computationally efficient and easier to train than RNNs.

On the other hand, RNNs are more difficult to parallelize than CNNs because they require the output of the previous time step as input to the current time step. This makes it difficult to process multiple time steps in parallel. However, recent

advances in hardware and software have made it possible to parallelize RNNs to some extent.

In summary, CNNs are better suited for tasks that involve spatial structure, while RNNs are better suited for tasks that involve temporal dependencies. CNNs are more computationally efficient and easier to train than RNNs, but RNNs are more flexible and can handle variable-length input sequences.

سوال ۵-

الف) فرمولی که همه پارامترهای stride, padding, dilation rate را در محاسبه ابعاد خروجی در نظر بگیرد، به شکل زیر است:

$$W_{out} = \left\lfloor \frac{(W_{in} + 2P - D(W_k - 1) - 1)}{S} \right\rfloor + 1$$

که P همان مقدار padding است که از هر طرف چقدر به آن اضافه میشود. D مقدار dilate کردن و Wk عرض کرنل است و S مقدار stride است.  
خروجی لایه اول:

256\*256\*64 (because of padding=same, input spatial dimensions will stay unchanged)

تعداد پارامتر لایه اول:

$$64 * (3 * 3 * 3 + 1) = 1792$$

خروجی لایه دوم:

We use the formula provided earlier

Win = 256, S=2, D=2, P=0(padding=valid), Wk = 5

Wout = 124 and Hout = Wout = 124

Output shape = 124\*124\*32

تعداد پارامتر لایه دوم:

$$32*(5*5*64 + 1) = 51232$$

خروجی لایه سوم:

$$62*62*32$$

تعداد پارامتر لایه سوم: صفر

خروجی لایه چهارم:

$$62*62*128 \text{ (spatial dimensions unchanged due to padding=same)}$$

تعداد پارامتر لایه چهارم:

$$128*(3*3*32 + 1) = 36992$$

خروجی لایه پنجم:

$$S=2, D=4, W_k=5, W_{in}=62, P=0$$

$$W_{out} = 23, H_{out} = W_{out} = 23$$

$$\text{Output shape: } 23*23*64$$

تعداد پارامتر لایه پنجم:

$$64*(5*5*128 + 1) = 204864$$

خروجی لایه ششم:

$$11*11*64$$

تعداد پارامتر لایه ششم: صفر

خروجی لایه هفتم:

$$11*11*256 \text{ (because of padding=same)}$$

تعداد پارامتر لایه هفتم:

$$256*(3*3*64 + 1) = 147712$$

خروجی لایه هشتم:

S=2, D=8, P=0, Wk=5, Win=11

Wout = -10

Output shape: -10\*-10\*128

تعداد پارامتر لایه هشتم:

$$128*(5*5*256 + 1) = 819328$$

خروجی لایه نهم: خروجی قبل منفی شد و معنا ندارد. وگرنه دو بعد اول نصف و بعد سوم ثابت میماند.

تعداد پارامتر لایه نهم: صفر

**Layer1: Conv(64, (3,3), stride=1, padding='same')**

**Layer2: Dilated-Conv(32, (5,5), stride=2, dilation rate=2, padding='valid')**

**Layer3: Max-pool (size=(2,2), stride=2)**

**Layer4: Conv(128, (3,3), stride=1, padding='same')**

**Layer5: Dilated-Conv(64, (5,5), stride=2, dilation rate=4, padding='valid')**

**Layer6: Max-pool (size=(2,2), stride=2)**

**Layer7: Conv(256, (3,3), stride=1, padding='same')**

**Layer8: Dilated-Conv(128, (5,5), stride=2, dilation rate=8, padding='valid')**

**Layer9: Max-pool (size=(2,2), stride=2)**

(ب) فرمول P از رابطه زیر محاسبه میشود:

$$P = \left\lfloor \frac{f}{2} \right\rfloor$$

که مقدار محاسبه شده مقداری از پدینگ است که باید به هر طرف (بالا، پایین، چپ، راست) اضافه شود.

اگر مقدار f عددی زوج بود، در آن صورت، جمع پدینگ های چپ و راست باید برابر با f-1 شود، همینطور برای بالا و پایین.

## سوال ۶-

(الف)

- غلط است. پردازش را سریع نمیکند زیرا یک سری پردازش بیشتر اضافه میکند (البته در زمان تست، این پردازش اضافه میتواند در لایه قبل از BN ادغام شود ولی در زمان آموزش قطعاً پردازش اضافه‌ای مثل محاسبه میانگین و واریانس دسته برای هر ویژگی را داریم و باید `moving avg` و `moving var` را نیز برای زمان تست محاسبه کنیم، همچنین یک لایه خطی اضافه تر با پارامترهای گاما و بتا داریم پس تعداد به روز رسانی‌ها از این جهت در `back propagation` افزایش پیدا میکند و ثابت نمی‌ماند. البته نرمال سازی دسته ای باعث میشود همگرایی بهتر و سریعتر صورت بگیرد ولی زمان هر `forward_pass` را در زمان آموزش افزایش میدهد.
- درست است.
- غلط است. نرمال سازی دسته ای چنین کاری انجام نمیدهد.

(ب) کد زده شد.

(ج) وجود مقدار اپسیلون بخاطر کدزنی است، ممکن است یک ویژگی برای تمام ورودی های آن دسته مقدار یکسان داشته باشد و واریانس آن صفر شود، در این صورت در فرمول زیر اگر اپسیلون اضافه نشود، مقدار مخرج برابر با صفر خواهد بود و به `Division by zero exception` میخوریم. اضافه کردن اپسیلون برای جلوگیری از این موضوع است.

$$Z\_norm = (Z.T - mu) / np.sqrt(var + eps)$$

(د) نرمال سازی دسته ای تکنیکی است که برای سریع تر و پایدارتر کردن آموزش شبکه های عصبی مصنوعی از طریق `rescale` و `re-center` کردن مقادیر دسته ورودی استفاده می شود. با این حال، استفاده از نرمال سازی دسته ای با اندازه دسته ای یک می تواند منجر به مشکلاتی شود.

هنگامی که اندازه دسته یک است، واریانس صفر خواهد بود و تقسیم بر صفر در هنگام نرمال سازی منجر به خطا می شود. با این حال، برخی از محققان پیشنهاد کرده اند که این خطا ممکن است به دلیل دو مورد رخ ندهد.



اول، خطا در یک بلوک try-catch رخ دهد. دوم، یک عدد گویا کوچک ( $1e-19$ ) به عبارت واریانس اضافه می شود تا هرگز صفر نشود.

با این حال، استفاده از نرمال سازی دسته ای با اندازه دسته ای یک توصیه نمی شود. دلایل نظری قوی علیه آن وجود دارد، و مقالات متعدد نشان داده اند که عملکرد نرمال سازی دسته ای برای اندازه های دسته ای کمتر از ۳۲ و به خصوص برای اندازه های دسته ای کمتر یا مساوی ۸ کاهش می یابد. همچنین معیارهای moving avg و moving var در این حالت از مقدار واقعی خود فاصله دارند.

(ه)

محاسبه تعداد پارامترهای خود لایه خطی با ۲۰ نورون:

$$20 * (10(\text{input dimension}) + 1(\text{one bias for each neuron})) = 220$$

محاسبه تعداد پارامترهای لایه نرمال سازی:

$$2(\text{gamma}, \text{beta}) * 20(\text{input channel dimension}) = 40$$

Because each channel has its own gamma and beta

پس تعداد کل پارامترهای قابل آموزش برابر است با:

$$220 + 40 = 260$$

البته میتوان برای لایه Dense در اینجا بایاس در نظر نگرفت، زیرا در لایه BN بایاس داریم که میتواند با لایه خطی قبل از خودش ترکیب شود. در آن صورت ۲۰ تا از پارامترها کم خواهد شد و ۲۴۰ پارامتر قابل آموزش خواهد ماند.

سوال ۷-

(الف)

دانلود دیتاست mnist:

```
from keras.datasets import mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

بر زدن دیتای آموزشی:

تنظیم seed باعث میشود نتایج قابل تولید دوباره باشند.

از مقدار shuffle\_index برای بر زدن هردوی ورودی و برچسب ها استفاده میشود تا هر برچسب ایندکس یکسانی با دیتای ورودی مربوط به خودش باشد.

```
import numpy as np
np.random.seed(0)
shuffle_index = np.random.permutation(60000)
x_train, y_train = x_train[shuffle_index], y_train[shuffle_index]
print(f"Training data dimensions: {x_train.shape}")
```

نمایش چند عکس با برچسب مربوط به آن به کمک matplotlib:

Label: 3      Label: 6      Label: 6      Label: 6      Label: 0



Label: 3      Label: 6      Label: 2      Label: 5      Label: 6



با کد زیر، دیتاست آموزشی و تست را نرمالایز میکنیم تا فرایند آموزش بهتر و سریعتر صورت بگیرد:

```
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255
```

از کد زیر برای one-hot encode کردن برچسب ها استفاده میشود تا تابع ضرر categorical cross entropy روی آن کار کند:

```
from keras.utils import to_categorical
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

تعریف مدل بر اساس لایه های ذکر شده در سوال:

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

model = Sequential([
    Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=(28, 28, 1)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu', padding='same'),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu', padding='same'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])

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

model.fit(x_train.reshape(-1, 28, 28, 1), y_train, epochs=15, batch_size=64)
```

استفاده از بهینه ساز Adam

حاصل آموزش:

```
938/938 [====] - 84s 89ms/step - loss: 0.0088 - accuracy: 0.9969
Epoch 10/15
938/938 [====] - 84s 89ms/step - loss: 0.0088 - accuracy: 0.9969
Epoch 11/15
938/938 [====] - 86s 92ms/step - loss: 0.0069 - accuracy: 0.9976
Epoch 12/15
938/938 [====] - 83s 88ms/step - loss: 0.0078 - accuracy: 0.9972
Epoch 13/15
938/938 [====] - 83s 88ms/step - loss: 0.0063 - accuracy: 0.9980
Epoch 14/15
938/938 [====] - 83s 88ms/step - loss: 0.0054 - accuracy: 0.9983
Epoch 15/15
938/938 [====] - 86s 91ms/step - loss: 0.0056 - accuracy: 0.9982
(keras.src.callbacks.History at 0x7b72590c2e90)
```

ب) برای الگوریتم Grad-CAM من از این [لینک](#) استفاده کردم و کد الگوریتمش به شکل زیر است:

```
def make_gradcam_heatmap(img_array, model, last_conv_layer_name, pred_index=None):
    # First, we create a model that maps the input image to the activations
    # of the last conv layer as well as the output predictions
    grad_model = keras.models.Model(
        model.inputs, [model.get_layer(last_conv_layer_name).output, model.output]
    )

    # Then, we compute the gradient of the top predicted class for our input image
    # with respect to the activations of the last conv layer
    with tf.GradientTape() as tape:
        last_conv_layer_output, preds = grad_model(img_array)
        if pred_index is None:
            pred_index = tf.argmax(preds[0])
            class_channel = preds[:, pred_index]

    # This is the gradient of the output neuron (top predicted or chosen)
    # with regard to the output feature map of the last conv layer
    grads = tape.gradient(class_channel, last_conv_layer_output)

    # This is a vector where each entry is the mean intensity of the gradient
    # over a specific feature map channel
    pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))

    # We multiply each channel in the feature map array
    # by "how important this channel is" with regard to the top predicted class
    # then sum all the channels to obtain the heatmap class activation
    last_conv_layer_output = last_conv_layer_output[0]
    heatmap = last_conv_layer_output @ pooled_grads[..., tf.newaxis]
    heatmap = tf.squeeze(heatmap)

    # For visualization purpose, we will also normalize the heatmap between 0 & 1
    heatmap = tf.maximum(heatmap, 0) / tf.math.reduce_max(heatmap)
    return heatmap.numpy()
```

ورودیهای این تابع، عکس ورودی، مدل آموزش دیده و اسم آخرین لایه کانولوشنی در آن مدل است.

ورودیهای بالا رو توی کد زیر به تابع دادم:

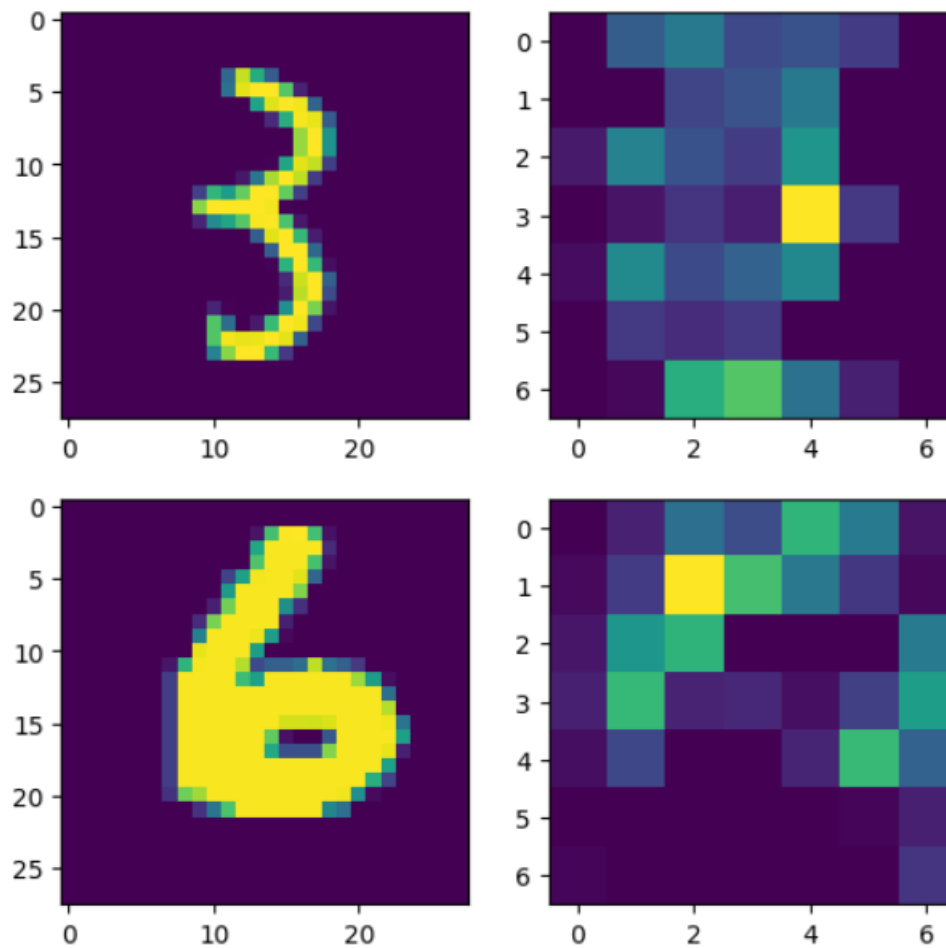
```

# Remove last layer's softmax
model.layers[-1].activation = None
# last convolutional layer
last_conv_layer_name = "conv2d_2"
for i in range(10):
    img_array = np.expand_dims(x_train[i], axis=0)
    heatmap = make_gradcam_heatmap(img_array, model, last_conv_layer_name)

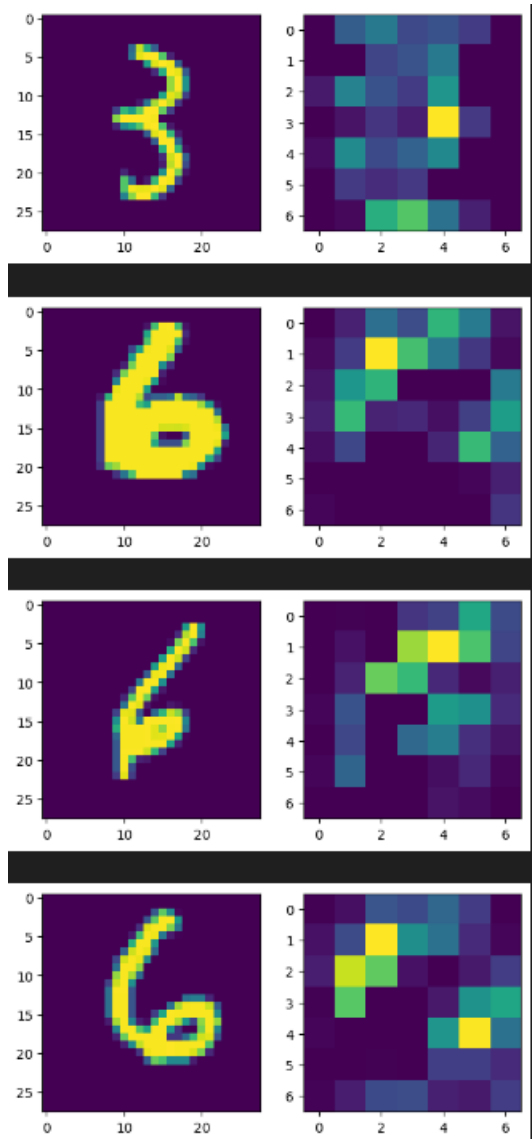
    # Display heatmap
    plt.matshow(heatmap)
    plt.show()

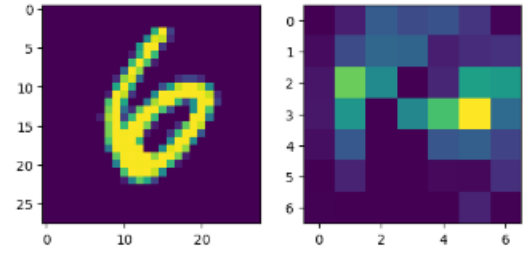
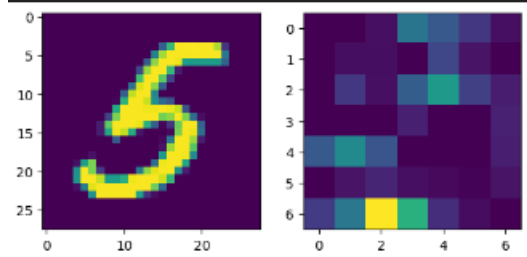
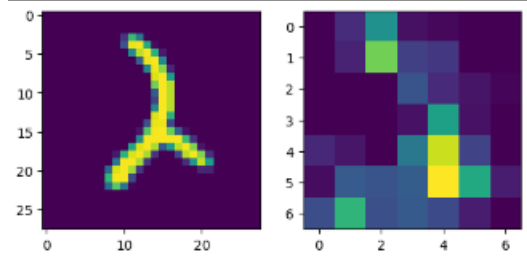
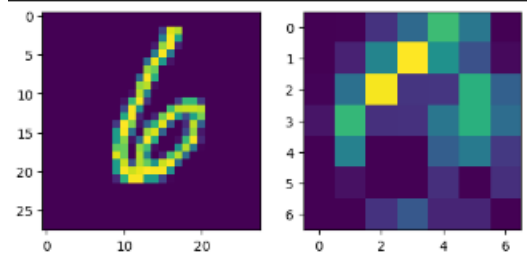
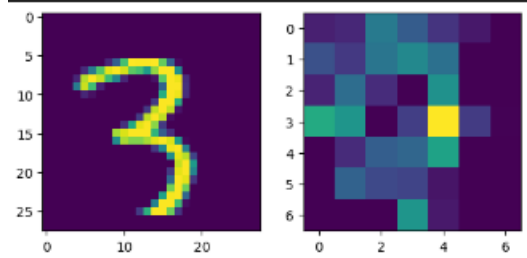
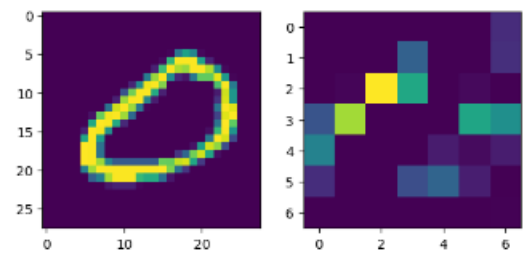
```

خروجی برای عکس اول و دوم:



(c)





نکاتی که درون این heat map ها میتوان به آن توجه کرد:

- خانه هایی از نقشه که برای یک رقم، مقادیر روشنی دارند، نشان دهنده ویژگی هایی هستند که مخصوص آن رقم است. برای مثال برای رقم ۶ در عکس دوم و چهارم پیکسل خانه (2,1) مقدار بزرگی دارد. یا برای رقم ۳ در دو عکسی که حاوی این رقم هستند، خانه (4,3) حاوی بیشترین مقدار هستند.
- نکته دیگر این است که ممکن است روشن بودن ترکیب چند خانه یا خاموش بودن ترکیب چند خانه، نشان دهنده یک رقم باشد، به همین دلیل بعضی خانه هایی که برای یک رقم ۶ روشن بودند، لزوماً برای عکسی دیگر حاوی این رقم روشن نخواهند بود.
- نقشه حرارتی روی ارقام یکسان با فونت های مختلف نتایج متفاوتی دارد ولی نتیجه گیری ای که کلیت نقشه در لایه های خطی بعدی میشود، همان رقم را با احتمال بالا پیش بینی میکند.