به نام خدا

تمرین چهارم

دستیاران : الناز رضایی، سید محمد موسوی، کمیل فتحی

سارا سادات یونسی – ۹۸۵۳۳۰۵۳

سوال ۱)

الف)

- Keras Tuner یک ابزار برای بهینهسازی هایپرپارامترهای مدلهای یادگیری عمیق با استفاده از کتابخانه Keras است. هایپرپارامترها عبارتند از پارامترهایی که قبل از آموزش مدل تعیین میشوند و تأثیر مستقیمی بر عملکرد و سرعت مدل دارند. مثلاً تعداد لایه ها، تعداد نورونها، نرخ یادگیری، تابع فعالسازی و غیره. Keras Tuner با ارائه چندین روش جستجوی هایپرپارامتر، از جمله جستجوی تصادفی، جستجوی فضای بایزی، جستجوی تقسیم شده و جستجوی هایپرباند، به شما امکان میدهد که به راحتی مدلهای خود را بر اساس معیارهای مورد نظر خود، مانند دقت یا خطا، بهینه کنید.

فراپارامترها پارامترهایی هستند که آموزش یک شبکه عصبی مصنوعی را دستکاری می‌کنند، با تنظیم آن‌ها می‌توانیم راه‌حل‌هایی با کیفیت بالا تولید کنیم. بر خلاف وزن های مدل، فراپارامترها در طول فرآیند آموزش توسط مدل تولید نمی شوند. هایپرپارامترها در طول زمان آموزش تغییر نمی کنند و ثابت می مانند و فرآیند آموزش یک مدل را دستکاری می کنند. یافتن فراپارامترهای بهینه یک مشکل جستجو است که با استفاده از آن ممکن است به بهترین راه حل ممکن که بهینه جهانی است یا خیر، برسیم.

فراپارامترهای یک شبکه عصبی مصنوعی عبارتند از

تعداد لایه ها برای انتخاب
تعداد نورون ها در یک لایه برای انتخاب
انتخاب تابع بهینه سازی
انتخاب نرخ یادگیری برای تابع بهینه سازی
انتخاب تابع ضرر
انتخاب معیارها
انتخاب تابع فعال سازی
انتخاب مقدار اولیه وزن لایه
اهمیت تنظیم فراپارامتر

تنظیم فراپارامتر فرآیند جستجوی مجموعه بهینه از فراپارامترها است. یافتن مجموعه بهینه ابرپارامترها به صورت دستی واقعاً سخت است، بنابراین الگوریتم‌های خاصی وجود دارند که جستجوی فراپارامتر ما را آسان‌تر می‌کنند. جستجوی شبکه‌ای یکی از الگوریتم‌هایی است که جستجوی جامع را انجام می‌دهد که طبیعتاً زمان‌بر است، بنابراین یک جایگزین برای آن الگوریتم جستجوی تصادفی است که به‌طور

تصادفی فضای جستجوی فراپارامتر را جستجو می‌کند اما یک راه‌حل بهینه جهانی را تضمین نمی‌کند. الگوریتم‌هایی که احتمال بیشتری برای ارائه راه‌حل‌های بهینه جهانی دارند، بهینه‌سازی بیزی، Hyperband و Hyperparameter با استفاده از الگوریتم‌های ژنتیک هستند.

فراپارامترها بر اساس تلفات پیش‌بینی‌های مدل، یعنی. فراپارامترها بر روی مدل تنظیم می شوند، مدل بر روی داده ها آموزش داده می شود و عملکرد مدل بر اساس تابع ضرر ارزیابی می شود. این مراحل به صورت تکراری دنبال می شوند تا مجموعه بهینه هایپرپارامترها را پیدا کنیم. همانطور که قبلاً گفته شد، ممکن است به یک راه حل بهینه جهانی دست پیدا کنیم یا نه.
KerasTuner یک چارچوب بهینه سازی هایپرپارامتر با کاربری آسان و مقیاس پذیر است که نقاط درد جستجوی فراپارامتر را حل می کند. به راحتی فضای جستجوی خود را با یک نحو تعریف شده پیکربندی کنید، سپس از یکی از الگوریتم های جستجوی موجود برای یافتن بهترین مقادیر فراپارامتر برای مدل های خود استفاده کنید. KerasTuner با الگوریتم‌های بهینه‌سازی بیزی، هایپرباند و جستجوی تصادفی داخلی ارائه می‌شود، و همچنین به گونه‌ای طراحی شده است که توسعه آن برای محققان برای آزمایش الگوریتم‌های جستجوی جدید آسان باشد.

- برای استفاده از Keras Tuner برای بهینهسازی شبکه همگشتی جهت دستهبندی دادهها، شما باید چند مرحله را طی کنید:

ابتدا باید دادههای خود را بارگذاری و پیشپردازش کنید. مثلاً برای دیتاست MNIST، شما میتوانید از تابع load_data کتابخانه Keras استفاده کنید تا دادههای آموزشی و آزمون را دریافت کنید. سپس باید دادهها را نرمال کنید و به شکل مناسب برای شبکه همگشتی تبدیل کنید.

سپس باید یک تابع برای ساخت مدل خود تعریف کنید. این تابع باید یک شیء از کلاس keras.Model را برگرداند. در این تابع، شما میتوانید از کلاسهای موجود در Keras Tuner برای تعریف هایپرپارامترهای مدل خود استفاده کنید. مثلاً برای تعیین تعداد لایهها، تعداد فیلترها، اندازه فیلترها، تابع فعالسازی و غیره. این کلاسها به شما امکان میدهند که یک فضای جستجوی هایپرپارامتر ایجاد کنید که Keras Tuner میتواند در آن به دنبال بهترین ترکیب از هایپرپارامترها باشد

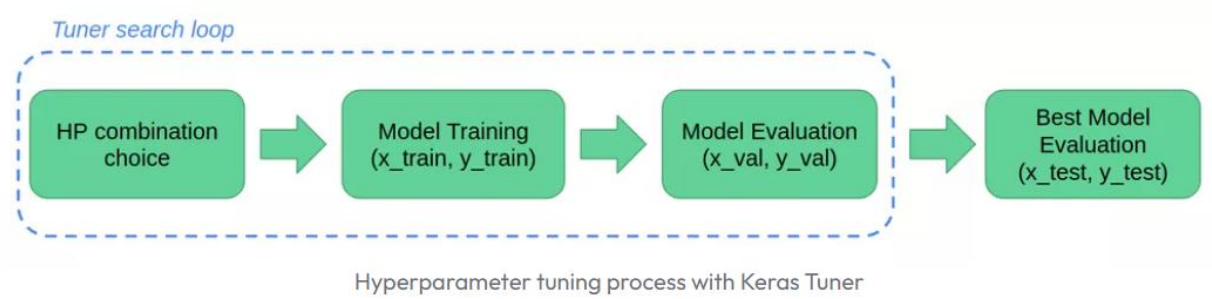
سپس باید یک شیء از کلاس Tuner را ایجاد کنید. این کلاس مسئول اجرای جستجوی هایپرپارامتر بر روی مدل شما است. شما میتوانید از یکی از روشهای جستجوی موجود در Keras Tuner انتخاب کنید، مانند RandomSearch، BayesianOptimization، Hyperband یا Sklearn. شما باید به این شیء تابع

ساخت مدل، دادههای آموزشی و آزمون، تعداد تکرارها، تعداد مدلهای مورد آزمایش و سایر پارامترهای مربوط به جستجو را بدهید.

در نهایت باید متد search را روی شیء Tuner فراخوانی کنید تا جستجوی هایپرپارامتر شروع شود. این متد مدلهای مختلف را با ترکیبهای مختلف از هایپرپارامترها آموزش میدهد و بر اساس معیارهای شما، بهترین مدل را انتخاب میکند. شما میتوانید نتایج جستجو را مشاهده کنید و مدل بهینه را بارگذاری کنید.

Main Steps:

1. Model-building function OR HyperModel subclass
2. Instantiate the tuner to be used [RandomSearch, Hyperband, or BayesianOptimization]
3. Run tuning using tuner.search()
4. Get optimal hyperparameters and models from the tuner
5. (optional) Try the newly found optimal hyperparameters by using simple methods provided by Keras Tuner



Hyperparameter tuning process with Keras Tuner

https://www.sicara.fr/blog-technique/hyperparameter-tuning-keras-tuner
/https://keras.io/keras_tuner
https://medium.com/swlh/hyperparameter-tuning-in-keras-tensorflow-2-with-keras-tuner-randomsearch-hyperband-3e212647778f

بتدا یک تیونر تعریف می شود. نقش آن تعیین این است که کدام ترکیب هایپرپارامتر باید آزمایش شوند. تابع جستجوی کتابخانه حلقه تکرار را انجام می دهد که تعداد معینی از ترکیب هایپرپارامتر را ارزیابی می کند. ارزیابی با محاسبه دقت مدل آموزش‌دیده بر روی یک مجموعه اعتبارسنجی انجام می‌شود.

در نهایت، بهترین ترکیب فراپارامتر از نظر دقت اعتبار سنجی را می توان بر روی یک مجموعه تست نگه داشته شده آزمایش کرد.

- Tuner های موجود در Keras Tuner عبارتند از:

RandomSearch: این Tuner به صورت تصادفی از فضای جستجوی هایپرپارامتر نمونهبرداری میکند و مدلها را بر اساس معیارهای شما رتبهبندی میکند. این Tuner ساده و سریع است، اما ممکن است به بهترین ترکیب از هایپرپارامترها نرسد\

BayesianOptimization: این Tuner از یک الگوریتم بهینهسازی بایزی برای انتخاب بهترین نمونهها از فضای جستجوی هایپرپارامتر استفاده میکند. این Tuner پیچیدهتر و کندتر از RandomSearch است، اما میتواند به نتایج بهتری برسد

Hyperband: این Tuner از یک الگوریتم بهینهسازی مبتنی بر چند باند برای انتخاب و آزمون نمونههای مختلف از فضای جستجوی هایپرپارامتر استفاده میکند. این Tuner از یک روش تقسیم و حل برای کاهش تعداد تکرارهای مدلها استفاده میکند و مدلهای ضعیف را حذف میکند. این Tuner سریعتر و کارآمدتر از RandomSearch و BayesianOptimization است.

Sklearn: این Tuner از یک رابط کاربری مشابه با کتابخانه Sklearn برای انتخاب و آزمون نمونههای مختلف از فضای جستجوی هایپرپارامتر استفاده میکند. این Tuner از روشهای جستجوی موجود در Sklearn، مانند GridSearchCV و RandomizedSearchCV پشتیبانی میکند

شما میتوانید بر اساس نیاز و هدف خود از هر یک از این Tuner ها استفاده کنید. اگر میخواهید یک روش ساده و سریع برای جستجوی هایپرپارامتر داشته باشید، میتوانید از RandomSearch استفاده کنید. اگر میخواهید یک روش دقیق و ریاضی برای جستجوی هایپرپارامتر داشته باشید، میتوانید از BayesianOptimization استفاده کنید. اگر میخواهید یک روش سریع و کارآمد برای جستجوی هایپرپارامتر داشته باشید، میتوانید از Hyperband استفاده کنید. اگر میخواهید یک روش شناخته شده و محبوب برای جستجوی هایپرپارامتر داشته باشید، میتوانید از Sklearn استفاده کنید.

از

```
tuner = RandomSearch(
    optimize_model,
    objective='val_accuracy',
    max_trials=10,
    directory='out',
    project_name='k_tuner'
)
```

تونر randomsearch به صورت تصادفی مقادیر هایپرپارامترها را از یک توزیع یا یک بازه مشخص انتخاب میکند و مدل را با آنها آموزش میدهد. سپس مدل را با استفاده از یک معیار هدف، مانند دقت یا خطا، ارزیابی میکند. این فرآیند را به تعدادی مشخص از آزمایشها یا تلاشها تکرار میکند و در نهایت بهترین مقادیر هایپرپارامترها را برمیگرداند.

مزایای استفاده از تونر randomsearch عبارتند از:

- سادگی و سرعت: تونر randomsearch نیازی به پیچیدگی یا حافظه زیادی ندارد و میتواند به سرعت مقادیر هایپرپارامترها را تولید و آزمایش کند.

- کاوش گسترده: تونر randomsearch میتواند مقادیر هایپرپارامترها را از یک فضای جستجوی گسترده انتخاب کند و احتمال بالاتری دارد که به نقاط بهینه یا نزدیک بهینه برسد.

- عدم وابستگی به ترتیب: تونر randomsearch به ترتیب انتخاب مقادیر هایپرپارامترها وابسته نیست و میتواند هر مقداری را در هر زمانی امتحان کند. این باعث میشود که تونر randomsearch به راحتی قابل موازیسازی باشد و از چندین پردازنده یا سیستم بهره ببرد.

nstantiate the tuner to perform the hypertuning. The Keras Tuner has four tuners available - RandomSearch, Hyperband, BayesianOptimization, and Sklearn. In this tutorial, you use the Hyperband tuner.

To instantiate the Hyperband tuner, you must specify the hypermodel, the objective to optimize and the maximum number of epochs to train (max_epochs).

RandomSearch concept:

One way to find the optimal parameters is to try every possible combinations of the available parameters [Grid Search], but the number of combinations would increase exponentially as the number of hyperparameters increases. Trying all the combinations one by one takes longer to explore the hyperparameter space. Random search helps to explore more hyperparameter space

in less time as compared to Grid Search. Exploring more hyperparameter space does not guarantee the absolute optimal results but close to optimal.

Hyperband Concept:

This technique tries to remove one of the problems in random search of hyperparameters. Problem is as follows: Random search may pick some values which are very obviously bad and will do full training and evaluation on it, which is wasteful. Hyperband provides one way to solve this problem. Hyperband Solution: Randomly sample all the combinations of hyperparameter and now instead of running full training and evaluation on it, train the model for few epochs (less than max_epochs) with these combinations and select the best candidates based on the results on these few epochs. It does this iteratively and finally runs full training and evaluation on the final chosen candidates. The number of iterations done depends on parameter 'hyperband_iterations' and number of epochs in each iteration are less than 'max_epochs'.

Bayesian Optimization tuner Concept:

This techniques addresses a common problem in RandomSearch and Hyperband. Problem: All the hyperparameter combinations are chosen randomly. Choosing hyperparameters randomly helps to explore the hyperparameter space but does not guarantee absolute optimal hyperparameters. Solution provided by BayesianOptimization tuner: Instead of all combinations being random, it chooses first few randomly, then based on the performance on these hyperparameters it chooses the next best possible hyperparameters. Hence it takes into account the history of the hyperparameters which were tried. The iterations of choosing next set of hyperparameters based on history and evaluating performance continues till the tuner reaches optimal hyperparameters or exhausts maximum number of allowed trails.

ب)

- مجموعه داده MNIST یک پایگاه داده بزرگ از ارقام دست نویس است که معمولاً برای آموزش و آزمایش سیستم های مختلف پردازش تصویر و یادگیری ماشین استفاده می شود. مجموعه داده MNIST شامل ۶۰۰۰۰ تصویر ارقام از ۰ تا ۹ برای آموزش و ۱۰۰۰۰ تصویر برای آزمایش است. اندازه هر تصویر ۲۸ در ۲۸ پیکسل است و دارای یک مقدار خاکستری است. مجموعه داده MNIST از مجموعه داده بزرگتری به نام NIST مشتق شده است که شامل ارقامی است که توسط کارمندان و دانشجویان آمریکایی نوشته شده است. مجموعه داده MNIST نرمالایز و مرکزی شده است تا استفاده از آن را آسان تر کند

مجموعه داده MNIST یکی از محبوب ترین و پرکاربردترین مجموعه داده ها در زمینه بینایی کامپیوتر و یادگیری ماشین است. اغلب به عنوان یک معیار برای ارزیابی عملکرد و دقت الگوریتم ها و مدل های مختلف استفاده می شود. برخی از کارهایی که می توان با مجموعه داده MNIST انجام داد عبارتند از:

تشخیص تصویر: هدف این است که تصاویر ارقام را به برچسب های مربوط به آنها، مانند ۰، ۱، ۲ و غیره طبقه بندی کنیم. این کار را می توان با استفاده از تکنیک های مختلف مانند شبکه های عصبی کانولوشن، ماشین های بردار پشتیبان، k-نزدیک ترین همسایگان انجام داد. ، و غیره.

تولید تصویر: هدف ایجاد تصاویر جدیدی از ارقام است که واقعی و متنوع به نظر می رسند، مانند ۰، ۱، ۲، و غیره. ، و غیره.

تبدیل تصویر: هدف این است که تصاویر ارقام را به نحوی اصلاح کنیم، مانند چرخش، مقیاس بندی، ترجمه، اضافه کردن نویز و غیره. ، انتقال سبک و غیره

پایگاه داده MNIST (پایگاه داده موسسه ملی استاندارد و فناوری اصلاح شده یک پایگاه داده بزرگ از ارقام دست نویس است که معمولاً برای آموزش سیستم های مختلف پردازش تصویر استفاده می شود پایگاه داده همچنین به طور گسترده برای آموزش و آزمایش در زمینه یادگیری ماشین استفاده می شودبا "آمیختن مجدد" نمونه‌های مجموعه داده‌های اصلی NIST ایجاد شد. سازندگان احساس کردند که از آنجایی که مجموعه داده آموزشی NIST از کارمندان اداره سرشماری آمریکا گرفته شده است، در حالی که مجموعه داده آزمایشی از دانش آموزان دبیرستانی آمریکایی گرفته شده است، برای آزمایش های یادگیری ماشین مناسب نیست.علاوه بر این، تصاویر سیاه و سفید از NIST نرمال سازی شدند تا در یک جعبه محدود کننده پیکسلی ۲۸x۲۸ قرار بگیرند و ضد مستعار شوند، که سطوح خاکستری را معرفی کرد

پایگاه داده MNIST شامل ۶۰۰۰۰ تصویر آموزشی و ۱۰۰۰۰ تصویر آزمایشی است. نیمی از مجموعه آموزشی و نیمی از مجموعه تست از مجموعه داده آموزشی NIST گرفته شده است، در حالی که نیمی دیگر از مجموعه آموزشی و نیمی دیگر از مجموعه تست از مجموعه داده های آزمایشی NIST گرفته شده است. سازندگان اصلی پایگاه داده فهرستی از برخی از روش های آزمایش شده بر روی آن را نگه می دارند در مقاله اصلی خود، آنها از یک ماشین بردار پشتیبان برای دریافت نرخ خطای ۰٫۸٪ استفاده می کنند.

Extended MNIST (EMNIST) مجموعه داده جدیدتری است که توسط NIST به عنوان جانشین (نهایی) MNIST توسعه و منتشر شده استMNIST فقط شامل تصاویر ارقام دست نویس می شد. EMNIST شامل تمام تصاویر پایگاه داده ویژه NIST 19 است که یک پایگاه داده بزرگ از حروف بزرگ و کوچک دست‌نویس و همچنین ارقام است. تصاویر در EMNIST با همان فرآیند مانند تصاویر MNIST به همان فرمت پیکسلی ۲۸x۲۸ تبدیل شدند. بر این اساس، ابزارهایی که با مجموعه داده‌های قدیمی‌تر و کوچک‌تر MNIST کار می‌کنند، احتمالاً بدون تغییر با EMNIST کار می‌کنند.

MNIST set is a large collection of handwritten digits. It is a very popular dataset in the field of image processing. It is often used for benchmarking machine learning algorithms. MNIST is short for Modified National Institute of Standards and Technology database

- CNN مخفف شبکه عصبی کانولوشن است که نوعی شبکه عصبی مصنوعی است که می تواند تصاویر را پردازش کرده و ویژگی هایی را از آنها استخراج کند. CNN از لایه های متعددی مانند لایه های کانولوشن، لایه های ادغام، لایه های نرمال سازی دسته ای و لایه های متراکم تشکیل شده است. لایه‌های کانولوشن فیلترهایی را روی تصاویر ورودی اعمال می‌کنند و نقشه‌های ویژگی را تولید می‌کنند که نشان دهنده حضور الگوها یا ویژگی‌های خاص در تصاویر است. لایه‌های ادغام، اندازه و پیچیدگی نقشه‌های ویژگی را با اعمال یک عملیات نمونه‌برداری کاهش می‌دهند، مانند حداکثر ادغام یا ادغام متوسط. لایه‌های نرمال‌سازی دسته‌ای، نقشه‌های ویژگی را برای بهبود پایداری و عملکرد شبکه مقیاس‌بندی و عادی می‌کنند. لایه های متراکم لایه های کاملاً متصل هستند که طبقه بندی نهایی تصاویر را بر اساس ویژگی های استخراج شده انجام می دهند

Keras Tuner ابزاری است که می تواند هایپرپارامترهای شبکه CNN مانند تعداد لایه ها، تعداد فیلترها، اندازه فیلترها، تابع فعال سازی، نرخ یادگیری و غیره را بهینه کند. هایپرپارامترها پارامترهایی هستند که قبل از آموزش شبکه تنظیم می شوند و بر عملکرد و دقت شبکه تاثیر می گذارند. Keras Tuner می تواند با استفاده از روش های مختلف مانند جستجوی تصادفی، بهینه سازی بیزی، Hyperband یا Sklearn بهترین ترکیب از هایپرپارامترها را جستجو کند. Keras Tuner می تواند عملکرد شبکه را بر اساس معیارهای مختلف ارزیابی کند، مانند دقت یا ضرر
برای استفاده از شبکه CNN بهینه شده با Keras Tuner برای طبقه بندی تصاویر این مجموعه داده MNIST، باید این مراحل را دنبال کنید:

ابتدا باید مجموعه داده MNIST را بارگیری و از قبل پردازش کنید. می توانید از کتابخانه Keras یا کتابخانه TensorFlow Datasets برای بارگذاری مجموعه داده استفاده کنید که شامل ۶۰۰۰۰ تصویر از ارقام دست نویس از ۰ تا ۹ برای آموزش و ۱۰۰۰۰ تصویر برای آزمایش است. شما باید تصاویر را عادی کنید و شکل آن ها را برای شبکه CNN مناسب کنید

دوم، باید تابعی را تعریف کنید که شبکه CNN را بسازد. در این تابع باید معماری شبکه را مشخص کنید و از کلاس های Keras Tuner برای تعریف فضای هایپرپارامتری که می خواهید جستجو کنید استفاده کنید. برای مثال، می‌توانید از کلاس‌های Int، Float، Boolean یا Choice برای تعریف محدوده یا گزینه‌هایی برای هایپرپارامترها مانند تعداد لایه‌ها، تعداد فیلترها، اندازه فیلترها، تابع فعال‌سازی و غیره استفاده کنید.

سوم، باید یک شی از کلاس Tuner ایجاد کنید. این کلاس وظیفه انجام جستجوی هایپرپارامتر در شبکه شما را بر عهده دارد. می‌توانید یکی از روش‌های جستجو را از Keras Tuner انتخاب کنید، مانند RandomSearch، BayesianOptimization، Hyperband یا Sklearn. باید این شیء را با تابعی که شبکه را می سازد، داده های آموزش و آزمایش، تعداد دوره ها، تعداد مدل های مورد آزمایش و سایر پارامترهای مربوط به جستجو را ارائه دهید.

در نهایت باید متد جستجو را روی شی Tuner فراخوانی کنید تا جستجوی هایپرپارامتر شروع شود. این روش مدل های مختلف را با ترکیب های مختلف فراپارامترها آموزش می دهد و بر اساس معیارهایی که شما مشخص می کنید بهترین مدل را انتخاب می کند. می توانید نتایج جستجو را مشاهده کرده و مدل بهینه را بارگذاری کنید.

۱.ابتدا یک مدل کانولوشنی میسازیم .

این لایه ی کانولوشنی دارای هایپرپارامتر ها و لایه ها و انواع فیلتر ها می باشد.

۲. سپس بازه ی هر یک از این هایپر پارامتر هارا تعیین می کنینم.

۳.حالا یک الگوریتم سرچ از کراس تونر انتخاب می کنیم که شامل تونر های مختلف است که بالا معرفی کردیم

۴.حالا تونر را روی مدل فیت می کنیم .

۵.فرایند بهینه سازی را را اجرا می کنیم روی مدل

۶یک مدل با هایپر پارامتر های بهینه می سازیم

۷روی ان ترین می کنیم و اموزش می دهیم

۸. توشط داده های تست عملکرد مدل را می سنجیم و در صورت نیاز ان را بهبود می دهیم

- Dropout و Pooling دو تکنیکی هستند که می‌توانند در برخی از لایه‌های ANN برای بهبود عملکرد و کاهش بیش از حد برازش اعمال شوند. برازش بیش از حد مشکلی است که زمانی رخ می دهد که ANN نویز یا الگوهای خاصی را در داده های آموزشی یاد می گیرد، که باعث می شود در داده های جدید یا دیده نشده عملکرد ضعیفی داشته باشد. Dropout و Pooling می تواند با کاهش پیچیدگی و اندازه ANN به منظم کردن و تعمیم آن کمک کند

Dropout تکنیکی است که به طور تصادفی برخی از گره ها یا نورون ها را در یک لایه در طول فرآیند آموزش حذف یا غیرفعال می کند. این بدان معنی است که برخی از گره ها هیچ ورودی دریافت نمی کنند یا خروجی برای یک تکرار آموزشی مشخص نمی کنند. نرخ انصراف کسری از گره هایی است که حذف می شوند که معمولاً بین ۰,۱ تا ۰,۵ است. Dropout باعث ایجاد یک شبکه کوچکتر و ساده تر برای هر تکرار آموزشی می شود که باعث کاهش انطباق مشترک گره ها و وابستگی به ویژگی های خاص می شود. ترک تحصیل همچنین به عنوان

شکلی از یادگیری گروهی عمل می کند، که در آن چندین شبکه فرعی آموزش داده می شوند و برای تولید خروجی نهایی میانگین می گیرند. Dropout می تواند عملکرد و دقت ANN را با جلوگیری از برازش بیش از حد و افزایش تنوع ویژگی ها بهبود بخشد.

pooling تکنیکی است که اندازه و ابعاد نقشه های ویژگی تولید شده توسط لایه های کانولوشن در یک شبکه عصبی کانولوشنال (CNN) را کاهش می دهد. CNN نوعی شبکه عصبی مصنوعی است که می تواند تصاویر را پردازش کرده و با استفاده از فیلترها یا هسته ها، ویژگی ها را از آنها استخراج کند. ادغام یک عملیات نمونه برداری پایین، مانند حداکثر ادغام یا ادغام میانگین، را در یک منطقه از نقشه ویژگی اعمال می کند و یک مقدار واحد را که نشان دهنده آن منطقه است، به دست می آورد. ادغام اثر کاهش تعداد پارامترها و محاسبات در CNN را دارد که آن را سریعتر و کارآمدتر می کند. ادغام همچنین مقداری تغییر ناپذیری ترجمه را معرفی می کند، به این معنی که CNN می تواند همان ویژگی ها را بدون توجه به موقعیت یا جهت آنها در تصویر تشخیص دهد. ادغام می تواند عملکرد و دقت CNN را با جلوگیری از نصب بیش از حد و گرفتن مرتبط ترین ویژگی ها بهبود بخشد.

Dropout یک تکنیک منظم سازی است که از برازش بیش از حد شبکه های عصبی جلوگیری می کند. روش‌های منظم‌سازی مانند L1 و L2 با اصلاح تابع هزینه، اضافه برازش را کاهش می‌دهند. از طرف دیگر، خود شبکه را تغییر دهید. در طول آموزش در هر تکرار به طور تصادفی نورون ها را از شبکه عصبی رها می کند. وقتی مجموعه های مختلفی از نورون ها را رها می کنیم، معادل آموزش شبکه های عصبی مختلف است. شبکه‌های مختلف به روش‌های مختلف اضافه برازش خواهند کرد، بنابراین اثر خالص ترک تحصیل کاهش بیش‌برازش خواهد بود.

این تکنیک در نمودار بالا نشان داده شده است. همانطور که می بینیم، حذف تصادفی نورون ها در حین آموزش شبکه عصبی استفاده می شود. این تکنیک ثابت کرده است که بیش از حد برازش را به مشکلات مختلفی از جمله طبقه بندی تصویر، تقسیم بندی تصویر، جاسازی کلمات، تطبیق معنایی و غیره کاهش می دهد.

Dropout is a technique for addressing this problem. The key idea is to randomly drop units (along with their connections) from the neural network during training. This prevents units from co-adapting too much. During training, dropout samples from an exponential number of different thinned networks. At test time, it is easy to approximate the effect of averaging the predictions of all these thinned networks by simply using a single unthinned network that has smaller weights. This significantly reduces overfitting and gives major improvements over other regularization methods. We show that dropout improves the performance of neural networks on supervised learning tasks in vision, speech recognition, document classification and computational biology, obtaining state-of-the-art results on many benchmark data sets.

علاوه بر این، ادغام توانایی یادگیری ویژگی های ثابت را فراهم می کند و همچنین به عنوان یک تنظیم کننده برای کاهش بیشتر مشکل بیش از حد مناسب عمل می کند.
ابعاد نقشه های ویژگی تولید شده توسط لایه های کانولوشنال توسط لایه های ادغام کاهش می یابد. این کمک می کند تا نیازهای محاسباتی شبکه را محدود کرده و از هرگونه پتانسیل بیش از حد برازش جلوگیری کند.
تطبیق بیش از حد ممکن است زمانی اتفاق بیفتد که مجموعه داده شما به اندازه کافی بزرگ نباشد که بتواند تعداد ویژگی های شما را برآورده کند. Max Pooling از یک عملیات حداکثر برای جمع کردن مجموعه‌ای از ویژگی‌ها استفاده می‌کند و تعداد کمتری از آنها را در اختیار شما قرار می‌دهد. بنابراین، max-pooling به طور منطقی باید اضافه فیت را کاهش دهد.

دراپ اوت با اطمینان از اینکه آن ویژگی همیشه در دسترس نیست، وابستگی به هر ویژگی را کاهش می‌دهد، و مدل را مجبور می‌کند تا به دنبال نکات بالقوه مختلف باشد، نه اینکه فقط به یکی بچسبد - که به راحتی به مدل اجازه می‌دهد تا در هر اشاره به ظاهر خوب بیش از حد مناسب شود. بنابراین، این نیز باید به کاهش اضافه فیت کمک کند. و دقت را افزایش دهد.

ج)

- من دو مدل را پیاده سازی کردم تا بتوان نحوه ی کار کراس تونر را بهتر مشاهده کرد اولی مدلی که با تعداد لایه های بسیار زیادی تنظیم شده و تعداد پرامتر ها بسیار زیاد است اما در نهایت میبینیم که دقت و عملکرد مدل ما توسط ان مدلی که با کرا س تونر تنظیم شده است بیش تر میباشد .

این تحلیل را با استفاده از نمودار های رسم شده و میزان لای برای داده های ترین و تست می توان متوجه شد.

حالا به بررسی گام ها میپردازم :

مدل عادی :

ایپورت کتابخانه ها

لود کردن دیتاست

تقسیم به داده های اموزش و تست و لیبل های اموزش و تست

بر زدن داده های اموزشی

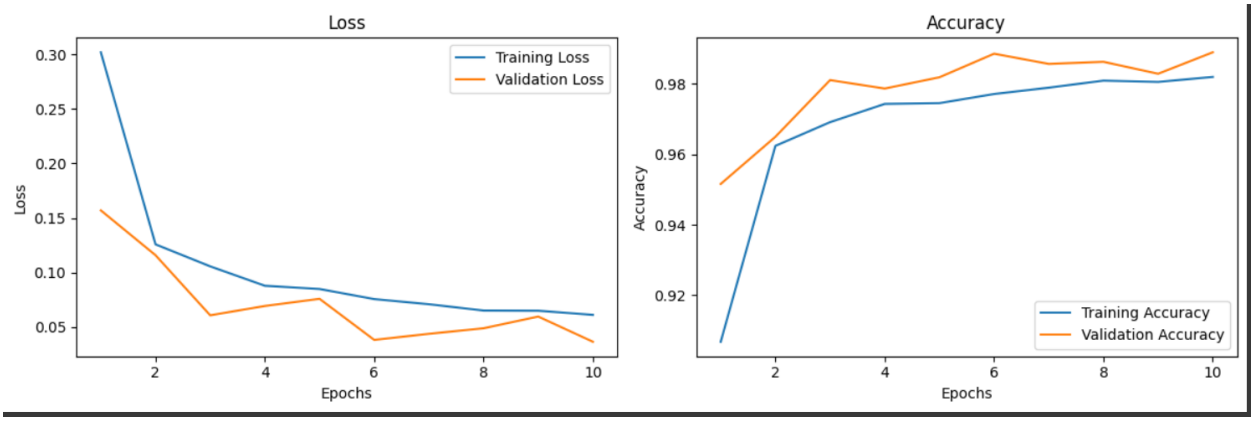
پلات کردن چند دیتا با لیبل ان ها

نرمال سازی داده ها و پیکسل عکس ها برای عملکرد بهتر مدل

تبدیل لیبل ها به کتگورکال برای استفاده از تابع لاس مورد نظر در ادامه

پیاده سازی مدل
کامپایل کردن مدل با بهینه ساز ادام
و تابع لاس کتگوریکال
و متریک دقت
و تست و اموزش مدل
و مقدار نهایی:

```
Epoch 10/10
938/938 [==============================] - 9s 10ms/step - loss: 0.0612 -
            accuracy: 0.9820 - val_loss: 0.0366 - val_accuracy: 0.9890
```



مدل بهینه :

از لایه های پولینگ و ادغام هم به صورتی استفاده شده که مقدار هیپر متری ان ها با استفاده از کراس تونر پیدا میشود.

مدل را با توجه به موارد گفته شده پیاده سازی کردیم.

این کد تابعی به نام optimize_model را تعریف می کند که یک مدل شبکه عصبی کانولوشن (CNN) را با استفاده از کتابخانه Keras ایجاد می کند و هایپرپارامترها را برای تنظیم با استفاده از آرگومان hp مشخص می کند. هایپرپارامترها متغیرهایی هستند که معماری و فرآیند آموزش مدل را کنترل می کنند، مانند تعداد لایه ها، تعداد فیلترها، اندازه هسته، تابع فعال سازی، نرخ یادگیری و غیره. این تابع مدل را برمی گرداند و می تواند برای جستجوی بهترین هایپرپارامترها با استفاده از تیونرهای مختلف مانند RandomSearch یا Hyperband استفاده می شود.

• model = keras.Sequential(): این قسمت یک شی از کلاس keras.Sequential ایجاد می کند و آن را در مدل متغیر ذخیره می کند. این شی یک مدل خطی از لایه های شبکه عصبی را نشان می دهد که می توان به آن اضافه کرد.

• num_class=10: این قسمت متغیری به نام num_class را با مقدار ۱۰ تعریف می کند. این مقدار نشان دهنده تعداد کلاس های خروجی مدل است. در این مورد، فرض می کنیم که مدل برای طبقه بندی تصاویر MNIST است که دارای ۱۰ کلاس هستند.

• for i in range(hp.Int('Conv_layer', 1, 5)):: این قسمت یک حلقه برای افزودن لایه های کانولوشن به مدل ایجاد می کند. این حلقه بسته به مقدار Hyperparameter Conv_layer که با استفاده از تابع hp.Int از یک محدوده صحیح انتخاب می شود، از یک تا پنج بار تکرار می شود. این تابع سه آرگومان می گیرد: نام هایپرپارامتر، کران پایین محدوده و کران بالای محدوده.

• model.add(layers.Conv2D(filters=hp.Int('Filter' + str(i), 32, 256, step=32), kernel_size=hp.Choice('Kernel' + str(i), values= [3, 5]), activation='relu')): این قسمت با استفاده از لایه های کلاس Conv2D یک لایه کانولوشنال اضافه می کند و به مدل اضافه می کند. این لایه ویژگی های تصاویر را با استفاده از فیلترهایی که روی آنها حرکت می کند استخراج می کند. این لایه چندین آرگومان می گیرد که برخی از آنها فراپارامترهای مدل هستند:

• filters=hp.Int('Filter' + str(i), 32, 256, step=32): این آرگومان تعداد فیلترهای لایه کانولوشن را مشخص می کند. این مقدار با استفاده از تابع hp.Int از یک محدوده عدد صحیح با گام ۳۲ انتخاب می شود. این تابع چهار آرگومان می گیرد: نام هایپرپارامتر، کران پایین محدوده، کران بالای محدوده، و مرحله از انتخاب.

• kernel_size=hp.Choice('Kernel' + str(i), values=[3, 5]): این آرگومان اندازه هسته های لایه کانولوشن را مشخص می کند. این مقدار با استفاده از تابع hp.Choice از لیستی از مقادیر ممکن انتخاب می شود. این تابع دو آرگومان می گیرد: نام هایپرپارامتر و لیست مقادیر.

• activation='relu': این آرگومان تابع فعال سازی لایه کانولوشن را مشخص می کند. این تابع مقدار خروجی هر نورون را بر اساس مقدار ورودی آن تغییر می دهد. در این حالت تابع relu انتخاب می شود که در صورت مثبت بودن مقدار ورودی و در صورت منفی بودن صفر را برمی گرداند.

• beta=1: این قسمت متغیری به نام beta را با مقدار ۱ تعریف می کند. این مقدار برای تعیین شرط افزودن لایه های ادغام و حذف به مدل استفاده می شود.

• if (i % 2 == beta):: این قسمت شرطی را برای افزودن لایه های جمع و حذف به مدل ایجاد می کند. این شرط بررسی می کند که آیا تعداد مدول لایه کانولوشنال ۲ برابر با مقدار بتا است یا خیر. اگر مساوی باشد به این معنی است که لایه کانولوشن فرد است و باید لایه های pooling و dropout به آن اضافه شود. اگر نه، به این معنی است که لایه کانولوشن یکنواخت است و نیازی به اضافه کردن لایه های pooling و dropout نیست.

• 'if hp.Choice('pooling_' + str(i), values=['max', 'avg']) == 'max':: این قسمت شرطی را برای افزودن یک لایه ادغام به مدل ایجاد می کند. این شرط بررسی می کند که آیا مقدار hyperparameter pooling_ که با استفاده از تابع hp.Choice از لیست مقادیر انتخاب شده است برابر با max است یا خیر. اگر برابر باشد به این معنی است که یک لایه max pooling باید به مدل اضافه شود. در غیر این صورت، به این معنی است که یک لایه جمع کننده متوسط باید به مدل اضافه شود.

• model.add(layers.MaxPooling2D()): این قسمت با استفاده از لایه های کلاس.MaxPooling2D یک لایه max pooling اضافه می کند و به مدل اضافه می کند. این لایه با گرفتن حداکثر مقدار هر زیر منطقه از داده ها، ابعاد داده های ورودی را کاهش می دهد.

• model.add(layers.AveragePooling2D()): این قسمت با استفاده از لایه های کلاس یک لایه ادغام متوسط را اضافه می کند.AveragePooling2D و به مدل اضافه می کند. این لایه با در نظر گرفتن مقدار متوسط هر زیر منطقه از داده ها، ابعاد داده های ورودی را کاهش می دهد.

• model.add(layers.Dropout(hp.Float('Dropout_1' + str(i), 0, 0., step=0.2))): این قسمت با استفاده از لایه های کلاس یک لایه dropout اضافه می کند. Dropout به آن اضافه می کند. مدل. این لایه به طور تصادفی برخی از نورون های لایه قبلی را در طول فرآیند آموزش از بین می برد. این به مدل کمک می کند تا بر ویژگی های مهم تمرکز کند و وابستگی بین نورون ها را کاهش دهد. در نتیجه، مدل در برابر برازش بیش از حد قوی تر می شود و دقت تعمیم آن را افزایش می دهد

model.add(layers.MaxPooling2D()): این قسمت با استفاده از لایه های کلاس.MaxPooling2D یک لایه max pooling اضافه می کند و به مدل اضافه می کند. این لایه با گرفتن حداکثر مقدار هر زیر منطقه از داده ها، ابعاد داده های ورودی را کاهش می دهد.

• model.add(layers.AveragePooling2D()): این قسمت با استفاده از لایه های کلاس یک لایه ادغام متوسط را اضافه می کند.AveragePooling2D و به مدل اضافه می کند. این لایه با در نظر گرفتن مقدار متوسط هر زیر منطقه از داده ها، ابعاد داده های ورودی را کاهش می دهد.

• model.add(layers.Dropout(hp.Float('Dropout_1' + str(i), 0, 0., step=0.2))): این قسمت با استفاده از لایه های کلاس یک لایه dropout اضافه می کند. Dropout به آن اضافه می کند. مدل. این لایه به طور تصادفی برخی از نورون های لایه قبلی را در طول فرآیند آموزش از بین می برد. این به مدل کمک می کند تا بر ویژگی های مهم تمرکز کند و وابستگی بین نورون ها را کاهش دهد. در نتیجه، مدل در برابر برازش بیش از حد قوی تر می شود و دقت تعمیم آن را افزایش می دهد. این لایه یک آرگومان می گیرد که یک هایپرپارامتر مدل است:

• hp.Float('Dropout_1' + str(i), 0, 0., step=0.2): این آرگومان میزان حذف لایه را مشخص می کند. این مقدار با استفاده از تابع hp.Float از محدوده float با گام ۰,۲ انتخاب می شود. این تابع چهار آرگومان می گیرد: نام هایپرپارامتر، کران پایین محدوده، کران بالای محدوده و مرحله انتخاب.

• model.add(layers.Flatten()): این قسمت با استفاده از لایه های کلاس یک لایه مسطح اضافه می کند. Flatten و به مدل اضافه می کند. این لایه داده های ورودی را به یک بردار یک بعدی مسطح می کند. این برای اتصال لایه های کانولوشن به لایه های کاملاً متصل ضروری است.

• for i in range(hp.Int('Dense_numbers', 1, 4)): این قسمت حلقه ای برای افزودن لایه های کاملا متصل به مدل ایجاد می کند. این حلقه بسته به مقدار هایپرپارامتر Dense_numbers که با استفاده از تابع hp.Int از یک محدوده صحیح انتخاب می شود، از یک تا چهار بار تکرار می شود. این تابع سه آرگومان می گیرد: نام هایپرپارامتر، کران پایین محدوده و کران بالای محدوده.

• model.add(layers.Dense(units=hp.Int('Units_numbers' + str(i), 32, 256, step=32), activation='relu')): این قسمت یک لایه کاملا متصل را با استفاده از class layers.Dense و آن را به مدل اضافه می کند. این لایه تمام نورون های لایه قبلی را به تمام نورون های لایه فعلی متصل می کند. این لایه دو آرگومان می گیرد که یکی از آنها یک هایپرپارامتر مدل است:

• units=hp.Int('Units_numbers' + str(i), 32, 256, step=32): این آرگومان تعداد نورون های لایه کاملا متصل را مشخص می کند. این مقدار با استفاده از تابع hp.Int از یک محدوده عدد صحیح با گام

۳۲ انتخاب می شود. این تابع چهار آرگومان می گیرد: نام هایپرپارامتر، کران پایین محدوده، کران بالای محدوده، و مرحله از انتخاب.

• activation='relu': این آرگومان تابع فعال سازی لایه کاملا متصل را مشخص می کند. این تابع مقدار خروجی هر نورون را بر اساس مقدار ورودی آن تغییر می دهد. در این حالت تابع relu انتخاب می شود که در صورت مثبت بودن مقدار ورودی و در صورت منفی بودن صفر را برمی گرداند.

• model.add(layers.Dropout(hp.Float('Dropout_2' + str(i), 0, 0.7, step=0.1))): این قسمت با استفاده از لایه های کلاس یک لایه dropout اضافه می کند. Dropout به آن اضافه می کند. مدل. این لایه به طور تصادفی برخی از نورون های لایه قبلی را در طول فرآیند آموزش از بین می برد. این به مدل کمک می کند تا بر ویژگی های مهم تمرکز کند و وابستگی بین نورون ها را کاهش دهد. در نتیجه، مدل در برابر برازش بیش از حد قوی تر می شود و دقت تعمیم آن را افزایش می دهد. این لایه یک آرگومان می گیرد که یک هایپرپارامتر مدل است:

• hp.Float('Dropout_2' + str(i), 0, 0.7, step=0.1): این آرگومان میزان حذف لایه را مشخص می کند. این مقدار با استفاده از تابع hp.Float از محدوده float با گام ۰,۱ انتخاب می شود. این تابع چهار آرگومان می گیرد: نام هایپرپارامتر، کران پایین محدوده، کران بالای محدوده و مرحله انتخاب.

• model.add(layers.Dense(num_class, activation='softmax')): این قسمت لایه خروجی را با استفاده از لایه های کلاس اضافه می کند.Dense و به مدل اضافه می کند. این لایه تمام نورون های لایه قبلی را به تمام نورون های لایه فعلی متصل می کند. این لایه دو آرگومان می گیرد:

• num_class: این آرگومان تعداد نورون های لایه خروجی را مشخص می کند. این مقدار برابر با متغیر num_class است که قبلاً تعریف شده بود. این مقدار تعداد کلاس های خروجی مدل را نشان می دهد. در این صورت ۱۰ است.

• activation='softmax': این آرگومان تابع فعال سازی لایه خروجی را مشخص می کند. این تابع مقدار خروجی هر نورون را بر اساس ورودی آن تغییر می دهد

• optimizer=keras.optimizers.Adam(hp.Choice('learning_rate', values=[0.001, 0.0001, 0.00001])): این آرگومان بهینهسازی که مدل برای بهروزرسانی وزنهای خود استفاده میکند را مشخص میکند. در این مورد، بهینهساز metrics-to-be-evaluated-by-the-model انتخاب شده است

که یکی از بهترین بهینهسازهای موجود برای شبکههای همگشتی است. این بهینهساز یک آرگومان میگیرد که یک هایپرپارامتر است:

- hp.Choice('learning_rate', values=[0.001, 0.0001, 0.00001]): این آرگومان نرخ یادگیری بهینهساز را مشخص میکند. این مقدار با استفاده از تابع hp.Choice از یک لیست از مقادیر ممکن انتخاب میشود. این تابع دو آرگومان میگیرد: نام هایپرپارامتر و لیست مقادیر. نرخ یادگیری تعیین میکند که مدل چقدر سریع بهینه میشود و باید با دقت انتخاب شود.

- loss='sparse_categorical_crossentropy': این آرگومان تابع هزینهای را که مدل برای محاسبه خطای خود استفاده میکند را مشخص میکند. در این مورد، تابع انتخاب شده است که مناسب برای مسئله دستهبندی چند کلاسه است. این تابع مقدار خروجی مدل را با مقدار واقعی کلاس مقایسه میکند و میزان اختلاف آنها را به عنوان خطا برمیگرداند.

- metrics=['accuracy']: این آرگومان یک لیست از معیارهایی را که مدل برای ارزیابی عملکرد خود استفاده میکند را مشخص میکند. در این مورد، معیار پ انتخاب شده است که نشان میدهد که مدل چه درصد از تصاویر را درست دستهبندی میکند.

کد شما در نهایت مدل را بازگردانده و میتوانید آن را با دادههای آموزشی و آزمون آموزش دهید.

```
def optimize_model(hp):
    model = keras.Sequential() #
    num_class=10

    for i in range(hp.Int('Conv_layer', 1, 5)):
        model.add(layers.Conv2D(
            filters=hp.Int('Filter' + str(i), 32, 256, step=32),
            kernel_size=hp.Choice('Kernel' + str(i), values=[3, 5]),
            activation='relu'
        ))

        beta=1
        if (i % 2 == beta):
            if hp.Choice('pooling_' + str(i), values=['max', 'avg']) == 'max':
                model.add(layers.MaxPooling2D())
            else:
```

```
                model.add(layers.AveragePooling2D())

            model.add(layers.Dropout(hp.Float('Dropout_1' + str(i), 0, 0., step=0.2)))

    model.add(layers.Flatten())


    for i in range(hp.Int('Dense_numbers', 1, 4)):
        model.add(layers.Dense(units=hp.Int('Units_numbers' + str(i), 32, 256,
step=32), activation='relu'))
        model.add(layers.Dropout(hp.Float('Dropout_2' + str(i), 0, 0.7, step=0.1)))

    model.add(layers.Dense(num_class, activation='softmax'))


    model.compile(
        optimizer=keras.optimizers.Adam(
            hp.Choice('learning_rate', values=[0.001, 0.0001, 0.00001])
        ),
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy']
    )

     return model
```

```
from keras_tuner.tuners import RandomSearch
tuner = RandomSearch(
    optimize_model,
    objective='val_accuracy',
    max_trials=10,
    directory='out',
    project_name='k_tuner'

)
```

حال می توان داده ها را ری شیپ و نرمالایز کرد و سپس

```
tuner.search(x_train, y_train, epochs=20, validation_data=(x_test,
y_test))
```

از تابع optimize_model برای ساختن و بازگرداندن یک مدل کراس استفاده میکند. این تابع باید یک پارامتر به نام hp دریافت کند که یک شیء از کلاس HyperParameters است. با استفاده از این شیء، شما میتوانید فضای جستجوی خود را با استفاده از متدهایی مانند hp.Choice، hp.Int، hp.Float و hp.Boolean تعریف کنید. این متدها به شما اجازه میدهند که یک مجموعه از مقادیر ممکن یا یک بازه از مقادیر را برای هر هایپرپارامتر مشخص کنید

https://blog.tensorflow.org/2020/01/hyperparameter-tuning-with-keras-tuner.html

.

کد شما از پارامتر objective برای مشخص کردن هدفی که میخواهید مدلهای خود را بر اساس آن ارزیابی کنید، استفاده میکند. در این مورد، شما از val_accuracy به عنوان هدف استفاده میکنید، که نشان میدهد که شما میخواهید دقت مدل خود را روی دادههای اعتبارسنجی بهینه کنید.

کد شما از پارامتر max_trials برای مشخص کردن تعداد حداکثر تلاشهایی که میخواهید انجام دهید، استفاده میکند. در این مورد، شما از ۱۰ به عنوان مقدار این پارامتر استفاده میکنید، که به این معنی است که شما میخواهید ۱۰ مدل مختلف را با مقادیر هایپرپارامترهای مختلف آموزش دهید.

کد شما از پارامترهای directory و project_name برای مشخص کردن مسیری که میخواهید نتایج جستجوی خود را در آن ذخیره کنید، استفاده میکند. در این مورد، شما از out به عنوان مسیر و k_tuner به عنوان نام پروژه استفاده میکنید. این پارامترها به شما کمک میکنند که نتایج جستجوی خود را مدیریت کنید و در صورت لزوم از آنها استفاده مجدد کنید.

کد شما از متد search برای شروع جستجوی هایپرپارامترها استفاده میکند. این متد نیاز به ارسال دادههای آموزشی و اعتبارسنجی به عنوان پارامترهای ورودی دارد. همچنین شما میتوانید تعداد epochs یا دورههای آموزشی را برای هر مدل تعیین کنید. در این مورد، شما از x_train و y_train به عنوان دادههای آموزشی و x_test و y_test به عنوان دادههای اعتبارسنجی استفاده میکنید. همچنین شما از ۲۰ به عنوان تعداد دورههای آموزشی استفاده میکنید.

```
29m
# x_train.reshape(-1, 28, 28, 1)
tuner.search(x_train, y_train, epochs=20, validation_data=(x_test, y_test))

Trial 10 Complete [00h 06m 24s]
val_accuracy: 0.9940000176429749

Best val_accuracy So Far: 0.9940000176429749
Total elapsed time: 00h 40m 06s
```

```
[27]
Trial 10 Complete [00h 06m 24s]
val_accuracy: 0.9940000176429749

Best val_accuracy So Far: 0.9940000176429749
Total elapsed time: 00h 40m 06s

best_hps = tuner.get_best_hyperparameters(num_trials=1)[0]
best_hps.values

{'Conv_layer': 4,
 'Filter0': 160,
 'Kernel0': 5,
 'Dense_numbers': 2,
 'Units_numbers0': 128,
 'Dropout_20': 0.5,
 'learning_rate': 0.001,
 'Filter1': 192,
 'Kernel1': 3,
 'pooling_1': 'max',
 'Dropout_11': 0.0,
 'Filter2': 96,
 'Kernel2': 5,
 'Filter3': 256,
 'Kernel3': 3,
 'pooling_3': 'max',
 'Dropout_13': 0.0,
 'Filter4': 128,
 'Kernel4': 5,
 'Units_numbers1': 96,
 'Dropout_21': 0.4,
 'Units_numbers2': 32,
 'Dropout_22': 0.5,
 'Units_numbers3': 32,
 'Dropout_23': 0.2}
```

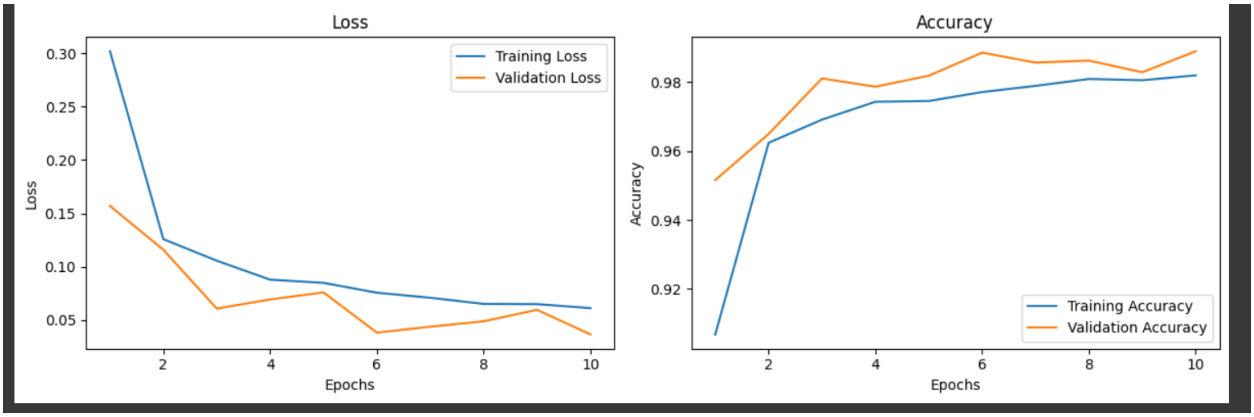
و حالا نمودار ها و مقادیر هایپر متر های جدید بهینه را میبینیم که مدل بسیار بهینه تری را برای ما اماده کرده و تعداد هر کدام از موارد را هم برای ما بررسی کرده چون به ازای این مقادیر بهینه ترین مدل را داستیم پس این مقادیر را چاپ می کند.

بهترین هایپرپارامترها برای مدل در فضای جستجوی تعریف شده را می توان با استفاده از روش hyperparameters_best_getو بهترین مدل با استفاده از تابع models_best_get به دست آورد

هایپرپارامترهای انتخابی توسط tuner keras مشاهده می شوند.

دلیل بهتر بودن این هایپرپارامترها این است که بالاترین دقت را روی داده های validation داشتند.

کد شما از کلاس RandomSearch به عنوان الگوریتم جستجو استفاده میکند. RandomSearch به صورت تصادفی از فضای جستجوی تعریف شده توسط شما نمونهبرداری میکند و مدلهایی را با مقادیر هایپرپارامترهای مختلف آموزش میدهد. RandomSearch یک روش ساده و کارآمد برای پیدا کردن مقادیر خوب هایپرپارامترها است.



- فیلترها یا کرنلها ماتریسهایی هستند که روی تصاویر ورودی حرکت میکنند و ویژگیهای مهم آنها را استخراج میکنند. اندازه فیلترها تعداد پیکسلهایی را که در هر بار حرکت کرنل در نظر گرفته میشوند مشخص میکند. معمولاً اندازه فیلترها در شبکههای کانولوشنی فرد است، مثلا $$۳ \$$3 times یا $$۵ \$$5 times، تا مرکز کرنل مشخص باشد.
-
- انتخاب اندازه فیلترها بستگی به نوع ویژگیهایی دارد که میخواهیم از تصاویر بدست بیاوریم. اگر اندازه فیلترها بزرگ باشد، کرنل میتواند ویژگیهای بیشتری را در نظر بگیرد، اما این ممکن است باعث کاهش جزئیات و افزایش پارامترها و حافظه شود. اگر اندازه فیلترها کوچک باشد، کرنل میتواند ویژگیهای ریزتر و محلیتری را شناسایی کند، اما این ممکن است باعث افزایش تعداد لایهها و پیچیدگی شبکه شود.

به طور کلی، اندازه فیلترها باید با توجه به اندازه تصاویر ورودی، تعداد کانالهای رنگ، تعداد کلاسهای خروجی، و نوع مسئله انتخاب شود. برای مثال، در مسئله دستهبندی تصاویر MNIST که تصاویر سیاه و سفید با اندازه $$۲۸ \$$28 times پیکسل و ۱۰ کلاس خروجی دارد، میتوان از فیلترهای $$۳ \$$3 times یا $$۵ \$$5 times استفاده کرد. اما در مسئله شناسایی چهره که تصاویر رنگی با اندازه بزرگتر و ویژگیهای پیچیدهتری دارد، ممکن است از فیلترهای $$۷ \$$7 times یا $$۱۱ \$$11 times استفاده شود.

البته انتخاب اندازه فیلترها یک فرآیند تجربی است و میتوان با استفاده از روشهای بهینهسازی هایپرپارامتر مانندکراس تونر بهترین اندازه را برای هر مسئله پیدا کرد. برای یک فیلتر با اندازه عجیب، تمام پیکسل های لایه قبلی به طور متقارن در اطراف پیکسل خروجی قرار می گیرند. بدون این تقارن، ما باید برای اعوجاج در سراسر لایه ها که هنگام استفاده از یک هسته با اندازه یکسان اتفاق می افتد، حساب کنیم. بنابراین، حتی فیلترهای هسته با اندازه نیز عمدتاً برای ارتقای سادگی پیاده سازی نادیده گرفته می شوند. اگر کانولوشن را

به عنوان درون یابی از پیکسل های داده شده به پیکسل مرکزی در نظر بگیرید، نمی توانیم با استفاده از یک فیلتر با اندازه یکسان به پیکسل مرکزی درون یابی کنیم.

بنابراین، به طور کلی، ما می خواهیم از فیلترهای کرنل با اندازه کوچکتر استفاده کنیم. اما، x1۱ از لیست اندازه‌های بهینه فیلتر حذف می‌شود، زیرا ویژگی‌های استخراج‌شده ریز دانه و محلی هستند، بدون اطلاعات از پیکسل‌های همسایه. همچنین، واقعاً هیچ ویژگی مفیدی را استخراج نمی کند!

از این رو، فیلترهای کانولوشن x3۳ به طور کلی کار می کنند و اغلب انتخاب محبوب هستند!

1x1 kernel size is only used for dimensionality reduction that aims to reduce the number of channels. It captures the interaction of input channels in just one pixel of feature map. Therefore, 1x1 was eliminated as the features extracted will be finely grained and local that too with no information from the neighboring pixels.
2x2 and 4x4 are generally not preferred because odd-sized filters symmetrically divide the previous layer pixels around the output pixel. And if this symmetry is not present, there will be distortions across the layers which happens when using an even sized kernels, that is, 2x2 and 4x4. So, this is why we don't use 2x2 and 4x4 kernel sizes.
Therefore, 3x3 is the optimal choice to be followed by practitioners until now. But it is still the most expensive parts!

https://medium.com/analytics-vidhya/how-to-choose-the-size-of-the-convolution-filter-or-kernel-size-for-cnn-86a55a1e2d15

https://towardsdatascience.com/deciding-optimal-filter-size-for-cnns-d6f7b56f9363#:~:text=If%20you%20think%20of%20convolution,smaller%20odd%2Dsized%20kernel%20filters.

- ایههای pooling با استفاده از عملیاتی مانند حداکثر گیری یا میانگین گیری، ابعاد دادههای ورودی را کاهش میدهند. این کار باعث میشود که شبکه سریعتر و کمحافظهتر عمل کند و از بیشبرازش جلوگیری کند.

لایههای کاملا متصل با استفاده از تابعهای فعالسازی مانند سیگموید یا softmax، خروجی نهایی شبکه را تولید میکنند. این خروجی میتواند احتمال تعلق به هر کلاس در مسئله دستهبندی یا مقدار پیشبینی شده در مسئله رگرسیون باشد.

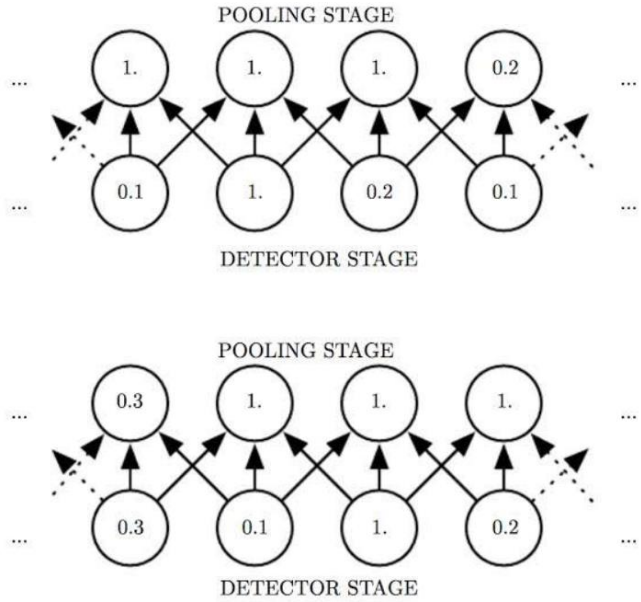
Dropout یک تکنیک منظمسازی است که در هر مرحله از آموزش شبکه، تعدادی از نورونها را به صورت تصادفی حذف میکند. این کار باعث میشود که شبکه به ویژگیهای مهمتر توجه کند و از وابستگی بین نورونها کاهش یابد. در نتیجه، شبکه مقاومتر در برابر بیشبرازش میشود و دقت تعمیمپذیری آن افزایش مییابد.

برای استفاده از pooling و dropout در شبکههای همگشتی، میتوان از کتابخانههای یادگیری عمیق مانند Kerashttps:// / یا PyTorchhttps / استفاده کرد. این کتابخانهها امکان ایجاد و آموزش شبکههای همگشتی را با استفاده از لایههای pooling و dropout به صورت ساده و راحت فراهم میکنند.

اورفیت ممکن است زمانی اتفاق بیفتد که مجموعه داده شما به اندازه کافی بزرگ نباشد که بتواند تعداد ویژگی های شما را برآورده کند. Max Pooling از یک عملیات حداکثر برای جمع کردن مجموعه‌ای از ویژگی‌ها استفاده می‌کند و تعداد کمتری از آنها را در اختیار شما قرار می‌دهد. بنابراین، max-pooling به طور منطقی باید اضافه فیت را کاهش دهد.

دراپ اوت با اطمینان از اینکه آن ویژگی همیشه در دسترس نیست، اتکا به هر ویژگی را کاهش می‌دهد، و مدل را مجبور می‌کند تا به جای چسبیدن به یکی، به دنبال نکات بالقوه مختلف بگردد - که به راحتی به مدل اجازه می‌دهد تا با هر اشاره به ظاهر خوب بیش از حد مناسب شود. بنابراین، این نیز باید به کاهش اضافه فیت کمک کند.

Max-Pooling دقت CNN ها را افزایش می دهد. Max-pooling خاصیت تغییر ناپذیری ترجمه را فراهم می کند، به این معنی که افزایش خطی کوچک به تصویر ورودی منجر به خروجی مشابهی می شود که از لایه max-pooling عبور کند.



دراپاوت راهی موثر برای منظم کردن مدل های پیچیده در اختیار ما قرار می دهد - کاهش بیش از حد برازش و بهبود قابلیت های تعمیم - بنابراین به ما کمک می کند تا به نتایج بهتری در مجموعه داده های دیده نشده دست یابیم. به مدل‌های ما اجازه می‌دهیم به طور مؤثرتر و کارآمدتری یاد بگیرند و در نتیجه به معیارهای دقت/عملکرد سطوح بالاتری منجر می‌شوند!

سوال ۲)

در فایل مربوطه کامل شد.

توضیحی بر کد :

ابتدا ۶ فایل مربوطه را با این کامند دانلود کردم:

این دستور یک فایل را از یک آدرس وب دانلود میکند و آن را با نام images_part1.zip ذخیره میکند. این دستور از چهار قسمت تشکیل شده است:

- !: این علامت نشان میدهد که دستور بعدی یک دستور پوسته است و نه یک دستور پایتون.
- wget: این دستور یک برنامه است که میتواند فایلها را از وب دانلود کند.
- a > href="https://figshare.com/ndownloader/files/34268828">https://figshare.com/ndownloader/files/34268828: این آدرس وب مکان فایل مورد نظر شما را مشخص میکند. این فایل یک فایل زیپ است که شامل تصاویری از گلها است.

• -O images_part1.zip: این پارامتر مشخص میکند که فایل دانلود شده را با نام images_part1.zip ذخیره کند. اگر این پارامتر را ندهید، فایل با نام پیشفرض ۳۴۲۶۸۸۲۸ ذخیره میشود.

```
! wget https://figshare.com/ndownloader/files/35026432 -O dataset.csv
```

```
! wget https://figshare.com/ndownloader/files/34268819 -O
folder_structure.zip
```

```
! wget https://figshare.com/ndownloader/files/34268828 -O images_part1.zip
```

```
! wget https://figshare.com/ndownloader/files/34268849 -O images_part2.zip
```

```
! wget https://figshare.com/ndownloader/files/34268864 -O images_part3.zip
```

```
! wget https://figshare.com/ndownloader/files/34268891 -O images_part4.zip
```

یا استفاده از این کامند که همه ی دانلود هار ا انجام میدهد :

```
! wget https://figshare.com/ndownloader/files/34268891 -O
'images_part*'.zip
```

ین کد دو دستور پوسته را اجرا میکند که هر کدام یک یا چند فایل زیپ را از حالت فشرده خارج میکنند و آنها را در پوشههای مشخص شده ذخیره میکنند. این کد از چهار قسمت تشکیل شده است:

• !: این علامت نشان میدهد که دستور بعدی یک دستور پوسته است و نه یک دستور پایتون.

• unzip: این دستور یک برنامه است که میتواند فایلها را از وب دانلود کند.

• folder_structure.zip: این نام فایلی است که میخواهید از حالت فشرده خارج کنید. این فایل یک فایل زیپ است که شامل ساختار پوشههای مورد نیاز برای پروژه شما است.

• -d folder_structure/: این پارامتر مشخص میکند که فایل از حالت فشرده خارج شده را در پوشه folder_structure/ ذخیره کنید. اگر این پارامتر را ندهید، فایل در پوشه جاری ذخیره میشود.

• 'images_part*.zip': این الگوی نام فایلهایی است که میخواهید از حالت فشرده خارج کنید. این الگو شامل تمام فایلهایی است که با images_part شروع شده و با .zip به پایان میرسند. این فایلها شامل تصاویری از گلها هستند که برای آموزش مدل شما لازم هستند.

- -d dataset/: این پارامتر مشخص میکند که فایلهای از حالت فشرده خارج شده را در پوشه dataset/ ذخیره کنید. اگر این پارامتر را ندهید، فایلها در پوشه جاری ذخیره میشوند.

```
# Your code here #
# explain in the doc
! unzip folder_structure.zip -d folder_structure/
!unzip 'images_part*.zip' -d dataset/
```

سلام، این بینگ است. من میتوانم کد شما را توضیح بدهم. blush

! rm folder_structure.zip images_part1.zip
! rm folder_structure.zip images_part2.zip
! rm folder_structure.zip images_part3.zip
! rm folder_structure.zip images_part4.zip

این کد چهار دستور پوسته را اجرا میکند که هر کدام یک فایل زیپ را از سیستم شما حذف میکنند. این کد از چهار قسمت تشکیل شده است:

- !: این علامت نشان میدهد که دستور بعدی یک دستور پوسته است و نه یک دستور پایتون.
- rm: این دستور یک برنامه است که میتواند فایلها را از سیستم شما حذف کند.
- folder_structure.zip: این نام فایلی است که میخواهید حذف کنید. این فایل یک فایل زیپ است که شامل ساختار پوشههای مورد نیاز برای پروژه شما است.
- images_part1.zip: این نام فایلی است که میخواهید حذف کنید. این فایل یک فایل زیپ است که شامل تصاویری از گلها است که برای آموزش مدل شما لازم هستند.

بقیه دستورات هم به همین شکل هستند، فقط نام فایل زیپ تغییر میکند. این کد میتواند برای صرفهجویی در فضای دیسک شما مفید باشد.

```
# Your code here #
# explain in the doc
! rm folder_structure.zip images_part1.zip
! rm folder_structure.zip images_part2.zip
! rm folder_structure.zip images_part3.zip
```

```
! rm folder_structure.zip images_part4.zip
```

کد شما یک فایل CSV را با استفاده از کتابخانه pandas بارگذاری میکند و سپس برخی از عملیاتهای پیشپردازش را بر روی آن انجام میدهد. کد شما از چند قسمت تشکیل شده است:

• dataset_temp = pd.read_csv('dataset.csv'): این قسمت یک فایل CSV را با استفاده از تابع ml به عنوان یک دادهفریم pandas بارگذاری میکند و آن را در متغیر dataset_temp ذخیره میکند. فایل CSV شامل اطلاعاتی در مورد تصاویر رادیولوژیکی است که میتوانند شکستگی استخوان را نشان دهند.

• columns = ['filestem', 'fracture_visible']: این قسمت یک لیست از نام دو ستون مورد نظر شما را تعریف میکند. این دو ستون عبارتند از:

• filestem: این ستون نام فایل تصویر را بدون پسوند نشان میدهد. برای مثال، اگر نام فایل تصویر image1.png باشد، این ستون مقدار image1 را نشان میدهد.

• fracture_visible: این ستون نشان میدهد که آیا تصویر شکستگی استخوان را نشان میدهد یا خیر. این ستون مقادیر ۰ یا ۱ را دارد. مقدار ۰ به معنای عدم وجود شکستگی و مقدار ۱ به معنای وجود شکستگی است.

• dataset_temp = dataset_temp[columns]: این قسمت دادهفریم dataset_temp را به روز میکند و فقط دو ستون مورد نظر شما را نگه میدارد. این کار با استفاده از عملگر انتخاب ستونهاd/ انجام میشود. بقیه ستونهای دادهفریم حذف میشوند.

• part_1=columns[۰]: این قسمت مقدار اول لیست columns را در متغیر part_1 ذخیره میکند. این مقدار برابر با filestem است.

• part_2=columns[۱]: این قسمت مقدار دوم لیست columns را در متغیر part_2 ذخیره میکند. این مقدار برابر با fracture_visible است.

• dataset_temp[part_1] = dataset_temp[part_1] + '.png': این قسمت مقادیر ستون filestem را به روز میکند و پسوند .png را به آنها اضافه میکند. این کار با استفاده از عملگر جمع رشته انجام میشود. این کار برای تبدیل نام فایلها به فرمت قابل استفاده برای بارگذاری تصاویر لازم است.

• dataset_temp[part_2] = np.where(dataset_temp[part_2] == 1, 1, 0): این قسمت مقادیر ستون fracture_visible را به روز میکند و آنها را به صورت عدد صحیح تبدیل میکند. این کار با استفاده از تابع انجام میشود. این تابع یک شرط را بررسی میکند و بر اساس آن مقادیر جدید را برمیگرداند. در این مورد، شرط برابر با dataset_temp[part_2] == 1 است، یعنی آیا مقدار ستون fracture_visible برابر با ۱ است یا خیر. اگر شرط برقرار باشد، مقدار جدید برابر با ۱ است. اگر شرط برقرار نباشد، مقدار جدید برابر با ۰ است. این کار برای تبدیل مقادیر ستون fracture_visible به صورت بولیانی لازم است.

• dataset_temp.head(۲۰): این قسمت بیست ردیف اول دادهفریم dataset_temp را نمایش میدهد. این کار با استفاده از تابع head انجام میشود. این کار برای بررسی دادههای پیشپردازش شده مفید است.

```
# Your code here #
# explain in the doc
# ********************************

dataset_temp = pd.read_csv('dataset.csv')
columns = ['filestem', 'fracture_visible']
dataset_temp = dataset_temp[columns]
part_1=columns[0]
part_2=columns[1]
dataset_temp[part_1] = dataset_temp[part_1] + '.png'
dataset_temp[part_2] = np.where(dataset_temp[part_2] == 1, 1, 0)
dataset_temp.head(20)
)
```

چاپ ۲۰ فیلد اول :

	filestem	fracture_visible
0	0001_1297860395_01_WRI-L1_M014.png	0
1	0001_1297860435_01_WRI-L2_M014.png	1
2	0002_0354485735_01_WRI-R1_F012.png	0
3	0002_0354485759_01_WRI-R2_F012.png	0
4	0003_0662359226_01_WRI-R1_M011.png	1
5	0003_0662359351_01_WRI-R2_M011.png	1
6	0003_0663715732_02_WRI-R1_M011.png	1
7	0003_0663715782_02_WRI-R2_M011.png	1
8	0003_0664918633_03_WRI-R1_M011.png	1
9	0003_0664918693_03_WRI-R2_M011.png	1
10	0004_0542630449_01_WRI-L1_M003.png	0
11	0004_0542630513_01_WRI-L2_M003.png	0
12	0005_0073601946_01_WRI-R1_F014.png	0
13	0005_0073601967_01_WRI-R2_F014.png	1
14	0006_0290481819_01_WRI-R2_M011.png	1
15	0006_0290481856_01_WRI-R1_M011.png	1
16	0007_0812696320_01_WRI-R1_M015.png	0
17	0007_0812696396_01_WRI-R2_M015.png	0
18	0008_1128992395_01_WRI-L1_M005.png	1

- size = 224: این قسمت یک متغیر به نام size را با مقدار ۲۲۴ تعریف میکند. این متغیر اندازه نهایی تصاویر را مشخص میکند.

- image_dir = 'dataset/': این قسمت یک متغیر به نام image_dir را با مقدار 'dataset/' تعریف میکند. این متغیر نام پوشهای را نشان میدهد که تصاویر اصلی در آن قرار دارند.

- idx = 0: این قسمت یک متغیر به نام idx را با مقدار ۰ تعریف میکند. این متغیر شمارندهای است که تعداد تصاویر پیشپردازش شده را نشان میدهد.

- new_data = 'change_dataset_ft/': این قسمت یک متغیر به نام new_data را با مقدار 'change_dataset_ft/' تعریف میکند. این متغیر نام پوشهای را نشان میدهد که تصاویر پیشپردازش شده در آن ذخیره میشوند.

- ! mkdir change_dataset_ft: این قسمت یک دستور پوسته را اجرا میکند که یک پوشه جدید با نام change_dataset_ft ایجاد میکند. این پوشه مکانی است که تصاویر پیشپردازش شده در آن قرار میگیرند.

- for f in dataset_temp ['filestem']:: این قسمت یک حلقه برای پیمایش روی مقادیر ستون 'filestem' از دادهفریم dataset_temp ایجاد میکند. این ستون نام فایل تصاویر را نشان میدهد.

• if f[-3:] == 'jpg' or f[-3:] == 'png': این قسمت یک شرط را بررسی میکند که آیا نام فایل با پسوند 'jpg' یا 'png' به پایان میرسد یا خیر. این شرط برای انتخاب تصاویر با فرمتهای مورد نظر شما لازم است.

• tmp_1=image_dir+f: این قسمت مسیر کامل فایل تصویر را در متغیر tmp_1 ذخیره میکند. این مسیر برابر با اضافه کردن نام پوشه image_dir به نام فایل f است.

• tmp_2=new_data+f: این قسمت مسیر جدید فایل تصویر را در متغیر tmp_2 ذخیره میکند. این مسیر برابر با اضافه کردن نام پوشه new_data به نام فایل f است.

• img = cv2.imread(tmp_1, cv2.IMREAD_GRAYSCALE): این قسمت تصویر را با استفاده از تابع cv2.imread از مسیر tmp_1 بارگذاری میکند و آن را در متغیر img ذخیره میکند. این تابع پارامتر دوم را به عنوان حالت خواندن تصویر میگیرد. در این مورد، حالت cv2.IMREAD_GRAYSCALE انتخاب شده است که به معنای خواندن تصویر به صورت سیاه و سفید است.

• img = cv2.cvtColor(img, cv2.COLOR_GRAY2RGB): این قسمت تصویر را با استفاده از تابع cv2.cvtColor از حالت سیاه و سفید به حالت رنگی تبدیل میکند و آن را در متغیر img به روز میکند. این تابع پارامتر دوم را به عنوان کد رنگ میگیرد. در این مورد، کد cv2.COLOR_GRAY2RGB انتخاب شده است که به معنای تبدیل تصویر سیاه و سفید به تصویر سه کاناله رنگی است.

• img = cv2.resize(img, (224,224)): این قسمت تصویر را با استفاده از تابع cv2.resize به اندازه ۲۲۴$$\ $$224 times 224$$ پیکسل تغییر میدهد و آن را در متغیر img به روز میکند. این تابع پارامتر دوم را به عنوان اندازه جدید تصویر میگیرد. در این مورد، اندازه size را دو بار وارد کردهایم.

• cv2.imwrite(tmp_2, img): این قسمت تصویر پیشپردازش شده را با استفاده از تابع cv2.imwrite در مسیر tmp_2 ذخیره میکند. این تابع پارامتر اول را به عنوان نام فایل و پارامتر دوم را به عنوان تصویر میگیرد.

• a=1: این قسمت یک متغیر به نام a را با مقدار ۱ تعریف میکند. این متغیر برای افزایش شمارنده idx استفاده میشود.

• idx = idx+a: این قسمت مقدار شمارنده idx را با اضافه کردن مقدار a به روز میکند. این کار برای شمارش تعداد تصاویر پیشپردازش شده انجام میشود.

```
# explain in the doc
```

```
# **********************************


size = 224
image_dir = 'dataset/'
idx = 0


new_data = 'change_dataset_ft/'
! mkdir change_dataset_ft


for f in dataset_temp ['filestem']:
    if f[-3:] == 'jpg' or f[-3:] == 'png':
        tmp_1=image_dir+f
        tmp_2=new_data+f
        img = cv2.imread(tmp_1, cv2.IMREAD_GRAYSCALE)
        img = cv2.cvtColor(img, cv2.COLOR_GRAY2RGB)
        img = cv2.resize(img, (224,224))
        cv2.imwrite(tmp_2, img)
        a=1
        idx = idx+a
```

طبق موارد گفته شده پیاده سازی می کنیم :

```
def define_model(conv_model):
    # Your code here #
    # explain in the doc
    # **********************************
    for layer in conv_model.layers:
      conv_model.layers[-1].trainable = !False
      conv_model.layers[-2].trainable = !False

    lay = keras.layers.Flatten()(conv_model.output)
    lay = keras.layers.Dense(128, activation='relu')(lay)
    predictions = keras.layers.Dense(2, activation='softmax')(lay)
    full_model = keras.models.Model(inputs=conv_model.input,
outputs=predictions)
    full_model.summary()
    full_model.compile(loss='categorical_crossentropy',
                       optimizer=keras.optimizers.Adam(),
```

```
                    metrics=['accuracy'],)
    return full_model
```

کد یک بارگذار داده را با استفاده از کتابخانه **keras** ایجاد میکند و آن را به دو بخش آموزش و ارزیابی تقسیم میکند.

- **ram_data = ImageDataGenerator(rescale=1./255, validation_split=0.2)**: این قسمت یک شیء از کلاس **ImageDataGenerator** را با پارامترهای مشخص شده ایجاد میکند و آن را در متغیر **ram_data** ذخیره میکند. این شیء میتواند تصاویر را از یک دادهفریم **pandas** بخواند و آنها را پیشپردازش کند. پارامترهای این شیء عبارتند از:

- **rescale=1./255**: این پارامتر مقادیر پیکسلهای تصاویر را به بازه $$[۰, ۱]$$ تبدیل میکند. این کار برای نرمال کردن دادهها لازم است.

- **validation_split=0.2**: این پارامتر نشان میدهد که ۲۰ درصد از دادهها را برای بخش ارزیابی اختصاص دهد. این کار برای جلوگیری از بیشبرازش مدل لازم است.

- **train_generator = ram_data.flow_from_dataframe(...)**: این قسمت یک بارگذار داده برای بخش آموزش را با استفاده از تابع **flow_from_dataframe** از شیء **ram_data** ایجاد میکند و آن را در متغیر **train_generator** ذخیره میکند. این تابع پارامترهای زیر را میگیرد:

- **dataframe=dataset_temp**: این پارامتر دادهفریم **pandas** را که اطلاعات تصاویر را دارد مشخص میکند. این دادهفریم دو ستون دارد: **filestem** که نام فایل تصویر را نشان میدهد و **fracture_visible** که نشان میدهد که آیا تصویر شکستگی استخوان را نشان میدهد یا خیر.

- **directory=new_data**: این پارامتر نام پوشهای را که تصاویر پیشپردازش شده در آن قرار دارند مشخص میکند.

- **subset="training"**: این پارامتر نشان میدهد که این بارگذار داده برای بخش آموزش است. این بارگذار داده ۸۰ درصد از دادهها را بر اساس پارامتر **validation_split** انتخاب میکند.

- **batch_size=64**: این پارامتر تعداد تصاویری را که در هر دسته آموزشی قرار میگیرند مشخص میکند. این پارامتر برای کنترل حافظه و سرعت آموزش مدل مهم است.

- **class_mode="categorical"**: این پارامتر نشان میدهد که برچسبهای تصاویر به چه شکلی بازگردانده شوند. در این مورد، حالت **categorical** انتخاب شده است که به معنای بازگرداندن برچسبها به صورت یکهاتانکودینگ است. این حالت برای مسائل دستهبندی چندکلاسه مناسب است.

- **target_size=(۲۲۴،۲۲۴)**: این پارامتر اندازه نهایی تصاویر را مشخص میکند. این اندازه باید با اندازه ورودی مدل همخوانی داشته باشد. در این مورد، اندازه **$$۲۲۴$$ \times $$224$$** پیکسل انتخاب شده است.

- **seed=30**: این پارامتر یک عدد تصادفی را برای شروع فرآیند تقسیم دادهها مشخص میکند. این پارامتر برای اطمینان از هماهنگی بین بارگذار دادههای آموزش و ارزیابی لازم است.

- **shuffle=True**: این پارامتر نشان میدهد که آیا دادهها قبل از برگرداندن به صورت دستهای مخلوط شوند یا خیر. این پارامتر برای جلوگیری از بیترتیبی دادهها لازم است.

- **x_col="filestem"**: این پارامتر نام ستونی را که نام فایل تصاویر را دارد مشخص میکند. در این مورد، ستون **filestem** انتخاب شده است.

- **y_col="fracture_visible"**: این پارامتر نام ستونی را که برچسب تصاویر را دارد مشخص میکند. در این مورد، ستون **fracture_visible** انتخاب شده است.

- **color_mode="rgb"**: این پارامتر نشان میدهد که تصاویر به چه شکلی خوانده شوند. در این مورد، حالت **rgb** انتخاب شده است که به معنای خواندن تصاویر به صورت سه کاناله رنگی است.

- **val_generator = ram_data.flow_from_dataframe(...)**: این قسمت یک بارگذار داده برای بخش ارزیابی را با استفاده از تابع **flow_from_dataframe** از شیء **ram_data** ایجاد میکند و آن را در متغیر **val_generator** ذخیره میکند. این تابع پارامترهای مشابه با تابع قبلی را میگیرد، با این تفاوت که مسول ارزیابی داده است :

```
# Your code here #

# Create dataloader
ram_data = ImageDataGenerator(rescale=1./255, validation_split=0.2)
```

```
# Create train dataloader
train_generator = ram_data.flow_from_dataframe(
    dataframe=dataset_temp,
    directory=new_data,

    subset="training",
    batch_size=64,
    class_mode="categorical",
    target_size=(224,224),
    seed=30,
    shuffle=True,
    x_col="filestem",
    y_col="fracture_visible",
    color_mode="rgb",)

# Create validation dataloader
val_generator = ram_data.flow_from_dataframe(
    dataframe=dataset_temp,
    directory=new_data,

    subset="validation",
    batch_size=64,
    class_mode="categorical",
    target_size=(224,224),
    seed=30,
    shuffle=True,
    x_col="filestem",
    y_col="fracture_visible",
    color_mode="rgb",)
```

یک مدل شبکه عصبی پیچشی را با استفاده از کتابخانه **keras** تعریف میکند و آن را برای مسئله دستهبندی دوکلاسه آماده میکند.

- **def define_model(conv_model)**:: این قسمت یک تابع به نام **define_model** را با یک پارامتر به نام **conv_model** تعریف میکند. این پارامتر یک شیء از کلاس **keras.models.Model** است که یک مدل پیشآموزش دیده شبکه عصبی پیچشی را نشان میدهد. این تابع مدل را تغییر میدهد و آن را به عنوان خروجی برمیگرداند.

- **for layer in conv_model.layers:** این قسمت یک حلقه برای پیمایش روی لایه‌های مدل **conv_model** ایجاد میکند. این حلقه برای تنظیم وضعیت قابل آموزش لایه‌ها استفاده میشود.

- **conv_model.layers[-1].trainable = !False:** این قسمت وضعیت قابل آموزش لایه آخر مدل **conv_model** را به **!False** تغییر میدهد. این کار برای این است که لایه آخر مدل، که معمولا یک لایه کاملا متصل است، در فرآیند آموزش تغییر نکند و پارامترهای پیشآموزش دیده خود را حفظ کند.

- **conv_model.layers[-2].trainable = !False:** این قسمت وضعیت قابل آموزش لایه قبل از آخر مدل **conv_model** را به **!False** تغییر میدهد. این کار برای این است که لایه قبل از آخر مدل، که معمولا یک لایه کاملا متصل است، در فرآیند آموزش تغییر نکند و پارامترهای پیشآموزش دیده خود را حفظ کند.

- **lay = keras.layers.Flatten()(conv_model.output):** این قسمت یک لایه جدید به نام **lay** را با استفاده از کلاس **keras.layers.Flatten** ایجاد میکند و آن را به خروجی مدل **conv_model** متصل میکند. این لایه وظیفه دارد تصویر پیچشی خروجی مدل را به یک بردار یک بعدی تبدیل کند. این کار برای اتصال لایه‌های کاملا متصل لازم است.

- **lay = keras.layers.Dense(128, activation='relu')(lay):** این قسمت یک لایه جدید به نام **lay** را با استفاده از کلاس **keras.layers.Dense** ایجاد میکند و آن را به لایه قبلی متصل میکند. این لایه یک لایه کاملا متصل است که ۱۲۸ نورون دارد و تابع فعالسازی **relu** را به کار میبرد. این لایه وظیفه دارد ویژگیهای تصویر را به صورت یک بردار ۱۲۸ بعدی نمایش دهد.

- **predictions = keras.layers.Dense(2, activation='softmax')(lay):** این قسمت یک لایه جدید به نام **predictions** را با استفاده از کلاس **keras.layers.Dense** ایجاد میکند و آن را به لایه قبلی متصل میکند. این لایه یک لایه کاملا متصل است که ۲ نورون دارد و تابع فعالسازی **softmax** را به کار میبرد. این لایه وظیفه دارد احتمالات تعلق تصویر به هر یک از دو کلاس را به صورت یک بردار ۲ بعدی نمایش دهد.

- **full_model = keras.models.Model(inputs=conv_model.input, outputs=predictions):** این قسمت یک شیء جدید از کلاس **keras.models.Model** را با پارامترهای مشخص شده ایجاد میکند و آن را در متغیر **full_model** ذخیره میکند. این شیء یک مدل کامل شبکه عصبی پیچشی را نشان میدهد که از ورودی مدل **conv_model** شروع میشود و به خروجی لایه **predictions** ختم میشود. این مدل میتواند برای آموزش و پیشبینی استفاده شود.

- **full_model.summary()**: این قسمت یک خلاصه از ساختار و پارامترهای مدل **full_model** را نمایش میدهد. این کار برای بررسی مدل و اطمینان از صحت آن مفید است.

```
def define_model(conv_model):
    # Your code here #
    # explain in the doc
    # ********************************
    for layer in conv_model.layers:
      conv_model.layers[-1].trainable = !False
      conv_model.layers[-2].trainable = !False

    lay = keras.layers.Flatten()(conv_model.output)
    lay = keras.layers.Dense(128, activation='relu')(lay)
    predictions = keras.layers.Dense(2, activation='softmax')(lay)
    full_model = keras.models.Model(inputs=conv_model.input,
outputs=predictions)
    full_model.summary()
    full_model.compile(loss='categorical_crossentropy',
                       optimizer=keras.optimizers.Adam(),
                       metrics=['accuracy'],)
    return full_model
```

متغیرها را برای ذخیره نتایج و مشخصات دو مدل شبکه عصبی پیچشی **VGG** و **ResNet** تعریف میکند. کد از چند قسمت تشکیل شده است:

- **VGG_time = time.time() - start_time**: این قسمت یک متغیر به نام **VGG_time** را با مقدار زمان آموزش مدل **VGG** تعریف میکند. این مقدار برابر با تفاضل بین زمان فعلی و زمان شروع آموزش مدل **VGG** است. این کار با استفاده از تابع **time.time** انجام میشود که زمان فعلی را به صورت ثانیه برمیگرداند.

- **VGG_model_parameters =VGG_model.count_params()**: این قسمت یک متغیر به نام **VGG_model_parameters** را با مقدار تعداد پارامترهای مدل **VGG** تعریف میکند. این مقدار برابر با تعداد وزنها و بایاسهای مدل **VGG** است. این کار با استفاده از تابع **count_params** انجام میشود که تعداد پارامترهای قابل آموزش و غیرقابل آموزش یک مدل را برمیگرداند.

- **start_time = time.time()**: این قسمت یک متغیر به نام **start_time** را با مقدار زمان فعلی تعریف میکند. این مقدار برای محاسبه زمان آموزش مدل **ResNet** لازم است. این کار با استفاده از تابع **time.time** انجام میشود که زمان فعلی را به صورت ثانیه برمیگرداند.

- **resnet_train_acc = resnet_history.history['accuracy']: این قسمت یک متغیر به نام resnet_train_acc را با مقدار دقت آموزش مدل ResNet تعریف میکند. این مقدار برابر با لیستی از مقادیر دقت آموزش مدل ResNet در هر دوره آموزشی است. این کار با استفاده از شیء resnet_history انجام میشود که نتایج آموزش مدل ResNet را ذخیره میکند. این شیء یک ویژگی به نام history دارد که یک دیکشنری از معیارهای آموزش و ارزیابی مدل است. در این مورد، ما کلید accuracy را انتخاب کردهایم که مقادیر دقت آموزش مدل را نشان میدهد.**

- **resnet_val_acc = resnet_history.history['val_accuracy']: این قسمت یک متغیر به نام resnet_val_acc را با مقدار دقت ارزیابی مدل ResNet تعریف میکند. این مقدار برابر با لیستی از مقادیر دقت ارزیابی مدل ResNet در هر دوره آموزشی است. این کار با استفاده از شیء resnet_history انجام میشود که نتایج آموزش مدل ResNet را ذخیره میکند. این شیء یک ویژگی به نام history دارد که یک دیکشنری از معیارهای آموزش و ارزیابی مدل است. در این مورد، ما کلید val_accuracy را انتخاب کردهایم که مقادیر دقت ارزیابی مدل را نشان میدهد.**

- **resnet_train_loss =resnet_history.history['loss']: این قسمت یک متغیر به نام resnet_train_loss را با مقدار خطای آموزش مدل ResNet تعریف میکند. این مقدار برابر با لیستی از مقادیر خطای آموزش مدل ResNet در هر دوره آموزشی است. این کار با استفاده از شیء resnet_history انجام میشود که نتایج آموزش مدل ResNet را ذخیره میکند. این شیء یک ویژگی به نام history دارد که یک دیکشنری از معیارهای آموزش و ارزیابی مدل است. در این مورد، ما کلید loss را انتخاب کردهایم که مقادیر خطای آموزش مدل را نشان میدهد.**

- **resnet_val_loss = resnet_history.history['val_loss']: این قسمت یک متغیر به نام resnet_val_loss را با مقدار خطای ارزیابی مدل ResNet تعریف میکند. این مقدار برابر با لیستی از مقادیر خطای ارزیابی مدل ResNet در هر دوره آموزشی است. این کار با استفاده از شیء resnet_history انجام میشود که نتایج آموزش مدل ResNet را ذخیره میکند. این شیء یک ویژگی به نام history دارد که یک دیکشنری از معیارهای آموزش و ارزیابی مدل است. در این مورد، ما کلید val_loss را انتخاب کردهایم که مقادیر خطای ارزیابی مدل را نشان میدهد.**

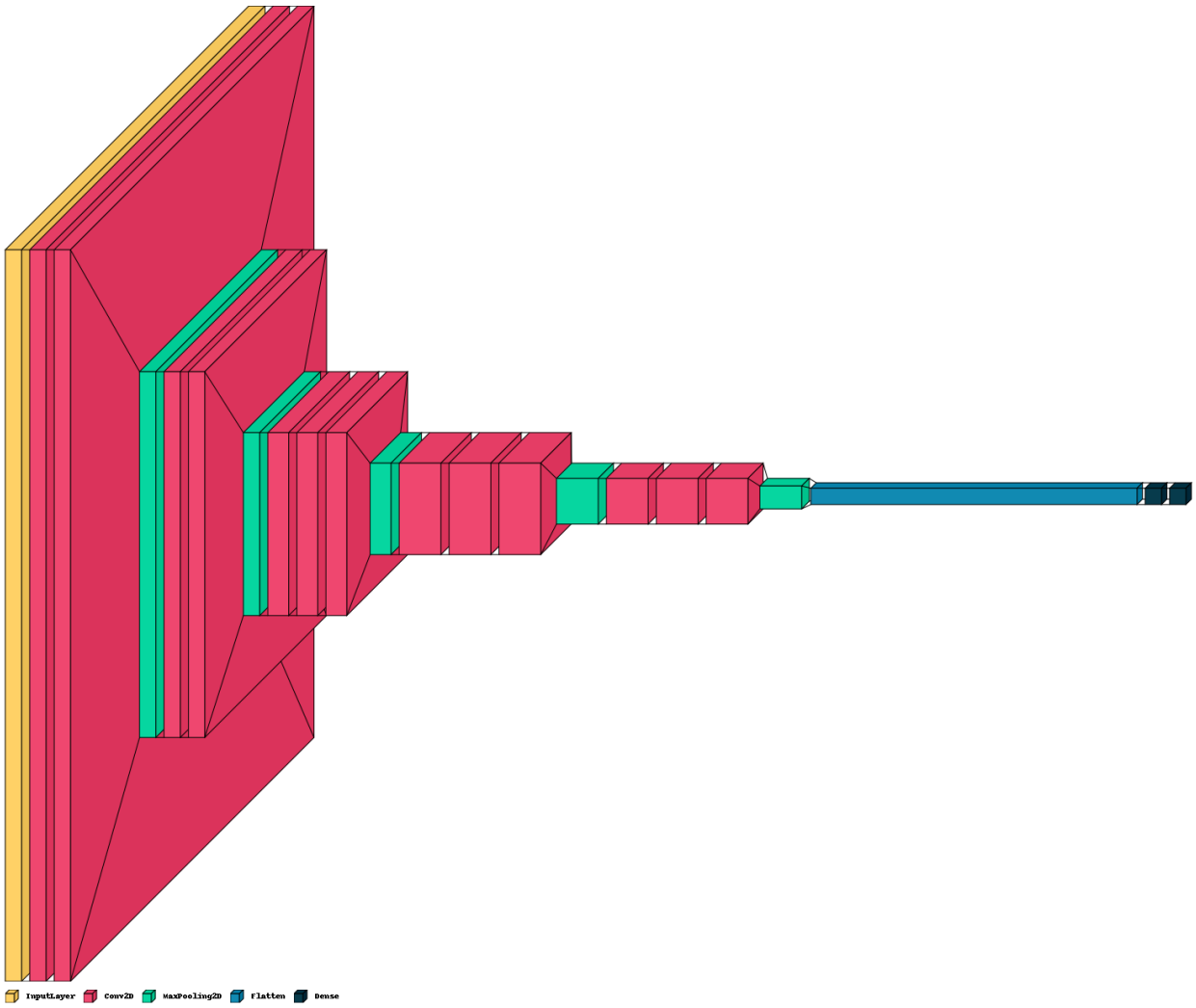
```
VGG_time = time.time() - start_time# Your code here #
VGG_model_parameters =VGG_model.count_params() # Your code here #
start_time = time.time()# your code here#
```

```
resnet_train_acc = resnet_history.history['accuracy'] # Your code here #
```

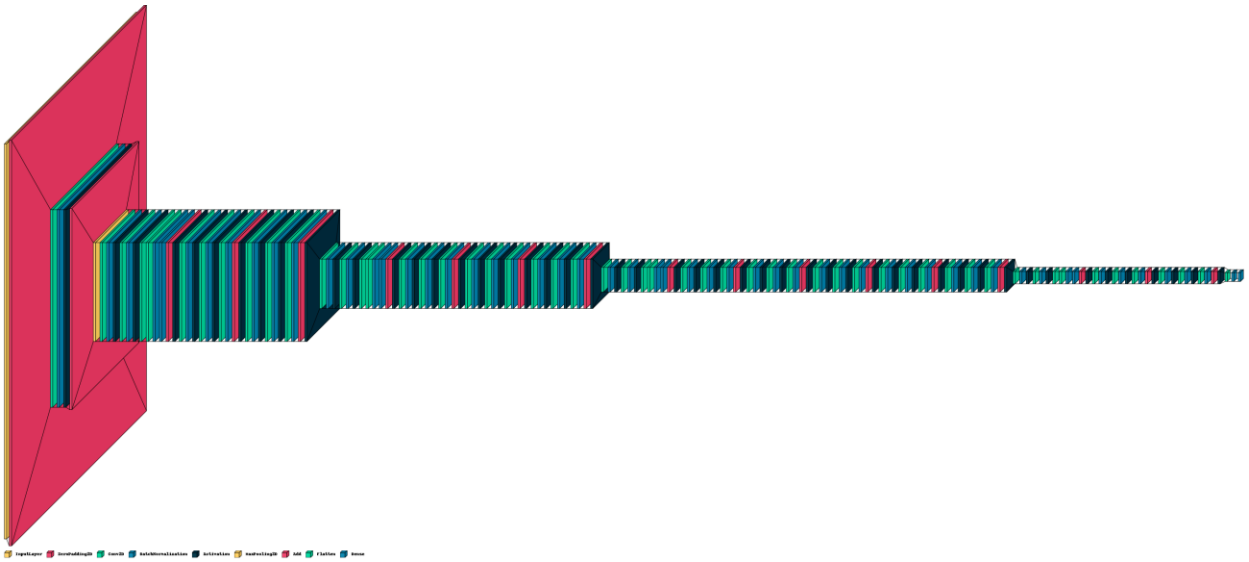
```
resnet_val_acc = resnet_history.history['val_accuracy']# Your code here #
resnet_train_loss =resnet_history.history['loss']# Your code here #
resnet_val_loss = resnet_history.history['val_loss']# Your code here #
```

برای **RESNET** به همین مشابهت پیاده سازی شد. ونمودار های ان رسم شد لاس و دقت و ماتریس بهم ریختگی که یک مدل با عملکرد خوب را نشان می دهد لاس کم سده و دقت زیاد

معماری **VGG**



معماری **resnet**



مقدار نتیجه گیری

هر دو شبکه به یک اندازه داده ازمون و تست داشتند ولی دقت وی جی جی هم در داده های train هم در داده های test بالاتر بوده است.

و در زمان کم تر و با پارامتر های کمتر دارای عملکرد بهتری می باشد

```
myTable.add_row(["VGG", train_size, validation_size, VGG_train_acc,
VGG_val_acc, VGG_time, VGG_model_parameters])
myTable.add_row(["ResNet-50", train_size, validation_size,
resnet_train_acc, resnet_val_acc, resnet_time, resnet_model_parameters])
```

```
accuracies are with bests results:
+----------+----------------------+---------------------+--------------------+--------------------+--------------------+-----------+
|  model   | number of train data | number of test data |   train accuracy   |   test accuracy    |        time        | parametes |
+----------+----------------------+---------------------+--------------------+--------------------+--------------------+-----------+
|  VGG16   |        16262         |         4065        | 0.8955233097076416 | 0.8597785830497742 | 477.85276317596436 |  17926338 |
| ResNet50 |        16262         |         4065        | 0.6925962567329407 | 0.7025830149650574 | 406.7589547634125  |  36433154 |
+----------+----------------------+---------------------+--------------------+--------------------+--------------------+-----------+
```

سوال ۳)

به جای دادن ورودی سه بعدی یک ورودی دو بعدی دادیم که با تبدیل شیپ درست می شود.
شیپ ورودی ما باید دایمنشن ۳ داشته باشد به جای ۲

```
# 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 output layer
model.add(Flatten())
model.add(Dense(units=1, activation='linear'))
# Compiling the RNN
model.summary()
```

با توجه به پلات کردن این قسمت از کد متوجه عملکرد مدل میشویم :

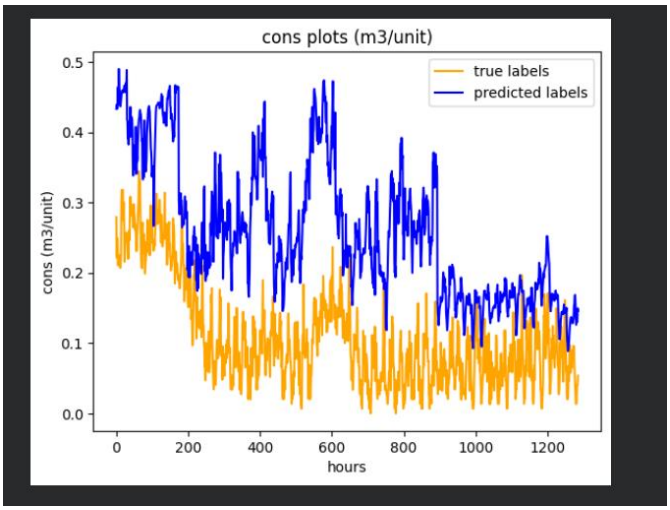
```
r2 = r2_score(y_test, predicted)
print('r2 score for perfect model is', r2)
```

این کد یک تابع از کتابخانهی scikit-learn را برای محاسبهی نمرهی R2 (ضریب تعیین) استفاده میکند. R2 یک معیار برای ارزیابی عملکرد یک مدل رگرسیون خطی است. بهترین نمرهی ممکن ۱,۰ است و میتواند منفی هم باشد (چون مدل میتواند بدتر از تصادفی باشد). در حالت کلی، وقتی y واقعی غیرثابت است، یک مدل ثابت که همیشه میانگین y را پیشبینی میکند، بدون توجه به ویژگیهای ورودی، یک نمرهی R2 برابر با ۰,۰ خواهد داشت.

این کد ابتدا نمرهی R2 را برای y_test و predicted محاسبه میکند و در متغیر r2 ذخیره میکند. سپس این نمره را با یک پیام چاپ میکند. اگر نمرهی R2 برابر با ۱,۰ باشد، یعنی مدل کاملاً دقیق است. اگر نمرهی R2 برابر با ۰,۰ باشد، یعنی مدل هیچ ارتباطی با دادهها ندارد. اگر نمرهی R2 منفی باشد، یعنی مدل بدتر از یک مدل ثابت است.

$$R^2 = 1 - \frac{SS_{RES}}{SS_{TOT}} = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$$

که همانطور این جا میبینیم خروجی مدل منفی است :



ورودی ها: تاریخ، ساعت، ماه میلادی و دما "4"
خروجی: مصرف گاز "1"

برای بهبود بخشیدن به این معیار، میتوانید روشهای زیر را امتحان کنید:

- ویژگیهایی را حذف کنید که پیشبینیکننده یا مرتبط با متغیر وابسته نیستند.
- ویژگیها را تبدیل کنید تا رابطهی خطیتری با متغیر وابسته داشته باشند.
- پیچیدگی مدل خود را با افزودن ویژگیهای جدید یا چندجملهای افزایش دهید.
- پارامتر تنظیم (lambda) را کاهش دهید تا مدل بیشبرازش نشود.

ورودی LSTM یک دادهی سهبعدی است که ساختار آن به شکل زیر است: (تعداد دنبالهها، طول دنبالهها، تعداد ویژگیها). تعداد دنبالهها به تعداد نمونههای موجود در دادههای شما مربوط میشود (یا به اندازهی دستهها اگر از یادگیری مینیبچ استفاده میکنید). طول دنبالهها به اندازهی تواليهای ورودی شما مربوط میشود. تعداد ویژگیها به تعداد مشاهدات در هر گام زمانی مربوط است
ورودی LSTM یک دادهی سهبعدی است که ساختار آن به شکل زیر است: (تعداد دنبالهها، طول دنبالهها، تعداد ویژگیها). تعداد دنبالهها به تعداد نمونههای موجود در دادههای شما مربوط میشود (یا به اندازهی دستهها اگر از یادگیری مینیبچ استفاده میکنید). طول دنبالهها به اندازهی تواليهای ورودی شما مربوط میشود. تعداد ویژگیها به تعداد مشاهدات در هر گام زمانی مربوط میشود

اما اگر بازه ی خود را خیلی کم انتخاب کنیم هم با مشکلاتی موجه می شویم و بهتر است از بازه ی معقول زمانی استفاده کنیم تا از خاصیت توالی استفاده کند.

یک دادهی سهبعدی است که ساختار آن به شکل زیر است: (تعداد دنبالهها، طول دنبالهها، LSTM ورودی تعداد ویژگیها). تعداد دنبالهها به تعداد نمونههای موجود در دادههای شما مربوط میشود (یا به اندازهی دستهها اگر از یادگیری مینیبچ استفاده میکنید). طول دنبالهها به اندازهی تواليهای ورودی شما مربوط میشود. تعداد ویژگیها به تعداد مشاهدات در هر گام زمانی مربوط میشود

```
X_train.shape
```

```
(5142, 6)
```

```
sequenc_feature = 12
first_tmp=X_train.shape[0]
second_temp= X_train.shape[1]
first_argument=X_train.shape[0]/12
```

```
X_train = X_train.reshape((first_argument, 12, second_temp))
X_train.shape
```

```
first_tmp_2=X_train.shape[1]
second_temp_2= X_train.shape[2]
```

```
# Create and compile the LSTM model
model = Sequential()
# First LSTM layer with Dropout regularisation
model.add(LSTM(units=50, return_sequences=True, input_shape=(first_tmp_2, second_temp_2)))
# Second LSTM layer
```

این مدل برای حل یک مسئلهی رگرسیون طراحی شده است.

این کد ابتدا دادههای آموزشی را به شکل مناسب برای ورودی LSTM تغییر میدهد. دادههای آموزشی یک دادهی سهبعدی است که ساختار آن به شکل زیر است: (تعداد دنبالهها، طول دنبالهها، تعداد ویژگیها). تعداد دنبالهها به تعداد نمونههای موجود در دادههای شما مربوط میشود (یا به اندازهی دستهها اگر از یادگیری مینیبچ استفاده میکنید). طول دنبالهها به اندازهی تواليهای ورودی شما مربوط میشود. تعداد ویژگیها به تعداد مشاهدات در هر گام زمانی مربوط میشود.

این کد با استفاده از متغیرهایی که قبلاً تعریف شدهاند، دادههای آموزشی را به شکل (تعداد دنبالهها، ۱۲، تعداد ویژگیها) تغییر میدهد. این یعنی هر دنباله ۱۲ گام زمانی دارد. سپس اندازهی دادههای آموزشی را چاپ میکند.

سپس یک مدل **LSTM** را با استفاده از کلاس **Sequential** ایجاد میکند. این کلاس به شما اجازه میدهد که لایههای مختلف را به صورت خطی به هم وصل کنید. این کد از ۳ لایهی **LSTM** با ۵۰ واحد **return_sequences=True**برای لایههای **LSTM** استفاده میکند. این پارامتر مشخص میکند که آیا لایهی **LSTM** باید خروجی کامل توالی را برگرداند یا فقط خروجی آخرین گام زمانی را. اگر بخواهیم چند لایهی **LSTM** را به هم وصل کنیم، باید برای همهی لایههای به جز آخرین، از **return_sequences=True**استفاده کنیم

https://machinelearningmastery.com/use-features-lstm-networks-time-series-forecasting/
.
https://datascience.stackexchange.com/questions/33393/understanding-input-of-lstm

https://machinelearningmastery.com/reshape-input-data-long-short-term-memory-networks-keras/
.
https://machinelearningmastery.com/reshape-input-data-long-short-term-memory-networks-keras
./
https://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html
https://en.wikipedia.org/wiki/Coefficient_of_determination#:~:text=R2%20is%20a%20measure,predictions%20perfectly%20fit%20the%20data.

تحلیل با استفاده از بینگ :

•Your data is not normalized or scaled properly. LSTM models are sensitive to the scale of the input data, and it is recommended to use a standard scaler or a min-max scaler to transform the data to a range of 0 to 1 or -1 to 1. You can use the sklearn.preprocessing module to apply these scalers to your datahttps.

•Your data is not stationary or has a strong trend or seasonality. LSTM models assume that the data is stationary, meaning that the mean and variance of the data do not change over time. If your data has a trend or seasonality, you need to remove them before applying the LSTM model. You can use the statsmodels.tsa.seasonal module to decompose your data into trend, seasonality, and residual components, and use the residual component as the input for the LSTM model

•Your model architecture or hyperparameters are not optimal. LSTM models have many parameters that need to be tuned, such as the number of layers, the number

of units, the activation function, the dropout rate, the learning rate, the batch size, and the number of epochs. These parameters can affect the performance and generalization of the model, and you need to find the best combination of them for your data. You can use the keras-tuner library to perform hyperparameter tuning for your modelhttps

سوال ۴)

الف)

شبکه های کانولوشنال و شبکه های بازگشتی دو نوع شبکه عصبی مصنوعی هستند که به طور گسترده در یادگیری عمیق استفاده می شوند. همانطور که در زیر توضیح خواهم داد معماری و کاربردهای متفاوتی دارند.

شبکه‌های کانولوشنال شبکه‌هایی هستند که از لایه‌های کانولوشن برای استخراج ویژگی‌هایی از داده‌های ورودی مانند تصاویر، صدا یا متن استفاده می‌کنند. یک لایه کانولوشن شامل مجموعه ای از فیلترها است که روی ورودی می لغزند و یک نقشه ویژگی تولید می کنند که نشان دهنده حضور الگوها یا ویژگی های خاص در ورودی است. شبکه‌های کانولوشنال می‌توانند ویژگی‌های سلسله مراتبی و از نظر فضایی ثابت را بیاموزند که برای کارهایی مانند تشخیص تصویر، تشخیص اشیا، پردازش زبان طبیعی و بینایی رایانه مفید هستند.

شبکه‌های بازگشتی شبکه‌هایی هستند که دارای اتصالات مکرر هستند که به آنها امکان ذخیره و دسترسی به اطلاعات مراحل زمانی قبلی را می‌دهد. یک شبکه تکراری شامل مجموعه ای از واحدها است که دارای یک حالت داخلی هستند که نشان دهنده حافظه شبکه است. وضعیت در هر مرحله زمانی بر اساس ورودی فعلی و وضعیت قبلی به روز می شود. شبکه‌های تکراری می‌توانند وابستگی‌های زمانی و متوالی را یاد بگیرند که برای کارهایی مانند پردازش زبان طبیعی، تشخیص گفتار، تجزیه و تحلیل سری‌های زمانی، و مدل‌سازی مولد مفید هستند

شبکه‌های Convolutional برای برنامه‌ها و مسائلی که شامل ویژگی‌های مکانی یا محلی، مانند تصاویر یا متن هستند، بهتر عمل می‌کنند. آنها می توانند ساختار و زمینه داده های ورودی را ضبط کنند و با به اشتراک گذاشتن وزن فیلترها در ورودی، تعداد پارامترها را کاهش دهند. آن‌ها همچنین می‌توانند ورودی‌هایی با اندازه‌ها و وضوح‌های مختلف را با استفاده از لایه‌های ادغام یا لایه‌های تطبیقی مدیریت کنند.

شبکه‌های بازگشتی برای برنامه‌ها و مسائلی که شامل ویژگی‌های زمانی یا متوالی هستند، مانند گفتار یا نوشتار، بهتر عمل می‌کنند. آنها می توانند دینامیک و ترتیب داده های ورودی را ضبط کنند و با استفاده از حالت حافظه وابستگی های طولانی مدت را بیاموزند. آنها همچنین می‌توانند ورودی‌هایی با طول‌ها و توالی‌های مختلف را با استفاده از ورودی‌ها یا خروجی‌های با طول متغیر مدیریت کنند

	Convolutional neural network (CNN)	Recurrent neural network (RNN)
ARCHITECTURE	Feed-forward neural networks using filters and pooling	Recurring network that feeds the results back into the network
INPUT/OUTPUT	The size of the input and the resulting output are fixed (i.e., receives images of fixed size and outputs them to the appropriate category along with the confidence level of its prediction)	The size of the input and the resulting output may vary (i.e., receives different text and output translations—the resulting sentences can have more or fewer words)
IDEAL USAGE SCENARIO	Spatial data (such as images)	Temporal/sequential data (such as text or video)
USE CASES	Image recognition and classification, face detection, medical analysis, drug discovery and image analysis	Text translation, natural language processing, language translation, entity extraction, conversational intelligence, sentiment analysis, speech analysis

CNN ها عمدتاً برای کارهای تشخیص تصویر و ویدیو استفاده می شوند، زیرا می توانند مقادیر زیادی از داده های تصویر را به صورت موازی پردازش کنند. آنها از چندین لایه کانولوشن تشکیل شده‌اند که استخراج ویژگی را روی تصویر ورودی انجام می‌دهند، به دنبال آن لایه‌های ادغامی که اندازه نقشه‌های ویژگی را کاهش می‌دهند، و در نهایت، یک یا چند لایه کاملاً متصل که وظیفه طبقه‌بندی یا رگرسیون را انجام می‌دهند. CNN ها در پردازش داده های تصویر بسیار کارآمد هستند و به طور گسترده در برنامه هایی مانند تشخیص اشیا، تشخیص چهره و خودروهای خودران استفاده می شوند.

از سوی دیگر، RNN ها برای کارهای مدل سازی توالی مانند مدل سازی زبان، تشخیص گفتار و پیش بینی سری های زمانی مناسب تر هستند. RNN ها معماری منحصر به فردی دارند که به آنها اجازه می دهد وابستگی های زمانی را در داده های متوالی ثبت کنند. آنها از یک لایه بازگشتی تشکیل شده اند که توالی ورودی را یک عنصر در یک زمان پردازش می کند و یک حالت پنهان را حفظ می کند که زمینه عناصر قبلی را رمزگذاری می کند و یک لایه کاملاً متصل که وظیفه طبقه بندی یا رگرسیون را انجام می دهد. RNN ها می توانند وابستگی های بلندمدت را در توالی ورودی مدل کنند و آنها را برای کارهایی مانند ترجمه زبان و تشخیص گفتار بسیار موثر می کند.

ب)

• تعداد پارامترها: تعداد پارامترها در یک شبکه عصبی، تعداد کل وزن ها و سوگیری هایی است که در طول آموزش آموخته می شوند. تعداد پارامترها به اندازه و ساختار شبکه و همچنین ابعاد ورودی و خروجی بستگی دارد. به طور کلی، **CNN ها پارامترهای کمتری نسبت به RNN دارند،** زیرا از فیلترهای اشتراک گذاری وزن و کانولوشنال استفاده می کنند که تعداد اتصالات بین لایه ها را کاهش می دهد. RNN ها پارامترهای بیشتری نسبت به CNN ها دارند، زیرا از اتصالات مکرر و سلول های حافظه استفاده می کنند که تعداد اتصالات بین لایه

ها را افزایش می دهد. به عنوان مثال، یک RNN ساده با یک لایه پنهان و ۱۰۰ واحد پنهان دارای حدود ۱۰۰۰۰ پارامتر است، در حالی که یک CNN ساده با یک لایه کانولوشن و ۱۰۰ فیلتر حدود ۱۰۰۰ پارامتر دارد .

• قابلیت موازی سازی: قابلیت موازی سازی یک شبکه عصبی میزانی است که محاسبات در شبکه را می توان به صورت همزمان یا مستقل انجام داد. قابلیت موازی سازی به وابستگی ها و ترتیب محاسبات در شبکه و همچنین در دسترس بودن منابع محاسباتی بستگی دارد. **به طور کلی، CNN ها توانایی موازی بالاتری نسبت به RNN دارند، زیرا از عملیات کانولوشنی استفاده می کنند** که می تواند به طور همزمان در قسمت های مختلف ورودی یا فیلترهای مختلف اعمال شود. RNN ها نسبت به CNN ها توانایی موازی کمتری دارند، زیرا از عملیات متوالی استفاده می کنند که به مراحل یا حالت های زمانی قبلی بستگی دارد.
NNCها به طور کلی در پردازش مقادیر زیادی از داده های تصویری به صورت موازی سریع تر و کارآمدتر هستند، در حالی که RNN ها در مدل سازی وابستگی های زمانی در داده های متوالی قدرتمندتر هستند. انتخاب نوع شبکه برای استفاده در نهایت به ماهیت مشکل و نوع داده مورد استفاده بستگی دارد.
RNN ها را می توان تا حدی موازی کرد اما نه به اندازه RNNهاCNN. برای پردازش توالی ها به صورت متوالی طراحی شده اند، بنابراین موازی سازی آنها چالش برانگیز می شود. از سوی دیگر، CNN ها مناطق مستقل را در یک تصویر پردازش می کنند، بنابراین می توان آنها را به طور موثر موازی کرد.

https://datascience.stackexchange.com/questions/11619/rnn-vs-cnn-at-a-high-level
https://www.techtarget.com/searchenterpriseai/feature/CNN-vs-RNN-How-they-differ-and-where-they-overlap
https://www.quora.com/What-is-the-difference-between-CNN-and-RNN-in-terms-of-power-and-performance
www.bing.com
https://d2l.ai/chapter_recurrent-neural-networks/rnn.html

Parameters of the RNN include the weights W xh ∈ R d × h , W hh ∈ R h × h , and the bias b h ∈ R 1 × h of the hidden layer, together with the weights W hq ∈ R h × q and the bias b q ∈ R 1 × q of the output layer. It is worth mentioning that even at different time steps, RNNs always use these model parameters.

The parameters involved are the number of convolution layers, the number of convolution kernels, the number of pooling layers, the number of the fully connected layer and the optimizer

سوال ۵)

الف)

For convolution:

$$\text{Output size} = \frac{\text{Input size} - \text{Filter size} + 2 \times \text{Padding}}{\text{Stride}} + 1$$

For pooling:

$$\text{Output size} = \frac{\text{Input size} - \text{Pooling size}}{\text{Stride}} + 1$$

Convolutional layer parameters:

$$\text{Parameters} = (\text{Filter size} \times \text{Filter size} \times \text{Input depth} + 1) \times \text{Number of filters}$$

Fc layer parameters:

If you have a Flatten layer followed by an FC layer with N neurons, and the flattened input has size M, then the number of parameters (P) in the FC layer is given by:

$$P = M \times N + N$$

The same padding is another technique used in convolutional neural networks (CNNs) to process the input data. Unlike valid padding, same padding adds additional rows and columns of pixels around the edges of the input data so that the size of the output feature map is the same as the size of the input data.

- With "SAME" padding, if you use a stride of 1, the layer's outputs will have the **same** spatial dimensions as its inputs.
- With "VALID" padding, there's no "made-up" padding inputs. The layer only uses **valid** input data.

layer	Out put layer	Output channel	parametr
Conv1D	In the image	Num of filters	Filters * (kernel size * input channel size +1)
Delated Conv		Num of filters	Filters * (kernel size * input channel size +1)
MaxPool1D	In the image	Stay same	0

Delated Conv output layer =

$(input\ size + 2*\text{Padding} - ((filter\ size - 1) * Dilation\ factor + 1) / \text{Stride}) + 1$

https://towardsdatascience.com/review-dilated-convolution-semantic-segmentation-9d5a5bd768f5

1. Dilated Convolution

$$(F * k)(\mathbf{p}) = \sum_{\mathbf{s}+\mathbf{t}=\mathbf{p}} F(\mathbf{s})\, k(\mathbf{t}). \quad (F *_l k)(\mathbf{p}) = \sum_{\mathbf{s}+l\mathbf{t}=\mathbf{p}} F(\mathbf{s})\, k(\mathbf{t}).$$

Standard Convolution (Left), Dilated Convolution (Right)

The left one is the standard convolution. The right one is the dilated convolution. We can see that at the summation, it is s+*l*t=p that we will skip some points during convolution.

When *l*=1, it is standard convolution.

When *l*>1, it is dilated convolution.

Layer	Activation Volume Dimensions	Number of parameters
Input	256,256,3	0
Conv2D	256,256,64	64*(3*3*3+1)
Conv2DDelated	124, 124, 32	32*(5*5*64+1)
MaxPool2D	62,62,32	0
Conv2D	62,62,128	128*(3*3*32+1)
Conv2DDelated	23,23,64	64*(5*5*128+1)
MaxPool1D	11,11,64	0
Conv2D	11,11,256	256*(3*3*64+1)
Conv2DDelated	-10,-10,128	128*(5*5*256+1)
MaxPool1D	-5,-5,128	0

چند نمونه از محاسبه ها برای لایه های اول تا سوم که دوباره همین محاسبات تکرار می شود :

لایه اول :

پارامتر: (یک بایاس + پارامتر ها به ازای هر فیلتریا اندازه کرنل * تعداد کانال های ورودی)تعداد کل فیلتر ها =

64*(3*3*3+1)

اندازه : چون پدینگ same و تعداد فیلتر ها ۶۴ است مساوی تعداد کانال ها پس همان اندازه عرض و طول ورودی را ثابت نگه می دارد برای اندازه دیمانسیون :

256,256,64

لایه دوم:

پارامتر: (یک بایاس + پارامتر ها به ازای هر فیلتریا اندازه کرنل * تعداد کانال های ورودی)تعداد کل فیلتر ها =

32*(5*5*64+1)

اندازه : عدد نهایی رو به پایین گرد کردم

تعداد فیلتر =۳۲

124,124,32

(256+2*0 – ((5-1)*2+1)/2)/2+1= 124.5 =124

لایه سوم :

پارامتر:ندارد

اندازه :تعداد فیلتر بی تغییر =۳۲

124-2/2 +1 = 62

62,62,32

لایه چهارم :

پارامتر: (یک بایاس + پارامتر ها به ازای هر فیلتریا اندازه کرنل * تعداد کانال های ورودی)تعداد کل فیلتر ها

128*(3*3*32+1)

اندازه : چون پدینگ same و تعداد فیلتر ها 128 است مساوی تعداد کانال ها تعداد فیلتر ها می شود پس همان اندازه عرض و طول ورودی را ثابت نگه می دارد برای اندازه دیمانسیون :

62,62,128

لایه پنجم :

پارامتر: : (یک بایاس + پارامتر ها به ازای هر فیلتریا اندازه کرنل * تعداد کانال های ورودی)تعداد کل فیلتر ها

64*(5*5*128+1)

اندازه :تعداد کانال = تعداد فیلتر

62+2*0 – ((5-1)*4+1)/2)/2+1= 23.5 =23

23,23,64

لایه ششم :

پارامتر: ندارد

اندازه : تعداد فیلتر بی تغییر =۶۴

23-2/2 +1 = 11.5 =11

11,11,64

لایه هفتم :

پارامتر : (یک بایاس + پارامتر ها به ازای هر فیلتریا اندازه کرنل * تعداد کانال های ورودی)تعداد کل فیلتر ها

256*(3*3*64+1)

اندازه : :تعداد کانال = تعداد فیلتر=۲۵۶

پدینگ same بدون تغییر

11,11,256

لایه هشتم :

پارامتر : (یک بایاس + پارامتر ها به ازای هر فیلتریا اندازه کرنل * تعداد کانال های ورودی)تعداد کل فیلتر ها

128*(5*5*256+1)

اندازه : :تعداد کانال = تعداد فیلتر=۱۲۸

11+2*0 – ((5-1)*8+1)/2)/2+1= -10

-10,-10,128

لایه نهم :

پارامتر: ندارد

اندازه : تعداد فیلتر بی تغییر =۱۲۸

-10-2 / 2 +1 =-5

-5,-5,128

اما در برخی کتابخانه ها که حذف را اعمال نمی کنند این مقدار ها می تواند منفی نشود

Delated Conv output layer =

$(input\ size$+2*Padding − (((

$filter\ size - 1) * Dilation\ factor$)- 1) / Stride) +1

Layer	Activation Volume Dimensions	Number of parameters
Input	256,256,3	0
Conv2D	256,256,64	64*(3*3*3+1)
Conv2DDelated	248, 248, 32	32*(5*5*64+1)
MaxPool2D	124,124,32	0
Conv2D	124,124, 128	128*(3*3*32+1)
Conv2DDelated	108,108,64	64*(5*5*128+1)
MaxPool1D	54,54,64	0
Conv2D	54,54,256	256*(3*3*64+1)
Conv2DDelated	22,22,128	128*(5*5*256+1)
MaxPool1D	11,11,128	0

ب)

بدون stride :=1

F =اندازه عرض یا طول فیلتر

S=اندازه گام استراد که یک مقدار ثابت است و می تواند برای طول یا عرض اعمال شود

P= میزان پدینگ یا حاشیه ای که به اطراف طول یا عرض صفحه اضافه می شود

p = (f – 1) / 2

(because (n + 2p – f)/S+ 1 = n).

(n + 2p – f)+ 1 = n).

Pad so that output size is the same as the input size.

$n+2p-f+1 \times n+2p-f+1$

$n+2p-f+1 = n \Rightarrow p = \frac{f-1}{2}$

با stride : =s

((input + 2p – f) /strid) 1 = output

(((Out put -1) * S)+f – input) /2 =p

چون کانولوشن دو بعدی است و می خواهیم اندازه ها ثابت بماند به مقدار پدینگ در طول و عرض باید توجه داشت:

عرض

(((Out put width-1) * S)+f width – input W) /2 =p width

طول

(((Out put height -1) * S)+f height – input H) /2 =p height

Same Padding: In this case, we add 'p' padding layers such that the output image has the same dimensions as the input image.
So, if we use a (3 x 3) filter on an input image to get the output with the same dimensions. the 1 layer of zeros must be added to the borders for the same padding. Similarly, if (5 x 5) filter is used 2 layers of zeros must be appended to the border of the image.

سوال ۶)

الف)

کدام یک از عبارات زیر در مورد نرمال سازی دسته ای درست است؟

•نرمال سازی دسته ای تنها پردازش یک دسته را سریع تر می کند و زمان آموزش را کاهش می دهد و درعین حال تعداد به روزرسانی ها را ثابت نگه می دارد. این به شبکه اجازه می دهد تا زمان مشابهی را صرف انجام به روزرسانی های بیشتر کند تا به حداقل برسد .

•**نرمال سازی دسته ای توزیع خروجی را نرمال می کند تا در ابعاد یکنواخت تر باشد .**

•به شبکه اجازه می دهد تا وزن های ما را به مقادیر کوچک تر نزدیک به صفر مقداردهی کند.

- نادرست دلیلی وجود ندارد که فرض کنیم عادی سازی دسته ای پردازش یک دسته را سریعتر می کند. کاملا برعکس، ما می توانیم انتظار دسته ای داشته باشیم نرمال سازی برای کندتر کردن هر تکرار

تمرین به دلیل پردازش اضافی مورد نیاز در طول پاس رو به جلو و فراپارامترهای اضافی که در حین تکثیر مجدد نیاز به آموزش دارند. همانطور که گفته شد، ما انتظار داریم شبکه سریعتر از آن چه بدون نرمال سازی دسته ای همگرا می شود انتظار می رود آموزش شبکه در کل سریعتر تکمیل شود.

- درست است، واقعی. این اساساً هدف عادی سازی دسته ای، عادی سازی است ورودی های هر لایه (که به نوبه خود خروجی های لایه قبل از آن هستند) داده می شود میانگین و واریانس مقادیر در مینی دسته فعلی. همانطور که در مواد درسی را معمولاً به صفر میانگین و واریانس واحد نرمال می کنی

- نادرست این یکی کمی مشکل است. قسمت اول قطعا درست است، به عنوان دسته نرمال سازی اثرات اولیه سازی ضعیف وزن را کاهش می دهد و به ما اجازه می دهد کمتر به این موضوع بپردازیم که چگونه وزن های خود را مقداردهی اولیه می کنیم، که به طور کلی اینطور است حل بهینه مشکل آسانی نیست، بنابراین عادی سازی دسته ای واقعا به ما کمک می کند اینجا اما بخش دوم این بیانیه برقرار نیست، زیرا ما هستیم به سادگی اثر اولیه سازی وزن خود را محدود می کنیم و سپس آن را عادی می کنیم ورودی به هر لایه، تضعیف تاثیر وزن های اولیه تعیین شده است اما لزوماً «اجازه دادن به شبکه برای مقداردهی اولیه وزن‌های ما به مقادیر نزدیک به صفر کوچک‌تر نیست ».

- منبع : https://cs230.stanford.edu/files/cs230exam_win20_soln.pdf

ب)

```
if mode == 'train':
        # TODO: Mean of Z across first dimension
        mu = np.mean(Z, axis=0, keepdims=True)

        # TODO: Variance of Z across first dimension
        var = np.var(Z, axis=0, keepdims=True)

        # Take moving average for cache_dict['mu']
        cache_dict['mu'] = beta_avg * cache_dict['mu'] + (1-beta_avg) * mu

        # Take moving average for cache_dict['var']
        cache_dict['var'] = beta_avg * cache_dict['var'] + (1-beta_avg) * var

        # X = (X - mu) / np.sqrt(var + eps)
        # out = gamma * X + beta

    elif mode == 'test':
        # TODO: Load moving average of mu
        mu = cache_dict['mu']
```

```
        # TODO: Load moving average of var
        var = cache_dict['var']

     # TODO: Apply z_norm transformation
    # Z_norm =
    Z_norm = (Z - mu) / np.sqrt(var + eps)

    # TODO: Apply gamma and beta transformation to get Z tiled
    out = gamma * Z_norm + beta

    return out
```

این کد یک تابع پایتون است که عملیات پخش را از طریق یک لایه BatchNorm انجام میدهد. BatchNorm یک تکنیک نرمالسازی دادهها است که در یادگیری عمیق به کار میرود. این تکنیک با کاهش انحراف داخلی و تغییر واریانس دادهها در هر لایه، به بهبود عملکرد و سرعت شبکههای عصبی کمک میکند این کد دو حالت را برای اجرای BatchNorm در نظر میگیرد: حالت train و حالت test. در حالت train، تابع میانگین و واریانس دادههای ورودی را محاسبه میکند و آنها را در یک دیکشنری به نام cache_dict ذخیره میکند. سپس از یک مقدار به نام beta_avg برای محاسبه میانگین متحرک از میانگین و واریانس استفاده میکند. این مقدار برای استفاده در حالت test مورد نیاز است. سپس دادهها را با کم کردن میانگین و تقسیم بر ریشه واریانس به همراه یک مقدار کوچک به نام eps نرمال میکند. این عملیات باعث میشود که دادهها دارای میانگین صفر و واریانس یک شوند. سپس از دو پارامتر به نام gamma و beta برای اعمال تبدیل خطی به دادهها استفاده میکند. این دو پارامتر قابل یادگیری هستند و به شبکه اجازه میدهند که بهترین مقیاس و جابهجایی را برای دادهها پیدا کنند. در نهایت خروجی را برمیگرداند. در حالت test، تابع میانگین و واریانس متحرک را از cache_dict بارگذاری میکند و همانند حالت train، دادهها را نرمال و تبدیل میکند. اما این بار از مقادیر محاسبه شده در حالت train برای میانگین و واریانس استفاده میکند. این کار باعث میشود که خروجی در حالت test با حالت train هماهنگ باشد و تغییرات ناخواسته در دادهها ایجاد نشود.

کد نوشته شده جهت تست :

```
X_size = 10
b_avg = 0.7
epsilon = 0.0001
bias = 200
X = np.random.randn(5, 4) + bias


gamma = np.random.rand(1, 4)
beta = np.random.rand(1, 4)
```

```
cache_dict = {
    'mu': np.zeros((1, 4)),
    'var': np.zeros((1, 4))
}

mode = 'train'
applied = forward_batchnorm(
    X, gamma, beta, epsilon, cache_dict, b_avg, 'train')
print('')
print('_____________________________________________________________________
__________')
var = np.var(applied, axis=0)
mu = np.mean(applied, axis=0)
print(var, mu)
print('_____________________________________________________________________
__________')
print('')

print(cache_dict)
print('_____________________________________________________________________
__________')
print('matris without')
print('_____________________________________________________________________
__________')
print(X)
print('_____________________________________________________________________
__________')
print('matris applied')
print('_____________________________________________________________________
__________')
print(applied)
```

این کد یک تابع به نام forward_batchnorm را فراخوانی میکند که وظیفه آن انجام عملیات نرمالسازی دستهای (batch normalization) بر روی ورودی X است. نرمالسازی دستهای یک تکنیک است که با تغییر مرکز و مقیاس دادههای ورودی هر لایه شبکه عصبی، باعث سریعتر و پایدارتر شدن فرآیند آموزش میشو

این کد از چند پارامتر برای انجام نرمالسازی دستهای استفاده میکند که در ادامه به آنها اشاره میکنم:

- X_size: اندازه دستهای که برای آموزش شبکه عصبی استفاده میشود. در این کد مقدار آن برابر با ۱۰ است.

• b_avg: ضریبی که برای محاسبه میانگین حرکتی (moving average) مورد استفاده قرار میگیرد. این میانگین حرکتی برای نگهداری میانگین و واریانس دستههای آموزش در زمان آزمون استفاده learn-/. در این کد مقدار آن برابر با ۰,۷ است.

• epsilon: یک مقدار کوچک مثبت که به واریانس دستهها اضافه میشود تا از تقسیم بر صفر جلوگیری کند.در این کد مقدار آن برابر با ۰,۰۰۰۱ است.

• bias: یک مقدار ثابت که به دادههای ورودی اضافه میشود تا از مقادیر منفی جلوگیری کند. در این کد مقدار آن برابر با ۲۰۰ است.

• X: یک آرایه دو بعدی از دادههای ورودی که برای آموزش شبکه عصبی استفاده میشود. در این کد اندازه آن برابر با (۵, ۴) است و مقادیر آن از توزیع نرمال با میانگین صفر و واریانس یک به دست میآیند. سپس مقدار bias به آنها اضافه میشود.

• gamma: یک آرایه یک بعدی از پارامترهای مقیاس که در مرحله نرمالسازی دستهای برای تنظیم مقیاس دادههای نرمالشده استفاده میشوند. در این کد اندازه آن برابر با (۱, ۴) است و مقادیر آن از توزیع یکنواخت بین صفر و یک به دست میآیند.

• beta: یک آرایه یک بعدی از پارامترهای انحراف که در مرحله نرمالسازی دستهای برای تنظیم مرکز دادههای نرمالشده استفاده می شوند. در این کد اندازه آن برابر با (۱, ۴) است و مقادیر آن از توزیع یکنواخت بین صفر و یک به دست میآیند.

• cache_dict: یک دیکشنری که برای نگهداری میانگین و واریانس دستههای آموزش استفاده میشود. در این کد اندازه هر کلید برابر با (۱, ۴) است و مقادیر آنها صفر هستند.

• mode: یک رشته که نشان میدهد که تابع forward_batchnorm در حالت آموزش یا آزمون است. در حالت آموزش، میانگین و واریانس دستهها محاسبه میشوند و در دیکشنری cache_dict ذخیره میشوند. در حالت آزمون، میانگین و واریانس میانگین حرکتی استفاده میشوند. در این کد مقدار آن برابر با 'train' است.

خروجی تابع forward_batchnorm یک آرایه دو بعدی از دادههای نرمالشده است که اندازه آن برابر با اندازه ورودی X است. در این کد اندازه آن برابر با (۵, ۴) است و مقادیر آن با استفاده از فرمول زیر به دست میآیند:

که در آن x هر عنصر از ورودی X، mu میانگین دستهای، sigma واریانس دستهای، epsilon مقدار کوچک مثبت، gamma پارامتر مقیاس و beta پارامتر انحراف است.

این کد بعد از فراخوانی تابع forward_batchnorm، چند خط چاپ میکند که در ادامه به آنها اشاره میکنم:

- خط سوم واریانس و میانگین خروجی تابع forward_batchnorm را بر اساس محور صفر (هر ستون) چاپ میکند. این مقادیر نشان میدهند که خروجی تابع چقدر نزدیک به توزیع نرمال با میانگین صفر و واریانس یک است.

```
_____
[0.02525136 0.0532453  0.00198642 0.09757538] [0.96033662 0.09796898 0.88215292 0.30749616]
_____

{'mu': array([[60.11437055, 60.05156571, 59.99811837, 59.99286275]]), 'var': array([[0.02287844, 0.15757392, 0.25910608, 0.0764227 ]])}
_____
matris without
_____
[[200.24086698 200.05284776 198.3689064  199.59337774]
 [200.19831952 200.81873641 199.73344368 200.32126731]
 [200.64816171 198.94874588 200.0894894  199.43741279]
 [200.05212682 200.0421421  200.87114681 200.7896458 ]
 [200.76670087 200.9969564  200.90565315 199.73934224]]
_____
matris applied
_____
[[ 0.87956504  0.06006851  0.80423036  0.07056229]
 [ 0.8550821   0.30391961  0.86967031  0.52105218]
 [ 1.11393321 -0.29146621  0.88674541 -0.02596421]
 [ 0.77095895  0.05665994  0.92423183  0.81093102]
 [ 1.18214378  0.36066303  0.92588667  0.16089952]]
PS C:\Users\user\Desktop\Term9\Deep learning\DL_4_SaraYounesi_98533053>
```

ج)

از تقسیم بر صفر جلوگیری می کند با ویژگی هایی که واریانس صفر دارند.

Small float added to variance to avoid dividing by zero
epsilon is small constant (configurable as part of the constructor arguments)

A small value (default, 0.001) added to the mini-batch variance when normalizing the inputs as I explained in Step 2. center: When True (default), this activates the beta parameter. scale: When True (default), this activates the gamma parameter.

هایپر پرامتر اپسیلون یک مقدار کوچک مثبت است که به واریانس هر دسته کوچک اضافه میشود تا از تقسیم بر صفر جلوگیری کند.این کار باعث میشود که نرمالسازی دسته ای بیشتر به مقادیر ورودی حساس نباشد و مقاومت شبکه را در برابر تغییرات توزیع داخلی افزایش دهد. مقدار اپسیلون معمولا بسیار کوچک است، مثلا ۰,۰۰۱ یا ۰,۰۰۰۱.

https://medium.com/data-science-365/batch-normalization-explained-in-plain-english-3436014f9e52#:~:text=epsilon%3A%20A%20small%20value%20(default,this%20activates%20the%20gamma%20parameter.

د)

ستفاده از نرمال سازی دسته ای با اندازه یک مشکلاتی را به همراه دارد که در ادامه به آنها اشاره میکنم:

• اگر اندازه دسته یک باشد، میانگین و واریانس هر دسته برابر با میانگین و واریانس هر نمونه خواهد بود. این باعث میشود که نرمال سازی دسته ای هیچ تاثیری در تغییر توزیع دادهها نداشته باشد و فقط مقیاس آنها را تغییر دهد

• اگر اندازه دسته یک باشد، نرمال سازی دسته ای نمیتواند از اثر شیفت توزیع داخلی جلوگیری کند. این اثر زمانی رخ میدهد که توزیع دادههای ورودی در طول آموزش تغییر کند و باعث کاهش کارایی شبکه عصبی شود

• اگر اندازه دسته یک باشد، نرمال سازی دسته ای نمیتواند از اثر منظمکننده برخوردار باشد. این اثر زمانی رخ میدهد که نرمال سازی دسته ای باعث کاهش بیشبرازش شبکه عصبی شود و بهبود تعمیم پذیری آن دهد

بنابراین، استفاده از نرمال سازی دسته ای با اندازه یک مزایای این تکنیک را از بین میبرد و ممکن است باعث کاهش عملکرد شبکه عصبی شود. به همین دلیل، معمولا اندازه دسته ها بزرگتر از یک انتخاب میشوند تا نرمال سازی دسته ای بتواند اثرات مثبت خود را نشان دهد.

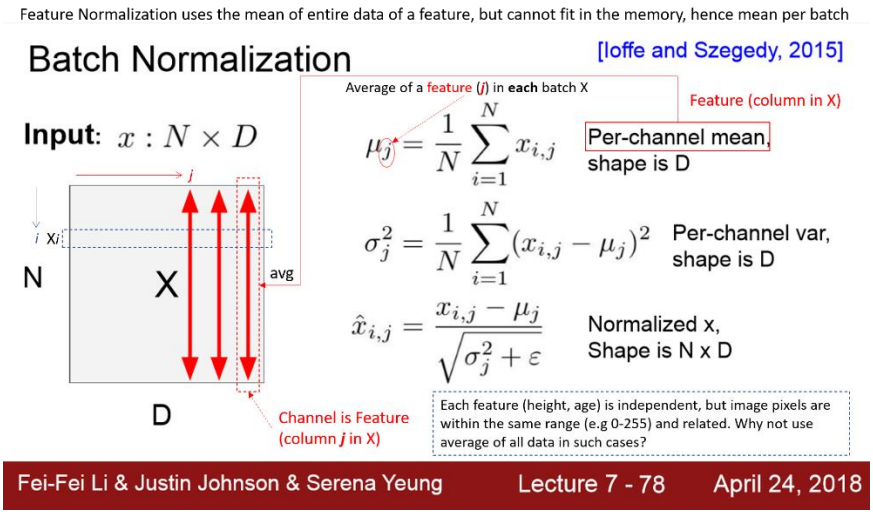
به طور خاص، اگر ما فقط یک مینی بچ ۱ داشته باشیم، خروجی از لایه BN همیشه ۰ خواهد بود ($x - \mu$ صفر است، ما آن را در γ ضرب می کنیم که همچنان ۰ خواهد بود و β را اضافه می کنیم. از آنجایی که β یک پارامتر "بایاس" است، به ۰ مقداردهی اولیه می شود.)
و بنابراین ما نمی توانیم ویژگی های معنی دار را یاد بگیریم.
به طور کلی، زمانی که شما فقط حول یک بچ نرمالایز می کنید منطقی نیست وقتی تعداد نمونه کمی در هر دسته است .
نرمال سازی دسته ای در زمان یادگیری به هم می ریزد، زیرا برآوردهای میانگین/واریانس شما فوق العاده نویزی خواهد بود (فقط با استفاده از تعداد کمی نمونه هایی که ما در تلاش هستیم تا جمعیت/توزیع واقعی را تخمین بزنیم). همینطور در زمان آزمون اعمال می شود - ما معمولاً از میانگین متحرک میانگین / واریانس جمع آوری

شده در حین آموزش در زمان آزمون استفاده می اما این میانگین متحرک نیز پر نویزدار خواهد بود و نشان دهنده توزیع واقعی نیست.

https://www.wikiwand.com/fa/%D9%86%D8%B1%D9%85%D8%A7%D9%84%E2%80%8C%D8%B3%D8%A7%D8%B2%DB%8C_%D8%AF%D8%B3%D8%AA%D9%87%E2%80%8C%D8%A7%DB%8C

https://blog.faradars.org/%D8%A2%D9%85%D9%88%D8%B2%D8%B4-%D9%86%D8%B1%D9%85%D8%A7%D9%84-%D8%B3%D8%A7%D8%B2%DB%8C-/%D8%AF%D8%B1-%D9%BE%D8%A7%DB%8C%DA%AF%D8%A7%D9%87-%D8%AF%D8%A7%D8%AF%D9%87

۵)



این پارامترها برای مقیاس مجدد (γ) و تغییر (β) بردار حاوی مقادیر از عملیات قبلی استفاده می شود. این دو پارامتر قابل یادگیری هستند، در طول آموزش شبکه عصبی تضمین می کند که مقادیر بهینه γ و β استفاده می شود. این امکان عادی سازی دقیق هر دسته را فراهم می کند.

نرمال سازی دسته ای معمولاً برای پیش فعال سازی ها در یک لایه متراکم اعمال می شود که ۲۰ مورد برای این لایه وجود دارد. گاما و بتا برای هر ورودی اسکالر هستند. به این ترتیب ما ۲۰ گاما و ۲۰ بتا داریم. این علاوه بر وزن‌های ۱۰ × ۲۰ = ۲۰۰ وزن در لایه متراکم است (هنگامی که بچ نرمالیزیشن اعمال می شود ، هیچ شرایط بایاسی وجود ندارد زیرا بتا آنها را اضافی می‌کند). این در مجموع ۲۴۰ پارامتر را به دست می دهد.
مقادیر لایه ی متراکم + لایه بچ نرمالیزشن (بتا و گاما) =۲۰۰+۴۰ = ۲۴۰

https://www.analyticsvidhya.com/

سوال ۷)

ابتدا به کمک api فراهم شده در کتابخانه Keras مجموعه داده MNIST را بارگذاری نمایید
این کد یک مجموعهدادهی مشهور به نام MNIST را بارگذاری میکند. این مجموعهداده شامل ۷۰ هزار تصویر دستنوشتهی اعداد از ۰ تا ۹ است- h. این کد این مجموعهداده را به دو بخش تقسیم میکند: یک بخش آموزشی که شامل ۶۰ هزار تصویر و برچسب آنها است، و یک بخش آزمون که شامل ۱۰ هزار تصویر و برچسب آنها است. این کد این دو بخش را به صورت دو جفت متغیر در اختیار میگذارد: (x_train, y_train) و (x_test, y_test).

x_train و x_test دو آرایهی سهبعدی هستند که شامل تصاویر دستنوشتهی اعداد هستند. هر تصویر یک آرایهی دوبعدی از پیکسلها است که اندازهی آن ۲۸ در ۲۸ است. هر پیکسل یک مقدار عددی بین ۰ تا ۲۵۵ دارد که نشاندهندهی روشنایی آن است. بنابراین، x_train و x_test به ترتیب شکل (۶۰۰۰۰, ۲۸, ۲۸) و (۱۰۰۰۰, ۲۸, ۲۸) دارند.

y_train و y_test دو آرایهی یکبعدی هستند که شامل برچسبهای تصاویر هستند. هر برچسب یک عدد صحیح بین ۰ تا ۹ است که نشاندهندهی عدد دستنوشتهی مربوطه است. بنابراین، y_train و y_test به ترتیب شکل (۶۰۰۰۰,) و (۱۰۰۰۰,) دارند.

```
(x_train, y_train), (x_test, y_test)= mnist.load_data()
```
[4]

```
print('x_train.shape =', x_train.shape)
print('y_train.shape =', y_train.shape)
print('x_test.shape =', x_test.shape)
print('y_test.shape =', y_test.shape)
```
[5]

```
x_train.shape = (60000, 28, 28)
y_train.shape = (60000,)
x_test.shape = (10000, 28, 28)
y_test.shape = (10000,)
```

•داده های آموزشی را shuffle کنید و ابعاد داده ها را چاپ کنید

• x_shuffled=np.random.shuffle(x_train) و y_shuffled=np.random.shuffle(y_train) دو خط کد هستند که به ترتیب آرایههای x_train و y_train را به صورت تصادفی میآمیزند. این کار میتواند برای جلوگیری از بیاس در دادههای آموزشی مفید باشد. توجه کنید که np.random.shuffle تابعی است که آرایه ورودی را در جای خود تغییر میدهد و خروجی نمیدهد. بنابراین، متغیرهای x_shuffled و y_shuffled برابر None خواهند بود. اگر میخواهید یک نسخه مخلوط شده از آرایه را بدون تغییر آرایه اصلی بسازید، میتوانید از تابع np.random.permutation استفاده کنید

```
np.random.seed(42)
x_shuffled=np.random.shuffle(x_train)
np.random.seed(42)
y_shuffled=np.random.shuffle(y_train)
```
[6]

```
print('x_train.shape =', x_train.shape)
print('y_train.shape =', y_train.shape)
```
[7]

```
x_train.shape = (60000, 28, 28)
y_train.shape = (60000,)
```

تصویر نخست موجود در مجموعه داده آموزشی را به همراه برچسب آن ها نمایش دهید.

fig = plt.figure(figsize=(10, 10)) تابعی است که یک شیء نمودار را با اندازه ۱۰ در ۱۰ اینچ ایجاد میکند. plt مخفف matplotlib.pyplot است که یک کتابخانه برای رسم نمودارهای دوبعدی در پایتون است.

- for i in range(۱۰): حلقهای است که ۱۰ بار تکرار میشود. در هر تکرار، مقدار i از ۰ تا ۹ تغییر میکند.

- img, lb= x_train[i], y_train[i] دو خط کد هستند که به ترتیب عکس و برچسب مربوط به نمونه i ام را از دادههای آموزشی در متغیرهای img و lb ذخیره میکنند.

- j=i+1 خط کدی است که مقدار j را برابر i+1 قرار میدهد. این کار برای شمارهگذاری زیرنمودارها لازم است، زیرا شمارهگذاری زیرنمودارها از ۱ شروع میشود، در حالی که شمارهگذاری حلقه از ۰ شروع میشود.

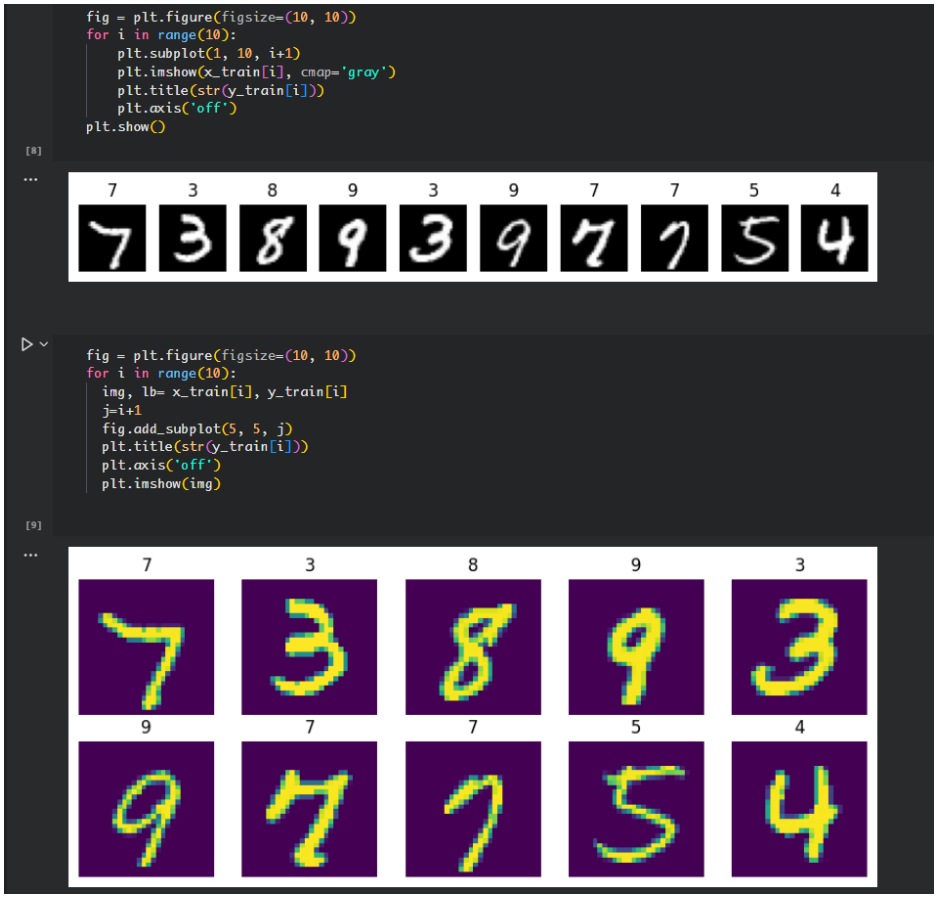
- fig.add_subplot(5, 5, j) تابعی است که یک زیرنمودار را به شیء نمودار اضافه میکند. پارامترهای ۵, ۵, j مشخص میکنند که زیرنمودار در یک شبکه ۵ در ۵ قرار میگیرد و شماره j را دارد.

- plt.title(str(y_train[i])) تابعی است که عنوان زیرنمودار را برابر برچسب نمونه i ام قرار میدهد. تابع str برای تبدیل برچسب به رشته استفاده میشود.

- plt.axis('off') تابعی است که محورهای زیرنمودار را غیرفعال میکند.

- plt.imshow(img) تابعی است که عکس را در زیرنمودار نشان میدهد.

به طور خلاصه، کد شما یک نمودار با ۲۵ زیرنمودار را رسم میکند که در هر زیرنمودار یک عکس از دادههای آموزشی و برچسب آن نشان داده میشود. دادههای آموزشی قبل از رسم نمودار به صورت تصادفی مخلوط شدهاند.



مقادیر پیکسل ها را به محدوده ۰ تا ۱ تغییر دهید.

- x_train = x_train.astype("float32") / 255 و x_test = x_test.astype("float32") / 255
دو خط کد هستند که به ترتیب آرایههای x_train و x_test را به نوع داده float32 تبدیل میکنند و سپس تمام مقادیر آنها را بر ۲۵۵ تقسیم میکنند. این کار برای نرمالسازی دادههای عکس که مقادیر پیکسلهای آنها بین ۰ تا ۲۵۵ است، انجام میشود. نرمالسازی دادهها میتواند به بهبود عملکرد مدل کمک کند.

- y_train = keras.utils.to_categorical(y_train) و y_test = keras.utils.to_categorical(y_test)
دو خط کد هستند که به ترتیب آرایههای y_train و y_test را به نمایش دستهای تبدیل میکنند. نمایش دستهای یک روش برای کدگذاری برچسبهای دستهای است که در آن هر برچسب به یک بردار تبدیل میشود که تمام عناصر آن صفر هستند به جز عنصری که با شماره دسته مطابقت دارد که مقدار آن یک است. برای مثال، اگر دستهها ۰ تا ۹ باشند، برچسب ۳ به بردار [۰, ۰, ۰, ۱, ۰, ۰, ۰, ۰, ۰, ۰] تبدیل میشود. این کار برای استفاده از تابع هزینهی cross-entropy در مدلهای دستهبندی لازم است

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

برچسب ها را به نمایش categorical تغییر دهید.

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


print(y_train.shape)
```

```
(60000, 10)
```

یک شبکه عصبی با ویژگی های زیر طراحی و آموزش دهید.
ین کد یک مدل شبکه عصبی پیچشی (CNN) را با استفاده از کتابخانه کراس ایجاد میکند. مدل شامل چهار لایه پیچشی، سه لایه حداکثر تجمیع (MaxPooling)، یک لایه فلتن (Flatten) و دو لایه تماما متصل (Dense) است. توضیحات بیشتر در زیر آمده است:

- model=Sequential() یک مدل خطی را تعریف میکند که لایهها را به ترتیب پشت سر هم قرار میدهد.

- model.add(Conv2D(32, 3, padding="same", activation='relu', input_shape=(28, 28, 1))) یک لایه پیچشی را به مدل اضافه میکند که ۳۲ فیلتر با اندازه ۳×۳ دارد. این لایه از تابع فعالسازی رلو (ReLU) استفاده میکند و برای ورودیهای با شکل ۲۸×۲۸×۱ مناسب است. این لایه از حالت پدینگ (padding) همانند (same) استفاده میکند که به این معنی است که اندازه خروجی همان اندازه ورودی خواهد بود.

- model.add(MaxPooling2D(pool_size=(2, 2))) یک لایه حداکثر تجمیع را به مدل اضافه میکند که اندازه پنجره تجمیع را ۲×۲ در نظر میگیرد. این لایه باعث کاهش اندازه ویژگیها و کاهش پیچیدگی محاسباتی میشود.

- model.add(Conv2D(64, 3, padding="same", activation='relu')) یک لایه پیچشی دیگر را به مدل اضافه میکند که ۶۴ فیلتر با اندازه ۳×۳ دارد. این لایه نیز از تابع فعالسازی رلو و حالت پدینگ همانند استفاده میکند.

• model.add(MaxPooling2D(pool_size=(2, 2))) یک لایه حداکثر تجمیع دیگر را به مدل اضافه میکند که اندازه پنجره تجمیع را ۲×۲ در نظر میگیرد.

• model.add(Conv2D(64, 3, padding="same", activation='relu')) یک لایه پیچشی سوم را به مدل اضافه میکند که ۶۴ فیلتر با اندازه ۳×۳ دارد. این لایه نیز از تابع فعالسازی رلو و حالت پدینگ همانند استفاده میکند.

• model.add(MaxPooling2D(pool_size=(2, 2))) یک لایه حداکثر تجمیع سوم را به مدل اضافه میکند که اندازه پنجره تجمیع را ۲×۲ در نظر میگیرد.

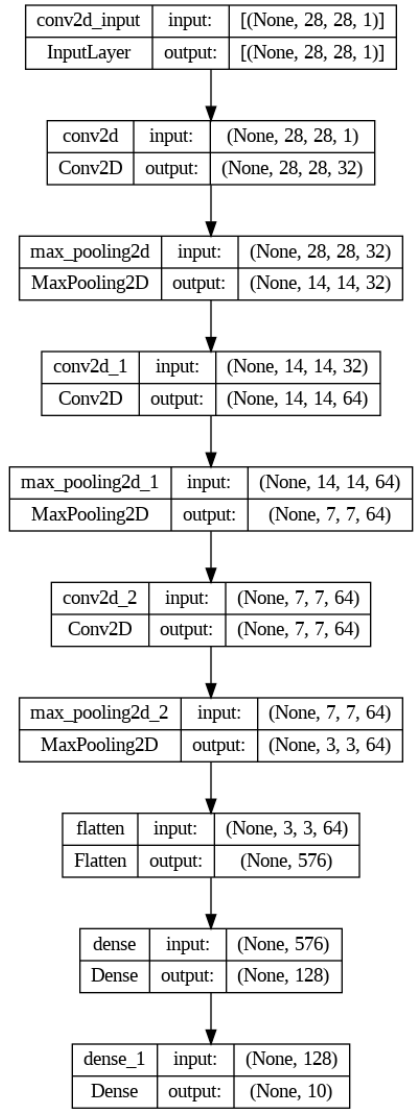
• model.add(Flatten()) یک لایه فلتن را به مدل اضافه میکند که خروجی لایه قبلی را به یک بردار یک بعدی تبدیل میکند. این لایه برای اتصال لایههای پیچشی به لایههای تماما متصل لازم است.

• model.add(Dense(128, activation='relu')) یک لایه تماما متصل را به مدل اضافه میکند که ۱۲۸ نورون دارد و از تابع فعالسازی رلو استفاده میکند. این لایه برای یادگیری ویژگیهای سطح بالاتر از دادهها مفید است.

• model.add(Dense(10, activation='softmax')) یک لایه تماما متصل دیگر را به مدل اضافه میکند که ۱۰ نورون دارد و از تابع فعالسازی سافتمکس (Softmax) استفاده میکند. این لایه برای دستهبندی دادهها به ۱۰ کلاس مختلف مناسب است.

```python
model=Sequential()
model.add(Conv2D(32, 3, padding="same", activation='relu', input_shape=(28, 28, 1)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, 3, padding="same", activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, 3, padding="same", activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(10, activation='softmax'))

summary = model.summary()
print(summary)
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 32)        320

max_pooling2d (MaxPooling2   (None, 14, 14, 32)        0
D)

conv2d_1 (Conv2D)            (None, 14, 14, 64)        18496

max_pooling2d_1 (MaxPoolin   (None, 7, 7, 64)          0
g2D)

conv2d_2 (Conv2D)            (None, 7, 7, 64)          36928

max_pooling2d_2 (MaxPoolin   (None, 3, 3, 64)          0
g2D)

flatten (Flatten)            (None, 576)               0

dense (Dense)                (None, 128)               73856

dense_1 (Dense)              (None, 10)                1290

...
Trainable params: 130890 (511.29 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```



مدل ساخته شده را با مشخصات زیر آموزش دهید.

سلام، این بینگ است. من میتوانم کد را برای شما توضیح بدهم. blush

کد شما یک مدل شبکه عصبی پیچشی را برای دستهبندی تصاویر دستنوشته اعداد از ۰ تا ۹ آموزش میدهد. کد شما از چهار قسمت تشکیل شده است:

- model.compile: این قسمت مشخص میکند که مدل چگونه آموزش داده شود. شما سه پارامتر را تعیین کردهاید:

• optimizer=Adam(learning_rate=1e-3): این پارامتر مشخص میکند که مدل از الگوریتم بهینهسازی آدام با نرخ یادگیری $$۱۰^{-۳}$$ استفاده کند. الگوریتم آدام یکی از رایجترین و موثرترین الگوریتمهای بهینهسازی برای شبکههای عصبی است.

• loss=CategoricalCrossentropy(): این پارامتر مشخص میکند که مدل از تابع هزینه انتروپی متقاطع چنددستهای استفاده کند. این تابع هزینه میزان اختلاف بین پیشبینیهای مدل و برچسبهای واقعی را اندازهگیری میکند. هدف مدل کاهش این هزینه است.

• metrics=['accuracy']: این پارامتر مشخص میکند که مدل از معیار دقت برای ارزیابی عملکرد خود استفاده کند. دقت نسبت تعداد پیشبینیهای صحیح به تعداد کل پیشبینیها است.

• num_classes = 10: این قسمت تعداد دستههای مختلف را مشخص میکند. در این مورد، مدل باید بین ۱۰ دسته از اعداد ۰ تا ۹ تشخیص دهد.

• E = 15: این قسمت تعداد دورههای آموزش را مشخص میکند. یک دوره به معنای این است که مدل تمام دادههای آموزش را یک بار میبیند و پارامترهای خود را بهروز میکند.

• BATCH_SIZE = 64: این قسمت اندازه دستههای آموزش را مشخص میکند. یک دسته شامل چند نمونه داده است که مدل به طور همزمان میبیند و گرادیان هزینه را برای آنها محاسبه میکند. اندازه دسته تأثیر بسزایی بر سرعت و کیفیت آموزش مدل دارد.

• history=model.fit(x_train.reshape(-1, 28, 28, 1), y_train, epochs=E, batch_size=BATCH_SIZE , validation_data=(x_test.reshape(-1, 28, 28, 1), y_test)): این قسمت مرحله اصلی آموزش مدل را انجام میدهد. شما پنج پارامتر را تعیین کردهاید:

• x_train.reshape(-۱, ۲۸, ۲۸, ۱): این پارامتر دادههای ورودی آموزش را مشخص میکند. شما دادههای خود را به شکل مناسب برای مدل تغییر دادهاید. دادههای شما تصاویر دستنوشته با ابعاد

• y_train: این پارامتر برچسبهای واقعی دادههای آموزش را مشخص میکند. برچسبها نشان میدهند که هر تصویر متعلق به کدام دسته است.

• epochs=E: این پارامتر تعداد دورههای آموزش را مشخص میکند. شما از مقدار قبلی که در قسمت E = 15 تعریف کردهاید استفاده کردهاید.

- batch_size=BATCH_SIZE: این پارامتر اندازه دستههای آموزش را مشخص میکند. شما از مقدار قبلی که در قسمت BATCH_SIZE = 64 تعریف کردهاید استفاده کردهاید.

- validation_data=(x_test.reshape(-1, 28, 28, 1), y_test): این پارامتر دادههای اعتبارسنجی را مشخص میکند. دادههای اعتبارسنجی برای ارزیابی عملکرد مدل بر روی دادههای جدید استفاده میشوند. شما دادههای ورودی و برچسبهای اعتبارسنجی را به همان شکلی که برای دادههای آموزش انجام دادهاید، تغییر دادهاید.

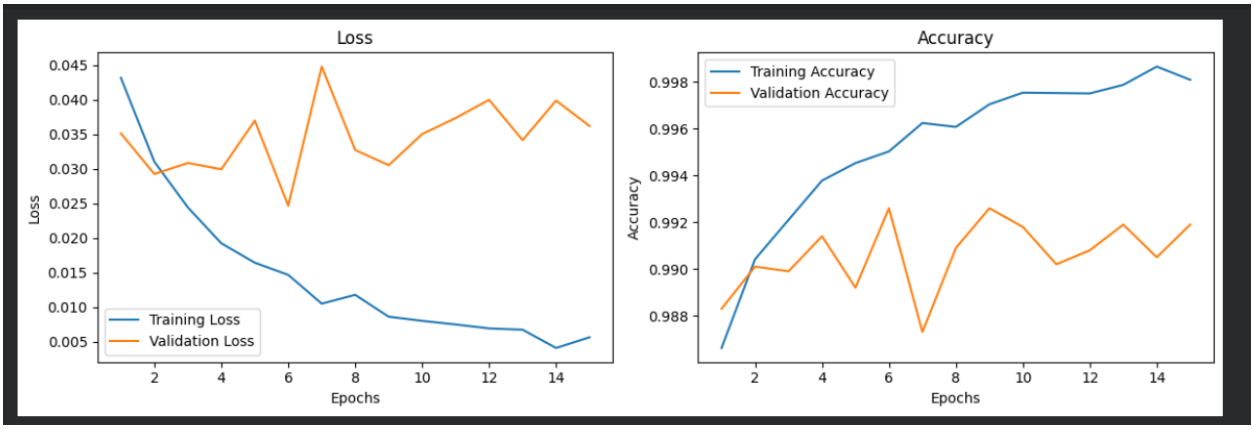
```
model.compile(
    optimizer=Adam(learning_rate=1e-3),
    loss=CategoricalCrossentropy(),
    metrics=['accuracy']
    )
```

```
num_classes = 10
E = 15
BATCH_SIZE = 64
```

```
history=model.fit(x_train.reshape(-1, 28, 28, 1), y_train, epochs=E, batch_size=BATCH_SIZE , validation_data=(x_test.reshape(-1, 28, 28, 1), y_test))
```

```
Epoch 1/15
938/938 [==============================] - 96s 102ms/step - loss: 0.0432 - accuracy: 0.9866 - val_loss: 0.0351 - val_accuracy: 0.9883
Epoch 2/15
938/938 [==============================] - 97s 104ms/step - loss: 0.0310 - accuracy: 0.9904 - val_loss: 0.0292 - val_accuracy: 0.9901
Epoch 3/15
938/938 [==============================] - 93s 99ms/step - loss: 0.0244 - accuracy: 0.9921 - val_loss: 0.0308 - val_accuracy: 0.9899
Epoch 4/15
938/938 [==============================] - 91s 97ms/step - loss: 0.0192 - accuracy: 0.9938 - val_loss: 0.0299 - val_accuracy: 0.9914
Epoch 5/15
938/938 [==============================] - 92s 98ms/step - loss: 0.0164 - accuracy: 0.9945 - val_loss: 0.0370 - val_accuracy: 0.9892
Epoch 6/15
938/938 [==============================] - 102s 109ms/step - loss: 0.0147 - accuracy: 0.9950 - val_loss: 0.0247 - val_accuracy: 0.9926
Epoch 7/15
938/938 [==============================] - 105s 112ms/step - loss: 0.0105 - accuracy: 0.9962 - val_loss: 0.0448 - val_accuracy: 0.9873
Epoch 8/15
938/938 [==============================] - 104s 111ms/step - loss: 0.0118 - accuracy: 0.9961 - val_loss: 0.0327 - val_accuracy: 0.9909
Epoch 9/15
938/938 [==============================] - 108s 115ms/step - loss: 0.0086 - accuracy: 0.9970 - val_loss: 0.0305 - val_accuracy: 0.9926
Epoch 10/15
938/938 [==============================] - 121s 129ms/step - loss: 0.0080 - accuracy: 0.9976 - val_loss: 0.0350 - val_accuracy: 0.9918
Epoch 11/15
938/938 [==============================] - 106s 113ms/step - loss: 0.0075 - accuracy: 0.9975 - val_loss: 0.0373 - val_accuracy: 0.9902
Epoch 12/15
938/938 [==============================] - 115s 122ms/step - loss: 0.0069 - accuracy: 0.9975 - val_loss: 0.0400 - val_accuracy: 0.9908
Epoch 13/15
...
Epoch 14/15
938/938 [==============================] - 111s 118ms/step - loss: 0.0041 - accuracy: 0.9987 - val_loss: 0.0399 - val_accuracy: 0.9905
Epoch 15/15
```

رسم توابع لاس و دقت برای داده های train , test مدل از عملکرد مناسبی برخوردار است و همانگونه که انتظار می رفت لاس در داده های تست بالا تر از دقت و در داده های ترین دقت بالا تر است.:
اما به میزان خوبی از دقت در داده ی تست رسیدیم که عملکرد خوب مدل را نمایش می دهد.



الگوریتم CAM – Grad را بر روی آخرین لایه هم گشتی اجرا کرده و خروجی آن را برای ۱۰ تصویر نمونه نمایش دهید.

```
model=Sequential()
model.add(Conv2D(32, 3, padding="same", activation='relu', input_shape=(28, 28, 1)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, 3, padding="same", activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, 3, padding="same", activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(10, activation='softmax'))

summary = model.summary()
print(summary)
```

[12]

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 32)        320

 max_pooling2d (MaxPooling2  (None, 14, 14, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 14, 14, 64)        18496

 max_pooling2d_1 (MaxPoolin  (None, 7, 7, 64)          0
 g2D)

 conv2d_2 (Conv2D)           (None, 7, 7, 64)          36928

 max_pooling2d_2 (MaxPoolin  (None, 3, 3, 64)          0
 g2D)

 flatten (Flatten)           (None, 576)               0

 dense (Dense)               (None, 128)               73856

 dense_1 (Dense)             (None, 10)                1290

...
Trainable params: 130890 (511.29 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```python
model.compile(
    optimizer=Adam(learning_rate=1e-3),
    loss=CategoricalCrossentropy(),
    metrics=['accuracy']
)
```

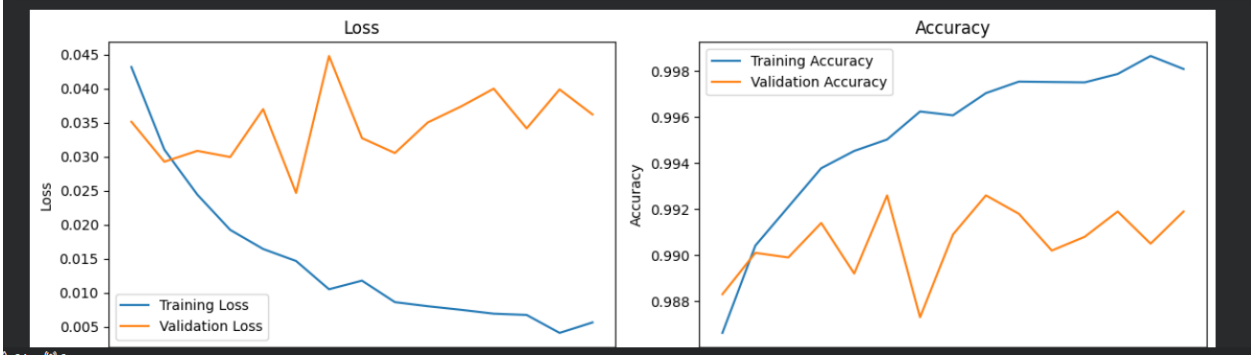
```python
num_classes = 10
E = 15
BATCH_SIZE = 64
```

```python
history=model.fit(x_train.reshape(-1, 28, 28, 1), y_train, epochs=E, batch_size=BATCH_SIZE , validation_data=(x_test.reshape(-1, 28, 28, 1), y_test))
```

```
Epoch 1/15
938/938 [==============================] - 96s 102ms/step - loss: 0.0432 - accuracy: 0.9866 - val_loss: 0.0351 - val_accuracy: 0.9883
Epoch 2/15
938/938 [==============================] - 97s 104ms/step - loss: 0.0310 - accuracy: 0.9904 - val_loss: 0.0292 - val_accuracy: 0.9901
Epoch 3/15
938/938 [==============================] - 93s 99ms/step - loss: 0.0244 - accuracy: 0.9921 - val_loss: 0.0308 - val_accuracy: 0.9899
Epoch 4/15
938/938 [==============================] - 91s 97ms/step - loss: 0.0192 - accuracy: 0.9938 - val_loss: 0.0299 - val_accuracy: 0.9914
Epoch 5/15
938/938 [==============================] - 92s 98ms/step - loss: 0.0164 - accuracy: 0.9945 - val_loss: 0.0370 - val_accuracy: 0.9892
Epoch 6/15
938/938 [==============================] - 102s 109ms/step - loss: 0.0147 - accuracy: 0.9950 - val_loss: 0.0247 - val_accuracy: 0.9926
Epoch 7/15
938/938 [==============================] - 105s 112ms/step - loss: 0.0105 - accuracy: 0.9962 - val_loss: 0.0448 - val_accuracy: 0.9873
Epoch 8/15
938/938 [==============================] - 104s 111ms/step - loss: 0.0118 - accuracy: 0.9961 - val_loss: 0.0327 - val_accuracy: 0.9909
Epoch 9/15
938/938 [==============================] - 108s 115ms/step - loss: 0.0086 - accuracy: 0.9970 - val_loss: 0.0305 - val_accuracy: 0.9926
Epoch 10/15
938/938 [==============================] - 121s 129ms/step - loss: 0.0080 - accuracy: 0.9976 - val_loss: 0.0350 - val_accuracy: 0.9918
Epoch 11/15
938/938 [==============================] - 106s 113ms/step - loss: 0.0075 - accuracy: 0.9975 - val_loss: 0.0373 - val_accuracy: 0.9902
Epoch 12/15
938/938 [==============================] - 115s 122ms/step - loss: 0.0069 - accuracy: 0.9975 - val_loss: 0.0400 - val_accuracy: 0.9908
Epoch 13/15
...
Epoch 14/15
938/938 [==============================] - 111s 118ms/step - loss: 0.0041 - accuracy: 0.9987 - val_loss: 0.0399 - val_accuracy: 0.9905
Epoch 15/15
938/938 [==============================] - 104s 111ms/step - loss: 0.0056 - accuracy: 0.9981 - val_loss: 0.0362 - val_accuracy: 0.9919
```

84 ⚠ 0

```python
train_loss = history.history['loss']
val_loss = history.history['val_loss']

# Get the training and validation accuracy from the history object
train_acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

# Plot the training and validation loss
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(range(1, len(train_loss) + 1), train_loss, label='Training Loss')
plt.plot(range(1, len(val_loss) + 1), val_loss, label='Validation Loss')
plt.title('Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

# Plot the training and validation accuracy
plt.subplot(1, 2, 2)
plt.plot(range(1, len(train_acc) + 1), train_acc, label='Training Accuracy')
plt.plot(range(1, len(val_acc) + 1), val_acc, label='Validation Accuracy')
plt.title('Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

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



یک نقشه حرارتی را برای نشان دادن میزان تأثیر هر پیکسل تصویر بر پیشبینی مدل ایجاد میکند. کد شما از چند قسمت تشکیل شده است:

- last_conv_layer = model.layers[-۵]: این قسمت لایه پیچشی آخر مدل را درون یک متغیر ذخیره میکند. این لایه خروجیهایی را تولید میکند که نشاندهنده ویژگیهای مهم تصویر هستند.

- grad_model = tf.keras.models.Model([model.inputs], [model.output, last_conv_layer.output]): این قسمت یک مدل جدید را با ورودیها و خروجیهای مورد نظر ایجاد میکند. این مدل خروجیهای لایه پیچشی آخر و لایه خروجی اصلی مدل را برمیگرداند.

- with tf.GradientTape() as tape:: این قسمت یک شیء گرادیاننوار را ایجاد میکند که میتواند گرادیانهای هزینه را نسبت به خروجیهای مدل محاسبه کند.

- predictions, conv_outputs = grad_model(x_test[:10].reshape(-1, 28, 28, 1)): این قسمت ده تصویر اول از دادههای آزمون را به مدل جدید میدهد و خروجیهای مربوطه را درون دو متغیر ذخیره میکند.

- sam =predictions[۰]: این قسمت پیشبینی مدل برای تصویر اول را درون یک متغیر ذخیره میکند.

- loss = predictions[:, np.argmax(sam)]: این قسمت هزینه را برای هر تصویر محاسبه میکند. هزینه برابر با احتمال پیشبینی شده برای دستهای است که مدل برای تصویر اول انتخاب کرده است.

- grads = tape.gradient(loss, conv_outputs): این قسمت گرادیانهای هزینه را نسبت به خروجیهای لایه پیچشی آخر محاسبه میکند. این گرادیانها نشان میدهند که هر خروجی چقدر بر هزینه تأثیر دارد.

- temp = K.mean(grads, axis=(0, 1, 2)): این قسمت میانگین گرادیانها را برای هر فیلتر پیچشی محاسبه میکند. این میانگین نشان میدهد که هر فیلتر چقدر بر پیشبینی مدل تأثیر دارد.

- heatmap = tf.reduce_mean(tf.multiply(temp, conv_outputs), axis=-1): این قسمت نقشه حرارتی را برای هر تصویر محاسبه میکند. نقشه حرارتی برابر با حاصلضرب میانگین گرادیانها و خروجیهای لایه پیچشی آخر است. این نقشه حرارتی نشان میدهد که هر پیکسل تصویر چقدر بر پیشبینی مدل تأثیر دارد.

- heatmap = np.maximum(heatmap, 0): این قسمت مقادیر منفی نقشه حرارتی را صفر میکند. این کار برای حذف پیکسلهایی است که بر پیشبینی مدل تأثیر منفی دارند.

- heatmap /= np.max(heatmap): این قسمت نقشه حرارتی را نرمال میکند. این کار برای تبدیل مقادیر نقشه حرارتی به بازه است.

```
last_conv_layer = model.layers[-5]

grad_model = tf.keras.models.Model([model.inputs], [model.output,
last_conv_layer.output])

with tf.GradientTape() as tape:
    predictions, conv_outputs = grad_model(x_test[:10].reshape(-1, 28, 28,
1))
    sam =predictions[0]
    loss = predictions[:, np.argmax(sam)]
    grads = tape.gradient(loss, conv_outputs)
```

```
    temp = K.mean(grads, axis=(0, 1, 2))

heatmap = tf.reduce_mean(tf.multiply(temp, conv_outputs), axis=-1)
heatmap = np.maximum(heatmap, 0)
heatmap /= np.max(heatmap)
```

کد شما یک نمودار را برای نشان دادن تصاویر دستنوشته و نقشه حرارتی مربوط به آنها رسم میکند. کد شما از چند قسمت تشکیل شده است:

- fig = plt.figure(figsize=(10, 10)): این قسمت یک شکل جدید با اندازه $$۱۰ \times 10$$ اینچ ایجاد میکند. این شکل میتواند چندین زیرنمودار را در خود جای دهد

- for img in range(۱۰):: این قسمت یک حلقه برای پیمایش روی ده تصویر اول از دادههای آزمون ایجاد میکند.

- plt.figure(): این قسمت یک شکل جدید با اندازه پیشفرض ایجاد میکند. این شکل فقط یک زیرنمودار را در خود جای میدهد ./

- plt.subplot(۳, ۲, ۱): این قسمت یک زیرنمودار را در شکل جاری ایجاد میکند. این زیرنمودار در سطر اول و ستون اول از یک شبکه
- plt.imshow(x_test[img].squeeze(), cmap='gray'): این قسمت تصویر دستنوشته را در زیرنمودار جاری نشان میدهد. این تصویر با رنگبندی خاکستری نمایش داده میشودh.

- s=np.argmax(y_test[img]): این قسمت برچسب واقعی تصویر را محاسبه میکند. برچسب واقعی برابر با اندیس بزرگترین عنصر در آرایه $$y_test[img]$$ است.

- plt.title(f" LABLE :{s}"): این قسمت عنوان زیرنمودار را تنظیم میکند. عنوان شامل کلمه LABLE و برچسب واقعی تصویر است.

- plt.subplot(۳, ۲, ۲): این قسمت یک زیرنمودار دیگر را در شکل جاری ایجاد میکند. این زیرنمودار در سطر اول و ستون دوم از یک شبکه $$۳ \times 2$$ قرار میگیرد.

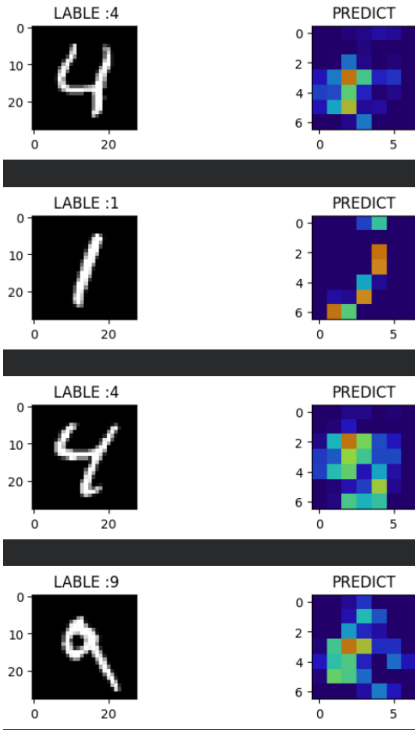
- plt.imshow(heatmap[img]): این قسمت نقشه حرارتی تصویر را در زیرنمودار جاری نشان میدهد. این نقشه حرارتی با رنگبندی پیشفرض نمایش داده

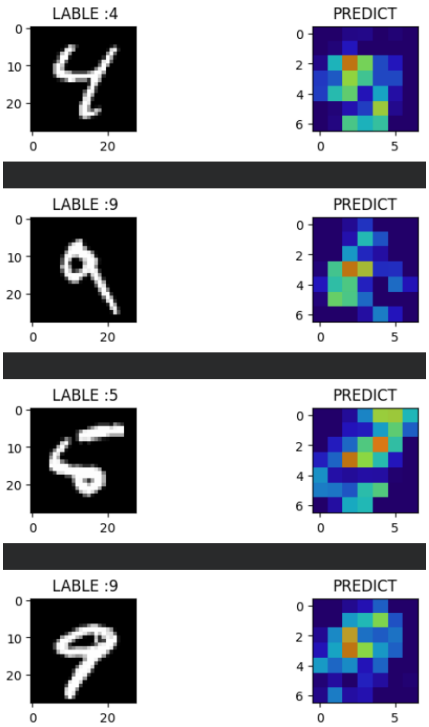
- plt.imshow(heatmap[img], cmap='jet', alpha=0.5): این قسمت نقشه حرارتی تصویر را دوباره در زیرنمودار جاری نشان میدهد. این بار با رنگبندی جت و شفافیت $$۰٫۵$$ نمایش داده میشود. این کار باعث میشود که نقشه حرارتی روی تصویر دستنوشته قرار گیرد

- plt.title("PREDICT"): این قسمت عنوان زیرنمودار را تنظیم میکند. عنوان شامل کلمه PREDICT است.

- plt.tight_layout(): این قسمت فاصله بین زیرنمودارها را بهینه میکند.

- plt.show(): این قسمت نمودار را نمایش میدهد.





خروجی الگوریتم را تحلیل کنید و نتیجه ی استفاده از CAM–Grad در تشخیص ویژگی های مهم در تصاویر MNIST را بررسی کنید:

این روش، یکی از معروف‌ترین و مجبوب‌ترین تکنیک‌های توسعه داده شده جهت تفسیر پیش‌بینی‌های تولید شده توسط مدل های یادگیری عمیق در بینایی کامپیوتر است. در این روش، از طریق بررسی گرادیان‌های پس انتشار شده (انتشار داده شده در جهت عکس جریان اطلاعاتی شبکه عصبی) به سمت «نقشه‌های فعال‌سازی کلاسی(Class Activation Maps | CAM) «، میزان تأثیر بخش‌های سمختلف تصویر در خروجی (پیش‌بینی) تولید شده توسط شبکه عصبی محاسبه می‌گردد.
نقشه‌های فعال‌سازی کلاسی، روش‌های ساده‌ای هستند و هدف آن‌ها، پیدا کردن نواحی «متمایزگر « (Discriminative)در تصویر است. این نواحی، توسط مدل شبکه عصبی پیچشی جهت شناسایی یک کلاس خاص در تصویر (و تخصیص برچسب کلاسی متناظر با آن کلاس به تصویر) استفاده می‌شوند. به عبارت دیگر، این روش به کاربران اجازه می‌دهد تا نواحی مرتبط با یک کلاس خاص در هر تصویر را شناسایی و مشاهده کنند.
خروجی این روش، پیکسل‌هایی خواهد بود که بیشترین تأثیرگذاری را در «بیشینه‌سازی (Maximization) « تابع هدف دارند.
برای مشخص کردن پیکسل‌های مرتبط با هر کلاس خاص در تصاویر ورودی، کافی است تا گرادیان‌های تابع هدف نسبت به خروجی‌های لایه‌های کانولوشن محاسبه شود. چنین کاری، در هنگام پس انتشار خطا به راحتی قابل انجام است.
به عنوان یک متخصص یادگیری عمیق، این مسئولیت شماست که مطمئن شوید مدل شما به درستی عمل می کند. یکی از راه‌هایی که می‌توانید این کار را انجام دهید این است که مدل خود را اشکال‌زدایی کنید و اعتبار بصری آن را تأیید کنید که در مکان‌های صحیح یک تصویر «به‌نظر می‌رسد» و «فعال می‌شود».
تصویری از شبکه های عمیق از طریق محلی سازی مبتنی بر گرادیان منتشر کرد.
این روش عبارت است از:

به راحتی اجرا می شود
تقریباً با هر معماری شبکه عصبی کانولوشنال کار می کند
می توان از آن برای اشکال زدایی بصری جایی که یک شبکه در یک تصویر نگاه می کند استفاده کرد
Grad-CAM با (۱) یافتن لایه کانولوشنال نهایی در شبکه و سپس (۲) بررسی اطلاعات گرادیان جریان در آن لایه کار می کند.

خروجی Grad-CAM یک تجسم نقشه حرارتی برای یک برچسب کلاس مشخص است (یا برچسب بالا، پیش بینی شده یا یک برچسب دلخواه که برای اشکال زدایی انتخاب می کنیم). ما می‌توانیم از این نقشه حرارتی برای بررسی بصری این تصویر استفاده کنیم که CNN به کجای تصویر نگاه می‌کند.
با استفاده از Grad-CAM، می‌توانیم به صورت بصری اعتبار شبکه ما را به کجا نگاه می‌کند، تأیید کنیم که واقعاً به الگوهای صحیح در تصویر نگاه می‌کند و در اطراف آن الگوها فعال می‌شود.

اگر شبکه حول الگوها/اشیاء مناسب در تصویر فعال نمی شود، می دانیم:

1.شبکه ما الگوهای اساسی در مجموعه داده ما را به درستی یاد نگرفته است
2.روند آموزشی ما نیاز به بازنگری دارد
3.ممکن است نیاز به جمع آوری داده های اضافی داشته باشیم
4.و مهمتر از همه، مدل ما برای استقرار آماده نیست.

همانطور که در شکل ها میبینیم در نواحی ای که پیکسل های سفید مربوط به عدد را داشتیم بیش ترین تاثیر پذیری بر روی تصمیم گیری اخر لایه اخر کانولوسنال داشته اند و بیشتر از همه فعال شده اند و پررنگ ترنداما پیکسل های حاشیه در هیچ تصویر فعال نشده اند از این طریق این طبقه بندی انجام شده است که کدام پیکسل ها در تصمیم گیری تاثیز گذارند و نقاطی پررنگ شده اند که تقریبا تا حدود خوبی با اعداد ما مپ میشونداز این رو برای بهبود عملکرد مدل یا تغییر ان اقدام می کنیم.

https://notebook.community/raghakot/keras-vis/examples/mnist/attention

/https://keras.io/examples/vision/grad_cam
https://notebook.community/raghakot/keras-vis/examples/mnist/attention