به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین پنجم

پرسش ۱	نام و نام خانوادگی	آرمین قاسمی
	شماره دانشجویی	810100198
پرسش ۲	نام و نام خانوادگی	امیرحسین ثمودی
	شماره دانشجویی	810100108
	مهلت ارسال پاسخ	**3/1404**

فهرست

شکل‌ها

جدول‌ها

پرسش 1 – طبقه بندی تصاویر با ViT

1-1: مقدمه

در این تمرین می خواهیم با توجه به دستور کار از دو مدل شبکه عصبی یکی مبتنی بر transformer ها (ViT) و یکی مبتنی بر CNN ها (Inception V3) برای تشخیص بیماری های گیاه گوجه فرنگی از روی تصویر برگ گیاه استفاده کنیم.

- **با توجه به مقاله، اصلی ترین تفاوت و مزیت استفاده از مدل های ویژن ترنسفورمر در مقایسه با مدل های سنتی چیست؟**

با توجه به چیزی که در مقاله مطرح شده، اصلی ترین تفاوت و مزیت مدل ViT نسبت به مدل های CNN سنتی، در نحوه پردازش تصویر و استفاده از مکانیزم self attention است.

می دانیم که مدل های CNN از عملیات کانولوشن برای استخراج ویژگی ها استفاده می‌کنند. در این روش انگار مدل دنبال الگو های کوچک و مشخص میگردد (مثلا لکه های قهوه ای کوچک، یا لبه های زرد شده) و هرجای تصویر که این الگوها پیدا شدند، مدل آنها را گزارش میدهد. به این مورد **بایاس القایی** (Inductive Bias) میگوییم. در واقع انگار به مدل می گوییم یک الگو هرجای تصویر باشد همان الگو است. در این روش مثلا اگر لکه ای در پایین برگ باشد فرقی با اینن ندارد که لکه در بالای برگ باشد، در واقع اینجا **هم‌ارزی انتقالی** (Translation Equivalence) داریم که نتیجه همان بایاس القایی است. یعنی موقعیت یا انتقال الگو در تصویر ماهیت آن را عوض نمیکند.

در عوض در مدل های ترنسفورمری مبنای کار ما مکانیزم توجه است. در این روش مدل به دنبال الگو های کوچک مشخص نمیگردد. در این روش انگار ما تصویر را تکه تکه می کنیم (به patch های یک اندازه تقسیم می کنیم) و همه تکه ها را جلوی مدل می گذاریم. انگار در این روش مدل ارتباط این تکه ها را با هم درک میکند. مثلا مدل ارتباط تکه ای که لکه دارد را با تکه ای که رگبرگ اصلی در آن است و تکه ای که زرد شده درک میکند و سپس می گویید که گیاه بیمار است یا نه. با این تفسیر مدل به کل تصویر نگاه می کند و ارتباط بخش های مختلف تصویر را باهم درک میکند و بدین ترتیب می تواند روابط کلی بین بخش های دور از هم را درک کند. مثلا ممکن است یک لکه به تنهایی معنی خاصی نداشته باشد ولی اگر در کنار رگبرگ اصلی باشد، نشانه بیماری باشد. در این موارد شبکه های ترنسفورمری تشخیص دقیق تری دارند.

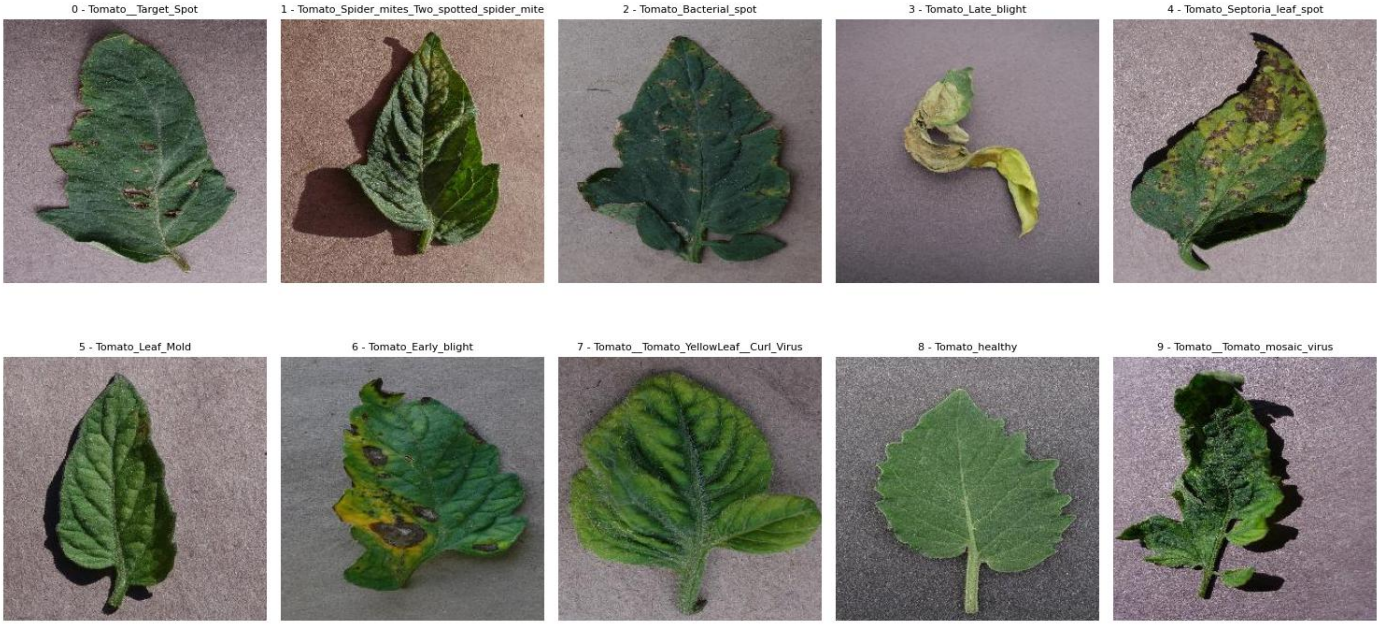
- **زمانی که دادگان موجود محدود است، کدام مدل بهتر عمل میکند؟ پاسخ خود را با توجه به ساختار و مکانیسم های به کار رفته در مدل ها توجیه کنید.**

با توجه به اینکه در مقاله مدل Inception V3 به صورت pre trained استفاده شده است و وزن های آن حین آموزش فریز شده اند و صرفا وزن های شبکه MLP آپدیت می شوند، قطعا این مدل نسبتا به مدل ترنسفورمری به داده کمتری برای آموزش نیاز خواهد داشت. با توجه به جدول 2 و جدول 4 موجود در مقاله، مدل Inception V3 به تعداد ۲۱،۸۰۲،۷۴۸ پارامتر دارد ولی این پارامتر ها در طول آموزش تمام فریز می شوند و شبکه MLP مربوط به head این شبکه است که آموزش می بیند. این شبکه تنها ۲0,490 پارامتر دارد. از طرفی مدل ViT به تعداد ۱۵،۲۳۴،۵۰۶ پارامتر دارد که این مقادیر تماما آموزش میبینند. بنابراین قطع به یقین مدل Inception V3 که pre train شده است برای آموزش به داده های کمتری نیاز خواهد داشت.

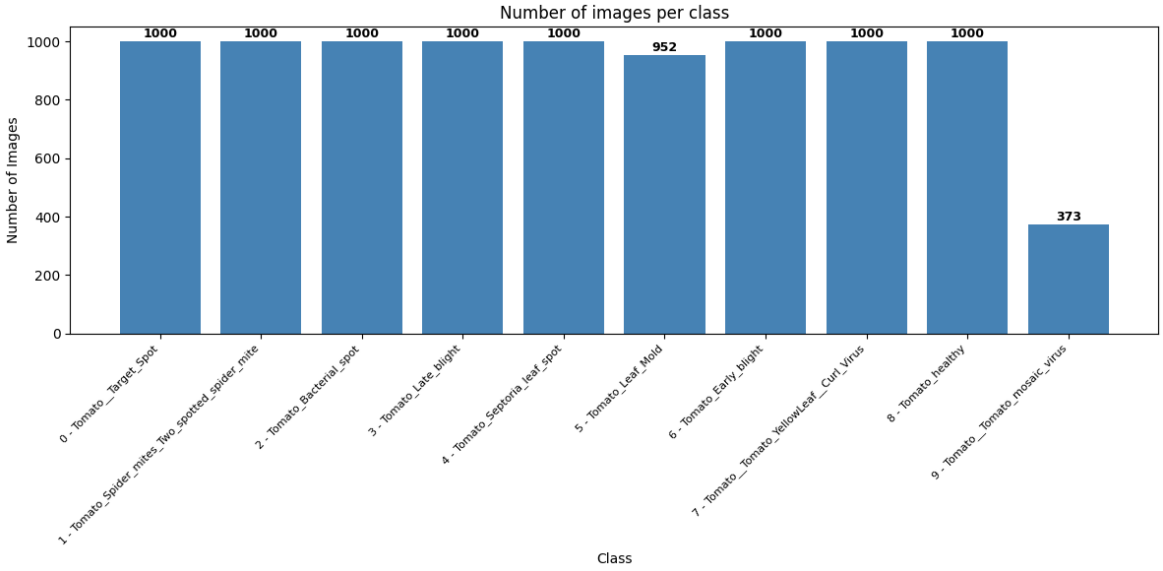
حال این مورد را بررسی میکنیم که اگر شبکه Inception V3 از پیش آموزش ندیده بود باز هم به داده های کمتری نیاز داشت؟ در نگاه اول ممکن است بنظر برسد با توجه به اینکه این شبکه حدود 6 میلیون پارامتر بیشتر از شبکه ViT دارد، بنابراین به تعداد داده های بیشتری نیز برای آموزش نیاز دارد ولی اینگونه نیست! مدل های CNN به طور ذاتی برای کار روی تصاویر طراحی شده اند. لایه‌های کانولوشنی به طور طبیعی الگوهایی مانند لبه‌ها، بافت‌ها، رنگ‌ها و... را جستجو می‌کنند و ساختار هندسی تصویر را حفظ می‌کنند. این ویژگی باعث می‌شود مدل حتی با داده‌های کمتر هم بتواند ویژگی‌های بصری را خیلی سریع یاد بگیرد. ولی یک مدل ترنسفورمری مانند ViT مدل هیچ دانش اولیه‌ای درباره ساختار دو بعدی تصاویر ندارد. این مدل یک تصویر را صرفاً به عنوان یک دنباله بلند از داده‌ها می‌بیند (مثل یک متن طولانی). ViT باید مفهوم پیکسل‌های همسایه، لبه‌ها، بافت ها، رنگ ها و... را کاملاً از صفر و فقط با دیدن داده‌های بسیار زیاد یاد بگیرد در صورتی که این موارد در معماری CNN ها به صورت ذاتی وجود دارد. بنابراین بر خلاف انتظار ما حتی اگر هر دو مدل خام باشند، مدل ViT به داده های بیشتری جهت آموزش نیاز دارد.

2-1: آماده سازی داده ها

در این بخش ابتدا از هر کلاس یک داده را نشان می دهیم و سپس به صورت نمودار میله ای توزیع داده ها در کلاس های مختلف را نشان میدهیم. این موارد در دو تصویر بعدی (شکل های1 و 2)آورده شده اند.



شکل 1: نمایش یک نمونه داد از هر کلاس



شکل 2: نمایش یک نمونه داد از هر کلاس

با توجه به این موارد تمامی کلاس ها داده های یکسانی دارند بجز کلاس شماره 9 که مربوط به بیماری Tomato__Tomato_mosaic_virus است. این کلاس تقریبا یک سوم بقیه کلاس ها داده دارد که باعث عدم تعادل در کلاس ها و بایاس شدن مدل می شود. بنابراین باید با استفاده از روش های مناسب داده ها را بیشتر کنیم. نکته مورد توجه این است که با توجه به تفاوت ساختار CNN و ViT ممکن است برخی از روش های Data augmentation برای یک مدل مفید و برای مدل دیگر آسیب زا باشد. بنابراین پیش

پردازش داده ها برای این دو مدل را به صورت جداگانه انجام میدهیم. از طرف دیگر با توجه به اینکه آموزش مناسب ترنسفورمر ها نیازمند داده های زیاد است، تصمیم میگیریم که با استفاده از برخی روش های Data augmentation تعداد داده های هر کلاس را تا 3000 داده افزایش دهیم. یعنی برای مدل ViT نهایتا 30000 داده خواهیم داشت که متعلق به 10 کلاس 3000 تایی خواهد بود. ولی با توجه به اینکه برای CNN ها تعداد داده ها نسبتا مناسب است بهتر است تعداد را زیاد نکنیم و برای CNN ها داده های هر کلاس را تا 1500 داد افزایش می دهیم تا کیفیت تصاویر حفظ شوند.

روی داده های مربوط به ViT کد رو به رو را اعمال میکنیم:

```
A.RandomResizedCrop(size=(256, 256), scale=(0.8, 1.0), p=0.7),
A.Rotate(limit=20, p=0.4),
A.HorizontalFlip(p=0.4),
A.ColorJitter(brightness=0.1, contrast=0.1, saturation=0.1, hue=0.1, p=0.5),
A.CoarseDropout(
    max_holes=8,
    max_height=25,
    max_width=25,
    p=0.5,
    fill_value=128
),
```

هر خط از این کد به این شکل عمل میکند:

`A.RandomResizedCrop(size=(256, 256), scale=(0.8, 1.0), p=0.7)` :

این بخش از کد به احتمال 0.7 یک بخش که از تصویر را میبرد و دوباره به اندازه اصلی resize میکند.

`A.Rotate(limit=20, p=0.4)` :

این بخش از کد به احتمال 0.4 تصویر را به اندازه نهایتا 20 درجه می چرخاند.

`A.HorizontalFlip(p=0.4)` :

این بخش از کد به احتمال 0.4 تصویر را به صورت افقی آینه میکند.

`A.ColorJitter(brightness=0.1, contrast=0.1, saturation=0.1, hue=0.1, p=0.5)` :

این بخش از کد با احتمال 0.5 این کار ها را انجام میدهد:

`brightness=0.1` : نور را تا 10 درصد کم یا زیاد میکند.

`contrast=0.1` : کنتراست (یا تضاد) بین نواحی تیره و روشن تصویر را تا 10٪ کم یا زیاد می کند.

`saturation=0.1` : غلظت (یا شادابی) رنگ ها را تا 10٪ کم یا زیاد می‌کند. با کاهش آن، تصویر به سمت سیاه‌وسفید میل می‌کند و با افزایش آن، رنگ‌ها پررنگ‌تر می‌شوند.

`A.CoarseDropout(max_holes=8,max_height=25,max_width=25,p=0.5,fill_value=128)` :

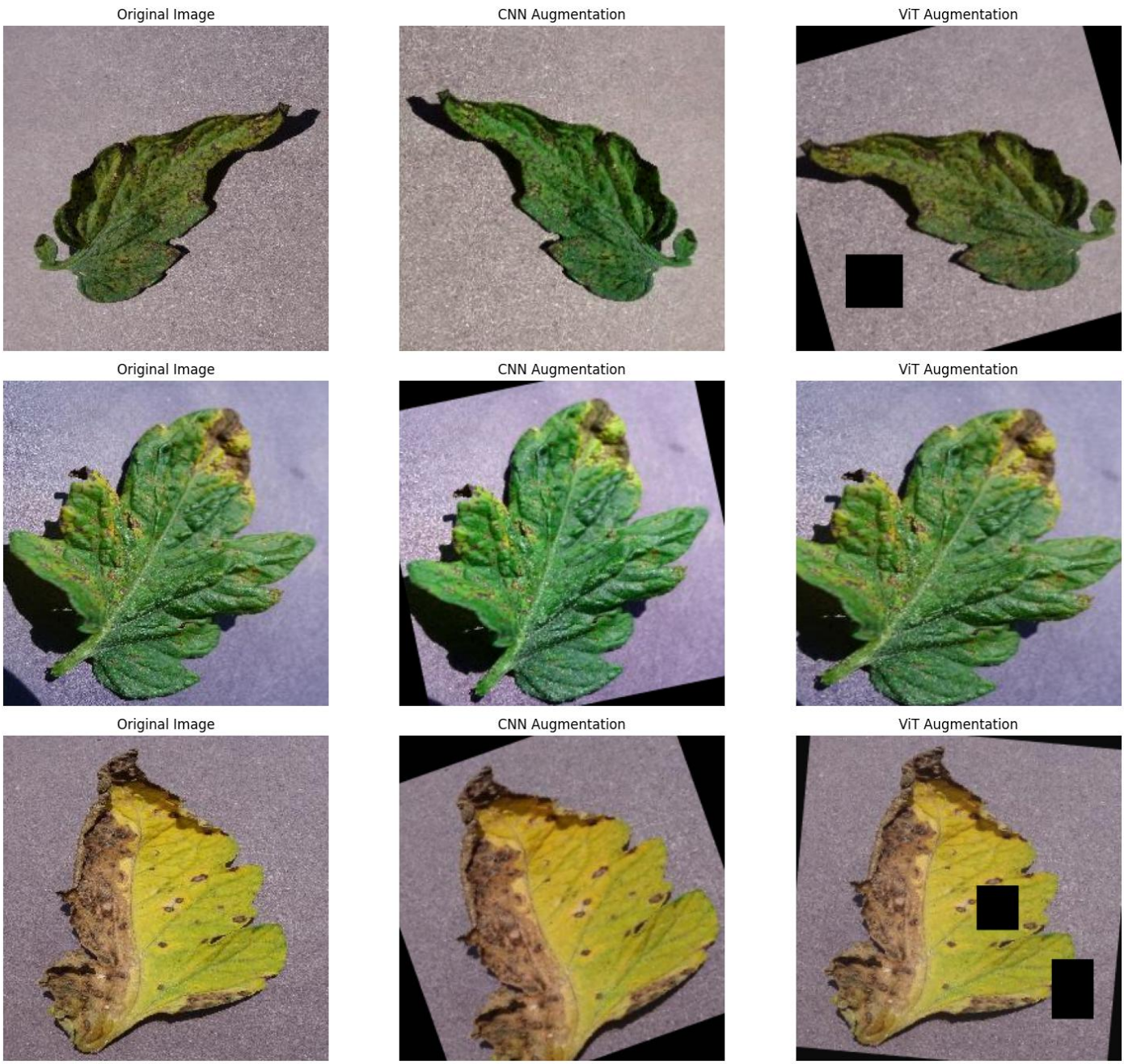
به احتمال 0.5 حداکثر به تعداد 8 مستطیل با حداکثر طول و عرض 25 را با مقدار 128 پر میکند(این بخش از تصویر را می پوشاند). این مورد در واقع نقطه تفاوت این تبدیلات با تبدیلات مربوط به CNN است. این تابع در واقع معادل تکنیک‌های معروف Cutout یا Random Erasing است که هدف اصلی آن، وادار کردن مدل به یادگیری از روی زمینه کلی تصویر است، نه اینکه فقط روی چند ویژگی خاص و محدود تمرکز کند. این روش برای ترنسفورمر ها عملکرد بسیار خوبی دارد. ترنسفورمر ها تصویر را به صورت دنباله‌ای از پچ‌ها می‌بینند. با این کار، ما عملاً یک یا چند پچ را از دید مدل مخفی می‌کنیم. این کار مدل را مجبور می‌کند تا برای تصمیم‌گیری به زمینه (context) و اطلاعات پچ‌های باقی‌مانده تکیه کند، که این دقیقاً همان کاری است که مکانیزم توجه به خود (self-attention) در ترنسفورمرها انجام می‌دهد. این روش یک تنظیم‌کننده (regularizer) بسیار قوی برای ترنسفورمر ها است.

روی داده های مربوط به CNN ها هم تبدیلات رو به رو را اعمال می کنیم:

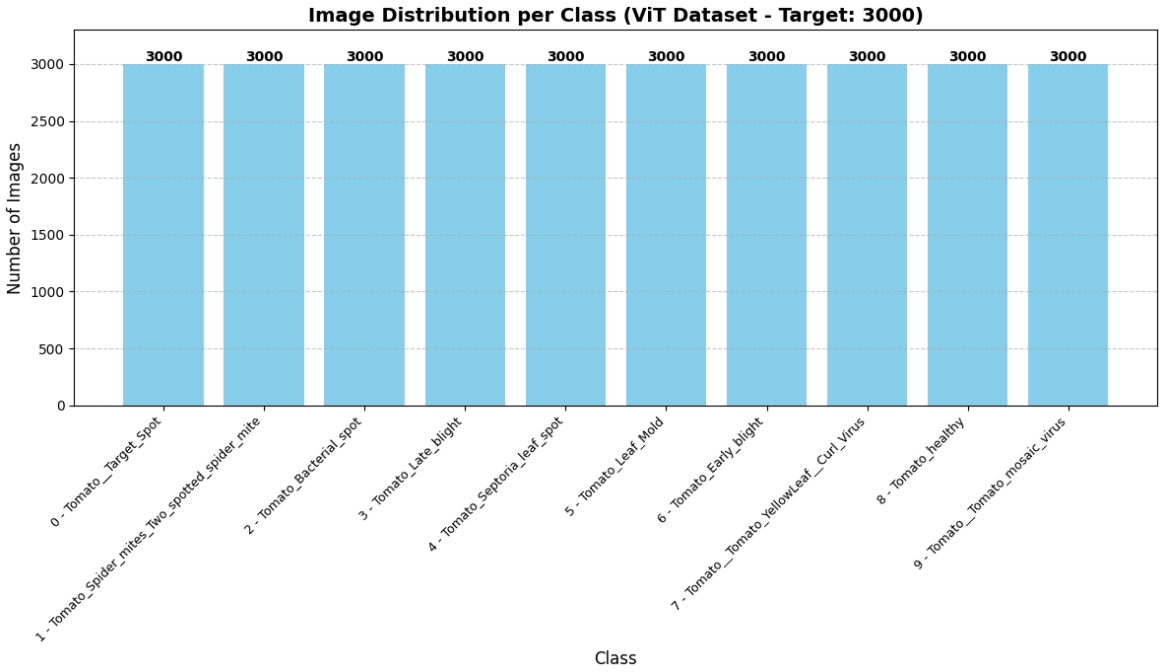
```
A.RandomResizedCrop(size=(256, 256), scale=(0.8, 1.0), p=1.0),
A.Rotate(limit=20, p=0.4),
A.HorizontalFlip(p=0.4),
A.ColorJitter(brightness=0.1, contrast=0.1, saturation=0.1, hue=0.1, p=0.7),
```

با توجه به این تبدیلات، واضحا تنها تغییر بین این تبدیلات و تبدیلات مربوط به ViT همان CoarseDropou است. در ادامه برخی داده از داده های اصلی را همراه با داده های augment شده از آنها نمایش میدهیم.

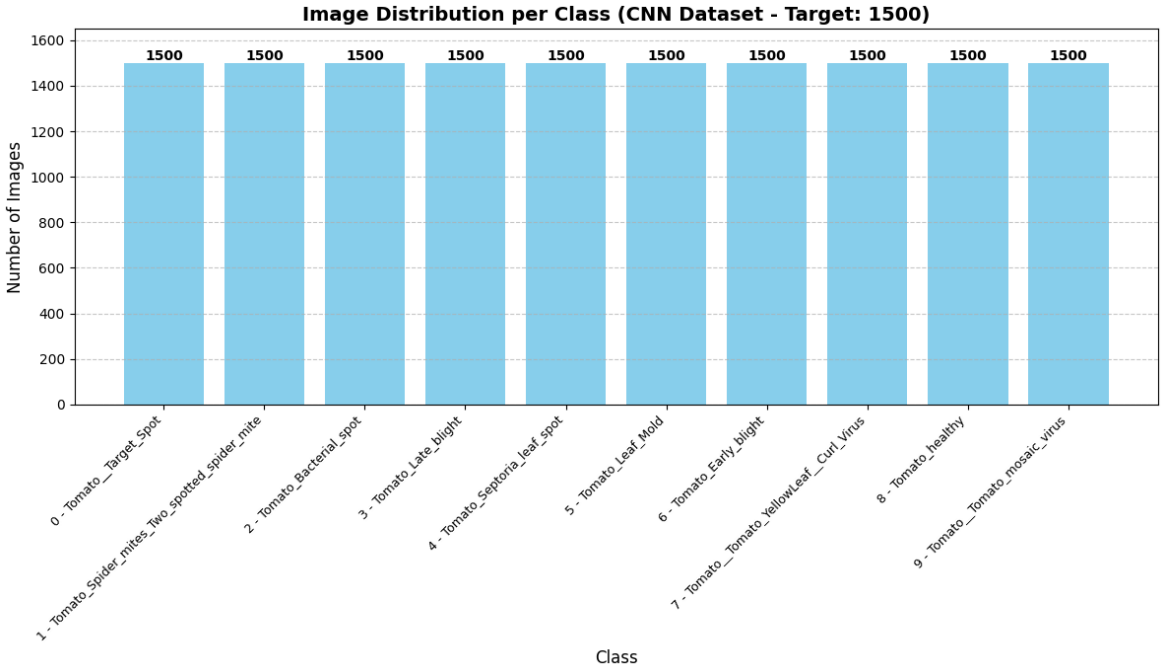
همینطور در ادامه مجددا نمودار ستونی توزیع داده ها در کلاس ها رسم شده است تا نشان داده شود داده ها به تعداد مورد نظر رسیده اند و تعداد آنها در کلاس ها بالانس شده است.



شکل 2: چند نمونه داده همراه با نسخه های **augment** شده از آنها



شکل 3:توزیع داده های مربوط به **ViT** در کلاس های مختلف



شکل 4: توزیع داده های مربوط به **CNN** در کلاس های مختلف

همانطور که مشخص است، همه چیز به درستی انجام شده است!

در ادامه بخش آماده سازی داده ها باید داده ها را resize و به دسته هی validation و train تقسیم کنیم. با توجه به اینکه در دستور کار صحبتی از داده های تست نشده ما نیز داده های تست را درنظر نمیگیریم و صرفا داده های ترین و ولیدیشن را درنظر میگیریم. با توجه به مقاله، 10٪ داده ها را به داده های ولیدیشن و 90٪ مابقی را به داده های ترین اختصاص میدهیم.

درباره سایز داده ها در مقاله ذکر شده که ورودی مدل ViT تصاویر را با اندازه 64 در 64 دریافت میکند و برای مدل Inception V3 چیزی در مقاله ذکر نشده است. با توجه به اینکه مدل Inception V3 روی داده های ImageNet آموزش دیده است و اندازه تصاویر این دیتاست برابر با 299 در 299 است، ما نیز داده های خود را برای این مدل با اندازه 299 در 299 resize میکنیم و سپس به مدل میدهیم. (درست است که ما نمی خواهیم از وزن های آموزش دیده توسط ImageNet استفاده کنیم و می توانیم Inception V3 را با هر انداه ورودی به کار بگیریم ولی باز بهتر است از این اندازه تبعیت کنیم). درباره مدل ViT نیز با توجه به اینکه داده های اصلی ما ابعاد 256 در 256 دارند و ما می خواهیم به ViT ورودی 64 در 64 بدهیم، باید ابعاد تصویر را یک چهارم کنیم. واضح است که این کار کیفیت تصاویر را بسیار خراب خواهد کرد و با این کار داده های بسیار زیادی را از دست خواهیم داد به همین منظور از داده های مربوط به ViT دو نسخه ایجاد میکنیم:

- در نسخه اول ابعاد تصاویر 256 در 256 است و اندازه تصاویر کوچکتر نمیشود تا دقت حفظ شود.
- در نسخه دوم ابعاد تصاویر 64 در 64 است تا متریک های بدست آمده با مقاله قابل قیاس باشند.

نهایتا مدل ViT را روی هر دو این داده ها فیت میکنیم و متریک های خروجی را ثبت میکنیم.

3-1: آموزش مدل CNN

مدل Inception V3 را بارگذاری کرده و روی خروجی آن یک لایه شبکه MLP برای طبقه بندی ایجاد میکنیم. summary مدل در صفحه بعد آورده شده است. مطابق summary ، شبکه MLP شامل یک لایه average pooling یک لایه مخفی با 512 نرون و یک لایه dropout است. حداکثر تعداد ایپاک ها را برابر با 30 ایپاک قرار میدهیم ولی از مکانیزم early stop برای جلوگیری از اورفیت شدن استفاده میکنیم. به این شکل که اگر در ده ایپاک متوالی مقدار loss روی داده های ولیدیشن کمتر نشد، یادگیری مدل متوقف و بهترین مدل ذخیره می‌شود. همچنین مطابق مقاله مقدار learning rate را برابر با 0.001 قرار میدهیم(البته مقاله در یک جای دیگر learning rate را 0.0001 بیان کرده است) ولی از مکانیزم کاهش learning rate نیز استفاده میکنیم. به این شکل که اگر در 5 ایپاک متوالی مقدار loss روی داده های

ولیدیشن کمتر نشد، learning rate نصف میشود. ولی مقدار learning rate نمی تواند از 0.00001 کوچکتر شود. توابع بهینه ساز و خطا نیز مطابق مقاله adam و crossentropy categorical انتخاب شده.

Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 299, 299, 3)	0
inception_v3 (Functional)	(None, 8, 8, 2048)	21,802,784
avg_pool (GlobalAveragePooling2D)	(None, 2048)	0
dense_hidden (Dense)	(None, 512)	1,049,088
dropout (Dropout)	(None, 512)	0
predictions (Dense)	(None, 10)	5,130

Total params: 22,857,002 (87.19 MB)
Trainable params: 22,822,570 (87.06 MB)
Non-trainable params: 34,432 (134.50 KB)

نحوه کارکرد کلی معماری Inception V3 بر استخراج سلسله‌مراتبی ویژگی‌ها است، به این صورت که لایه‌های اولیه شبکه، ویژگی‌های سطح پایین مانند لبه‌ها، بافت‌ها و رنگ‌ها را تشخیص می‌دهند و لایه‌های عمیق‌تر، این ویژگی‌های ساده را با یکدیگر ترکیب کرده و الگوهای پیچیده‌تر و معنادارتری را شناسایی می‌کنند.(مانند تشخیص لبه های زرد رنگ یک لکه روی برگ)

نوآوری کلیدی در این معماری، ماژول Inception است. معماری‌های CNN سنتی در هر لایه از یک اندازه فیلتر ثابت استفاده می‌کنند ولی ماژول Inception چندین عملیات کانولوشن با ابعاد مختلف کرنل (مانند 1×1, 3×3, 5×5)و یک عملیات Max-Pooling را به صورت موازی اجرا می‌کند. بعد نتایج حاصل از این مسیرهای پردازشی موازی را با یکدیگر الحاق (Concatenate) میکند و به عنوان خروجی آن لایه به لایه بعدی منتقل می کند. این پردازش به شبکه اجازه می‌دهد تا ویژگی‌ها را در سطوح و ابعاد مختلف به طور همزمان ثبت کند. این امر منجر به افزایش کارایی مدل می شود. در حالی که هزینه محاسباتی آنچنان افزایش پیدا نمی کند.

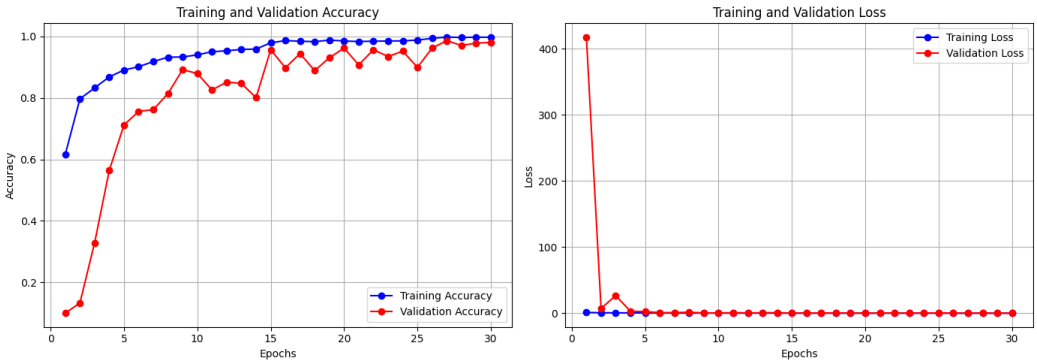
تابع خطای استفاده در مقاله را توضیح دهید. چه توابع دیگری برای این کار مناسب است؟

بر اساس مقاله مدل Inception V3 با استفاده از تابع هزینه categorical_crossentropy آموزش داده شده است. با توجه به اینکه ماهیت مسئله یک طبقه‌بندی چندکلاسه (multi-class classification) با ۱۰

کلاس مجزا است، این تابع هزینه کاملاً برای این مسئله مناسب می باشد. تابع هزینه categorical_crossentropy برای اندازه‌گیری اختلاف یا واگرایی بین توزیع احتمال پیش‌بینی‌شده توسط مدل (که معمولاً خروجی یک لایه فعال‌سازی Softmax است) و برچسب‌های واقعی که به فرمت one-hot کدگذاری شده‌اند، طراحی شده است.

تابع هزینه جایگزین و برای این تابع هزینه، sparse_categorical_crossentropy است. تفاوت اصلی این تابع با categorical_crossentropy در فرمت مورد انتظار برای برچسب‌ها است. تابع categorical_crossentropy با برچسب‌های one-hot کار میکند و sparse_categorical_crossentropy با برچسب‌هایی که به صورت عدد صحیح ارائه می‌شوند (مثلاً اعداد ۰ تا ۹ برای ۱۰ کلاس)، کار می‌کند. این تابع هزینه می‌تواند از نظر مصرف حافظه بهینه‌تر باشد، زیرا نیازی به ساخت و نگهداری بردارهای one-hot حجیم نیست. انتخاب بین این دو تابع، تنها به نحوه آماده‌سازی و کدگذاری برچسب‌ها در خط لوله داده بستگی دارد و اصولاً بر عملکرد نهایی مدل تأثیری ندارد.

نهایتا مدل را با پارامتر های مورد بحث آموزش میدهیم و نمودار یادگیری مدل را بدست می آوریم:

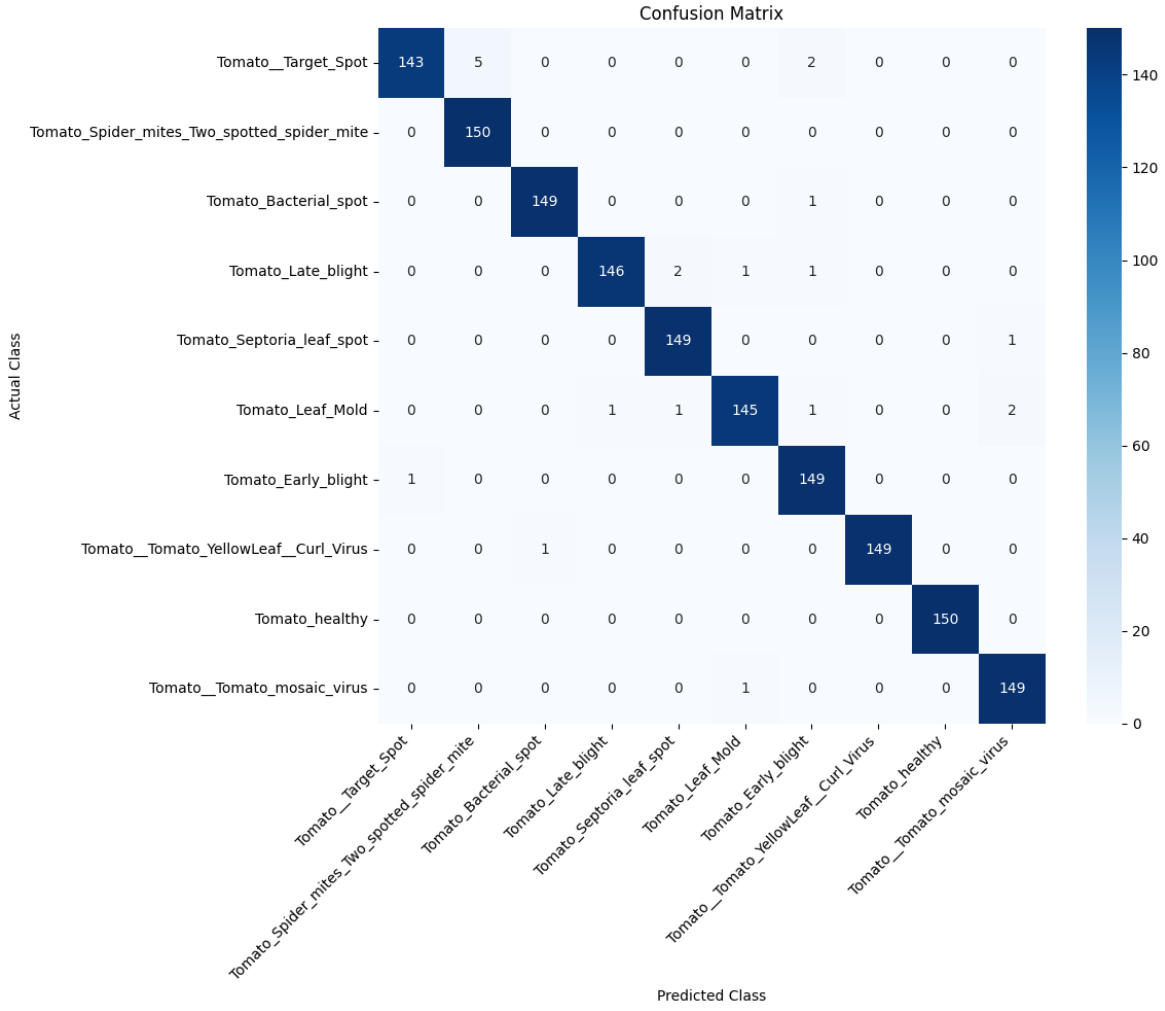


شکل 5: نمودار یادگیری مدل روی داده های **train , validation**

سپس مدل را با تمام داده های ولیدیشن تست می کنیم و گزارش ها و ماتریس آشفتگی را بدست می آوریم:

```
Classification Report:
```

	precision	recall	f1-score	support
Tomato__Target_Spot	0.99	0.95	0.97	150
Tomato_Spider_mites_Two_spotted_spider_mite	0.97	1.00	0.98	150
Tomato_Bacterial_spot	0.99	0.99	0.99	150
Tomato_Late_blight	0.99	0.97	0.98	150
Tomato_Septoria_leaf_spot	0.98	0.99	0.99	150
Tomato_Leaf_Mold	0.99	0.97	0.98	150
Tomato_Early_blight	0.97	0.99	0.98	150
Tomato__Tomato_YellowLeaf__Curl_Virus	1.00	0.99	1.00	150
Tomato_healthy	1.00	1.00	1.00	150
Tomato__Tomato_mosaic_virus	0.98	0.99	0.99	150
accuracy			0.99	1500
macro avg	0.99	0.99	0.99	1500
weighted avg	0.99	0.99	0.99	1500



شکل 6: ماتریس آشفتگی مدل **CNN**

1-4: آموزش مدل ViT

دو مدل را مطابق چیزی که قبل تر گفتیم (یکی با ورودی 255*255 و دیگری با ورودی 64*64) می سازیم. Summary مدل ها در ادامه آمده است. در این بخش نیز از پارامتر های مطابق مقاله استفاده میکنیم. تعداد ایپاک ها را در این بخش افزایش میدهیم و به 50 ایپاک میرسانیم.

Model: "functional_1"

Layer (type)	Output Shape	Param #
input_layer_1 (InputLayer)	(None, 64, 64, 3)	0
overlapping_patches_1 (OverlappingPatches)	(None, None, 192)	0
patch_and_position_encoder_1 (PatchAndPositionEncoder)	(None, 100, 64)	18,752
transformer_encoder_block_1 (TransformerEncoderBlock)	(None, 100, 64)	83,200
layer_normalization_5 (LayerNormalization)	(None, 100, 64)	128
flatten_1 (Flatten)	(None, 6400)	0
dropout_9 (Dropout)	(None, 6400)	0
dense_9 (Dense)	(None, 2048)	13,109,248
dropout_10 (Dropout)	(None, 2048)	0
dense_10 (Dense)	(None, 1024)	2,098,176
dropout_11 (Dropout)	(None, 1024)	0
dense_11 (Dense)	(None, 10)	10,250

Total params: 15,319,754 (58.44 MB)
Trainable params: 15,319,754 (58.44 MB)
Non-trainable params: 0 (0.00 B)

Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 256, 256, 3)	0
overlapping_patches (OverlappingPatches)	(None, None, 3072)	0
patch_and_position_encoder (PatchAndPositionEncoder)	(None, 100, 64)	203,072
transformer_encoder_block (TransformerEncoderBlock)	(None, 100, 64)	83,200
layer_normalization_2 (LayerNormalization)	(None, 100, 64)	128
flatten (Flatten)	(None, 6400)	0
dropout_3 (Dropout)	(None, 6400)	0
dense_3 (Dense)	(None, 2048)	13,109,248
dropout_4 (Dropout)	(None, 2048)	0
dense_4 (Dense)	(None, 1024)	2,098,176
dropout_5 (Dropout)	(None, 1024)	0
dense_5 (Dense)	(None, 10)	10,250

Total params: 15,504,074 (59.14 MB)
Trainable params: 15,504,074 (59.14 MB)
Non-trainable params: 0 (0.00 B)

لایه Patch Embedding در شبکه های مبدل تصویر به چه منظوری استفاده میشود؟ کاهش یا افزایش اندازه ی هر patch در این تمرین چه تاثیراتی بر روی خروجی دارد؟

لایه Patch Embedding در حالت کلی دو هدف اصلی دارد:

1. تبدیل پچ به بردار :یک پچ تصویر در حالت خام، صرفاً یک ماتریس از مقادیر پیکسل‌هاست. این فرمت خام برای معماری ترنسفورمر که با دنباله‌ای از بردارها کار می‌کند، معنادار نیست. لایه Patch Embedding(که معمولاً یک لایه Dense ساده است) این بردار بلند پیکسلی را به یک بردار با ابعاد ثابت و معنادارتر (مثلاً با طول ۶۴) تبدیل می‌کند. این فرآیند، اطلاعات مهم پچ را در یک فضای ویژگی فشرده و غنی‌تر کدگذاری می‌کند.

2. ایجاد ورودی با ابعاد یکسان : این لایه تضمین می‌کند که تمام پچ‌ها، صرف نظر از محتوایشان، به بردارهایی با طول یکسان تبدیل شوند تا بتوانند به عنوان یک دنباله استاندارد وارد بلوک‌های ترنسفورمر شوند. همینطور می توان گفت این لایه معمولاً باعث کاهش ابعاد می‌شود.

با توجه به جدول4 مقاله که نشان دهنده خروجی ماژول self attention تا خروجی نهایی است، hidden state ما 64 بعدی و تعداد پچ های هر تصویر 100 عدد است. با توجه به اینکه ورودی ما یک تصویر 64 در 64 است برای اینکه تصویر را به 100 پچ تقسیم کنیم، تصویر را به پچ های 8 در 8 که 2 پیکسل با هم اورلپ دارند تقسیم میکنیم. بنابر این هر پچ ما یک تصویر 8در8 با سه کانال رنگی است که مجموعا شامل مقادیر 192 پیکسل است. ماژول Patch Embedding این 192 پیکسل را به یک بردار 64 بعدی map میکند.

اندازه پچ یکی از مهم‌ترین هایپرپارامترها در طراحی یک مدل ViT است و به طور مستقیم روی عملکرد و هزینه محاسباتی مدل تأثیر می‌گذارد. اندازه کوچکتر یا بزرگتر یک پچ در واقع یک ترید آف است:

1. پچ‌های کوچک : جزئیات بیشتری را می‌بینند و پتانسیل دقت بالاتری دارند، اما هزینه محاسباتی را به شدت افزایش می‌دهند .
2. پچ‌های بزرگ : باعث آموزش بسیار سریع‌تر و سبک‌تر مدل می‌شوند، اما این کار به قیمت از دست دادن جزئیات ریز تصویر تمام می‌شود.

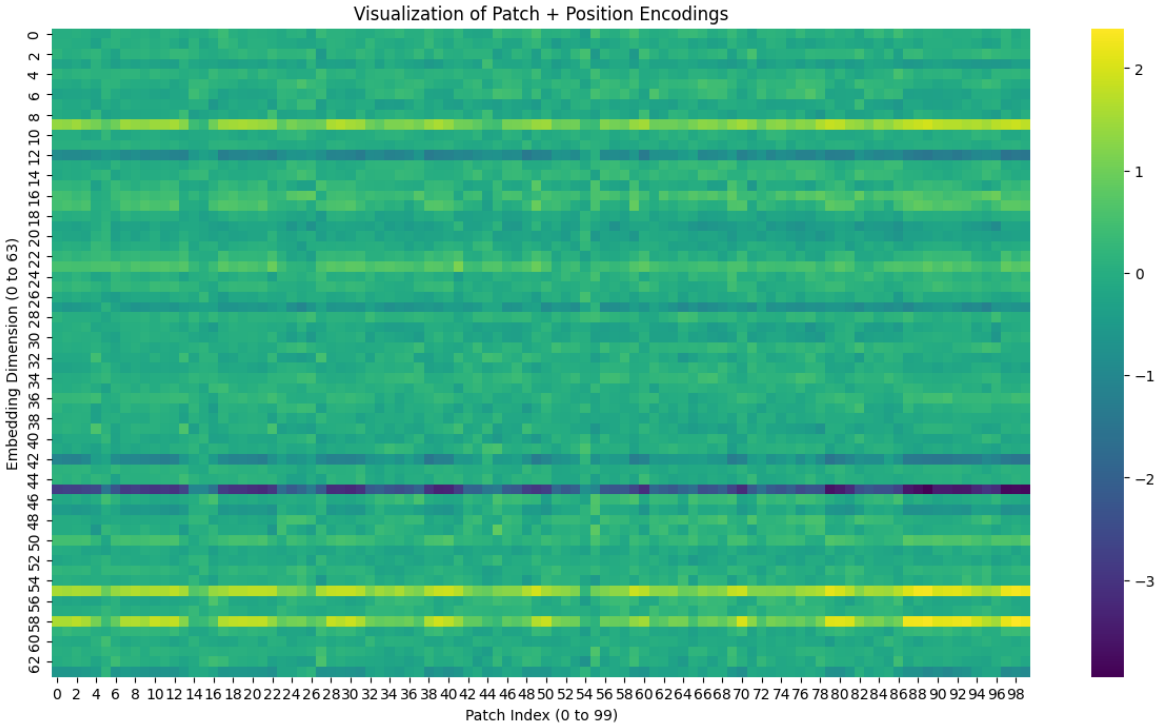
(امتیازی) در لایه Patch Embedding یک قابلیت خروجی پیکسلی تعبیه کنید و به استفاده از آن خروجی این لایه را به صورت تصویر نشان دهید.

با توجه به چیزی که گفتیم خروجی لایه path embedding یک بردار 64 تایی است و خروجی آن عملا نمایش پیکسلی ندارد چون صرفا 64 عدد است. احتمالا منظور سوال خروجی این لایه pos and patch encoding بوده است.

برای اینکه ببینیم در خروجی لایه pos encoding چه خبر است، این فرآیند چهار مرحله‌ای را طی می کنیم:

1. بارگذاری مدل آموزش‌دیده :ابتدا، مدلی که قبلاً با موفقیت آموزش داده بودیم را به همراه تمام دانش و وزن‌های یادگرفته شده‌اش، در حافظه بارگذاری کردیم.(مدل 64*64 را لود می کنیم)

2. ایجاد یک میان‌بُر به لایه میانی :سپس، یک مدل مجازی جدید می سازیم. این مدل از همان ورودی مدل اصلی استفاده می‌کند، اما به جای اینکه تا انتها برود و پیش‌بینی نهایی را انجام دهد، خروجی آن دقیقاً در لایه‌ای که ما می خواهیم (یعنی لایه کدگذاری پچ و موقعیت) متوقف می‌شود.
3. آماده‌سازی و ارسال یک تصویر نمونه : یک تصویر را به صورت تصادفی از مجموعه داده انتخاب کرده، آن را برای مدل آماده می کنیم و به ورودی این مدل میان‌بر می دهیم.
4. دریافت و نمایش خروجی :خروجی این مدل میان‌بُر، که همان اطلاعات لایه مورد نظر ماست را دریافت کرده و آن را به صورت یک نقشه حرارتی (Heatmap) رنگی نمایش می دهیم تا قابل فهم باشد.



شکل 7: خروجی لایه **pos and patch encoding**

این ماتریس، خروجی لایه کدگذاری پچ و موقعیت برای یک تصویر نمونه است. معنی هر بخش از نمودار به این شکل است:

محور افقی (ستون‌ها): هر ستون در این نمودار، یکی از ۱۰۰ پچ تصویر نمونه است. از چپ به راست، در حال حرکت روی تمام تکه‌هایی هستیم که تصویر نمونه به آن‌ها تقسیم شده است.

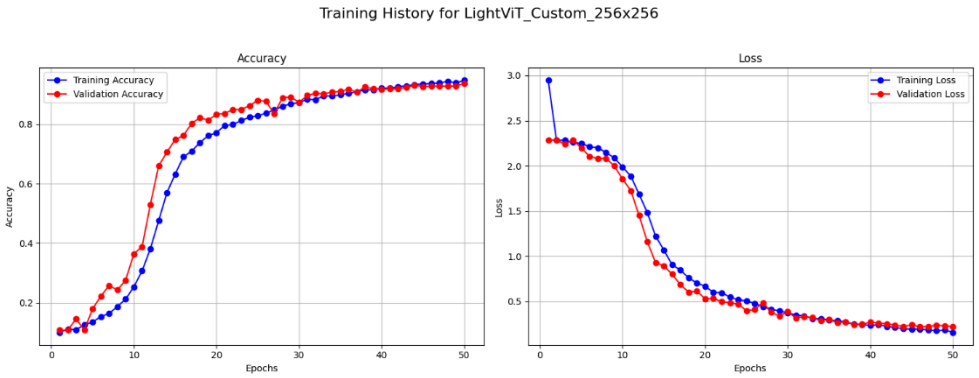
محور عمودی (ردیف‌ها): هر ردیف، یکی از ابعاد ۶۴ گانه بردار ویژگی است که برای توصیف هر پچ استفاده می‌شود. این بردارها خلاصه‌ای از اطلاعات بصری (محتوای پچ) و اطلاعات مکانی (موقعیت پچ) هستند.

با نگاه کردن به الگوهای رنگی عمودی و افقی در این تصویر می‌توان به نتایج جالبی رسید:

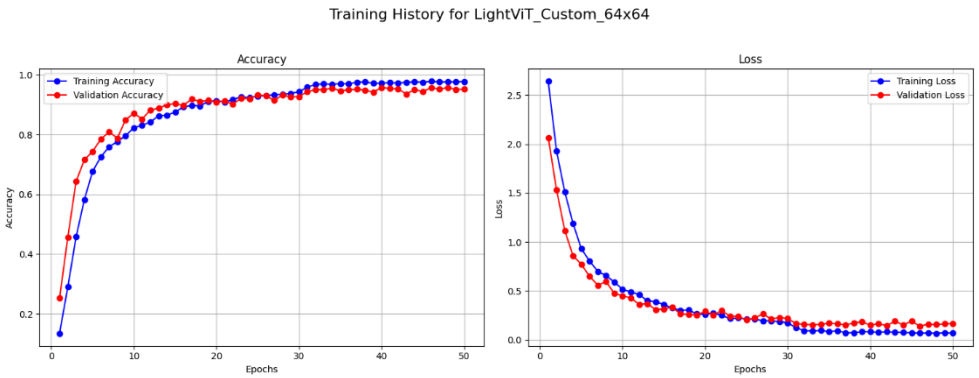
الگوهای عمودی :وجود نوارهای عمودی واضح می‌تواند به این معنی باشد که برخی پچ‌ها (مثلاً پچ‌هایی که حاوی لبه برگ یا یک لکه بیماری هستند) به طور کاملاً متفاوتی از پچ‌های دیگر (مثلاً پچ‌های پس‌زمینه ساده) کدگذاری شده‌اند. این نشان می‌دهد که مدل در همان ابتدا توانسته تفاوت بین بخش‌های مهم و غیرمهم تصویر را تشخیص دهد.

الگوهای افقی :وجود نوارهای افقی واضح نشان می‌دهد که برخی از ابعاد ۶۴ گانه بردار ویژگی (مثلاً ویژگی شماره ۱۰ یا ویژگی شماره ۵۰) در کل تصویر فعال‌تر یا مهم‌تر از بقیه هستند. این ویژگی‌ها ممکن است به الگوهای خاصی مانند وجود بافت یا یک رنگ خاص مرتبط باشند.

نهایتا مدل را آموزش داده و تست می کنیم. سپس نمودار های یادگیری، ماتریس آشفتگی و گزارش طبقه بندی را بدست می آوریم:



شکل 8: نمودار یادگیری مدل **ViT** با ورودی 256 در 256



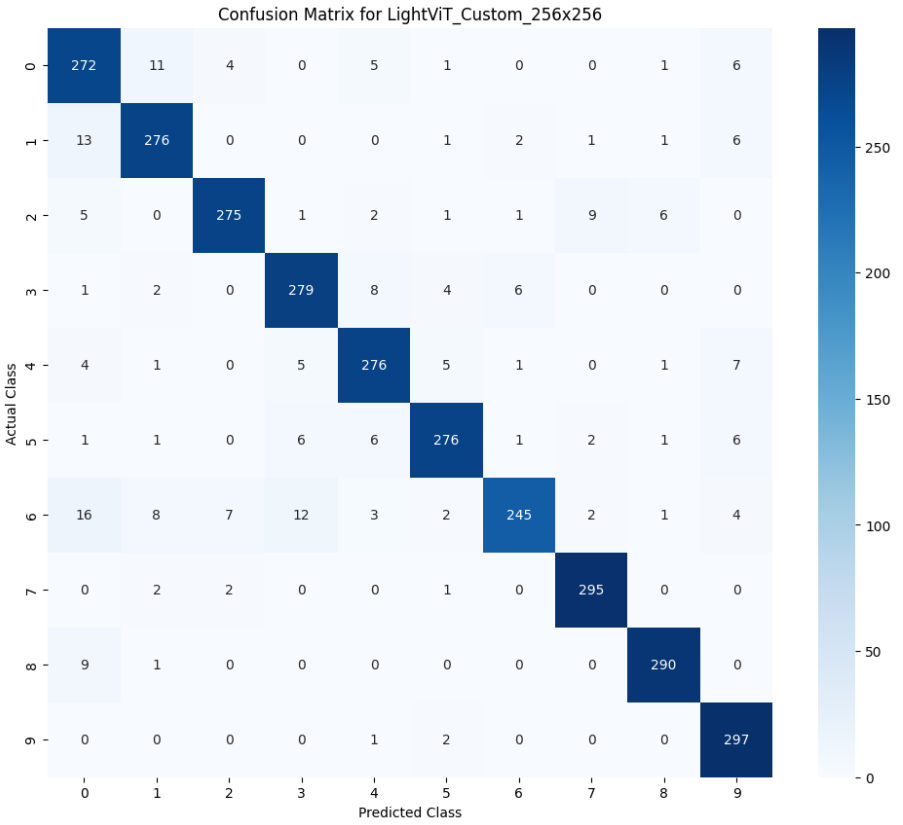
شکل 9: نمودار یادگیری مدل **ViT** با ورودی 64 در 64

Classification Report(256*256):

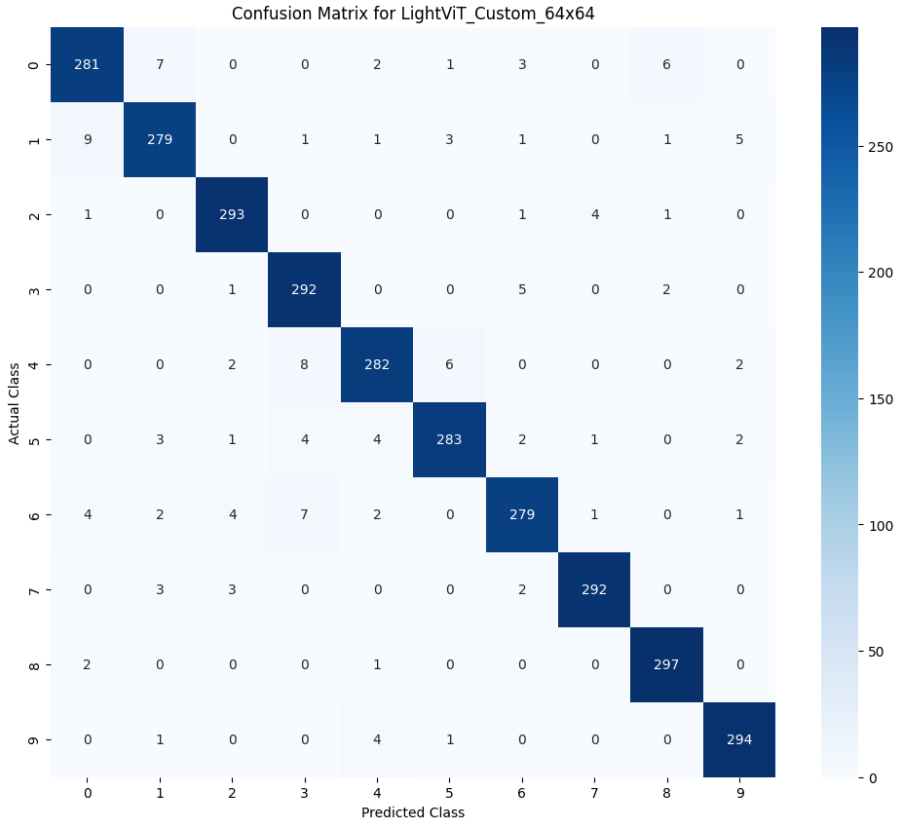
	precision	recall	f1-score	support
0	0.85	0.91	0.88	300
1	0.91	0.92	0.92	300
2	0.95	0.92	0.94	300
3	0.92	0.93	0.93	300
4	0.92	0.92	0.92	300
5	0.94	0.92	0.93	300
6	0.96	0.82	0.88	300
7	0.95	0.98	0.97	300
8	0.96	0.97	0.97	300
9	0.91	0.99	0.95	300
accuracy			0.93	3000
macro avg	0.93	0.93	0.93	3000
weighted avg	0.93	0.93	0.93	3000

Classification Report(64*64):

	precision	recall	f1-score	support
0	0.95	0.94	0.94	300
1	0.95	0.93	0.94	300
2	0.96	0.98	0.97	300
3	0.94	0.97	0.95	300
4	0.95	0.94	0.95	300
5	0.96	0.94	0.95	300
6	0.95	0.93	0.94	300
7	0.98	0.97	0.98	300
8	0.97	0.99	0.98	300
9	0.97	0.98	0.97	300
accuracy			0.96	3000
macro avg	0.96	0.96	0.96	3000
weighted avg	0.96	0.96	0.96	3000



شکل 10: ماتریس آشفتگی مدل **ViT** با ورودی 256 در 256



شکل 11: ماتریس آشفتگی مدل **ViT** با ورودی 64 در 64

5-1: تحلیل و نتیجه گیری

با توجه به چیزی که بررسی کردیم، دقت مدل ViT با تصاویر 64 در 64 از دقت مدل ViT با تصاویر 256 در 256 بیشتر بود. ابتدا این مورد غیر منطقی به نظر میرسد ولی با کمی تامل بیشتر می توان متوجه شد که وقتی معماری مدل ترنسفورمری یکسان است اندازه تصاویر شاید تاثیر خاصی نداشته باشد. در مدل CNN هرچقدر اندازه تصویر بزرگتر باشد لایه های pooling و کانولوشنی بیشتر میشود تا اندازه feature map ها رفته رفته کوچکتر شوند ولی در شبکه های ترنسفورمری هر ورودی تصویر با هر اندازه ای بلاخره به یک بردار 64 بعدی مپ خواهد شد و اندازه تصاویر تاثیر چندانی روی معماری شبکه نخواهد گذاشت. صرفا ماژول pos and patch encoder سنگین تر خواهد شد و پارامتر های بیشتری خواهد داشت. شاید به این دلیل است که دقت مدل 64 در 64 از دقت مدل 256 در 256 بیشتر شده است.

به هر حال، با توجه به اینکه دقت مدل 64 در 64 از دقت مدل 256 در 256 بیشتر است، مدل ViT با ورودی 64 در 64 را با مدل CNN در یک جدول مقایسه می کنیم.

جدول**1** : مقایسه متریک های مدل های **ViT , CNN**

معیار(Metric)	مدل Inception V3 (CNN)	مدل ViT (64x64)
دقت کلی(Overall Accuracy)	99%	96%
میانگین F1-Score (Macro Avg)	0.99	0.96
تعداد کل پارامترها	22.8~میلیون	15.3~میلیون
تعداد داده‌های ارزیابی	1500تصویر	3000تصویر
اندازه ورودی تصویر	299x299	64x64
معماری پایه	CNNبسیار عمیق	Transformerبسیار سبک

با توجه به گزارش‌ها و جدول بالا می توان گفت مدل Inception V3 (CNN) با دقت 99% عملکرد به مراتب بهتری نسبت به مدل ViT با دقت 96% داشته است. این برتری با وجود اینکه روی داده‌های کمتری آموزش دیده، چشمگیرتر هم هست. معماری Inception V3 یک طراحی بسیار پخته و بهینه در دنیایCNN هاستCNN .ها به دلیل داشتن بایاس‌های القایی (Inductive Biases) مانند درک محلی بودن ویژگی‌ها و ثبات در برابر جابجایی، در یادگیری از داده‌های تصویری بسیار کارآمد هستند. حتی وقتی به صورت خام آموزش داده می‌شوند، ساختار کانولوشنی آن‌ها یک مزیت ذاتی برایشان ایجاد می‌کند.

همچنین مدل Inception V3 با داشتن پارامترهای بیشتر (حدود ۲۳ میلیون در برابر ۱۵ میلیون)، ظرفیت بالاتری برای یادگیری الگوهای پیچیده داشته است.

با اینکه دقت 96٪ برای مدل سبک ViT بسیار خوب است، برای بهبود آن می‌توانیم چندین راهکار را امتحان کنیم:

1. افزایش عمق و پیچیدگی مدل :مدل فعلی ما فقط یک بلوک انکودر دارد. می‌توانیم با افزایش پارامتر num_encoder_blocks (مثلاً به ۴، ۶ یا حتی ۱۲ لایه مانند ViT-Base) و افزایش hidden_size (مثلاً به ۲۵۶)، ظرفیت مدل را برای یادگیری الگوهای پیچیده‌تر به شدت افزایش دهیم.
2. Transfer Learning: آموزش مدل‌های ViT از ابتدا نیازمند داده‌های بسیار زیاد است. بهترین راه برای رسیدن به بالاترین دقت، استفاده از یک مدل ViT است که روی مجموعه داده عظیمی مانند ImageNet از پیش آموزش دیده باشد. ما می‌توانستیم به جای ساخت مدل از صفر، یک مدل ViT-Base از پیش آموزش‌دیده را از کتابخانه transformers بارگذاری کرده و آن را روی داده‌های گوجه‌فرنگی خودمان Fine-tune کنیم. این روش معمولاً نتایج بهتری به همراه دارد.
3. تکنیک‌های آموزش پیشرفته‌تر : می‌توان از روش‌های پیشرفته‌تری مانند استفاده از زمان‌بند نرخ یادگیری (Learning Rate Scheduler) با فاز گرم کردن (warmup) که برای ترنسفورمرها بسیار رایج است، استفاده کرد.

مدل سبک ViT ما در سناریوهای زیر می‌توانست عملکرد بهتری داشته باشد یا حتی از CNN پیشی بگیرد:

1. در صورت وجود مجموعه داده عظیم (Big Data) : نقطه قوت اصلی ترنسفورمرها، مقیاس‌پذیری آن‌هاست. اگر به جای ۳۰ هزار تصویر، ۳ میلیون تصویر در اختیار داشتیم، ViTبه دلیل داشتن بایاس‌های القایی کمتر، می‌توانست الگوهای بنیادی‌تری را مستقیماً از داده‌ها یاد بگیرد و احتمالاً از CNN که ممکن است به یک سقف عملکردی برسد، بهتر عمل کند.
2. در صورت استفاده از Transfer Learning : همانطور که در بالا ذکر شد، یک مدل ViT-Base که روی ImageNet پیش‌آموزش دیده، به احتمال قریب به یقین مدل Inception V3 که از صفر آموزش دیده را شکست می‌دهد، زیرا از دانش استخراج شده از میلیون‌ها تصویر متنوع بهره می‌برد.
3. در مسائلی با نیاز به درک زمینه کلی (Global Context) : اگر تشخیص بیماری نیازمند درک رابطه بین بخش‌های دور از هم در یک تصویر بود (مثلاً یک الگو در بالای برگ به یک پژمردگی در

پایین آن مرتبط باشد)، مکانیزم توجه سراسری ViT به آن یک برتری ذاتی نسبت به دید محلی‌تر CNNها می‌داد.

پرسش 2 – Robust Zero-Shot Classification

2-1: آشنایی با مدل CLIP، طبقه بندی تک ضرب و حملات خصمانه

1: روش‌های تولید نمونه‌های تخاصمی (FGSM و PGD)

روش های تولید نمونه های تخاصمی برای تولید یک نویز کوچک (و معمولا نامحسوس برای انسان) استفاده می‌شوند که با اعمال آن به تصویر اصلی، تابع خطای شبکه عصبی بیشترین افزایش را دارد تا مدل را به اشتباه وادارد.

روش های FGSM و PGD هر دو Foundational Gradient-based attack هستند، به این معنا که از گرادیان های خود مدلی که قصد حمله به آن را داریم استفاده میکنند.

روش FGSM (Fast Gradient Sign Method):

یک روش تک مرحله ای و سریع برای بدست آوردن نویز می‌باشد. نحوه عملکرد آن به این صورت هست که ابتدا Loss را محاسبه میکند. سپس گرادیان Loss را نسبت به تک تک پیکسل های تصویر ورودی محاسبه میکند. این روش تنها از علامت گرادیان ها استفاده میکند و یک ماتریس می سازد که هر درایه آن مثبت یا منفی یک است (با توجه به علامت گرادیان) و جهت بهینه تغییر برای هر پیکسل را نشان میدهد. در نهایت با ضرب این ماتریس در مقدار کوچک اپسیلون (که حداکثر مقدار مجاز نویز است) و افزودن آن به تصویر، نمونه متخاصم را می‌سازیم.

$$x_{adv} = x + \varepsilon * sign\left(\nabla_x J(\theta, x, y)\right)$$

این روش فرض میکند که با برداشتن یک گام نسبتا بزرگ در جهت گرادیان میتوان تابع هزینه را به مقدار زیادی افزایش داد.

روش PGD (Projected Gradient Descent):

یک روش Iterative می باشد. نحوه عملکرد آن مشابه FGSM هست ولی با این تفاوت که در چند مرحله اجرا شده و هر مرحله گام کوچکی برمی‌دارد. بعد از برداشتن هر قدم هم مطمئن میشود که از محدوده مجاز ε خارج نشده باشد.

برای شروع معمولا به تصویر اولیه یک نویز رندوم کوچک اضافه می کنند تا از گیر کردن مدل داخل یک Local Optimum جلوگیری کند. سپس به تعداد Steps مراحل زیر را تکرار می‌کنیم:

ابتدا مانند روش FGSM گرادیان را برای هر پیکسل محاسبه کرده و علامت آن را تعیین میکنیم. سپس آن را در یک ضریب کوچک آلفا ضرب کرده و نویز حاصل را به تصویر اضافه می‌کنیم.

سپس بررسی میکنیم که نویز اضافه شده به تصویر از حد ε خارج نشده باشد که در این صورت دوباره نویز را به آن محدوده Project می‌کنیم.

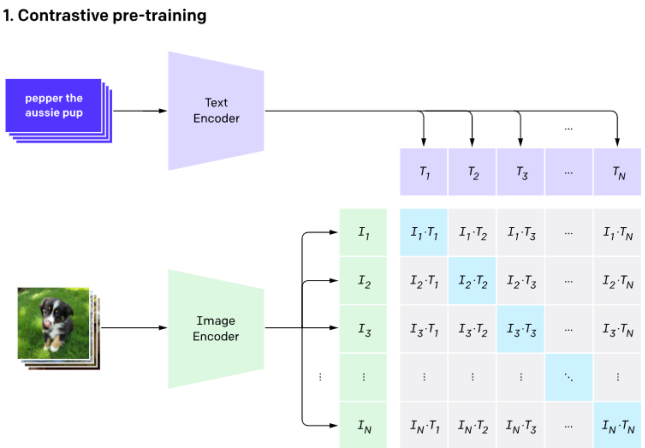
این روش از نظر محاسباتی سنگین تر از روش قبل هست ولی به دلیل اینکه تعداد زیادی قدم کوچک در جهت افزایش Loss بر می‌دارد، نمونه های تخاصمی قوی تر را پیدا می‌کند و در عمل برای تولید تصاویر Adversarial از همین روش استفاده می‌کنند.

2: معماری CLIP

مدل CLIP یک مدل برای ارتباط میان تصویر و متن هست. این مدل روی مجموعه داده بسیار بزرگی از تصاویر و متن های مرتبط با آنها از روی اینترنت آموزش داده شده است. نکته جالب درباره این مدل این است که با فراهم کردن نام دسته ها برای مدل، میتواند برروی هر دیتاست تصویری Classification یی اعمال شود و نیازی به آموزش مستقیم مدل روی آن دیتاست نیست.

انگیزه ایجاد این مدل به تعدادی از مشکلات مدل های معمول یادگیری عمیق در وظایف Computer Vision برمیگردد. از مشکلات تهیه کردن دیتاست های لیبل دار است که کار وقت گیر و پرهزینه ای معمولا می‌باشد. یکی دیگر از مشکلات بزرگ مدل های معمول این است که آنها برروی دیتاستی که روی آن آموزش دیده اند عملکرد خوبی از خود نشان میدهند، ولی آنها را نمیتوانیم برروی داده های جدیدی اعمال کرده و انتظار عملکرد خوب داشته باشیم. به عبارت دیگر این مدل ها برای یک وظیفه خاص طراحی شده و تنها همان را میتوانند به خوبی انجام دهند و قدرت Generalization ندارند. این مشکل همچنین باعث میشود عملکرد این مدل ها در دنیای واقعی نیز دچار افت شود، زیرا به گونه ای این مدل ها عملکرد خود را برروی دیتاست آموزشی بهینه میکنند.

معماری:



شکل 12: معماری مدل **CLIP**

معماری مدل CLIP دو بخشی اصلی دارد که مجزا از هم میباشند: کدگذار تصویر (Image Encoder) و کدگذار متن (Text Encoder). این دو بخش تصویر/ متن را دریافت کرده و به بردار ویژگی (با سایز های یکسان) تبدیل می‌کنند (بردار های I و T در شکل بالا).

هدف از این کار این است که بردار های ویژگی در یک فضای ویژگی مشترک چندوجهی به یکدیگر نزدیک شوند. به عبارتی تصاویر و متون همگی به یک فضای مشترکی منتقل می‌شوند که در آنجا نزدیکی بردار ویژگی آنها به هم میتواند معنی‌دار باشد.

مطابق شکل بالا ابتدا تصاویر و متون متناظرشان کدگذاری می‌شوند. سپس بر اساس معیار شباهت و روش Contrastive Learning، شباهت میان همه جفت تصویر-متن ها محاسبه میشود. از آنجایی که تصویر i ام متناظر با متن i ام میباشد، درایه های قطر اصلی این ماتریس شباهت باید بیشینه شوند و مطابق تابع هزینه ای که تعریف میشود مدل آموزش می‌بینند.

تابع زیان و نوع آموزش:

```
# image_encoder - ResNet or Vision Transformer
# text_encoder  - CBOW or Text Transformer
# I[n, h, w, c] - minibatch of aligned images
# T[n, l]       - minibatch of aligned texts
# W_i[d_i, d_e] - learned proj of image to embed
# W_t[d_t, d_e] - learned proj of text to embed
# t             - learned temperature parameter

# extract feature representations of each modality
I_f = image_encoder(I) #[n, d_i]
T_f = text_encoder(T)  #[n, d_t]

# joint multimodal embedding [n, d_e]
I_e = l2_normalize(np.dot(I_f, W_i), axis=1)
T_e = l2_normalize(np.dot(T_f, W_t), axis=1)

# scaled pairwise cosine similarities [n, n]
logits = np.dot(I_e, T_e.T) * np.exp(t)

# symmetric loss function
labels = np.arange(n)
loss_i = cross_entropy_loss(logits, labels, axis=0)
loss_t = cross_entropy_loss(logits, labels, axis=1)
loss   = (loss_i + loss_t)/2
```

شکل 13: پیاده‌سازی شبه‌کد مدل **CLIP**

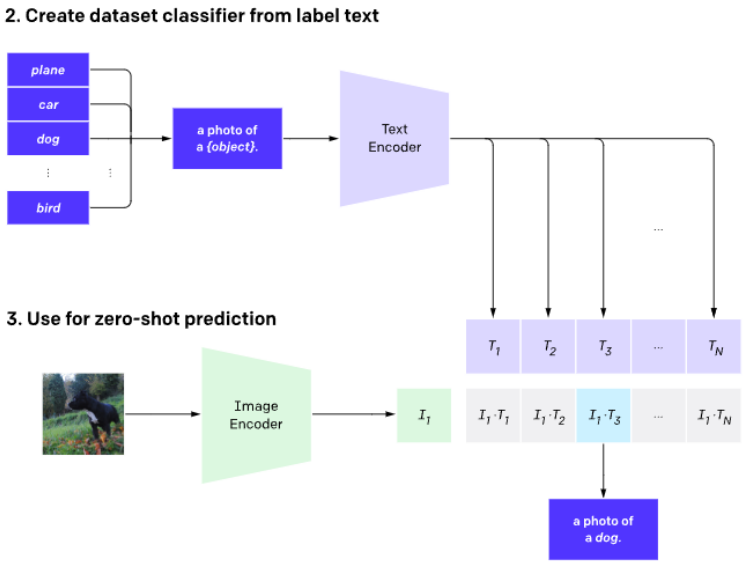
همانطور که می‌بینیم تصاویر و متون ورودی کدگذاری شده و به فضای مشترک نگاشت می‌شوند. سپس با ضرب داخلی بردار های ویژگی های تصاویر و متون، ماتریس شباهت را تشکیل میدهد (معیار شباهت کسینوسی). مفهموم یادگیری تقابلی نیز همین است که مدل بجای یادگرفتن لیبل برای عکس، از طریق مقایسه زوج ها (یک زوج مثبت و بقیه زوج ها منفی) کلاس تصویر را یاد می‌گیرد.

تابع هزینه از دو بخش تشکیل شده است. Loss_i که زیان هر تصویر داخل آن mini-batch را حساب میکند و مدل باید از میان متن های موجود بهترین متن را انتخاب کند. بر خلاف آن Loss_t زیان برای هر متن را حساب میکند و مدل باید از میان تصاویر، تصویر درست را پیشبینی بکند. در نهایت این دو مقدار را میانگین می‌گیرد، برای همین مدل به صورت دوطرفه یاد می‌گیرد که هم تصویر را به متن صحیح متناظر کند و هم بالعکس.

3: تفاوت طبقه‌بندی عادی و تک ضرب

در طبقه بندی عادی، ابتدا مدل برروی یک مجموعه داده لیبل گذاری شده آموزش داده می‌شود که هر تصویر آن به یکی از کلاس های از پیش مشخص شده تعلق دارد. مدل یاد می‌گیرد که وقتی تصویر جدید به آن اعمال می‌شود، آن را به یکی از این کلاس ها نسبت بدهد. واضحا این نوع طبقه بندی محدود به کلاس هایی است که بر روی آن آموزش انجام شده و مدل نمیتواند کلاس جدیدی را پیشبینی کند.

در طبقه بندی تک‌ضرب (Zero-shot Classification) مدل برروی مجموعه بزرگی از زوج داده تصویر-متن آموزش می‌بیند و یاد میگیرد ارتباط مفهومی بین محتوای تصویر و متن آن را درک کند. این مدل ها به واسطه داده های آموزش وسیع، دارای دانش نهان زیادی میشوند که میتواند باعث Generalization مدل بشود.



شکل 14: نحوه انجام **Zero-shot prediction** در مدل **CLIP**

از مدل CLIP برای Zero-shot Classification میتوان استفاده نمود. نحوه پیشبینی کلاس با استفاده از این مدل در شکل بالا نشان داده شده است. ابتدا یک فضای متنی با استفاده از لیبل های ممکن برای مدل میسازیم و آن را کد گذاری میکنیم (این کلاس ها میتوانند هر کلاسی باشند). سپس هر تصویر

ورودی را نیز به کدگذار تصویر اعمال کرده و شباهت بردار ویژگی آن را با تک تک بردارهای ویژگی متون محاسبه کرده و متنی که بیشترین شباهت را انتخاب میکنیم.

بر خلاف طبقه بندی عادی، این نوع طبقه بندی بسیار منعطف است و میتواند روی هر مجموعه دلخواهی از کلاس ها اعمال شود (حتی اگر مدل صراحتا روی آنها آموزش دیده نشده باشد)

4: حملات جعبه سفید و جعبه سیاه

حملات جعبه سفید:

در حمله خصمانه جعبه سفید، مهاجم دسترسی کاملی به مدلی که میخواهد به آن حمله کند و جزئیات آن (معماری مدل و ساختار دقیق لایه‌ها، پارامترهای مدل و همچنین قابلیت محاسبه گرادیان تابع هزینه بر حسب ورودی) دارد.

در این حمله به دلیل دسترسی به گرادیان ها، میتوان از روش های موثری مانند PGD و FGSM استفاده نمود تا مشخص شود کدام پیکسل‌ها چطور تغییر کنند تا بیشترین تاثیر را برروی تابع هزینه داشته باشند.

از آنجایی که این حملات نیاز به دسترسی کامل به مدل دارند، معمولا در امور تحقیقاتی و برای بدست آوردن مقاومت مدل در بدترین حالت ممکن استفاده می‌شوند.

حملات جعبه سیاه:

در این حملات، مدل مانند یک جعبه سیاه است که فقط ورودی گرفته و خروجی تولید می‌کند و مهاجم هیچ اطلاعی از داخل مدل ندارد. به دلیل عدم دسترسی به گرادیان ها، از روش هایی مانند PGD نمیتوان استفاده نمود. دونوع حمله رایج از این نوع وجود دارد:

1. حملات مبتنی بر پرس‌وجو

در این حملات مهاجم تعداد بسیار زیادی ورودی مختلف به مدل اعمال کرده و سعی میکند رفتار آن را تخمین زده و جهت گرادیان ها را حدس بزند که روش زمانبر و پرهزینه ای است.

2. حملات انتقالی (Transfer Attack)

این روش بسیار رایج است و نمونه های تخاصمی را روی یک مدل جایگزین که خود تعریف می‌کند (و به آن دسترسی کامل دارد) به روش جعبه سفید تولید کرده و همان نمونه ها را به مدل اصلی میفرستد. در خیلی از موارد این نمونه تخاصمی توانایی فریب مدل جعبه سیاه را نیز دارد.

مقایسه:

از نظر سطح دسترسی به مدل، در حمله جعبه سفید نیاز به دسترسی کامل به مدل است، در صورتی که در حمله جعبه سیاه هیچ اطلاعاتی از داخل مدل نداریم.

حملات جعبه سفید معمولا ساده تر و سر راست تر میباشند، در حالی که حملات جعبه سیاه از تکنیک های جایگزین و پیچیده تری استفاده میکنند.

از نظر کاربرد به نظر می‌رسد حملات جعبه سیاه به شدت کاربردی‌تر باشند. زیرا در دنیای واقعی مهاجمان در اکثر مواقع به ساختار داخلی مدل و پارامترها دسترسی ندارند و تنها میتوانند ورودی به آن اعمال کنند.

5: حملات انتقالی

همانطور که گفته شد برای انجام حمله جعبه سفید نیاز به دسترسی کامل به جزئیات مدل می‌باشد ولی بسیاری از مدل ها تجاری بوده و شرکت ها جزئیاتی از آنها منتشر نمیکنند و این حملات بیشتر در همان فاز توسعه مدل برای سنجش آسیب پذیری کاربرد دارند.

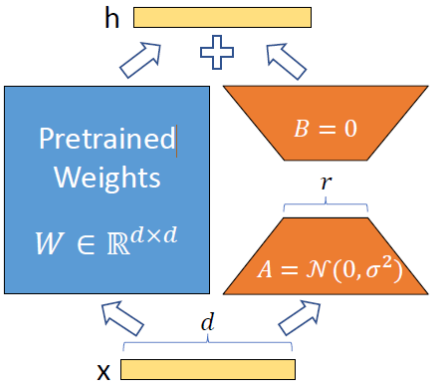
از طرف دیگر حملات انتقالی تنها نیاز به اعمال ورودی به مدل دارند. همچنین معمولا در حملات انتقالی با اینکه نمونه خصمانه روی مدلی متفاوت از مدل اصلی ساخته شده است، قابلیت تاثیرگذاری مخرب برروی مدل اصلی را نیز تا حد بالایی دارد. دلیل این امر شباهت کلی مدل ها و ویژگی هایی هست که یاد می‌گیرند و نویز مخرب برای یک مدل، به احتمال زیاد عملکرد مدل دیگر را نیز تحت تاثیر قرار می‌دهد.

6: روش تنظیم دقیق LoRA

بسیاری از اپلیکیشن ها از مدل های بزرگ از پیش آموزش داده شده استفاده میکنند. برای تطبیق مدل با وظیفه جدید آن را Finetune می‌کنند. تنظیم دقیق کامل مدل که شامل تغییر همه پارامترهای مدل می‌شود اصلا روش بهینه ای نیست و هزینه محاسباتی بسیاری دارد (از آنجا که بسیاری از مدل های بزرگ امروزی چندین میلیارد پارامتر دارند و آموزششان زمان و منابع زیادی می‌طلبد).

روش تنظیم دقیق LoRA پیشنهاد می‌کند که تغییرات وزن های مدل هنگام Adaptation آن، دارای رتبه ذاتی پایینی می‌باشد و می‌توان با آموزش ماتریس های Rank decomposition یک لایه، به طور غیر مستقیم وزن های آن را تنظیم نمود، در حالی که وزن های اولیه مدل را ثابت نگه میداریم.

این ماتریس های تجزیه شده رنک پایین اضافه شده، تعداد پارامترهای بسیار کمتری را برای یادگیری شامل می‌شوند.



شکل 15: نحوه عملکرد **LoRA** بر یک لایه

همانطور که در شکل بالا می‌بینیم، ماتریس وزن های از پیش آموزش داده شده بدون تغییر باقی میماند. در یک شاخه موازی، دو ماتریس رتبه پایین A و B اضافه میشوند که وزن هایشان آموزش داده میشود. حاصلضرب این ماتریس ها تغییرات وزن این لایه را به ما میدهد. همانطور که در روابط زیر میبینیم ابعاد این ماتریس ها بسیار کوچک تر از ماتریس وزن اصلی می‌باشد.

$$W_0 \in R^{d \times k}, \qquad B \in R^{d \times r}, \qquad A \in R^{r \times k} \ (where\ r \ll \min(d, k))$$

$$W_0 + \Delta W = W_0 + BA$$

$$h = W_0 x + \Delta W x = W_0 x + BAx$$

معمولا این ماتریس ها را به ساختار های Transformer و وزن های Attention (مانند ماتریس های Query, Key, Value) اعمال می‌کنند.

مزیت ها:

1. بعد از آموزش مدل، تغییرات وزن ΔW هر ماتریس را به وزن های اصلی اضافه میکنند و شاخه موازی را از ساختار حذف می‌کنند، بنابراین این روش هیچ تاخیری در مرحله Inference اضافه نمی‌کند و مدل مانند قبل و در همان زمان استنتاج می‌کند.
2. می‌توان مدل را برای چندین وظیفه مختلف تنظیم دقیق کرد و برای هر کدام تغییرات وزن ΔW که حجم به مراتب کمتری از کل پارامترهای مدل دارد را ذخیره نمود و با اضافه کردن هر کدام به مدل اصلی (که سربار کمی هم دارد) بین تسک های مختلف به سرعت جابجا شد.
3. هزینه محاسباتی را در مقایسه با روش های معمول Finetune که شامل تغییر همه وزن ها میشود، به شدت کاهش میدهد و با سرعت بیشتری مدل تنظیم میشود. همچنین عملکرد آن نیز معمولا به حد همان مدل های تنظیم دقیق شده به روش معمول می‌رسد.

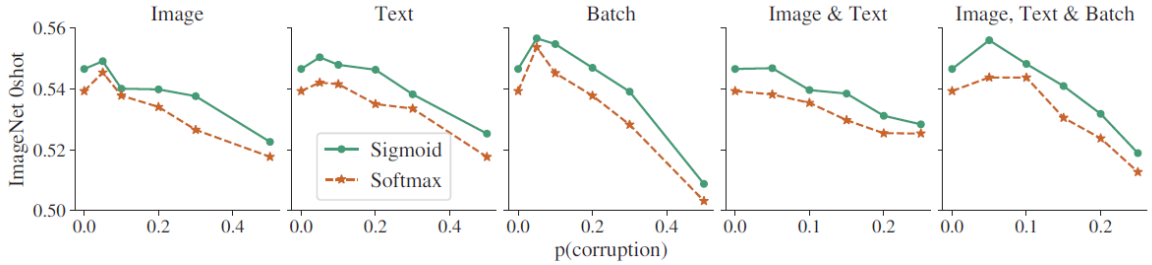
7: توابع زیان بهبود یافته CLIP

1. مقاله [Sigmoid Loss for Language Image Pre-Training]:

در این مقاله یک تابع Pairwise Sigmoid Loss را برای آموزش مدل های تصویر-متن پیشنهاد داده‌است. بر خلاف تابع Softmax یی که در CLIP تعریف شده بود و برای نرمال سازی شباهت های متقابل یک تصویر به متون (و بالعکس)، نیاز به دید کامل از همه Pairwise similarities داخل Batch داشت، این تابع هزینه تنها برروی جفت های تصویر-متن به صورت مستقل عمل می‌کند و برای زوج های جفت برچسب مثبت و برای دیگر جفت ها برچسب منفی درنظر می‌گیرید.

$$-\frac{1}{|\mathcal{B}|}\sum_{i=1}^{|\mathcal{B}|}\sum_{j=1}^{|\mathcal{B}|}\underbrace{\log\frac{1}{1+e^{z_{ij}(-t\mathbf{x}_i\cdot\mathbf{y}_j+b)}}}_{\mathcal{L}_{ij}}$$

مزیت این تابع هزینه نسبت به CLIP، این است که نیاز به عملیاتی برروی کل داده های بچ ندارد، و عملا مسئله را به یک Binary Classification برای جفت های تصویر-متن تبدیل میکند (این جفت میتواند مثبت یا منفی باشد). همچنین عملیات ما مستقل از سایز بچ شده و میتوان هم برای سایز بچ های کوچک و هم سایز های بزرگ پیاده سازی را انجام داد. نتایج نشان می‌دهد که این تابع هزینه مقاومت مدل نسبت به نویز داده ها (تصاویر و متون) افزایش می‌دهد.



شکل 16: مقایسه تاثیر نویز برروی توابع هزینه **Sigmoid** و **Softmax**

همانطور که نشان داده شده، عملکرد این تابع هزینه در مواجه با انواع نویز روی ورودی، بهتر از تابع هزینه برمبنای Softmax می‌باشد.

2. مقاله [Debiased Contrastive Learning] :

مدل های یادگیری تقابلی (Contrastive learning) مانند CLIP یک نقص بنیادی دارند. این مدل ها هر تصویر درون بچ را به یک نمونه مثبت که متن متناظرش هست نزدیک میکند و از بقیه متن ها دور میکند (تابع هزینه CLIP نیز بر همین اساس 1 نمونه مثبت و N-1 نمونه منفی درست شده بود). حال

آنکه نمونه های منفی داخل یک بچ میتوانند از همان کلاس نمونه مثبت باشند ولی مدل دارد به اشتباه آن را نمونه منفی در نظر میگیرد که باعث سوگیری مدل می‌شود و دقت آن را کاهش می‌دهد.

در این مقاله برای حل این مشکل، تابع Unbiased Loss را تعریف میکنند تا اثر منفی های کاذب را خنثی کند.

$$L^N_{\text{Unbiased}}(f) = \mathbb{E}_{\substack{x\sim p, x^+\sim p^+_x \\ x^-_i \sim p^-_x}} \left[-\log \frac{e^{f(x)^T f(x^+)}}{e^{f(x)^T f(x^+)} + \frac{Q}{N}\sum_{i=1}^{N} e^{f(x)^T f(x^-_i)}} \right],$$

شکل 17: تابع هزینه **Unbiased**

```
# pos: exponential for positive example
# neg: sum of exponentials for negative examples
# N  : number of negative examples
# t  : temperature scaling
# tau_plus: class probability

standard_loss = -log(pos / (pos + neg))
Ng = max((-N * tau_plus * pos + neg) / (1-tau_plus), N * e**(-1/t))
debiased_loss = -log(pos / (pos + Ng))
```

شکل 18: شبه کد نحوه پیاده سازی تابع هزینه **Unbiased**

احتمالات نمونه مثبت و نمونه های منفی مانند قبل محاسبه میشوند. ولی مدل با استفاده از هایپرپارامتر tau_plus که نشان دهنده احتمال پیشین (Prior) آن کلاس هست، از مجموع کل امتیازهای منفی، سهم منفی های کاذب را تخمین زده و کم می‌کند. (اگر دقت کنیم امتیاز مثبت ها را هنگام تخمین تعداد منفی های کاذب اعمال می‌کند.

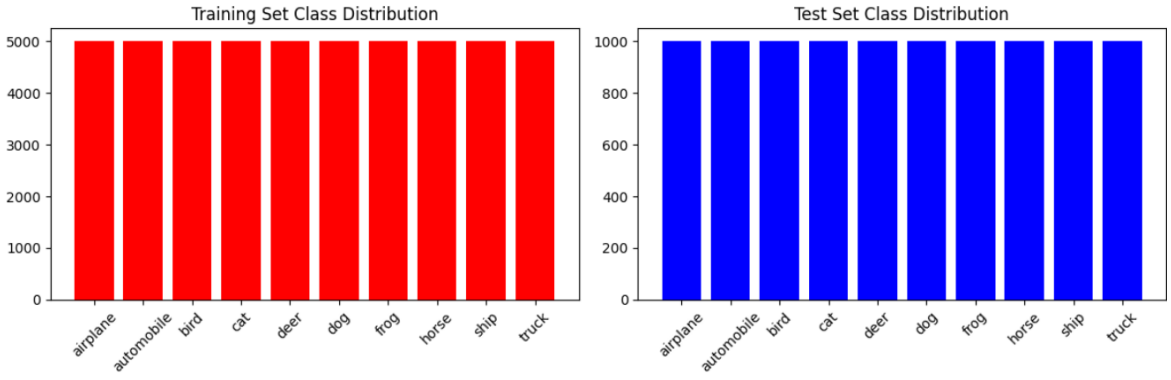
مدل در این شرایط مقاوم تر شده و قدرت تعمیم بهتری پیدا می‌کند. در مقاله نشان داده شده که Cluster های پیشبینی شده در فضای ویژگی ها، فاصله بیشتری نسبت به مدل Biased پیدا میکنند که میتواند باعث مقاوم تر شدن مدل نسبت به نویز و یا نمونه‌های تخاصمی بشود.

2-2: پیاده سازی و مقایسه روش های آموزش خصمانه

1: بارگذاری مجموعه دادگان CIFAR-10

در این بخش دیتا ست CIFAR10 با بارگذاری کرده و پیش پردازش های لازم را به آن اعمال میکنیم.

ابتدا برای آشنایی با دیتاست، نمودار فراوانی کلاس ها در مجموعه های آموزش و تست را رسم میکنیم.



شکل 19: نمودار فراوانی کلاس ها در مجموعه های آموزش و تست **CIFAR-10**

همانطور که مبینیم، دیتاست شامل 10 کلاس با لیبل های مشخص شده در تصویر هست. همچنین داده های هر کلاس در هر دو مجموعه آموزش و تست کاملا بالانس می‌باشند. حال 5 نمونه تصادفی از دادگان را انتخاب کرده و همراه لیبل آن نمایش می‌دهیم.

شکل 20: نمونه های تصادفی از دادگان آموزش دیتاست

حال تصاویر را پردازش میکنیم. برای این کار از مقادیر میانگین و انحراف معیار مختص CLIP برای نرمال سازی استفاده می‌کنیم. همچنین تصاویر را به ابعاد 224 در 224 در می‌آوریم.

```
CLIP_MEAN = [0.48145466, 0.4578275, 0.40821073]
CLIP_STD = [0.26862954, 0.26130258, 0.27577711]

transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(CLIP_MEAN, CLIP_STD)
])
```

حال مجموعه دادگان را به سه دسته آموزش، اعتبارسنجی و آزمایش تقسیم میکنیم. داده های آموزش دانلود شده را با نسبت 0.9 به دادگان آموزش و اعتبار سنجی تقسیم میکنیم. سایز بچ را برای دیتاست آموزش 64، و برای دیتاست اعتبارسنجی و آزمایش 32 قرار می‌دهیم.

```
Train samples: 45000
Validation samples: 5000
Test samples: 10000
Batch image shape: torch.Size([64, 3, 224, 224])
Batch labels shape: torch.Size([64])
```

شکل 21: سایز دسته های داده و ابعاد بچ آموزش

2: بارگذاری CLIP و مدل مولد نمونه‌های خصمانه

ابتدا مدل CLIP را دانلود کرده و در حالت ارزیابی قرار میدهیم.

```
model_name = "openai/clip-vit-base-patch32"

CLIP_model = CLIPModel.from_pretrained(model_name)
CLIP_processor = CLIPProcessor.from_pretrained(model_name)

CLIP_model.to(device)
CLIP_model.eval()
```

وظیفه CLPProcessor، آماده سازی ورودی ها برای اعمال به مدل هست. البته ما در بخش قبل تصاویر را نرمالایز کرده ایم ولی برای پردازش متن های ورودی از این تابع کمک میگیریم (کارهایی مانند توکنایز کردن، تبدیل توکن ها به ID و اعمال پدینگ یا برش باید در آن صورت میگیرد).

مدل ResNet-20 که روی CIFAR-10 آموزش دیده شده را نیز به عنوان مدل مولد نمونه های تخاصمی بارگذاری می‌کنیم. در حقیقت ما میخواهیم یک حمله انتقالی به مدل CLIP انجام دهیم، بنابراین از یک مدل دیگر برای تولید نمونه های خصمانه استفاده می‌کنیم.

```
adversarial_model = torch.hub.load("chenyaofo/pytorch-cifar-models",
"cifar10_resnet20", pretrained=True)
adversarial_model.to(device)
adversarial_model.eval()

for param in adversarial_model.parameters():
        param.requires_grad = False
```

حال بردارهای متنی متناظر با 10 کلاس را تولید کرده و آن ها را برای اعمال به مدل CLIP پردازش میکنیم.

```
Text prompts:
a photo of a airplane
a photo of a automobile
a photo of a bird
a photo of a cat
a photo of a deer
a photo of a dog
a photo of a frog
a photo of a horse
a photo of a ship
a photo of a truck
```

شکل 22: متن های ایجاد شده برای مدل **CLIP** برحسب کلاس های **CIFAR-10**

حال این متن ها را مطابق کد زیر پردازش می‌کنیم تا آماده اعمال به مدل CLIP شوند.

```
text_inputs = CLIP_processor(
    text=prompts,
    return_tensors="pt",
    padding=True
)
text_inputs = {key: value.to(device) for key, value in
text_inputs.items()}
```

سایز بردار هر یک از متن ها بعد از پردازش 512 میشود. همچنین بردارها را نرمالایز میکنیم. با اینکار اندازه همه بردار های متن یک شده و تنها جهت آن ها تفاوت دارد و از معیاری مثل شباهت کسینوسی میتوان برای یافتن شباهت بین بردارهای ویژگی استفاده نمود.

3: ارزیابی CLIP روی تصاویر تمیز و تخاصمی

تصاویر تمیز:

ابتدا مدل CLIP را روی تصاویر تست تمیز (10 هزار تصویر) ارزیابی میکنیم. برای اینکار تابعی مینویسیم که ابتدا بردار ویژگی تصویر را محاسبه کند، سپس آن را ضرب داخلی در بردار های ویژگی متن ها کند و متنی که بیشترین شباهت به آن را دارد به عنوان کلاس پیشبینی شده در نظر بگیرد. سپس این کلاس را با کلاس اصلی تصویر مقایسه کرده و دقت مدل را بسنجد.

دقت مدل روی داده های تمیز تست برابر 87.83٪ به دست آمد.

```
def evaluate_clip(model, dataloader, text_features, device):
    model.eval()
    total_correct = 0
    total_samples = 0
    with torch.no_grad():
        for images, labels in dataloader:
            images = images.to(device)  # Already normalized
```

```
            labels = labels.to(device)
            image_features =
model.get_image_features(pixel_values=images)
            image_features /= image_features.norm(dim=-1, keepdim=True)

            similarity = image_features @ text_features.T

            predictions = similarity.argmax(dim=1)

            total_correct += (predictions == labels).sum().item()
            total_samples += labels.size(0)

    accuracy = (total_correct / total_samples) * 100
    return accuracy
```

تصاویر تخاصمی:

ابتدا با اعمال حمله PGD به مدل ResNet-20 با تنظیمات ذکر شده در صورت پروژه، نمونه های تخاصمی تصاویر تست را ساخته و برای راحتی کار در ادامه، آن ها را در یک دیتالودر ذخیره میکنیم.

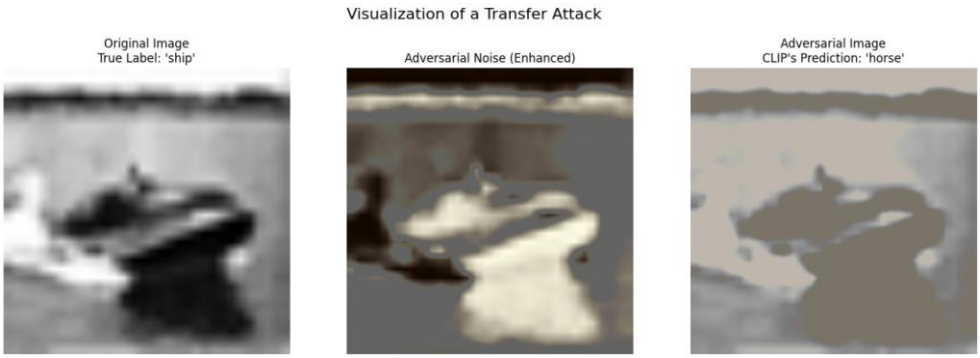
```
atk = torchattacks.PGD(adversarial_model, eps=8/255, alpha=2/255,
steps=7)
```

به همه داده های تست، تابع بالا را اعمال کرده و خروجی را ذخیره میکنیم.

```
adversarial_loader = DataLoader(adversarial_dataset, batch_size=32,
shuffle=False)
```

حال از همان تابع evaluate_clip استفاده میکنیم ولی اینبار داده های تخاصمی را ارزیابی میکنیم. دقت مدل روی این داده ها 54.67٪ بدست می‌آید که نشان دهنده افت دقت محسوس (حدود 33 درصد) است.

حال یک تصویر را انتخاب کرده و نمونه تخاصمی آن را محاسبه میکنیم. سپس با کم کردن این دو از هم نویز اضافه شده به تصویر را بدست می‌آوریم. نویز را برای نمایش بهتر نرمالایز می‌کنیم و هر سه تصویر را نمایش میدهیم.



شکل 23: نمونه ای از یک حمله تخاصمی

همانطور که میبینیم، نویز اضافه شده به تصویر کلیت آن را دچار تغییر نکرده و با چشم هنوز هم میتوان کلاس صحیح (کشتی) را تشخیص داد. ولی مدل دچار تشخیص اشتباه شده و کلاس تصویر را اسب پیشبینی کرده است!

همچنین میتوان درکی شهودی از نویز اضافه شده هم پیدا کرد. به نظر میرسد که نویز، کنتراست رنگی خلاف تصویر اصلی پیدا میکند (در جاهایی که پیکسل های تصویر اصلی سفیدترند، نویز به سمت طیف سیاه مقدار گرفته و بالعکس).

4: تنظیم دقیق خصمانه معمولی با روش LoRA

در این بخش میخواهیم با روش LoRA که در قسمت قبل توضیح دادیم، مدل CLIP را تنظیم دقیق کنیم تا عملکرد آن برروی داده های خصمانه بهتر شود. برای اینکار ابتدا تنظیمات Low-Rank Adaptation را روی ماژول بینایی مطابق زیر اعمال میکنیم. روش LoRA را به لایه های Attention معمولا اعمال میکنند. در این جا نیز ما به ماتریس های Key, Query, Value در لایه های توجه اعمال کرده ایم.

```
lora_config = LoraConfig(
    r=8,
    lora_alpha=32,
    target_modules=["q_proj", "k_proj", "v_proj"],
    lora_dropout=0.1,
    bias="none"
)

model = CLIPModel.from_pretrained(model_name).to(device)
lora_model = get_peft_model(model.vision_model, lora_config)
model.vision_model = lora_model

lora_model.print_trainable_parameters()
```

تعداد پارامترهای قابل آموزش اضافه شده به مدل به صورت زیر میباشد:

```
trainable params: 442,368 | all params: 87,898,368 | trainable%: 0.5033
```

همانطور که میبینیم تعداد پارامتر های قابل آموزش، کمتر از نیم درصد پارامتر های بخش بینایی ماشین می‌باشد.

برای آموزش مطابق صورت سوال، به تعداد نمونه های مجموعه داده تست (10 هزار) از داده های آموزش را انتخاب کرده و روی آنها آموزش میدهیم.

آموزش را یک ایپاک انجام میدهیم. سایز بچ ورودی ها 64 میباشد. همچنین بهینه ساز Adam با نرخ یادگیری 1e-4 را تنظیم میکنیم (البته در مقاله اشاره شده از بهینه سازی SGD با نرخ یادگیری 1e-5 و

momentum=0.9 استفاده شده. ما نیز ابتدا این تنظیمات را اعمال کردیم ولی نتایج خوبی مخصوصا روی داده های تخاصمی نمیداد).

برای آموزش هم مطابق خواسته سوال، ابتدا تصویر خصمانه بچ تصاویر ورودی را محاسبه کرده و نرمالایز میکنیم، سپس آنها را در بردارهای ویژگی متن ها ضرب داخلی کرده و تابع CrossEntropy را به عنوان تابع هزینه روی برچسب های اصلی اعمال میکنیم.

مقدار Loss مدل بعد از آموزش 0.78 شد. دقت مدل تنظیم شده در ارزیابی روی داده های تمیز برابر 93.29٪ و برروی داده های تخاصمی 83.68٪ بدست آمد که نشان دهنده حدود 30 درصد بهبود روی داده های تخاصمی است. نکته جالب اینکه دقت مدل روی داده های تمیز نیز حدود 5 درصد بهبود یافت. این امر میتواند به دلیل افزایش کلی مقاومت مدل نسبت به انواع نویز در تصویر ورودی باشد و به نظر میرسد پیشبینی مدل بر روی داده های ورودی نویزی نیز بهبود یافته است و به طور کلی قابلیت تعمیم مدل بهبود یافته است.

از طرف چون با داده های آموزشی دیتاست مدل را آموزش داده ایم، به نحوی مدل را برای این وظیفه خاص (پیشبینی کلاس های دیتاست CIFAR-10) تنظیم کرده ایم که باعث بهبود عملکرد کلی مدل می‌شود.

5: پیاده سازی تابع هزینه TeCoA

در این بخش الگوریتم TeCoA را مطابق رابطه زیر پیاده سازی کرده و مدل را با استفاده از این تابع هزینه جدید آموزش میدهیم.

$$\mathcal{L}_s(\mathbf{x}, \mathbf{t}, \mathbf{y}) = -\mathbb{E}_{i,j}\left[\mathbf{y}_{ij} \log \frac{\exp(\cos(\mathbf{z}_i^{(I)}, \mathbf{z}_j^{(T)})/\tau)}{\sum_k \exp(\cos(\mathbf{z}_i^{(I)}, \mathbf{z}_k^{(T)})/\tau)}\right]$$

تحلیل رابطه: در رابطه بالا، شباهت کسینوسی بردار های ویژگی تصاویر و متنون محاسبه میشود. شباهت کسینوسی را میتوان به سادگی با ضرب داخلی بردارهای نرمالایز شده ویژگی ها محاسبه کرد. سپس آن را در پارامتر τ ضرب کرده تا امتیازات را اسکیل دهی کنیم (هر چه مقدارش کوچکتر باشد، خروجی نهایی Softmax تیزتر می‌شود و یک کلاس با احتمال بالا تشکیل داده و بقیه احتمالات را نزدیک صفر میکند، و هرچه بزرگتر باشد احتمالات پخش تر می‌شوند). در مدل CLIP پارامتر logit_scale تعریف شده است که قابل یادگیری هست و همین نقش را دارد. برای عملکرد بهتر مدل، بجای اینکه مقدار τ را خودمان تعیین کنیم، آن را برابر پارامتر ذکر شده در CLIP قرار می‌دهیم.

همچنین اگر دقت کنیم بخشی که عبارت های نمایی را در مخرج جمع کرده و در صورت عبارت نمایی متناظر با نمونه مثبت (کلاس صحیح) را قرار داده، در حقیقت دارد تابع Softmax را به بردار های شباهت کسینوسی اعمال میکند. همینطور به مبنای لگاریتمی نیز تبدیل صورت گرفته است. پس ما میتوانیم بردار شباهت کسینوسی را محاسبه کرده و سپس تابع CrossEntropy را روی آن اعمال کنیم (در اصل عملکرد این بخش نیز تا حدی مشابه بخش 4 شد).

آموزش را یک ایپاک انجام میدهیم. سایز بچ ورودی ها 64 میباشد. همچنین بهینه ساز Adam را با سه نرخ یادگیری 1e-3 و 5e-4 و 1e-4 استفاده می‌کنیم.

کد بخش آموزش به صورت زیر می‌باشد:

```
model.train()
optimizer = torch.optim.AdamW(model.parameters(), lr=1e-4)
loss_fn = nn.CrossEntropyLoss()
num_epochs = 1

for epoch in range(num_epochs):
    progress_bar = tqdm(train_loader_10k, desc=f"Epoch {epoch+1}/{num_epochs} (TeCoA)")

    for images, labels in progress_bar:
        images, labels = images.to(device), labels.to(device)

        adv_images = atk(images, labels)


        image_features = model.get_image_features(pixel_values=adv_images)
        normalized_image_features = image_features / image_features.norm(dim=-1, keepdim=True)

        logits = normalized_image_features @ text_features.T * model.logit_scale.exp()

        loss = loss_fn(logits, labels)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        progress_bar.set_postfix({"Loss": loss.item()})
```

شکل 24: آموزش مدل با تابع هزینه **TeCoA**

همانطور که میبینیم برای محاسبه شباهت کسینوسی، بردارهای ویژگی تصاویر را نرمالایز کرده و در بردار ویژگی متون (که قبلا نرمالایز شده بودند) ضرب داخلی می‌کنیم.

برای نرخ یادگیری 1e-3، دقت مدل روی داده های تمیز 90.08٪ و برروی داده های تخاصمی 85.28٪ بدست آمد. همچنین مقدار Loss برابر 0.413 شد.

برای نرخ یادگیری 5e-4، دقت مدل روی داده های تمیز 88.14٪ و برروی داده های تخاصمی 86.50٪ بدست آمد. همچنین مقدار Loss برابر 0.319 شد.

برای نرخ یادگیری 1e-4، دقت مدل روی داده های تمیز 93.09٪ و برروی داده های تخاصمی 83.54٪ بدست آمد. همچنین مقدار Loss برابر 0.319 شد.

در این حالت نتایج تا حدی مشابه تنظیم دقیق عادی در بخش قبل شد که به دلیل شباهت الگوریتمی که در آن بخش برای محاسبه خطا اعمال کردیم به تابع خطای TeCoA، منطقی به نظر می‌آید.

همچنین از بین نرخ های یادگیری تست شده، 5e-4 بهترین دقت را روی داده های تخاصمی و نرخ 1e-5 بهترین دقت را روی داده های تمیز نتیجه داد.

6: مقایسه نتایج

در جدول زیر نتایج بخش های قبل را میتوان مشاهده نمود.

جدول 2: مقایسه نتایج بخش های 3 تا 5

Model	**Accuracy on clean images**	**Accuracy on adversarial images**
Base (CLIP)	87.83	54.67
LoRA + CrossEntropy	**93.29**	83.68
LoRA + TeCoA (lr=5e-4)	88.14	**86.50**
LoRA + TeCoA (lr=1e-3)	90.08	85.28

همانطور که میبینیم در روش تنظیم دقیق معمولی با LoRA توانستیم با آموزش تعداد بسیار کمی پارامتر (در مقایسه با پارامترهای مدل اصلی) عملکرد مدل را روی نمونه های متخاصم به شدت بهبود دهیم (حدود 30 درصد). همچنین همانطور که در بخش 4 توضیح دادیم، عملکرد مدل روی داده های تمیز نیز بهبود می‌یابد.

از طرفی با اعمال تابع هزینه TeCoA توانستیم به بهترین دقت روی داده های تخاصمی برسیم (حدود 3 درصد بهتر از تابع CrossEntropy). ولی در مقایسه با تنظیم دقیق معمولی، مدل روی داده های تمیز عملکرد ضعیف تری دارد (مدل در این حالت روی داده های تمیز تقریبا مشابه مدل پایه عمل میکند).

بنابراین اگر مقاومت مدل برایمان اهمیت داشته باشد و بخواهیم حتی در صورت حمله مدل دچار افت عملکرد نشود، به نظر مدل بهینه شده با TeCoA انتخاب مناسب تری باشد. ولی اگر دنبال بهترین عملکرد مدل روی داده های CIFAR-10 باشیم (مخصوصا تصاویر تمیز)، مدل بهینه شده با CrossEntropy را باید انتخاب کنیم.

از نظر محاسبات زمان آموزش، هر دو تقریبا مشابه یکدیگرند و برتری خاصی به هم ندارند (البته در تابع هزینه TeCoA، بردار ویژگی تصاویر نیز باید نرمالایز بشود).

7: تنظیم دقیق به روش VPT (Visual Prompt Tuning)

در این بخش می خواهیم بدون تغییر دادن وزن های مدل CLIP و تنها با اضافه کردن یک سری توکن قابل یادگیری به توکن های ورودی مدل، عملکرد آن را روی داده های خصمانه بهبود دهیم.

در مقاله دوروش کلی برای این کار ذکر شده است: اضافه کردن این اطلاعات به صورت پیکسل به تصویر اولیه و یا اضافه کردن آن ها به صورت توکن به دنباله توکن های ورودی. روش دوم گفته شده که کارآمدتر است، برای همین نیز ما در اینجا از همین روش استفاده میکنیم.

```
prompt_length = 10
embed_dim = vpt_model.config.vision_config.hidden_size
visual_prompt = nn.Parameter(torch.randn(1, prompt_length, embed_dim,
device=device))
visual_prompt.requires_grad = True
```

تعداد توکن هایی که اضافه میکنیم هایپرپارامتر مسئله ما میباشد. در ابتدا 10 توکن را به ورودی اضافه میکنیم. سایز توکن ها در مدل CLIP برابر 768 میباشد. بنابراین با اضافه کردن 10 توکن، در کل 7680 پارامتر قابل یادگیری به مدل اضافه کرده ایم.

```
def get_image_features_with_prompt(model, pixel_values, prompt):

    image_embeds = model.vision_model.embeddings(pixel_values)

    prompt_batch = prompt.expand(image_embeds.shape[0], -1, -1)
    input_with_prompt = torch.cat([
        image_embeds[:, :1, :],
        prompt_batch,
        image_embeds[:, 1:, :]
    ], dim=1)

    hidden_states = model.vision_model.pre_layrnorm(input_with_prompt)


    encoder_outputs = model.vision_model.encoder(hidden_states)

    pooled_output = encoder_outputs.last_hidden_state[:, 0, :]
    pooled_output = model.vision_model.post_layernorm(pooled_output)

    projected_output = model.visual_projection(pooled_output)

    return projected_output
```

تابع بالا را برای افزودن توکن های قابل یادگیری افزوده شده به دنباله توکن های تصویر استفاده میکنیم. برای اینکار تصویر ورودی را ابتدا امبد میکنیم. سپس توکن های افزوده شده را به ابتدای دنباله توکن ها اضافه میکنیم (البته توکن اولیه که CLS نام دارد و در نهایت برای کلاس بندی استفاده میشود را همان اول دنباله نگه میداریم). سپس بقیه لایه های بخش کدگذار تصویر مانند Pre_LayerNorm و Post_LayerNorm و Encoder را به این دنباله جدید توکن اعمال کرده تا در نهایت خروجی بخش کدگذار تصویر (با اعمال توکن های اضافه شده) تولید شود

در هنگام آموزش نیز ابتدا تصویر را به نمونه متخاصم تبدیل کرده و سپس بردار ویژگی جدید را به تابع بالا اعمال کرده و بردار ویژگی های استخراج شده را به تابع هزینه TeCoA اعمال کرده و گرادیان ها را محاسبه میکنیم.

آموزش را یک ایپاک انجام میدهیم. سایز بچ ورودی ها 64 میباشد. همچنین بهینه ساز Adam با نرخ یادگیری 1e-2 را تنظیم میکنیم. (نرخ یادگیری در این مسئله باید بزرگ باشد، چون تغییری که ایجاد میکنیم به پارامتر های اصلی مدل اعمال نمیشود و داریم تعدادی پارامتر را از نو بهینه سازی میکنیم. نتایجی که با نرخ های یادگیری کوچکتر بدست آمدند به خوبی نتایج نهایی نبودند)

مقدار Loss مدل بعد از آموزش 0.642 شد. دقت مدل در ارزیابی روی داده های تمیز برابر 86.82٪ و برروی داده های تخاصمی 64.14٪ بدست آمد که نشان دهنده حدود 10 درصد بهبود روی داده های تخاصمی است.

برای تعداد توکن های کمتر، عملکرد مدل برروی داده های متخاصم ضعیف تر شد و عملکرد آن روی داده های تمیز تغییر محسوسی نکرد. برای تعداد توکن بیشتر (20 توکن) نیز مدل را آموزش دادیم که نتایج تغییری با حالت 10 توکن نکرد.

در مقایسه با دو روش قبلی، این روش نتوانست بهبود قابل ملاحضه ای روی داده های متخاصم بدهد و تنها 10 درصد عملکرد بهتری از مدل پایه دارد. ولی از طرف دیگر روش VPT با 10 توکن تنها 7680 پارامتر قابل یادگیری دارد، که در مقایسه با 440 هزار پارامتر در روش های قبل ناچیز است، بنابراین سرعت آموزش بیشتری دارد.

برای عملکرد بهتر مدل، تعداد ایپاک های آموزش را از یک به پنج افزایش میدهیم.

```
Epoch 1/5 (VPT + TeCoA): 100%| 157/157 [2.64s/it, Loss=0.835]
Epoch 2/5 (VPT + TeCoA): 100%| 157/157 [2.61s/it, Loss=0.645]
Epoch 3/5 (VPT + TeCoA): 100%| 157/157 [2.61s/it, Loss=0.675]
Epoch 4/5 (VPT + TeCoA): 100%| 157/157 [2.61s/it, Loss=0.75]
Epoch 5/5 (VPT + TeCoA): 100%| 157/157 [2.61s/it, Loss=0.974
```

دقت مدل در این حالت ارزیابی روی داده های تمیز برابر 88.52٪ و برروی داده های تخاصمی 74.02٪ بدست آمد که نشان دهنده حدود 20 درصد بهبود روی داده های تخاصمی است.

این مدل جدید با حفظ دقت مدل برروی داده های تمیز، دقت آن روی داده های متخاصم را نیز تا حد خوبی افزایش داده است و با توجه به حجم بسیار کمتر پارامترهای آموزش داده شده نسبت به بخش های 4 و5، میتواند به عنوان روش مناسبی استفاده بشود.