
 <p>دانشگاه صنعتی خواجه نصیرالدین طوسی دانشکده مهندسی برق - گروه مهندسی کنترل</p>	<p>به نام خدا دانشگاه تهران - دانشکده صنعتی خواجه نصیرالدین طوسی تهران دانشکده مهندسی برق و کامپیوتر</p>	
<p>طبقه بندی و تحلیل تصاویر ماهواره ای با یادگیری عمیق</p>		

<p>محمد جواد احمدی</p>	<p>نام و نام خانوادگی</p>
<p>۴۰۱۰۰۰۸۶</p>	<p>شماره دانشجویی</p>

فهرست مطالب

۴	پاسخ پرسش اول	۱
۴	پاسخ قسمت ۱ - گزارشی از مقاله	۱.۱
۵	پاسخ قسمت ۲ - توضیح معماری شبکه و پیش‌پردازش‌های لازم	۲.۱
۱۰	پاسخ قسمت ۳ - توضیح دامنه معتبر تصاویر	۳.۱
۱۱	پاسخ قسمت ۴ - دریافت و تحلیل دادگان	۴.۱
۱۵	پاسخ قسمت ۵ - پیاده‌سازی شبکه و نتایج	۵.۱
۱۵	پیاده‌سازی اول	۱.۵.۱
۳۱	پیاده‌سازی دوم	۲.۵.۱

فهرست تصاویر

۶	شماری از تصاویر مجموعه داده در مقاله	۱
۶	میدان پذیرنده	۲
۷	معماری VGGNet	۳
۸	اطلاعات مدل‌های مختلف معماری VGGNet	۴
۱۴	نمایش تصاویر مجموعه داده در باندهای مختلف	۵
۲۰	نمودار توزیعی داده‌ها	۶
۲۳	تصاویر مربوط به یک دسته	۷
۲۷	نمودار تابع اتلاف	۸
۲۷	نمودار دقت	۹
۲۹	نتیجه ارزیابی مدل آموزش دیده روی یک دسته تصادفی از تصاویر آزمون	۱۰
۳۱	نمودار اصلی ماتریس درهم‌ریختگی	۱۱
۳۲	نتیجه ارزیابی مدل آموزش دیده روی یک دسته تصادفی از تصاویر آزمون	۱۲
۳۴	نمودار توزیعی داده‌ها در پیاده‌سازی دوم	۱۳
۳۵	نمودار دقت و اتلاف و ماتریس درهم‌ریختگی در پیاده‌سازی دوم	۱۴
۳۵	نتایج برخی آزمایش‌ها در پیاده‌سازی دوم	۱۵

پرسش ۱. آشنایی با یادگیری انتقالی – مورد دوم

۱ پاسخ پرسش اول

توضیح پوشه کدهای آشنایی با یادگیری انتقالی

کدهای مربوط به این قسمت، علاوه بر پوشه محلی کدها در این لینک گوگل کولب آورده شده است. مدل‌های ذخیره شده هم از طریق این لینک در دسترس هستند.

۱.۱ پاسخ قسمت ۱ – گزارشی از مقاله

در مقاله این‌گونه قید شده است که در علم سیاره‌شناسی، شناسایی و طبقه‌بندی ویژگی‌های توپوگرافی و ژئومورفولوژی از داده‌های بزرگ، یک کار پایه‌ای و مهم است. این مقاله یک مدل سبک مبتنی بر VGG-16 پیشنهاد می‌دهد که می‌تواند برخی ویژگی‌های تصاویر سیارات را به صورت منتخب استخراج کند، اطلاعات اضافی را حذف کرده و تصاویر را شناسایی و طبقه‌بندی کند. این مدل نه تنها دقت را تضمین می‌کند، بلکه پارامترهای مدل را هم کاهش می‌دهد. طبق نتایج آزمایشی مقاله، مدل در طبقه‌بندی تصاویر بهبود قابل ملاحظه‌ای داشته است و دقت آن به ۸۵ تا ۹۸ درصد رسیده است. در عین حال، این مدل سرعت همگرایی و عملکرد طبقه‌بندی را بهبود بخشیده است. با ورود داده‌های تصویر حسگری با پیکسل‌های فوق پایین (64×64) به مدل، ثابت شده که مدل پیشنهادی مقاله برای تصاویر با پیکسل‌های فوق پایین و دارای کمترین نقاط ویژگی، هم‌چنان دارای نرخ دقت بالا است. به همین دلیل، این مدل دارای چشم‌انداز کاربرد خوبی در طبقه‌بندی دقیق تصاویر ماهواره‌ای، حتی با پیکسل‌های بسیار پایین و کمترین ویژگی‌هاست.

این مقاله در قسمت مقدمه خود به برخی کاربردهای طبقه‌بندی تصاویر ماهواره‌ای اشاره می‌کند. از جمله این کاربردها می‌توان به شناسایی خطرات طبیعی، بازیابی تصاویر جغرافیایی و برنامه‌ریزی مناسب شهری اشاره کرد. این مقاله اشاره می‌کند که در ابتدا، استفاده از ویژگی‌های دست‌ساز و کلاسیک مانند هیستوگرام‌های رنگ رواج زیادی در طبقه‌بندی تصاویر سنجش از راه دور داشته است. اما این روش‌های قابلیت تعمیم‌پذیری بالایی ندارند و کاملاً به شرایط محیطی خاصی وابسته هستند. مخصوصاً که تنوع درون‌کلاسی در این نوع تصاویر بالا و تنوع بین‌کلاسی بعضاً پایین است.

این مقاله این‌گونه ادامه می‌دهد که در سال‌های اخیر، شبکه‌های عصبی پیچشی یا کانولوشنی (CNN) قدرت تعمیم‌پذیری بالایی در دسته‌بندی تصاویر طبیعی از خود نشان داده‌اند. این مقاله دلیل اصلی دقت بالای این مدل‌ها در دسته‌بندی تصاویر را افزایش تعداد مجموعه داده‌های آموزشی با برچسب مناسب می‌داند؛ چراکه، روش‌های مبتنی بر یادگیری عمیق به داده‌های بالایی نیاز دارند. در سال‌های اخیر، استفاده از مدل‌های کانولوشنی برای دسته‌بندی تصاویر ماهواره‌ای سنجش از راه دور نیز رواج یافته است. با این حال، به دلیل محدودیت در تعداد دیتاست‌های برچسب‌دار در این حوزه، مدل کانولوشنی می‌بایست از ابتدا برای این دیتاست‌ها آموزش داده شود. برای حل این مشکل، از روش یادگیری انتقالی استفاده می‌شود. برخی محققان از تصاویر از راه دور موجود برای رفع این مشکل استفاده کرده‌اند. در عین حال، برخی محققان از مدل‌های کانولوشنی‌ای که از پیش با مجموعه داده‌هایی بزرگ آموزش دیده‌اند استفاده کرده‌اند. از جمله مدل‌های پیش‌آموزش دیده استفاده شده می‌توان به AlexNet، GoogleNet و VGGNet اشاره کرد. به طور کلی، به دلیل قابلیت پیاده‌سازی بالای لایه‌های پیچشی مدل کانولوشنی، برای

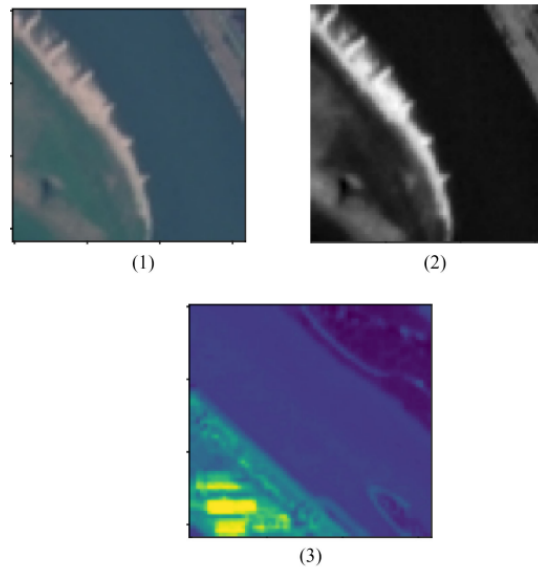
سازگار کردن معماری با وظایف دسته‌بندی جدید، از روش تنظیم مجدد یا تدقیق استفاده می‌شود. نسبت به روش‌های دسته‌بندی سنتی، این مدل‌ها عملکرد دسته‌بندی بهتری دارند. این شبکه‌های پیش‌آموزش دیده توانایی بالایی در استخراج ویژگی‌های اولیه دارد و این به کار طبقه‌بندی روی مجموعه داده هدف کمک به‌سزایی می‌کند.

این مقاله این‌گونه بیان می‌کند که شبکه‌های عصبی کانولوشن در حوزه ماهواره‌ای و سنجش از دور به‌صورت گسترده استفاده می‌شوند. از جمله زمینه‌های کاربردی اشاره شده می‌توان به کاوش معدن، پایش کاربری اراضی و غیره اشاره کرد. این مقاله در ادامه به این نکته اشاره می‌کند که در کاربردهای مربوط به شبکه‌های کانولوشنی، تصاویر و اشیاء باید در اشکال و اندازه‌های مختلف به یک اندازه ثابت تغییر شکل داده شوند، زیرا این شبکه‌ها به یک اندازه ثابت نیاز دارند و این عمل باعث از دست دادن اطلاعات شکل و اندازه شیء می‌شود. این شبکه به برخی مشکلات استفاده از این مدل‌ها و شبکه‌ها هم اشاره می‌کند. مشکلاتی مانند نیازمندی به ویژگی‌های غنی و مناسب، تعداد پارامترهای بالا و درگیری بالای حافظه. این مقاله مدعی شده که با استفاده از مدل VGG-16 نیم‌نگاهی به حل این پالش‌ها و مشکلات داشته است. این مقاله مدعی است که مدل بهبودیافته‌اش، توانایی سازگاری کلی با تصاویر حسگری با دقت بالا و پایین را داراست. همچنین، هنگام استفاده از تصاویر حسگری با تعداد پیکسل کم، همچنان قابل اعتنا است. در عین حال، این مدل، تعداد پارامترها و اندازه حافظه مورد نیاز برای مدل را نیز کاهش می‌دهد. بنابراین، در این مقاله، یک مدل سبک بر پایه VGG-16 پیشنهاد شده است که با حل مشکلات فوق طراحی شده است.

با ذکر این که توضیحات مقاله در مورد مدل و پیش‌پردازش مورد استفاده را در **بخش ۲.۱** اشاره خواهیم کرد، به سراغ قسمت توضیحات آموزش در مقاله می‌رویم. این مقاله این‌گونه عنوان کرده که در آموزش مدل، اندازه دسته را ۳۲، مومنتوم را ۰.۹ و نرخ یادگیری را ۰.۰۰۱ در نظر گرفته است. آموزش با کاهش وزن و رگولایزاسیون Dropout با مقدار ۰.۵ و دو لایه Dense انجام شده است. هم‌چنین این‌گونه اشاره شده که در فرآیند کدنویسی از چهارچوب Keras استفاده شده است و میانگین زمان آموزش هر مدل شبکه ۲۵ ساعت ذکر شده است. این مقاله در بخش آزمون، با ورودی تصاویر، شبکه آموزش داده شده را تغذیه کرده است. ابتدا، تصویر را به جهت‌های مختلف خودش چرخانده تا کوچکترین لبه مشخص شود. سپس، شبکه به تصاویر آزمونی مقیاس دار با استفاده از روش مشابهی بکار گرفته می‌شود. به عبارت دیگر، ابتدا لایه تماماً متصل به یک لایه پیچشی یا کانولوشنی تبدیل می‌شود. سپس، شبکه تماماً کانولوشنی حاصل، به تمام تصویر (بدون کوچک کردن) اعمال می‌شود. نتیجه، یک نقشه کسر کلاس است که تعداد کانال‌های آن برابر با تعداد کلاس‌ها است و متغیر رزولوشن فضایی وابسته به اندازه تصویر ورودی است. در نهایت، برای به‌دست آوردن برداری با اندازه ثابت از تصویر، تصویر به صورت فضایی میانگین‌گیری شده و به اصطلاح sum pool شده است. این مقاله این‌گونه ذکر کرده که نمودار تابع نرخ خطا و نمودار تابع دقت کمی شلوغ و با هم اشتباه گرفته می‌شوند. این به دلیل این است که پیکسل‌های تصویر در مجموعه داده خود به اندازه کافی کم هستند و تعداد نقاط ویژگی بارز در تصویر کم است، که باعث می‌شود تصاویر شبیه به هم باشند. بنابراین، تصاویر نرخ خطا و تصویر نرخ دقت کمی غیرطبیعی هستند. به عنوان مثال، برای جنگل‌ها و چمنزارها، دریاچه و رودخانه. در **شکل ۱** شماری از تصاویر مربوط به مقاله آورده شده است.

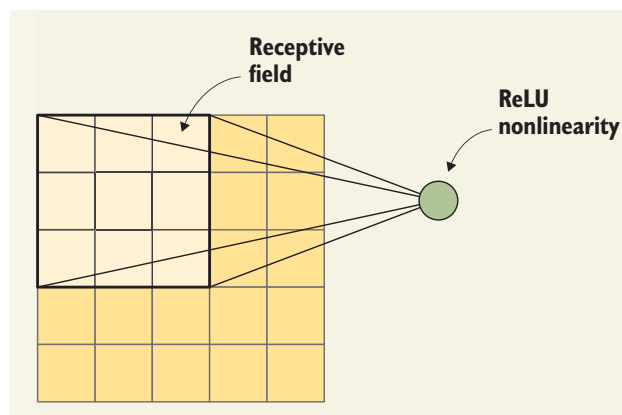
۲.۱ پاسخ قسمت ۲ - توضیح معماری شبکه و پیش‌پردازش‌های لازم

VGG ها از سال ۲۰۱۴ و در دانشگاه آکسفورد توسعه داده شده‌اند. مولفه‌های این مدل شبیه به LeNet و AlexNet بوده، به جز این که این شبکه در تعداد لایه‌های کانولوشن، رأی‌گیری و تماماً متصل با مدل‌های پیشین تفاوت دارد. VGG-16 از ۱۶ لایه شامل ۱۳ لایه کانولوشنی و سه لایه تماماً متصل تشکیل شده است. در این شبکه، AlexNet با جایگزینی فیلترهای اندازه کرنل بزرگ (به ترتیب ۱۱ و ۵ در اولین و دومین لایه‌های کانولوشن)، با چند فیلتر اندازه رأی‌گیری سه‌درسه یکی پس از دیگری بهبود داده می‌شود. معماری از یک سری واحدهای کانولوشنی همگن بهم‌پیوسته با یک لایه اجماع یا رأی‌گیری متحد تشکیل



شکل ۱: شماری از تصاویر مجموعه داده در مقاله.

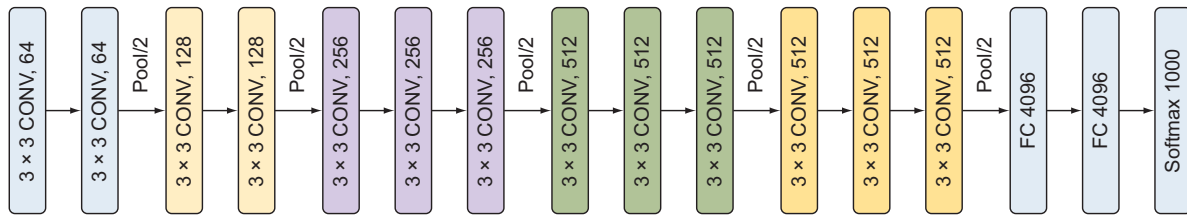
شده است. همه لایه‌های کانولوشنی فیلترهایی با اندازه کرنل سه‌درسه و پرش یک داشته و در لایه‌گذاری هم مشابه هستند. همه لایه‌های اِجماع هم اندازه دودردو داشته و پرش دو دارند. تصمیم در استفاده از کرنل‌های کوچکتر سه‌درسه اجازه به شبکه برای استخراج ویژگی‌های سطح دقیق‌تر تصویر در مقایسه با کرنل‌های بزرگ‌تر AlexNet (یازده و پنج‌تایی) می‌دهد. ایده این است که با یک میدان پذیرند کانولوشنی خاص، چند کرنل اندازه کوچک‌تر انباشت شده، بهتر از یک کرنل بزرگ‌تر بوده، چون لایه‌های چندگانه غیرخطی عمق شبکه را افزایش داده و به علت تعداد کمتری پارامتر یادگیری، می‌تواند ویژگی‌های پیچیده‌تری را با هزینه کم‌تر یاد بگیرد. به عنوان مثال در آزمایش‌های خود سازندگان توجه شده که انباشت دو لایه کانولوشنی با اندازه سه‌درسه (بدون اِجماع مکانی در بین)، میدان پذیرندگی اثربخش پنج‌در پنج داشته و سه لایه سه‌درسه، اثر میدان پذیرندگی هفت‌در هفت را داراست. میدان پذیرنده سطح موثر تصویر ورودی بوده که خروجی به آن وابستگی دارد (شکل ۲). بنابراین با استفاده از



شکل ۲: میدان پذیرنده.

کانولوشن‌های سه‌درسه با عمق بیش‌تر، مزایای استفاده از لایه‌های هم‌سوسازی غیرخطی بیش‌تری حاصل شده که قدرت تمایز

بیش‌تری برای تابع تصمیم ایجاد می‌کند. هم‌چنین این امر تعداد پارامترهای آموزش را کاهش می‌دهد؛ چراکه، در هنگام استفاده از کانولوشن با اندازه سه‌درسه با C کانال، انباشت با $27C^2 = 3^2C^2$ وزن ایجاد می‌شود که در مقایسه با لایه کانولوشنی هفت‌درهفت تکی پارامتریته شده که مستلزم $49C^2 = 7^2C^2$ وزن است، ۸۱ درصد پارامتر کم‌تری دارد. معماری VGGNet با انباشت کردن لایه‌های کانولوشن سه‌درسه با گنجانیدن لایه‌های رأی‌گیری دودردو پس از چند لایه کانولوشنی توسعه داده می‌شود. به دنبال این معماری، طبقه‌گر سنتی می‌آید که از لایه‌های تماماًمتصل و یک سافت‌مکس تشکیل شده است (شکل ۴). چند پیکربندی برای



شکل ۳: معماری VGGNet.

معماری VGGNet با طراحی ژنریک مشابه ایجاد شده است (؟؟). VGG-16 و VGG19 متداول‌ترین نوع مورد استفاده در این خانواده هستند. VGG-16 به دلیل عملکرد مشابه VGG19 و تعداد پارامتر کم‌تر محبوبیت بیش‌تری دارد. در برنامه ۱ پیاده‌سازی معماری VGG16 آورده شده است. سازندگان این معماری برای جلوگیری از بیش‌برازش از تنظیم $L2$ با زوال وزن 0.0005 و تنظیم دورریز برای اولین دو لایه تماماًمتصل با نسبت 0.5 استفاده کرده‌اند.

Program 1: VGG-16 Implementation

```
1 model = Sequential()
2 # block #1
3 model.add(Conv2D(filters=64, kernel_size=(3,3), strides=(1,1),
4 activation='relu',
5 padding='same', input_shape=(224,224, 3)))
6 model.add(Conv2D(filters=64, kernel_size=(3,3), strides=(1,1),
7 activation='relu',
8 padding='same'))
9 model.add(MaxPool2D((2,2), strides=(2,2)))
10 # block #2
11 model.add(Conv2D(filters=128, kernel_size=(3,3), strides=(1,1),
12 activation='relu',
13 padding='same'))
14 model.add(Conv2D(filters=128, kernel_size=(3,3), strides=(1,1),
15 activation='relu',
16 padding='same'))
17 model.add(MaxPool2D((2,2), strides=(2,2)))
18 # block #3
19 model.add(Conv2D(filters=256, kernel_size=(3,3), strides=(1,1),
20 activation='relu',
21 padding='same'))
22 model.add(Conv2D(filters=256, kernel_size=(3,3), strides=(1,1),
23 activation='relu',
24 padding='same'))
25 model.add(Conv2D(filters=256, kernel_size=(3,3), strides=(1,1),
26 activation='relu',
27 padding='same'))
28 model.add(MaxPool2D((2,2), strides=(2,2)))
29 # block #4
30 model.add(Conv2D(filters=512, kernel_size=(3,3), strides=(1,1),
31 activation='relu',
32 padding='same'))
33 model.add(Conv2D(filters=512, kernel_size=(3,3), strides=(1,1),
34 activation='relu',
35 padding='same'))
```

ConvNet configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
Input (224 x 224 RGB image)					
conv3-64	conv3-64 LRN	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 conv3-256	conv3-256 conv3-256 conv3-256	conv3-256 conv3-256 conv3-256
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512
maxpool					
FC-4096					
FC-4096					
FC-1000					

Network	A, A-LRN	B	C	D	E
No. of parameters	133	133	134	138	144

شکل ۴: اطلاعات مدل‌های مختلف معماری VGGNet (تعداد پارامترها به میلیون ذکر شده).

```

36 model.add(Conv2D(filters=512, kernel_size=(3,3), strides=(1,1),
37 activation='relu',

```



```

38 padding='same'))
39 model.add(MaxPool2D((2,2), strides=(2,2)))
40 # block #5
41 model.add(Conv2D(filters=512, kernel_size=(3,3), strides=(1,1),
42 activation='relu',
43 padding='same'))
44 model.add(Conv2D(filters=512, kernel_size=(3,3), strides=(1,1),
45 activation='relu',
46 padding='same'))
47 model.add(Conv2D(filters=512, kernel_size=(3,3), strides=(1,1),
48 activation='relu',
49 padding='same'))
50 model.add(MaxPool2D((2,2), strides=(2,2)))
51 # block #6 (classifier)
52 model.add(Flatten())
53 model.add(Dense(4096, activation='relu'))
54 model.add(Dropout(0.5))
55 model.add(Dense(4096, activation='relu'))
56 model.add(Dropout(0.5))
57 model.add(Dense(1000, activation='softmax'))
58 model.summary()
59 -----
60 Total params: 138,357, 544
61 Trainable params: 138,357, 544
62 Non-trainable params: 0

```

در ادامه توضیحات جامانده از مقاله در بخش ۱.۱ در مورد مدل استفاده را ذکر می‌کنیم. مقاله این‌گونه بیان کرده که مدل VGG یک مدل کانولوشنی است که در مقاله "شبکه‌های کانولوشنی عمیق برای تشخیص تصاویر بزرگ" توسط سیمونیان و زیسرمن پیشنهاد شده است. هسته کانولوشن کوچکتر از هسته کانولوشن بزرگ بهتر است؛ زیرا لایه‌های غیرخطی چندلایه می‌توانند عمق شبکه را افزایش داده و برای یادگیری الگوهای پیچیده‌تر اطمینان بیشتری حاصل کنند و پارامترها و هزینه نسبتاً کمتری هم داشته باشند. اما VGG مصرف منابع محاسباتی بیشتری دارد و از پارامترهای بیشتری استفاده می‌کند که منجر به مصرف حافظه بیشتر می‌شود. بیش‌تر پارامترها از لایه تماماًمتصل اول نشأت می‌گیرند و VGG-16 سه لایه تماماًمتصل دارد. تصاویر مجموعه داده آزمایشی به تصاویر از راه دوری تعلق دارند که تعداد پیکسل پایینی دارد و نیازی به استخراج چندین ویژگی و پارامترهای بیشتر نیست. بنابراین، مقاله VGG-16 اصلی را با مدلی کاملاً پیچشی ترکیب کرده و پارامترهای مدل و تعداد لایه‌های تماماًمتصل را کاهش داده است. این علاوه بر اطمینان از دقت استخراج ویژگی مدل، سبک شدن مدل و بهبود سرعت آموزش مدل را نیز به دنبال خواهد داشت. مدل پیشنهادی مقاله بر اساس مدل اصلی VGG-16 است و با مدل کاملاً کانولوشنی سنتی ترکیب شده است. مقاله ابتدا با لایه کانولوشنی 32×32 شروع کرده و پس از هر لایه کانولوشن، یک لایه اجماع بیشینه با ابعاد 2×2 اضافه می‌کند. سپس، لایه‌های کانولوشنی ۴۴، ۱۲۸، ۲۵۶ و ۵۱۲ تایی را اضافه کرده و پس از هر لایه کانولوشنی یک لایه اجماع بیشینه با همان ابعاد قبلی ذکر شده اضافه می‌کند. مقاله لایه کانولوشنی ۵۱۲ تایی اصلی را که در vgg-16 نیاز به دو بار عبور دارد، یکبار از آن عبور می‌دهد. در آن اندازه لایه کانولوشن 3×3 و اندازه گام ۲ است. لایه تماماًمتصل اول ۴۰۹۶ کانال دارد، لایه دوم طبقه‌بندی ۱۰۰۰ راه ILSVRC^۱ را انجام می‌دهد و آخرین لایه، لایه سافت مکس است.

از مزایای مدل VGG-16 می‌توان به دقت بالا، امکان استفاده در یادگیری انتقالی (وجود مدل‌های متعدد پیش‌آموزش دیده)، معماری ساده و کم‌تر چپیده نسبت به برخی معماری‌های دیگر مانند اینسپشن و رزنت، و انعطاف‌پذیری برای تغییر و تدقیق اشاره کرد. از معایب این مدل هم می‌توان به تعداد پارامتر بالا، نیاز به حافظه بالا (محدودیت حافظه)، زمان اجرای بالا (سرعت پایین)، احتمال میل به بیش‌برازش مخصوصاً برای مجموعه داده‌های کوچک، و توانایی تعمیم‌پذیری پایین اشاره کرد.

^۱ طبقه‌بندی ۱۰۰۰ شاخه‌ای ILSVRC به معنی طبقه‌بندی تصاویر به ۱۰۰۰ دسته مختلف است که بر اساس مجموعه داده ImageNet Large Scale Visual Recognition Challenge (ILSVRC) انجام می‌شود. ILSVRC یک مسابقه بینایی ماشین سالانه است که در آن شرکت‌کنندگان الگوریتم‌هایی را برای کارهای تشخیص تصاویر، شناسایی و مکان‌یابی توسعه می‌دهند. ۱۰۰۰ دسته در ILSVRC متنوع هستند و گستره‌ای از اشیاء و صحنه‌ها را پوشش می‌دهند، از جمله حیوانات، وسایل نقلیه و غذا.

این مقاله برای پیش‌پردازش داده‌ها، ابتدا اندازه تصاویر مجموعه داده را به 64×64 تغییر می‌دهد، سپس تصویر را خاکستری می‌کند و به یک ماتریس داده $n \times p$ (n = تعداد نمونه‌ها و p = تعداد پیکسل‌ها در هر تصویر) تبدیل می‌کند. سپس بردار تخت را به آرایه ایجادشده درج کرده و ویژگی‌های مشخص تصویر را با عملیات رشته‌ای نمایش می‌دهد. در نهایت، پیکسل‌های تصویر را بر ۲۲۵ تقسیم کرده و آن را مقیاس و نرمال می‌کند. مجموعه داده مورد استفاده EuroSAT نام دارد. تصاویر این مجموعه داده توسط ماهواره Sentinel-2 جمع‌آوری شده و شامل ۱۳ باند طیفی، ۱۰ دسته‌بندی و در مجموع ۲۷۰۰۰ تصویر کاربردی با برچسب و مختصات جغرافیایی است. این مجموعه داده برای طبقه‌بندی و تشخیص تغییر پوشش زمین و مشکلات دیگری که به بهبود محیط زیست جغرافیایی کمک می‌کند، استفاده می‌شود. مجموعه داده شامل ده دسته‌بندی زیر است، که هر کدام شامل ۲۰۰۰ تا ۳۰۰۰ عکس با اندازه پیکسلی 64×64 هستند. پس از پردازش تصویر، تصویر را با نسبت ۲ : ۸ برای مجموعه‌های آموزشی و آزمایشی تقسیم‌بندی می‌کنیم.

۳.۱ پاسخ قسمت ۳ - توضیح دامنه معتبر تصاویر

یک مدل VGG-16 پیش‌آموزش دیده قابلیت تشخیص تصاویر در بسیاری از دسته‌بندی‌ها را داراست، از جمله حیوانات، وسایل نقلیه، غذا، مناظر، و غیره. اما در صورتی که عکس مورد نظر در هیچکدام از دسته‌بندی‌های مدل قرار نگیرد، مدل نمی‌تواند به درستی آن را تشخیص دهد و احتمالاً به عنوان دسته "نامعلوم" یا "دیگر" شناسایی خواهد شد. در این صورت، برای تشخیص صحیح تصویر، می‌توان از مدل‌های دیگری که در دسته‌بندی مورد نظر عکس کارایی دارند استفاده کرد و یا مدل خود را با تصاویر مربوط به دسته‌بندی جدیدی که عکس مورد نظر در آن قرار می‌گیرد، آموزش داد و تدقیق کرد. علاوه بر این‌ها، یکی از چالش‌های مهم مجموعه داده این مقاله، سروکار داشتن با تصاویر دارای ابعاد پیکسلی و تعداد ویژگی پایین است. مدل انتخابی ممکن است توانایی خودی در برابر این تصاویر از خود نشان ندهد. برای این منظور پیشنهادهایی ارائه شده است. مقاله این گونه بیان داشته که یک لایه تصحیح غیرخطی پس از هر لایه کانولوشن اضافه کرده است، به گونه‌ای که هر لایه از دو لایه تصحیح غیرخطی بهره می‌برد و به جای یک لایه تصحیح، دو لایه تصحیح را استفاده می‌کند. این کار به دلیل پیکسل‌های بسیار پایین و اندکی که در مجموعه داده مورد استفاده قرار می‌گیرند و ویژگی کافی برای تشخیص بخش‌های مختلف تصاویر فراهم نیستند، صورت گرفته است. اضافه کردن دو لایه تصحیح غیرخطی مشکلات ناشی از ناپدید شدن گرادیان را رفع کرده، احتمال بیش‌برازش را کاهش داده و سرعت آموزش مدل را بهبود می‌بخشد. مقاله به طور کلی روی تصاویر با پیکسل بسیار پایین تمرکز دارد؛ بنابراین، نیاز به افزایش اندازه مجموعه داده حس می‌شود، در غیر این صورت تصویر خود به خاطر اندازه کوچک، پس از چندین لایه پیچش، نمی‌تواند تصویر را خروجی دهد یا دقت مدل را کاهش می‌دهد. علاوه بر این، در تصویر با پیکسل پایین، تعداد نقاط ویژه و ویژگی‌ها کم است، بنابراین استخراج دقیق آن‌ها دشوار است. برای حل این مشکلات، مقاله یک لایه zero padding در اول هر لایه کانولوشن با اندازه 1×1 اضافه می‌کند. افزودن این لایه یک روش است برای افزایش ناخطینگی تابع تصمیم‌گیری بدون اینکه حوزه پذیرش لایه حجم را تحت تأثیر قرار دهد. از یک طرف، این لایه شروع به کنترل خروجی لایه حجمی و ساختار شبکه می‌کند و از طرف دیگر، می‌تواند ویژگی‌های جزئی‌تری را استخراج کند. بنابراین، از آن برای گسترش داده‌ها و استخراج ویژگی‌های تصویر با نقاط کم استفاده می‌شود. به این ترتیب، دقت مدلی که طراحی شده، بهبود می‌یابد. مدلی که مقاله طراحی کرده، سه لایه VGG-16 را به دو لایه تغییر داده است که می‌تواند به طور موثری پارامتر مدل را کاهش دهد، در عین این که دقت شناسایی و طبقه‌بندی مدل را کاهش ندهد و هم‌چنین سرعت آموزش مدل را بهبود بخشد. علاوه بر این‌ها اگر مقصود سوال به مباحث کلی‌تری هم چون پیش‌پردازش برگردد با انجام پیش‌پردازش‌های ذکر شده در بخش ۲.۱ می‌توان تصویر را آماده برای تغذیه به مدل و شبکه کرد.

۴.۱ پاسخ قسمت ۴ - دریافت و تحلیل دادگان

در گام اول، مجموعه داده را روی [گوگل درایو](#) بارگذاری می‌کنیم. این کار به این دلیل انجام می‌شود که دریافت مستقیم از لینک مذکور به دلیل احتمالاً محدودیت‌های سرور مرجع بسیار کند بوده است. هم‌چنین، فراخوانی این فایل بدون نیاز به Mount کردن و استفاده از gdown در محیط گوگل کولب ممکن است با خطا مواجه شود که با استفاده از دستورات زیر مشکل حل می‌شود (منبع). در ادامه هم دستوراتی برای خروج داده‌ها از حالت فشرده و رفتن به یک پوشه مخصوص مجموعه داده نوشته‌ایم و در انتها فایل فشرده را حذف کرده‌ایم.

```
1 !pip install --upgrade --no-cache-dir gdown
2 !gdown 1NQtY1BBVQy0h43hblIF9H6cJigtVdzSm
3
4 import zipfile
5 zip_file_path = '/content/EuroSATallBands.zip'
6 folder_path = '/content/EuroSAT'
7 # Extract the zip file to the specified folder
8 with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
9     zip_ref.extractall(folder_path)
10
11 import os
12 file_path = "/content/EuroSATallBands.zip"
13 if os.path.exists(file_path):
14     os.remove(file_path)
15     print(f"{file_path} has been deleted successfully.")
16 else:
17     print(f"{file_path} does not exist.")
```

ما هم‌چنین از کتابخانه rasterio استفاده می‌کنیم و آن را از طریق دستور زیر نصب می‌کنیم. این کتابخانه برای کار با داده‌های رستری (مانند تصاویر سنجش از راه دور) است. با استفاده از آن می‌توانیم تصاویر رستری را با فرمت‌های مختلفی مانند GeoTIFF و JPEG بخوانیم، ویرایش کنیم و به فرمت‌های دیگری تبدیل کنیم. هم‌چنین می‌توانیم ویژگی‌های تصاویر سنجش از دور، از جمله اندازه، پوشش‌های رنگی و جلوه‌های دیگر را نمایش دهیم و ویرایش کنیم. عملیات جغرافیایی بر روی داده‌های رستری، از جمله برش، ردیابی موقعیت مکانی و تحلیل‌های مکانی هم با استفاده از این کتابخانه ممکن خواهد بود.

```
1 !pip install rasterio
```

برای نمایش ابعاد تصویر از دستورات زیر استفاده می‌کنیم که برای خواندن چند باند از تصویر رستری با استفاده از کتابخانه rasterio و نمایش اندازه داده‌های مربوطه است. در این دستور، ابتدا با استفاده از glob فایل‌های tif در مسیر datapath را پیدا کرده و آدرس فایل‌ها را در متغیر path_to_preview ذخیره می‌کنیم. سپس با استفاده از rasterio.open، فایل اول در مسیر ذخیره شده در متغیر را با حالت خواندن باز می‌کنیم و سپس داده‌های دلخواه را از بین باندهای تصویر رستری با استفاده از شماره باندها (در اینجا ۲، ۳، ۴ و ۸) به متغیر tempdata تخصیص می‌دهیم. در نهایت هم اندازه تصویر را خروجی می‌گیریم. دستورات و نتایج آن به شرح زیر است:

```
1 # Import necessary libraries
2 import os
3 import glob
```

```

4 import rasterio
5
6 # Define the path to the directory containing the raster data
7 datapath = "/content/EuroSAT/ds/images/remote_sensing/otherDatasets/sentinel_2/tif"
8
9 # Find all the TIFF files in the directory and its subdirectories
10 path_to_tiffs = glob.glob(os.path.join(datapath, "**", "*.tif"), recursive=True)
11
12 # Choose the first TIFF file as an example
13 tiff_file = path_to_tiffs[0]
14
15 # Open the TIFF file in read-only mode using Rasterio
16 with rasterio.open(tiff_file, "r") as src:
17
18     # Read the data from the specified bands (2, 3, 4, 8) and store it in a variable
19     tempdata = src.read((2, 3, 4, 8))
20
21     # Print the shape of the data array
22     print("Shape of the data array:", tempdata.shape)
23
24 -----
25
26 Shape of the data array: (4, 64, 64)

```

از آن‌جا که بشخصه تاکنون با تصاویر سنجش از راه دور کار نکرده بودم و صرفاً با تصاویر پزشکی آشنایی داشتم، در ادامه توضیحاتی در خصوص این تصاویر می‌آورم که برای خودم که اولین بار با هم‌چنین مجموعه‌داده‌ای روبه‌رو می‌شدم جالب بوده است. تصاویر رستری، اطلاعات تصویر را در قالب پیکسل‌ها نگه می‌دارند، که هر پیکسل در آن یک مقدار شناور را به عنوان اطلاعات رنگ یا بازتابش نشان می‌دهد. هر تصویر رستری، می‌تواند شامل یک یا چند باند باشد. باند به معنی یکی از مولفه‌های تصویر است که مربوط به یک مقدار شناور در هر پیکسل می‌باشد. در بسیاری از تصاویر رستری، هر باند، نشان دهنده یک مشخصه مختلف مانند رنگ، بازتابش، دما و غیره می‌باشد. به عنوان مثال، در تصویر رنگی، هر باند نشان دهنده رنگ آبی، سبز یا قرمز در هر پیکسل می‌باشد. در تصاویر سنجش از دور هم، باندها می‌توانند نشان دهنده بازتابش در طول طیف الکترومغناطیس باشند که به ما اطلاعاتی درباره سطح زمین، گیاهان، آب، و غیره ارائه می‌دهند. تعداد و نوع باندهای تصویر رستری، وابسته به موارد مختلفی می‌باشد، از جمله نوع تصویر رستری و نیازهای کاربری. در بسیاری از موارد، باندهای خاصی به منظور بررسی و تحلیل خاص تصویر انتخاب می‌شوند. به طور کلی، با افزایش تعداد باندهای تصویر رستری، می‌توان دقت تحلیل را بیشتر کرد و اطلاعات بیشتری را از تصویر استخراج کرد. در مورد مجموعه‌داده EuroSAT و Sentinel-2، باندهای مختلفی وجود دارند که هر کدام اطلاعات مفیدی درباره موارد خاصی از زمین ارائه می‌دهند. مثلاً در این دیتاست، باند ۲ نشان دهنده بازتابش در طول موج نوری آبی، باند ۳ نشان دهنده بازتابش در طول موج نوری سبز، باند ۴ نشان دهنده بازتابش در طول موج نوری قرمز و باند ۸ نشان دهنده بازتابش در طول موج نوری نزدیک مادون قرمز (آلایند در هوا) می‌باشد. با استفاده از این باندها، می‌توان اطلاعات مفیدی را از تصاویر استخراج کرد. باندهای ۲ و ۳ برای تحلیل گیاهان و کشاورزی، باند ۴ برای تحلیل آب و دریا، و باند ۸ برای تحلیل آلودگی هوا و آلاینده‌های صنعتی استفاده می‌شوند. این دیتاست مجموعاً باید ۱۳ باند داشته باشد؛ اما فقط برخی از آن‌ها در نسخه نهایی مجموعه‌داده پوشش داده شده‌اند. لازم به ذکر است که هر یک از این باندها می‌تواند برای

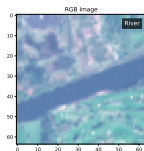
تحلیل خاصی به کار رود. مثلاً باند چهارم می‌تواند برای تحلیل وضعیت استرس و سلامتی گیاهان و باند ششم می‌تواند برای شناسایی نوع پوشش گیاهی، تحلیل زیست‌توده و فعالیت فتوسنتزی مورد استفاده قرار گیرد. هم‌چنین مثلاً باند یازدهم می‌تواند برای بررسی میزان رطوبت خاک استفاده شود که خود این می‌تواند به حضور یک رودخانه در نزدیکی محل عکس‌برداری شده ارتباط داشته باشد. برای پاسخ بهتر به این کنجکاوی، دستورات زیر را نوشته تا تصاویر مربوط به باندهای مختلف به همراه تصویر RGB با عنوان کلاس بر روی آن را رسم کنیم. چند نمونه از نتایج در شکل ۵ آورده شده است.

```
1 import rasterio
2 import glob
3 import os
4 import matplotlib.pyplot as plt
5
6 # Define the path to the directory containing the data
7 datapath = "/content/EuroSAT/ds/images/remote_sensing/otherDatasets/sentinel_2/tif"
8
9 # Find all TIFF files in the directory and subdirectories
10 path_to_preview = glob.glob(os.path.join(datapath, "**", "*.tif"), recursive=True)
11
12 # Read in the specified bands from the first TIFF file
13 with rasterio.open(path_to_preview[8000], "r") as src:
14     # Note that band numbering in rasterio starts at 1, not 0
15     band_nums = [1, 4, 8, 11]
16     tempdata = src.read(band_nums)
17
18 # Plot each band separately and save as a PDF file
19 for i, band in enumerate(tempdata, start=1):
20     # Get the band number from the source file
21     with rasterio.open(path_to_preview[8000], "r") as src:
22         band_num = src.indexes[band_nums[i-1]-1]
23
24     # Plot the band and set the title to the band number
25     plt.imshow(band)
26     plt.title("Band {}".format(band_num))
27
28     # Save the plot as a PDF file with the band number in the filename
29     plt.savefig("band{}.pdf".format(band_num))
30
31     # Show the plot in the notebook
32     plt.show()
33
34 # Display and save the RGB image
35 with rasterio.open(path_to_preview[8000], "r") as src:
36     # Read in the red, green, and blue bands
37     rgb = src.read([4, 3, 2])
38
39     # Normalize the bands to values between 0 and 1
```

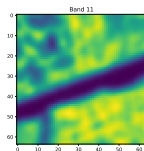
```

40  rgb = rgb / rgb.max()
41
42  # Display the RGB image
43  fig, ax = plt.subplots()
44  im = ax.imshow(rgb.transpose(1,2,0))
45  plt.title("RGB Image")
46
47  # Get the class name from the file path and add it as a legend on the top right corner
48  class_name = os.path.basename(os.path.dirname(path_to_preview[8000]))
49  ax.text(0.95, 0.95, class_name, transform=ax.transAxes, ha='right', va='top', color='white',
50         fontsize=12, bbox=dict(facecolor='black', alpha=0.7, pad=4))
51
52  # Save the RGB image as a PDF file
53  plt.savefig("rgb.pdf")
54
55  # Show the plot in the notebook
56  plt.show()

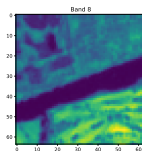
```



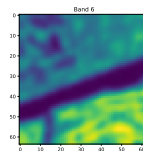
RGB (ا)



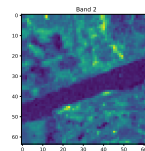
B11 (د)



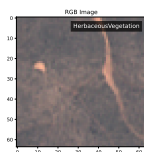
B08 (ج)



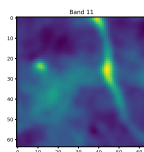
B06 (ب)



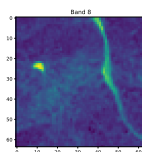
B02 (ا)



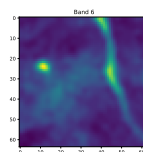
RGB (ی)



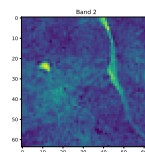
B11 (ط)



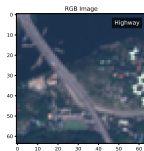
B08 (ح)



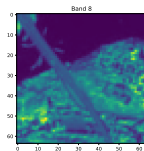
B06 (ز)



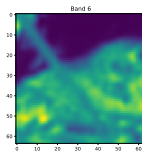
B02 (و)



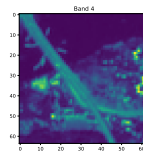
RGB (س)



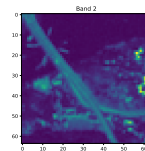
B08 (ن)



B05 (م)



B04 (ل)



B02 (ک)

شکل ۵: نمایش نمونه‌هایی از داده‌های مجموعه داده در باندهای مختلف.

در ادامه دستورات زیر را برای تعیین کلاس‌ها می‌نویسیم:

```

1  import os
2
3  # Get list of class names in alphabetical order

```

```

4 classes = (os.listdir(datapath))
5
6 # Print list of classes
7 print(classes)
8
9 # Check that the list of classes is not empty
10 assert len(classes) > 0
11
12 # Check that each class name is valid (for example, does not contain whitespace)
13 for class_name in classes:
14     assert class_name.strip() == class_name
15
16 -----
17
18 ['HerbaceousVegetation',
19  'Industrial',
20  'Pasture',
21  'River',
22  'AnnualCrop',
23  'Highway',
24  'Residential',
25  'Forest',
26  'SeaLake',
27  'PermanentCrop']

```

۵.۱ پاسخ قسمت ۵ - پیاده‌سازی شبکه و نتایج

۱.۵.۱ پیاده‌سازی اول

همان‌طور که در **بخش ۴.۱** بیان شد، به دلیل این که اولین بار بود که با تصاویر سنجش از راه دور مواجه می‌شدم، توضیحاتی در خصوص این نوع تصاویر و مفهوم باند در این تصاویر آوردم. از دانش بیان‌شده در **بخش ۴.۱** در قسمت پیاده‌سازی شبکه هم استفاده کرده‌ام. در ادامه دستورات زیر برای تعریف و نمایش شبکه و مدل می‌نویسیم.

```

1 import torch
2 import torch.nn as nn
3 from torchsummary import summary
4
5 # Load the pre-trained VGG-16 model
6 model = models.vgg16(pretrained=True)
7
8 # Move the model to the GPU
9 model.cuda()
10
11 # Print the model summary

```

```

12 summary(model, input_size=(3, 64, 64))
13
14
15 import torch
16 import torch.nn as nn
17 from torchsummary import summary
18
19 # Load the pre-trained VGG-16 model
20 model = models.vgg16(pretrained=True)
21
22 # Modify the first convolutional layer to accept 4 channels instead of 3
23 model.features[0] = nn.Conv2d(4, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
24
25 # Modify the last fully connected layer to output 10 classes instead of 1000
26 model.classifier[-1] = nn.Linear(in_features=4096, out_features=10, bias=True)
27
28 # Move the model to the GPU
29 model.cuda()
30
31 # Print the model summary
32 summary(model, input_size=(4, 64, 64))
33
34 -----
35      Layer (type)                Output Shape          Param #
36 =====
37      Conv2d-1                  [-1, 64, 64, 64]         2,368
38      ReLU-2                    [-1, 64, 64, 64]           0
39      Conv2d-3                  [-1, 64, 64, 64]        36,928
40      ReLU-4                    [-1, 64, 64, 64]           0
41      MaxPool2d-5               [-1, 64, 32, 32]           0
42      Conv2d-6                  [-1, 128, 32, 32]       73,856
43      ReLU-7                    [-1, 128, 32, 32]          0
44      Conv2d-8                  [-1, 128, 32, 32]      147,584
45      ReLU-9                    [-1, 128, 32, 32]          0
46      MaxPool2d-10              [-1, 128, 16, 16]          0
47      Conv2d-11                 [-1, 256, 16, 16]      295,168
48      ReLU-12                   [-1, 256, 16, 16]          0
49      Conv2d-13                 [-1, 256, 16, 16]      590,080
50      ReLU-14                   [-1, 256, 16, 16]          0
51      Conv2d-15                 [-1, 256, 16, 16]      590,080
52      ReLU-16                   [-1, 256, 16, 16]          0
53      MaxPool2d-17              [-1, 256, 8, 8]           0
54      Conv2d-18                 [-1, 512, 8, 8]      1,180,160
55      ReLU-19                   [-1, 512, 8, 8]           0
56      Conv2d-20                 [-1, 512, 8, 8]      2,359,808

```


57	ReLU-21	[-1, 512, 8, 8]	0
58	Conv2d-22	[-1, 512, 8, 8]	2,359,808
59	ReLU-23	[-1, 512, 8, 8]	0
60	MaxPool2d-24	[-1, 512, 4, 4]	0
61	Conv2d-25	[-1, 512, 4, 4]	2,359,808
62	ReLU-26	[-1, 512, 4, 4]	0
63	Conv2d-27	[-1, 512, 4, 4]	2,359,808
64	ReLU-28	[-1, 512, 4, 4]	0
65	Conv2d-29	[-1, 512, 4, 4]	2,359,808
66	ReLU-30	[-1, 512, 4, 4]	0
67	MaxPool2d-31	[-1, 512, 2, 2]	0
68	AdaptiveAvgPool2d-32	[-1, 512, 7, 7]	0
69	Linear-33	[-1, 4096]	102,764,544
70	ReLU-34	[-1, 4096]	0
71	Dropout-35	[-1, 4096]	0
72	Linear-36	[-1, 4096]	16,781,312
73	ReLU-37	[-1, 4096]	0
74	Dropout-38	[-1, 4096]	0
75	Linear-39	[-1, 10]	40,970
76	=====		
77	Total params: 134,302,090		
78	Trainable params: 134,302,090		
79	Non-trainable params: 0		
80	-----		
81	Input size (MB): 0.06		
82	Forward/backward pass size (MB): 18.21		
83	Params size (MB): 512.32		
84	Estimated Total Size (MB): 530.59		
85	-----		

در این دستورات، معماری مدل پیش‌آموزش‌دیده VGG-16 برای استفاده در یک مسأله دسته‌بندی تصویر ۱۰ کلاس تغییر داده شده است. در این کد ابتدا مدل VGG-16 در حالت پیش‌آموزش‌دیده روی تصاویر ImageNet بارگیری شده است. سپس با استفاده از دستور `nn.Conv2d`، لایه کانولوشنی اول مدل با نام `model.features[0]` به گونه‌ای تغییر داده شده است که این لایه قابلیت پردازش ۴ کانال (۴ باند رنگی در تصاویر) را دارا باشد. در مرحله بعد، با استفاده از دستور `nn.Linear`، لایه تماماًمتصل آخر مدل با نام `model.classifier[1]` تغییر داده شده و به جای ۱۰۰۰ کلاس، اکنون تنها ۱۰ کلاس خروجی دارد. در ادامه، با استفاده از دستور `model.cuda`، مدل به GPU منتقل شده است تا بتواند بر روی تصاویر به صورت کارآمدتری پردازش شود. در نهایت با استفاده از تابع `summary`، جزئیات مدل و اطلاعات مربوط به لایه‌ها و تعداد پارامترهای آن‌ها چاپ می‌شود. در اینجا، ورودی به عنوان سائز تصویر، با ۴ باند و اندازه 64×64 تعیین شده است.

در ادامه، تابعی برای جمع‌بندی تمام تصاویر و برچسب کلاس مربوط به آن‌ها می‌نویسیم. این تابع که `get_list_of_samples` نام دارد، مسیری که شامل زیردایرکتوری‌های هر کلاس از تصاویر است را به عنوان ورودی می‌گیرد. این تابع همه تصاویر و برچسب کلاس مربوط به آن‌ها را در یک لیست از دیکشنری‌ها جمع‌آوری می‌کند. این تابع تعداد تصاویر مربوط به هر کلاس را به ۲۰۰۰ تصویر محدود می‌کند تا داده‌ها برای یادگیری مدل متوازن باشند. در غیر این صورت، راهکار برای این موضوع می‌تواند در تابع اتلاف تعبیه شود. در نهایت، برای هر تصویر، یک دیکشنری با دو ورودی `X` و `Y` ساخته می‌شود که به ترتیب، مسیر فایل

تصویر و برچسب کلاس آن تصویر را نشان می‌دهد. لیستی از این دیکشنری‌ها به عنوان خروجی تابع بازگردانده می‌شود.

```
1 def get_list_of_samples(datapath):
2
3     sample_toupels = [] # a list to hold sample-tuples
4
5     # Loop over each class in the dataset
6     for ijk, _class in enumerate(classes):
7
8         # Get a list of all sample files for the current class
9         all_samples_in_class = glob.glob(os.path.join(datapath, _class, "*.tif"))
10
11        # Limit the number of samples to 2000 per class to create a balanced dataset
12        all_samples_in_class = all_samples_in_class[:2000]
13
14        # Create a dictionary for each sample that includes the file path and class label
15        for sample in all_samples_in_class:
16            sample_toupels.append({"X":sample, "Y":ijk})
17
18        # Return the list of dictionaries containing the image file paths and class labels
19    return sample_toupels
```

در ادامه و بر مبنای تابع `get_list_of_samples`، دستوراتی را می‌نویسیم که داده‌ها را در سه دسته آموزش، اعتبارسنجی و آزمون تقسیم کند. این دستورات پس از دریافت لیست نمونه‌ها از طریق تابع تعریف شده، آن‌ها را به صورت تصادفی مخلوط می‌کند. سپس نسبت‌های مجموعه‌های آموزش، اعتبارسنجی و آزمون تعریف می‌شود. تعداد نمونه‌ها برای هر مجموعه را بر اساس نسبت‌ها محاسبه می‌گردد. در نهایت نمودار توزیعی داده‌ها برای هر سه مجموعه به صورتی که در شکل ۶ نشان داده شده رسم می‌گردد.

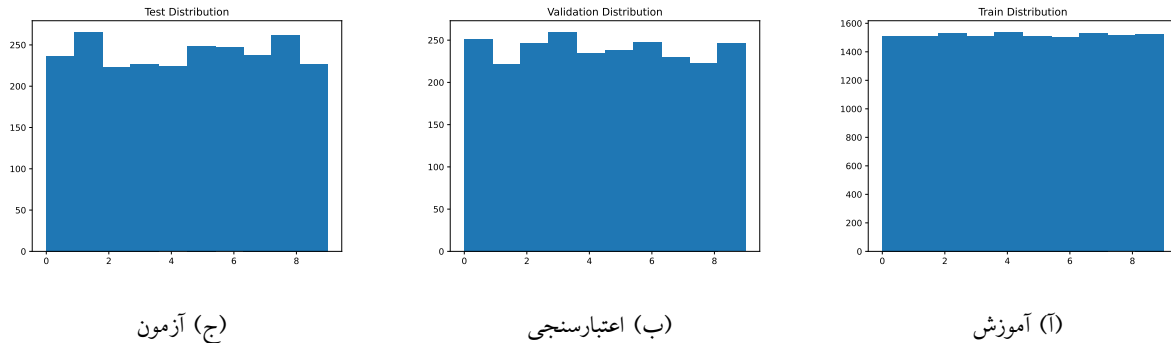
```
1 import random as RS
2 import matplotlib.pyplot as plt
3
4 # Define a function to get a list of samples from a given datapath
5 # def get_list_of_samples(datapath):
6 #     # Code to get a list of samples from the datapath
7 #     pass
8
9 # Get a list of all samples from the datapath
10 all_samples = get_list_of_samples(datapath)
11
12 # Shuffle the samples randomly
13 RS.shuffle(all_samples)
14
15 # Define the ratios for train, validation, and test sets
16 train_ratio = 0.76
17 val_ratio = 0.12
18 test_ratio = 0.12
```

```

19
20 # Calculate the number of samples for each set based on the ratios
21 num_samples = len(all_samples)
22 num_train = int(num_samples * train_ratio)
23 num_val = int(num_samples * val_ratio)
24 num_test = num_samples - num_train - num_val
25
26 # Split the shuffled samples into train, validation, and test sets
27 train_locations = all_samples[:num_train]
28 val_locations = all_samples[num_train:num_train+num_val]
29 test_locations = all_samples[num_train+num_val:]
30
31 # Print the number of samples in each set
32 print("# Train Images", len(train_locations))
33 print("# Val Images", len(val_locations))
34 print("# Test Images", len(test_locations))
35
36 # Plot the distribution of the Y values for the train set and save as a PDF
37 temp_hist = []
38 for d in train_locations:
39     temp_hist.append(d["Y"])
40 plt.hist(temp_hist)
41 plt.title("Train Distribution")
42 plt.savefig("traindistribution.pdf")
43 plt.show()
44
45 # Plot the distribution of the Y values for the validation set and save as a PDF
46 temp_hist = []
47 for d in val_locations:
48     temp_hist.append(d["Y"])
49 plt.hist(temp_hist)
50 plt.title("Validation Distribution")
51 plt.savefig("validationdistribution.pdf")
52 plt.show()
53
54 # Plot the distribution of the Y values for the test set and save as a PDF
55 temp_hist = []
56 for d in test_locations:
57     temp_hist.append(d["Y"])
58 plt.hist(temp_hist)
59 plt.title("Test Distribution")
60 plt.savefig("testdistribution.pdf")
61 plt.show()

```

سپس، کلاس و دستورات زیر را تعریف می‌کنیم تا داده‌ها فراخوانی، باندهایشان مشخص، و نرمال شوند. در انتها هم با استفاده از این کلاس و دیتالودر در پایتورچ، داده‌ها برای به‌کارگیری در قالب سه مجموعه آموزش، اعتبارسنجی و آزمون آماده



شکل ۶: نمودار توزیعی داده‌ها.

می‌شوند. به عنوان توضیحی از برخی متغیرهای داخل دیتالودر؛ از `num_workers` برای تعیین تعداد پردازش‌های موازی که برای بارگذاری داده‌ها به کار گرفته می‌شود استفاده می‌شود. به این صورت که، زمانی که داده‌ها برای آموزش شبکه باید از دیسک بارگذاری شوند، تعداد ورکرهایی که برای بارگذاری داده‌ها باید به کار گرفته می‌شود با توجه به مواردی قابل تعیین است: تعداد ورکرها باید کمتر از تعداد کل هسته‌های پردازنده سیستم باشد تا برنامه به صورت صحیح و با عملکرد بهینه اجرا شود. تعداد ورکرها باید به اندازه کافی بزرگ باشد تا بتوانند با توجه به مواردی مانند حجم داده و سرعت بارگذاری آن‌ها، داده‌ها را به صورت موازی بارگذاری و تحویل دهند. بنابراین، با افزایش این پارامتر و متغیر، تعداد ورکرها بیشتر و در نتیجه توانایی بارگذاری موازی داده‌ها به صورت بیشتری افزایش می‌یابد. البته این مقدار نباید بیش از تعداد هسته‌های پردازنده و میزان منابع دیگر مانند حافظه‌ی سیستمی را در نظر گرفته شود. متغیر `drop_last` هم اگر `True` باشد، آخرین دسته از داده‌ها که شاید کوچکتر از اندازه دسته باشد را حذف خواهد شد، و اگر `False` باشد، دسته آخر با داده‌های باقی‌مانده تکمیل می‌شود.

```

1 import rasterio
2 import numpy as np
3 import torch
4 from torch.utils import data
5 from torch.utils.data import DataLoader
6
7 # Define a dataset class for the Super Resolution (SR) model
8 class class_dataset(data.Dataset):
9
10     # Constructor to initialize the samplelist for the dataset
11     def __init__(self, samplelist):
12         self.samplelist = samplelist
13
14     # Method to get the data and label tensors for a given index
15     def __getitem__(self, i):
16
17         # Open the image file using rasterio
18         with rasterio.open(self.samplelist[i]["X"], "r") as src:
19
20             # Read the data from bands 2, 3, 4, and 8 as a float32 array
21             data = src.read((2,3,4,8)).astype("float32")

```

```

22
23         # Normalize the data to [0,1] range
24         data = data / 10000
25
26         # Clip the data to [0,1] range
27         data = np.clip(data, 0, 1)
28
29         # Return the data tensor and the label tensor as a tuple
30         return torch.Tensor(data), torch.Tensor([self.samplelist[i]["Y"]]).long()
31
32     # Method to get the length of the dataset
33     def __len__(self):
34         return len(self.samplelist)
35
36 # Create datasets and dataloaders for train, validation, and test sets
37 train_set = class_dataset(train_locations)
38 val_set = class_dataset(val_locations)
39 test_set = class_dataset(test_locations)
40
41 # Setup the dataloader with the given number of worker threads, batch size, and shuffle option
42 training_data_loader = DataLoader(dataset=train_set, num_workers=threads, batch_size=batch_size,
43                                   shuffle=True, drop_last=True)
44 val_data_loader = DataLoader(dataset=val_set, num_workers=threads, batch_size=batch_size, shuffle
45                              =True, drop_last=True)
46 test_data_loader = DataLoader(dataset=test_set, num_workers=threads, batch_size=batch_size,
47                               shuffle=True, drop_last=True)

```

دستورات زیر را هم برای نمایش یک دسته از داده‌ها می‌نویسیم و نتیجه به صورتی است که در شکل ۷ نشان داده شده است:

```

1 # Loop over the first batch of images and their corresponding labels
2 for img, label in val_data_loader:
3     # Only process the first batch
4     break
5
6 # Print the shape of the first image and its label
7 print(img.shape, label.shape)
8
9 # Create a 4x4 grid for the images with margins of 0.2 on all sides
10 fig, axs = plt.subplots(4, 4, figsize=(10,10))
11 fig.subplots_adjust(hspace=0.4, wspace=0.4, top=0.95, bottom=0.05, left=0.05, right=0.95)
12
13 # Loop over each image in the batch
14 for batchindex in range(batch_size):
15
16     # Extract the B, G, R color channels from the image
17     B,G,R = img[batchindex,:3,:,:]

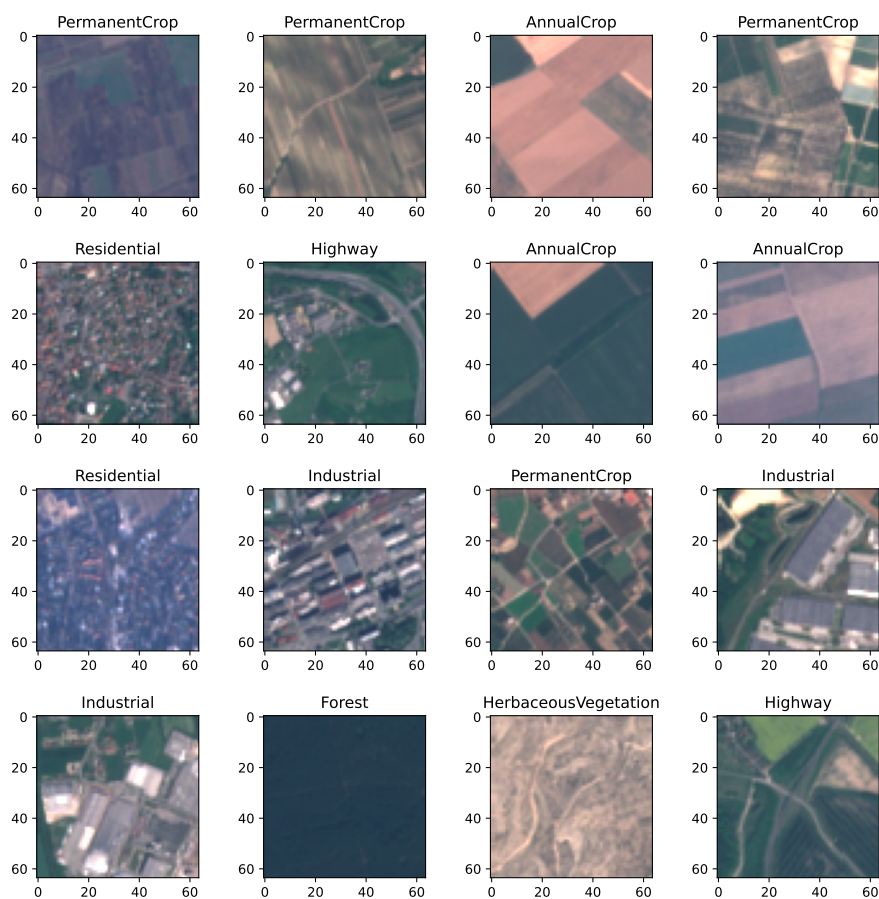
```

```

18 # Stack the channels to form a colored image
19 plot_img = np.stack([R,G,B],axis=-1)
20 # Increase the brightness of the image by a factor of 4
21 plot_img = np.clip(plot_img*4,0,1)
22
23 # Display the image along with its corresponding label in the grid
24 row = batchindex // 4
25 col = batchindex % 4
26 axs[row, col].imshow(plot_img)
27 axs[row, col].set_title(classes[label[batchindex].numpy()[0])
28
29 # Set the title of the plot
30 fig.suptitle('First batch of validation images')
31 # Hide the x and y axis labels for all subplots
32 for ax in axs.flat:
33     ax.set(xlabel='', ylabel='')
34 # Save the plot
35 plt.savefig('batchimages.pdf')
36 # Show the plot
37 plt.show()

```

در ادامه، دستوراتی را برای آموزش و ارزیابی شبکه با توجه به ساختار داده‌ای تعریف شده می‌نویسیم (برنامه ۲). این دستورات شامل مراحل آموزش شبکه، ارزیابی دقت شبکه بر روی داده‌های آموزش، ارزیابی دقت شبکه بر روی داده‌های اعتبارسنجی، و ارزیابی دقت شبکه بر روی داده‌های تست هستند. در این دستوران ابتدا ایجاد چهار لیست خالی برای ذخیره‌سازی مقادیر خطا و دقت شبکه در داده‌های آموزش، اعتبارسنجی و تست ایجاد می‌گردد. هم‌چنین دو لیست خالی برای ذخیره‌سازی برچسب‌های پیش‌بینی شده و برچسب‌های واقعی برای داده‌های تست هم ایجاد می‌شود. سپس با تعریف تعداد دوره‌های آموزش، حلقه آموزش تعریف می‌گردد. در واقع در این بخش با یک حلقه تکرار، مدل به مدت تعداد مشخصی از دوره‌ها آموزش داده می‌شود. در هر دوره، ابتدا مدل به حالت آموزش قرار می‌گیرد و سپس با استفاده از داده‌های آموزشی، پارامترهای مدل بهینه‌سازی می‌شوند. پس از پایان حلقه آموزش، مدل به حالت ارزیابی تغییر حالت می‌دهد و با استفاده از داده‌های اعتبارسنجی، عملکرد مدل ارزیابی می‌شود. دستور `torch.no_grad with` برای غیرفعال کردن محاسبات گرادیان در زمان ارزیابی و آزمایش استفاده می‌شود. چاپ و نمایش عملکرد مدل شامل هزینه و دقت برای دو دسته آموزش و ارزیابی در هر چرخه آموزش انجام می‌شود. در ادامه دستوراتی برای ارزیابی دقت و خطای شبکه در مجموعه داده‌ی آزمون نوشته شده است. این دستورات شباهت زیادی به دستورات ارزیابی برای مجموعه داده‌های آموزش و اعتبارسنجی دارند با این تفاوت که به جای مجموعه داده‌ی آموزش و اعتبارسنجی، از مجموعه داده‌ی آزمون استفاده شده است. با استفاده از دستور `model.eval`، مدل به حالت ارزیابی درآمده و شبکه در هر مرحله دیگر دستکاری نمی‌شود. سپس در هر مرحله از یک حلقه، داده‌ها از مجموعه داده‌ی تست به صورت دسته‌ای برای ارزیابی ورودی شبکه قرار می‌گیرند. سپس برای هر دسته، خروجی‌های شبکه با استفاده از دستور `model(inputs) = outputs` حساب می‌شود. در پایان، میانگین خطا و دقت مجموعه داده‌ی تست محاسبه می‌گردد. هم‌چنین در این دستورات برای نمایش بهتر نتایج، با استفاده از کتابخانه `matplotlib`، نمودارهای دقت و اتلاف را برای داده‌های آموزش، اعتبارسنجی و آزمون رسم می‌کنیم. برای این کار، ابتدا میانگین خطا و اتلاف برای هر مجموعه در هر دوره را محاسبه کرده و سپس نمودارهای اتلاف بر حسب دوره برای هریک به دست می‌آوریم. در ادامه، از شاخص‌های دیگری که برای ارزیابی یک مدل استفاده می‌شوند، مانند `pre-accuracy`، `recall`، `cision` و `f1-score` استفاده می‌کنیم. برای محاسبه‌ی این متریک‌ها، از کتابخانه `learnscikit` استفاده شده است.



شکل ۷: تصاویر مربوط به یک دسته.

در ادامه، با استفاده از کتابخانه `seaborn`، ماتریس درهم‌ریختگی را برای داده‌های تست بر حسب دسته‌بندی‌های پیش‌بینی شده و واقعی به دست آوردیم. این ماتریس در کنار نمودارهای قبلی برای ارزیابی بهتر مدل به کار گرفته می‌شود. در انتها، مدل آموزش داده‌شده را ذخیره کردیم تا بتوان در آینده از آن برای پیش‌بینی‌های جدید استفاده کرد.

Program 2: Main training Code

```

1 # Create empty lists for storing loss and accuracy for train, val, and test
2 train_loss = []
3 train_acc = []
4 val_loss = []
5 val_acc = []
6 test_loss = []
7 test_acc = []
8
9 # Create a list to store predicted and true labels for test set
10 pred_labels = []
11 true_labels = []
12

```

```

13 num_epochs = 60
14
15 # Training loop
16 for epoch in range(num_epochs):
17     # Train the model
18     model.train()
19     running_loss = 0.0
20     running_corrects = 0
21     total = 0
22     for i, (inputs, labels) in enumerate(training_data_loader):
23         inputs = inputs.cuda()
24         labels = labels.squeeze().long().cuda()
25         optimizer.zero_grad()
26         outputs = model(inputs)
27         loss = criterion(outputs, labels)
28         loss.backward()
29         optimizer.step()
30         running_loss += loss.item() * inputs.size(0)
31         _, preds = torch.max(outputs, 1)
32         running_corrects += torch.sum(preds == labels.data)
33         total += labels.size(0)
34     train_loss.append(running_loss / len(train_set))
35     train_acc.append(running_corrects.double() / total)
36
37 # Evaluate on the validation set
38 model.eval()
39 running_loss = 0.0
40 running_corrects = 0
41 total = 0
42 with torch.no_grad():
43     for i, (inputs, labels) in enumerate(val_data_loader):
44         inputs = inputs.cuda()
45         labels = labels.squeeze().long().cuda()
46         outputs = model(inputs)
47         loss = criterion(outputs, labels)
48         running_loss += loss.item() * inputs.size(0)
49         _, preds = torch.max(outputs, 1)
50         running_corrects += torch.sum(preds == labels.data)
51         total += labels.size(0)
52     val_loss.append(running_loss / len(val_set))
53     val_acc.append(running_corrects.double() / total)
54
55 # Evaluate on the test set
56 model.eval()
57 running_loss = 0.0

```



```

58     running_corrects = 0
59     total = 0
60     with torch.no_grad():
61         for i, (inputs, labels) in enumerate(test_data_loader):
62             inputs = inputs.cuda()
63             labels = labels.squeeze().long().cuda()
64             outputs = model(inputs)
65             loss = criterion(outputs, labels)
66             running_loss += loss.item() * inputs.size(0)
67             _, preds = torch.max(outputs, 1)
68             running_corrects += torch.sum(preds == labels.data)
69             total += labels.size(0)
70             pred_labels.extend(preds.cpu().numpy())
71             true_labels.extend(labels.cpu().numpy())
72     test_loss.append(running_loss / len(test_set))
73     test_acc.append(running_corrects.double() / total)
74
75     # Print epoch, loss, and accuracy for train and val sets
76     print('Epoch [{}/{}], Train Loss: {:.4f}, Train Acc: {:.4f}, Val Loss: {:.4f}, Val Acc: {:.4f}'
77           .format(epoch+1, num_epochs, train_loss[-1], train_acc[-1], val_loss[-1], val_acc[-1]))
78
79     # Plot the loss and accuracy curves for train, val, and test sets
80     plt.figure(figsize=(10, 5))
81     plt.plot(train_loss, label='train')
82     plt.plot(val_loss, label='val')
83     plt.plot(test_loss, label='test')
84     plt.title('Loss vs Epochs')
85     plt.xlabel('Epochs')
86     plt.ylabel('Loss')
87     plt.legend()
88     plt.savefig('loss.pdf')
89     plt.show()
90
91     train_acc = torch.tensor(train_acc)
92     val_acc = torch.tensor(val_acc)
93     test_acc = torch.tensor(test_acc)
94
95     # Plot the loss and accuracy curves for train, val, and test sets
96     plt.figure(figsize=(10, 5))
97     plt.plot(train_acc, label='train')
98     plt.plot(val_acc, label='val')
99     plt.plot(test_acc, label='test')
100    plt.title('Accuracy vs Epochs')
101    plt.xlabel('Epochs')

```

```

102 plt.ylabel('Accuracy')
103 plt.legend()
104 plt.savefig('Accuracy.pdf')
105 plt.show()
106
107 from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
    confusion_matrix
108
109 accuracy = accuracy_score(true_labels, pred_labels)
110 error = 1 - accuracy
111 precision = precision_score(true_labels, pred_labels, average='weighted')
112 recall = recall_score(true_labels, pred_labels, average='weighted')
113 f1score = f1_score(true_labels, pred_labels, average='weighted')
114
115 print('Test Accuracy: {:.4f}'.format(accuracy))
116 print('Test Error: {:.4f}'.format(error))
117 print('Test Precision: {:.4f}'.format(precision))
118 print('Test Recall: {:.4f}'.format(recall))
119 print('Test F1-score: {:.4f}'.format(f1score))
120
121 import seaborn as sns
122 confusion_matrix = np.zeros((10, 10))
123 for i in range(len(true_labels)):
124     true_idx = true_labels[i]
125     pred_idx = pred_labels[i]
126     confusion_matrix[true_idx][pred_idx] += 1
127
128 plt.figure(figsize=(8, 6))
129 sns.heatmap(confusion_matrix, annot=True, cmap='Blues', fmt='g', xticklabels=['
    HerbaceousVegetation', 'Industrial', 'Pasture', 'River', 'AnnualCrop', 'Highway', '
    Residential', 'Forest', 'SeaLake', 'PermanentCrop'])
130 plt.xlabel('Predicted')
131 plt.ylabel('True')
132 plt.title('Confusion Matrix')
133 plt.savefig('confusion_matrix.pdf')
134 plt.show()
135
136 # Save the trained model
137 torch.save(model.state_dict(), 'model.pth')

```

با ارائه این توضیحات، نتیجه نمودارهای اتلاف و دقت به صورتی که به ترتیب در شکل ۸، شکل ۹ و؟؟ نشان داده شده خواهد بود. باید توجه شود که ماتریس درهم‌ریختگی فقط برای داده‌های آزمون در ادامه و در شکل ۱۱ رسم شده است. نتایج سایر شاخصه‌ها هم به شرح زیر است:

```

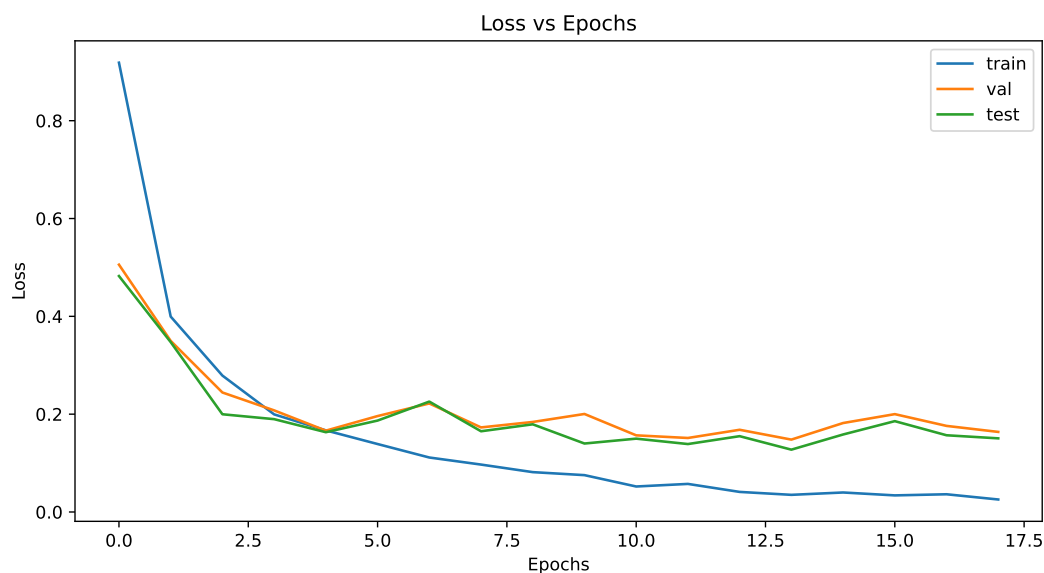
1 Test Accuracy: 0.9395
2 Test Error: 0.0605

```

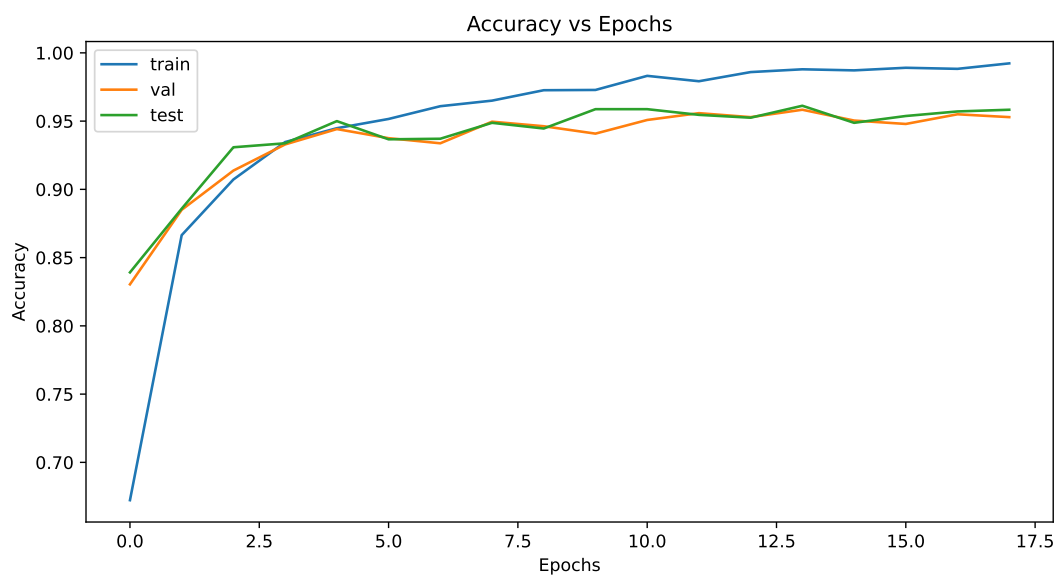
3 Test Precision: 0.9399

4 Test Recall: 0.9395

5 Test F1-score: 0.9396



شکل ۸: نمودار تابع اتلاف.



شکل ۹: نمودار دقت.

در پایان دستورات موجود در برنامه ۳ را نوشته‌ایم تا مدل آموزش دیده را روی مجموعه‌های تصادفی از داده‌های آزمون ارزیابی

و اختلاف میان برچسب (کلاس) واقعی و کلاس پیش‌بینی شده را نشان دهد. در این دستورات مدل آموزش‌دیده و ذخیره‌شده در model با استفاده از eval و torch.no_grad، برای جلوگیری از محاسبه‌ی مشتق در مرحله‌ی پیش‌بینی، روی دسته‌ای تصادفی از تصاویر مجموعه‌ی آزمون اعمال شده است. برای پیش‌بینی، از تابع torch.max استفاده شده که برای هر تصویر، برچسبی با بیشترین احتمال پیش‌بینی شده یافته و در predicted_label ذخیره می‌کند. نتیجه برای یک مجموعه تصادفی از داده‌های آزمون به‌صورتی است که در شکل ۱۰ نشان داده شده است.

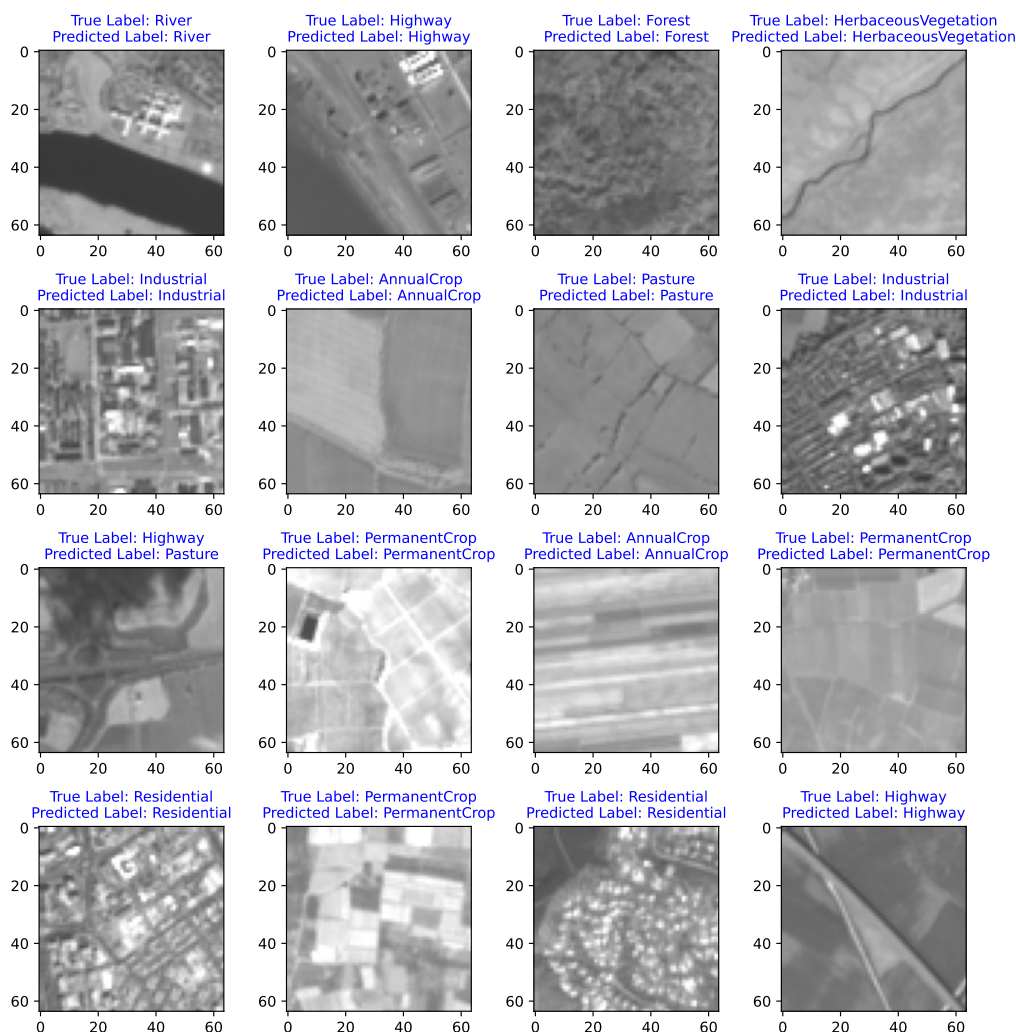
Program 3: Test Code

```

1 # Choose 16 random samples from the test set
2 samples = np.random.choice(len(test_set), size=16, replace=False)
3
4 # Create a 4x4 grid of subplots to display the images
5 fig, axes = plt.subplots(nrows=4, ncols=4, figsize=(12, 12))
6
7 # Evaluate the model on the chosen samples and display the images with true and predicted labels
8 for i, ax in enumerate(axes.flat):
9     image, true_label = test_set[samples[i]]
10    model.eval()
11    with torch.no_grad():
12        output = model(image.unsqueeze(0).cuda())
13    _, predicted_label = torch.max(output.data, 1)
14
15    # Convert the image to grayscale
16    gray_img = torch.mean(image, dim=0, keepdim=True)
17    plot_img = gray_img.repeat(3, 1, 1)
18
19    # Increase the brightness of the image by a factor of 4
20    plot_img = np.clip(plot_img.cpu().numpy() * 4, 0, 1)
21
22    ax.imshow(plot_img.transpose(1, 2, 0))
23
24    # Set title with True and Predicted labels in different colors and smaller font size
25    true_label_text = 'True Label: {}'.format(classes[true_label])
26    predicted_label_text = 'Predicted Label: {}'.format(classes[predicted_label])
27    ax.set_title('{}\n{}'.format(true_label_text, predicted_label_text), fontsize=10, color='blue')
28
29    ax.axis('on')
30
31 plt.subplots_adjust(wspace=0.1, hspace=0.4)
32 plt.savefig('Test2.pdf')
33 plt.show()

```

از آن‌جا که ساختار دستورات به گونه‌ای نوشته شده که به راحتی می‌توان فرآیند آموزش را ادامه داد، آموزش را برای ۱۰ دوره دیگر هم ادامه می‌دهیم برای نمایش ماتریس درهم‌ریختگی فقط روی داده‌های آزمون از دستور زیر استفاده کرده و نتایج به صورتی



شکل ۱۰: نتیجه ارزیابی مدل آموزش دیده روی یک دسته تصادفی از تصاویر آزمون.

خواهد بود که در شکل ۱۱ آمده است. همان طور که مشخص است، پیش‌بینی برای کلاس‌هایی مانند جنگل، دریاچه و رودخانه که ویژگی‌های بارزتری دارند بهتر بوده است.

```
1 from sklearn.metrics import confusion_matrix
2 import seaborn as sns
3 import matplotlib.pyplot as plt
4
5 # Set the model to evaluation mode
6 model.eval()
```

```

7
8 # Create empty lists to store predicted and true labels
9 pred_labels = []
10 true_labels = []
11
12 # Loop through the test set and get predictions and true labels
13 with torch.no_grad():
14     for i, (inputs, labels) in enumerate(test_data_loader):
15         inputs = inputs.cuda()
16         labels = labels.squeeze().long().cuda()
17         outputs = model(inputs)
18         _, preds = torch.max(outputs, 1)
19         pred_labels.extend(preds.cpu().numpy())
20         true_labels.extend(labels.cpu().numpy())
21
22 from matplotlib.backends.backend_pdf import PdfPages
23
24 # Compute the confusion matrix
25 confusion_matrix = confusion_matrix(true_labels, pred_labels)
26
27 # Plot the confusion matrix
28 plt.figure(figsize=(8, 6))
29 sns.heatmap(confusion_matrix, annot=True, cmap='Blues', fmt='g', xticklabels=['
    HerbaceousVegetation', 'Industrial', 'Pasture', 'River', 'AnnualCrop', 'Highway', '
    Residential', 'Forest', 'SeaLake', 'PermanentCrop'], yticklabels=['HerbaceousVegetation', '
    Industrial', 'Pasture', 'River', 'AnnualCrop', 'Highway', 'Residential', 'Forest', 'SeaLake',
    'PermanentCrop'])
30 plt.xlabel('Predicted')
31 plt.ylabel('True')
32 plt.title('Confusion Matrix')
33 plt.tight_layout()
34
35 # Save the figure to a PDF file
36 with PdfPages('confusion_matrix.pdf') as pdf:
37     pdf.savefig(bbox_inches='tight')
38     plt.show()

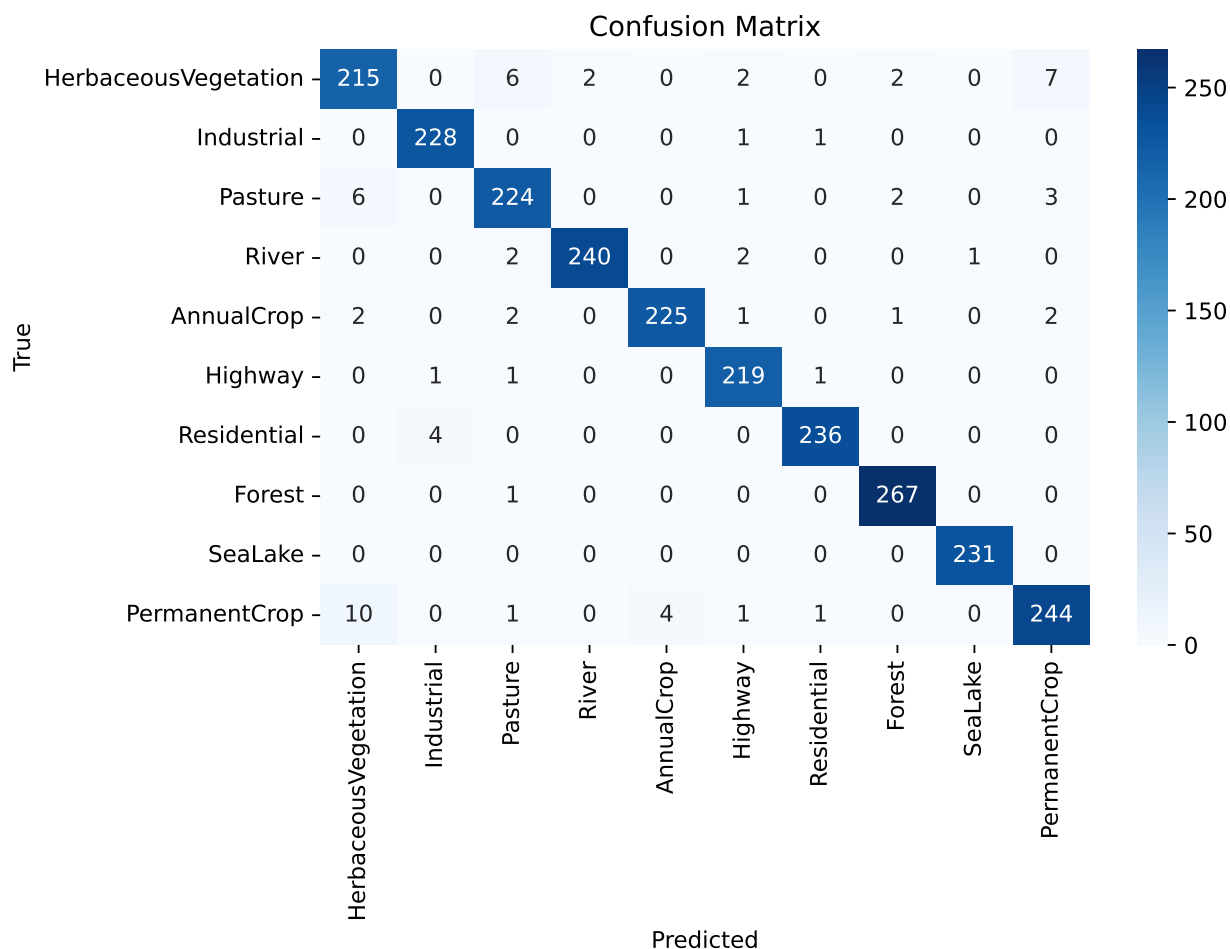
```

```

1 Test Accuracy: 0.9637
2 Test Error: 0.0363
3 Test Precision: 0.9638
4 Test Recall: 0.9637
5 Test F1-score: 0.9637

```

حال یک بار دیگر نتیجه ارزیابی روی ۱۶ داده تصادفی دیگر از مجموعه آزمون را در شکل ۱۲ نشان می‌دهیم. می‌توانیم هم‌چنان و برای دوره‌های بیش‌تری به آموزش ادامه دهیم؛ اما به نظر می‌رسد همین مقدار کافی است.

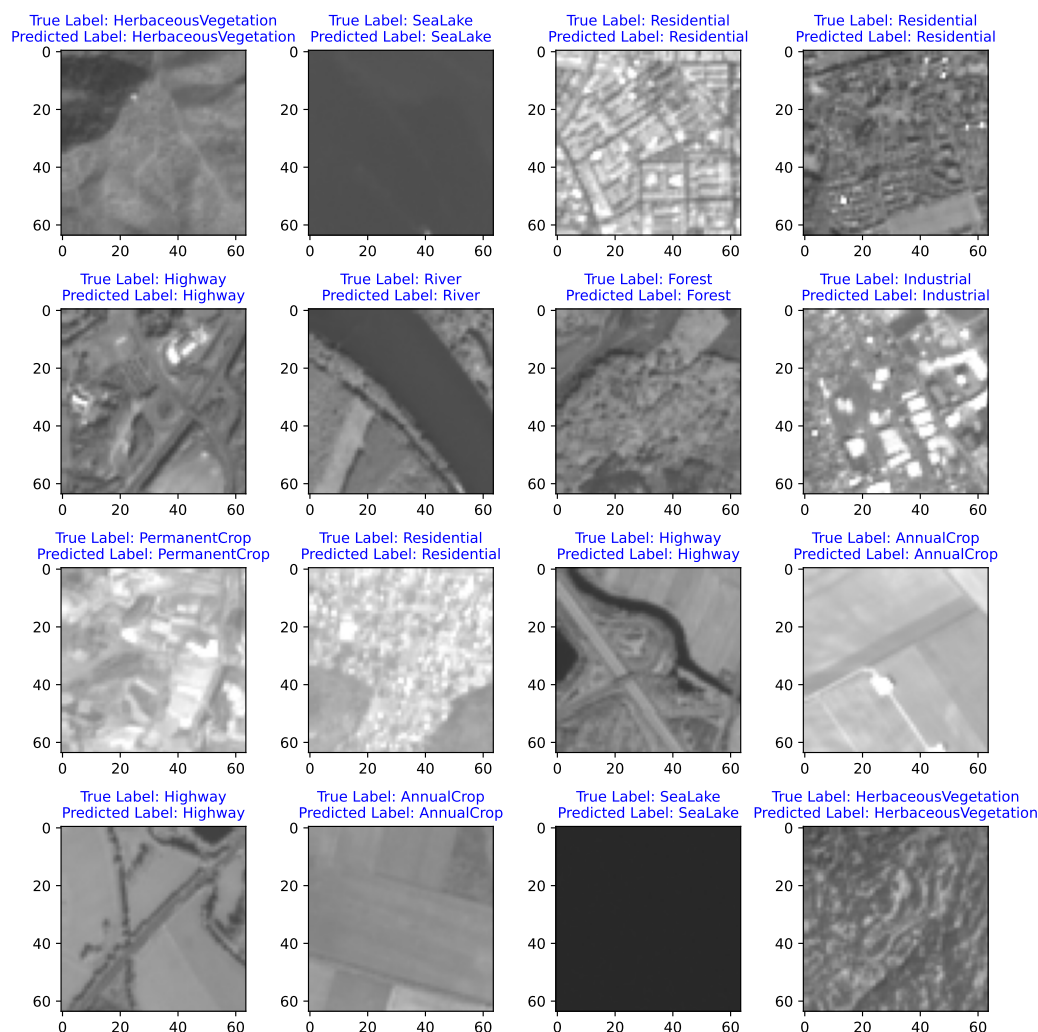


شکل ۱۱: نمودار اصلی ماتریس درهم‌ریختگی.

۲.۵.۱ پیاده‌سازی دوم

با انجام موفق این پیاده‌سازی، یک پیاده‌سازی دیگر با تمرکز بر تصویر RGB صرف (باند‌های ۴ و ۲ و ۱) و پارامترهای دقیقاً منطبق بر مقاله انجام (حتی اندازه‌دسته) می‌دهیم. بنابراین این بار مدل به صورت زیر تعریف می‌شود:

```
1 import torch
2 import torch.nn as nn
3 import torchvision.models as models
4 from torchsummary import summary
5
6 # Load the pre-trained VGG-16 model
7 model = models.vgg16(pretrained=True)
8
9 # Modify the last fully connected layer to output 10 classes instead of 1000
10 model.classifier[-1] = nn.Linear(in_features=4096, out_features=512, bias=True)
11 model.classifier.add_module("dropout1", nn.Dropout(p=0.5))
```



شکل ۱۲: نتیجه ارزیابی مدل آموزش دیده روی یک دسته تصادفی از تصاویر آزمون.

```

12 model.classifier.add_module("fc2", nn.Linear(in_features=512, out_features=10, bias=True))
13 model.classifier.add_module("dropout2", nn.Dropout(p=0.5))
14
15 # Move the model to the GPU
16 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
17 model.to(device)
18
19 # Print the model summary
20 summary(model, input_size=(3, 64, 64))

```


Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 64, 64]	1,792
ReLU-2	[-1, 64, 64, 64]	0
Conv2d-3	[-1, 64, 64, 64]	36,928
ReLU-4	[-1, 64, 64, 64]	0
MaxPool2d-5	[-1, 64, 32, 32]	0
Conv2d-6	[-1, 128, 32, 32]	73,856
ReLU-7	[-1, 128, 32, 32]	0
Conv2d-8	[-1, 128, 32, 32]	147,584
ReLU-9	[-1, 128, 32, 32]	0
MaxPool2d-10	[-1, 128, 16, 16]	0
Conv2d-11	[-1, 256, 16, 16]	295,168
ReLU-12	[-1, 256, 16, 16]	0
Conv2d-13	[-1, 256, 16, 16]	590,080
ReLU-14	[-1, 256, 16, 16]	0
Conv2d-15	[-1, 256, 16, 16]	590,080
ReLU-16	[-1, 256, 16, 16]	0
MaxPool2d-17	[-1, 256, 8, 8]	0
Conv2d-18	[-1, 512, 8, 8]	1,180,160
ReLU-19	[-1, 512, 8, 8]	0
Conv2d-20	[-1, 512, 8, 8]	2,359,808
ReLU-21	[-1, 512, 8, 8]	0
Conv2d-22	[-1, 512, 8, 8]	2,359,808
ReLU-23	[-1, 512, 8, 8]	0
MaxPool2d-24	[-1, 512, 4, 4]	0
Conv2d-25	[-1, 512, 4, 4]	2,359,808
ReLU-26	[-1, 512, 4, 4]	0
Conv2d-27	[-1, 512, 4, 4]	2,359,808
ReLU-28	[-1, 512, 4, 4]	0
Conv2d-29	[-1, 512, 4, 4]	2,359,808
ReLU-30	[-1, 512, 4, 4]	0
MaxPool2d-31	[-1, 512, 2, 2]	0
AdaptiveAvgPool2d-32	[-1, 512, 7, 7]	0
Linear-33	[-1, 4096]	102,764,544
ReLU-34	[-1, 4096]	0
Dropout-35	[-1, 4096]	0
Linear-36	[-1, 4096]	16,781,312
ReLU-37	[-1, 4096]	0
Dropout-38	[-1, 4096]	0
Linear-39	[-1, 512]	2,097,664
Dropout-40	[-1, 512]	0
Linear-41	[-1, 10]	5,130

```

66 Dropout-42 [-1, 10] 0
67 =====
68 Total params: 136,363,338
69 Trainable params: 136,363,338
70 Non-trainable params: 0
71 -----
72 Input size (MB): 0.05
73 Forward/backward pass size (MB): 18.21
74 Params size (MB): 520.18
75 Estimated Total Size (MB): 538.45
76 -----

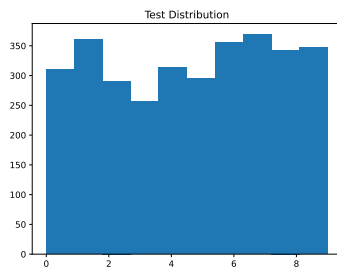
```

نتایج به صورتی است که در زیر و در شکل ۱۳، شکل ۱۴ و شکل ۱۵ آمده است. علت بهتر شدن نتایج مجموعه داده‌های ارزیابی می‌تواند به تفاوت پیچیدگی داده‌ها و ساده‌تر بودن داده‌های آن مجموعه برگردد. البته ما دخالتی در آن نداشته‌ایم اما مطابق آن چه استاد در کلاس درس گفته می‌توان با تعیین شاخص‌هایی مانند پیچیدگی و امکان تفکیک‌پذیری مجموعه داده را به لحاظ پیچیدگی کاملاً متوازن ساخت.

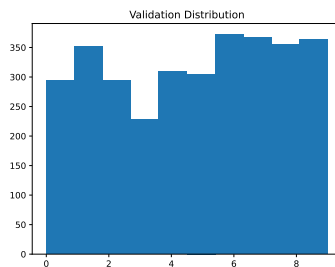
```

1 Test Accuracy: 0.9321
2 Test Error: 0.0679
3 Test Precision: 0.9328
4 Test Recall: 0.9321
5 Test F1-score: 0.9321

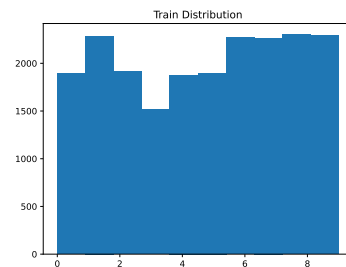
```



(ج) آزمون

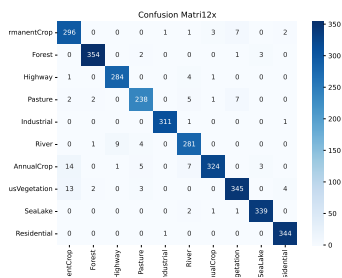


(ب) اعتبارسنجی

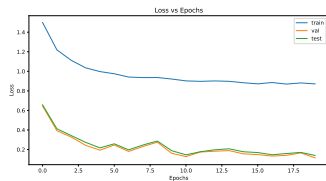


(آ) آموزش

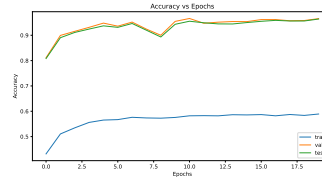
شکل ۱۳: نمودار توزیعی داده‌ها در پیاده‌سازی دوم.



(ج) ماتریس درهم‌ریختگی

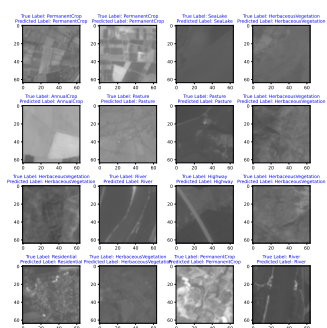


(ب) اتلاف

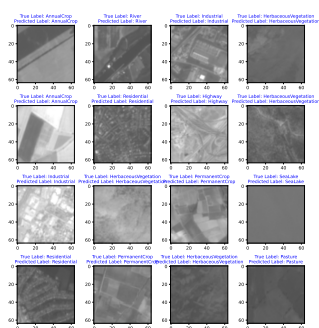


(آ) دقت

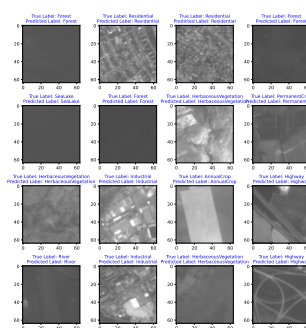
شکل ۱۴: نمودار دقت و اتلاف و ماتریس درهم‌ریختگی در پیاده‌سازی دوم.



(ج) ۳



(ب) ۲



(آ) ۱

شکل ۱۵: نتایج برخی آزمایش‌ها در پیاده‌سازی دوم.

