گزارش تشخیص ناهنجاری موجود در تصاویر شیشه به کمک مدل‌های پردازش تصویر بدون نظارت

نام و نام‌خانوادگی : کوروش خاوری مقدم

عنوان مقالات مورد بررسی :

Roth, K., Pemula, L., Zepeda, J., Schölkopf, B., Brox, T., & Gehler, P. (2022). **Towards Total Recall in Industrial Anomaly Detection**. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 14318-14328)

.

T. Defard, A. Setkov, A. Loesch, and R. Audigier, "**PaDiM: a Patch Distribution Modeling Framework for Anomaly Detection and Localization**," in ICPR Workshops, 2020

,

J. Yu, Y. Zheng, X. Wang, W. Li, Y. Wu, and R. Zhao, "**FastFlow: Unsupervised Anomaly Detection and Localization via 2D Normalizing Flows**," arXiv, vol. abs/2111.07677, 2021

فهرست عناوین

بررسی خط به خط دفترچه

ساخت مجموعه دادگان آموزش و تست

در خطوط [19-10] به ساخت دادگان آموزش مدل‌ها پرداخته‌ایم، بدین منظور از ماژول Folder جهت ساخت مجموعه دادگان غیراستاندارد استفاده کرده‌ایم، این ماژول با دریافت ویژگی‌های نام دادگان (دلخواه برای گزارش‌دهی و نام‌گذاری پوشه‌ها)، پوشه پایه، پوشه‌ی مربوط به داده‌های معمولی و ناهنجار، نوع فعالیت و اندازه‌ی عکس‌ها شی بارگذار داده را می‌سازد که جهت آموزش مدل‌ها استفاده می‌شود. در خط [20] با فراخوانی متد setup این شی را آماده استفاده نمودیم.

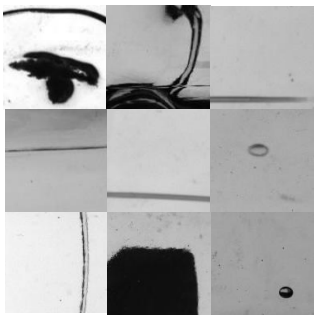
فرمت درختی مجموعه‌ی داده در شکل ۱ نشان داده شده است.

به صورت مشابه در خطوط [32-23] به ساخت دادگان تست جهت ارزیابی مدل پرداخته‌ایم. این دادگاه که در شکل ۲ نشان داده شده اند به نحوی دستچین شده اند که توانایی عمومی‌سازی (برخی از الگوها در داده‌های آموزش وجود ندارند)، و دقت مدل‌های (الگوهای خطی هم در داده آموزش و هم در داده‌های آزمون وجود دارند و تنها در صورتی ناهنجاری هستند که پر رنگ باشند) آموزش داده‌شده را به چالش بکشند.

```python
# Train Data module
train_datamodule = Folder(
    name="GlassDefect",
    root="/content/drive/MyDrive/Simple-AI-Task/GlassDefect",
    normal_dir="normal_train",
    abnormal_dir="abnormal_train",
    task="classification",
    image_size=(128,128),
    num_workers=1,
)
train_datamodule.setup()

# Test Data module
test_datamodule = Folder(
    name="GlassDefect",
    root="/content/drive/MyDrive/Simple-AI-Task/GlassDefect",
    normal_dir="normal_test",
    abnormal_dir="abnormal_test",
    task="classification",
    image_size=(128,128),
    num_workers=1,
)
test_datamodule.setup()
```

```
GlassData/
├── normal_train/
│   ├── 1_2450_887.png
│   └── ...
├── normal_test/
│   ├── 1_2400_829.png
│   └── ...
├── abnormal_train/
│   ├── 3_2286_1121.png
│   └── ...
├── abnormal_test/
│   ├── 6_2284_883.png
│   └── ...
└── mask/
    ├── 3_2286_1121.png
    └── ...
```



آموزش و ارزیابی مدل‌ها

با استفاده از موتور کتابخانه آنومالیب، عملیات آموزش و ارزیابی مدل‌ها به سادگی امکان پذیر است. با توجه به سادگی مجموعه دادگان، مدل‌های با تنظیمات اولیه انتخاب شده اند. این درحالیست که در صورت نیاز، می‌توان با تغییر مشخصات این مدل‌ها که در فایل کانفیگ آن‌ها وجود دارد، ساختارهای دیگر را نیز امتحان کرد. در خطوط [70-56] با تعریف تابع آموزش، تابعی تعریف کرده ایم که بتواند مدل‌های گوناگون را آموزش و آزمون کند.

```python
# test and train models
def training(model, train_datamodule, test_datamodule, ckpt):
    logger = AnomalibTensorBoardLogger("tb_logs", name=f"{type(model).__name__}")
    engine = Engine(
        max_epochs=20,
        logger=logger,
    )
    # train model
    engine.fit(datamodule=train_datamodule, model=model)
    # test model
    tests = engine.test(
        datamodule=test_datamodule,
        model=model,
        ckpt_path=ckpt,
    )
```

آنومالیب، از logger های گوناگون (MLFlow, Tensorboard, Weights And Biases) جهت ثبت فرآیند آموزش مدل‌ها پشتیبانی می‌کند. در این دفترچه درخط [58] ما با استفاده از Tensorboard به ثبت نتایج حاصل از مدل‌ها و آموزش آن‌ها پرداخته‌ایم. با تعریف موتور دلخوا در خطوط [59-62] می‌توان تنظیمات کلی مربوط به فرآیند آموزش را انجام داد، به دلیل استفاده ازمدل‌های پیش‌فرض، تنظیمات آموزش نیز به صورت پیش‌فرض برای هر مدل در نظر گرفته شده است که در صورت نیاز قابل تغییر چه در فایل‌های کانفیگ و چه در API کتابخانه می‌باشد. آموزش مدل‌ها در خط [64] با تعیین ساختار مدل و مجموعه‌دادگان آموزش صورت می‌گیرد. ارزیابی مدل‌ها با تعیین مجموعه‌دادگان آزمون، ساختار مدل و همچنین مقدار پارامترهای مدل که در فایل ckpt (check point) زخیره شده است، صورت می‌پذیرد.

در خطوط [72-84] به تعریف تابعی جهت استنتاج به کمک مدل‌های آموزش دادهشده پرداخته‌ایم. به کمک این تابع می‌توانیم معیارهای فنی لازم همانند AUPR, AUROC را ترسیم نماییم تا به کمک آن‌ها مدل‌ها را بررسی کنیم. خطوط [74-77] همانند قبل به تعریف موتور و لاگر پرداخته، بررسی تمام دادگان تست در خطوط [78-83] صورت می‌گیرد که حاصلش تنسور احتمال ناهنجاری است که توسط مدل تولید شده. با در دست داشتن این تنسور و تنسور مقادیر واقعی، می‌توانیم معیارهای مورد نظر را ترسیم کنیم که در خط [84] توسط تابع draw_graphs انجام شده است. از بررسی جزئی توابع draw_graphs و plot_images خودداری می‌کنیم چرا که این توابع پیچیدگی نداشته و صرفا جهت ترسیم نتایج هستند.

```
72  # inference using models
73  def inference(model, test_datamodule, ckpt):
74      logger = AnomalibTensorBoardLogger("/content/drive/MyDrive/Simple-AI-
75      engine = Engine(
76          logger=logger,
77      )
78      predictions = engine.predict(
79          datamodule=test_datamodule,
80          model=model,
81          ckpt_path=ckpt,
82          return_predictions=True,
83      )
84      draw_graphs(predicted=predictions[0]['pred_scores'], ground_truth=pre
```

Batch Training

در قسمت مربوط به Batch Training دفترچه، با تعریف مدل‌های دلخواه و فراخوانی تابع training که در بالا بررسی کردیم، در خطوط [1,2] به آموزش و آزمون اولیه ۶ مدل پرداخته‌ایم.

```
1  model_list = [Patchcore(), Padim(), Stfpm(), Dfm(), Fast
2  for model in model_list: training(model, copy.deepcopy(
```

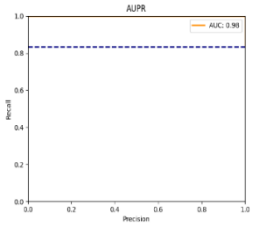
	Name	Type	Params
0	model	PatchcoreModel	24.9 M
1	_transform	Compose	0
2	normalization_metrics	MinMax	0
3	image_threshold	F1AdaptiveThreshold	0
4	pixel_threshold	F1AdaptiveThreshold	0
5	image_metrics	AnomalibMetricCollection	0
6	pixel_metrics	AnomalibMetricCollection	0

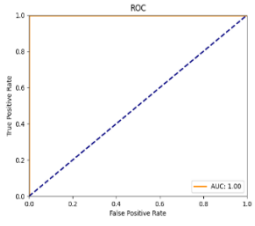
Trainable params: 24.9 M
Non-trainable params: 0
Total params: 24.9 M
Total estimated model params size (MB): 99

اطلاعات کلی ساختار هر مدل در زمان آموزش در کنسول نمایش داده می‌شود، برای مثال مدل PatchCore با 24.9 میلیون پارامتر به همراه سایر ویژگی‌های مرتبط و تعداد پارامترهای مربوط به هر کدام از آن ها نمایش داده شده است. همچنین سایز مدل نیز در خط آخر به صورت تقریبی برآورد شده که می‌تواند جزو معیارهای کلیدی ارزیابی تلقی شود.

Test metric	DataLoader 0
image_AUROC	1.0
image_F1Score	0.8888888955116272

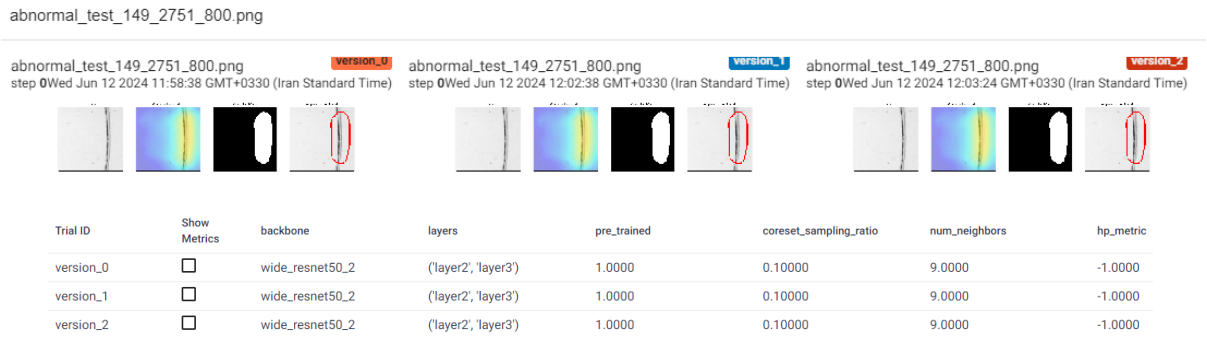
بعد از اتمام آموزش هر مدل و در مرحله‌ی تست، نتایج به صورت جدولی در کنسول نمایش داده می‌شود، برای مثال، نتایج حاصل از تست مدل PatchCore به صورت مقابل است که برای مجموعه‌ی دادگان ساده مورد بررسی و همچنین دادگان تست به نسبت جامع، منطقی به نظر می‌رسد، البته سایر مدل‌ها توانسته اند به امتیاز F1 یک نیز دست بیابند.





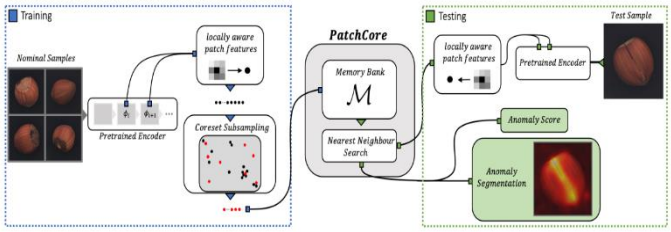
در پایان با فراخوانی تابع استنتاج در قسمت Training Curves به ترسیم نمودارهای آموزش پرداخته ایم، که به عنوان مثال برای مدل PatchCore در شکل ۳ نمایش داده شده است.

همچنین مقادیر لاگ آموزش مدل PatchCore شامل پارامترهای مربوط به مدل درنسخه‌های گوناگون آموزش آن به همراه نتایج حاصل از اجرای مدل بر عکس‌های مختلف و کانفیگ هر مدل است که در پوشه‌ی results زخیره شده است.



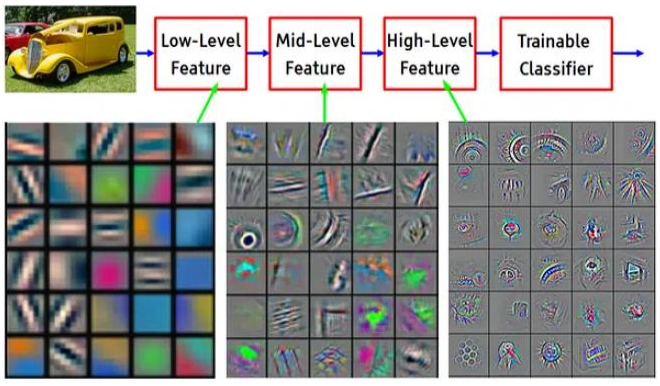
Trial ID	Show Metrics	backbone	layers	pre_trained	coreset_sampling_ratio	num_neighbors	hp_metric
version_0	☐	wide_resnet50_2	('layer2', 'layer3')	1.0000	0.10000	9.0000	-1.0000
version_1	☐	wide_resnet50_2	('layer2', 'layer3')	1.0000	0.10000	9.0000	-1.0000
version_2	☐	wide_resnet50_2	('layer2', 'layer3')	1.0000	0.10000	9.0000	-1.0000

لازم به ذکر است که امکان تهیه‌ی benchmark برای یک مجموعه‌ی داده دلخواه در این کتابخانه مهیا است که به کاربر اجازه می‌دهد با تعریف فایل کانفیگ دلخواه و تعیین نوع مدل‌ها، پارامتر و ساختار آن‌ها و روند کلی فرآیند آموزش به آموزش و بررسی یک مجموعه از مدل شناسایی عیوب بپردازد. همچنین ماژول‌های دسترسی سطح پایین نیز همانند Trainer وجود دارند که امکان انجام عملیات سطح پایین آموزش را به کاربر می‌دهند که در سناریوهای پیچیده کاربردی است.



بررسی سطح بالای مدل‌های آموزش داده‌شده:

در این قسمت به صورت مختصر به بررسی مدل‌های آموزش داده‌شده خواهیم پرداخت. برای اطلاعات بیشتر درباره‌ی هرکدام می‌توانید به مقالات ارجاع داده شده مراجعه کنید.



الگوریتم PatchCore

این الگوریتم شامل سه مرحله کلی است:

۱- جمع‌آوری ویژگی‌های محلی در بانک داده

الگوریتم با استفاده از یک شبکه از پیش آموزش داده شده همانند ResNet به جمع آوری ویژگی‌های **میانی** می‌پردازد. ویژگی‌های لایه‌های آغازین بسیار کلی و ویژگی‌های لایه‌های پایانی بسیار جزئی است و نویسندگان پیشنهادداده اند که خروجی لایه‌ی میانی جهت یافتن ناهنجاری استفاده شود. با جمع آوری خروجی های لایه‌های میانی از دادگان نرمال در یک بانک، تعداد بسیار زیادی بردار ویژگی (پچ) در این بانک زخیره می‌شود که بیان‌کننده ویژگی‌های عکس‌های نرمال هستند.

۲- کاهش دادگان به یک مجموعه اصلی جهت بازدهی بیشتر

الگوریتم اصلی (بخش ۳) مورد استفاده از یک رویکرد نزدیک ترین همسایه (KNN) جهت تشخیص داده‌های ناهنجار از داده‌های عادی استفاده می‌کند، به همین دلیل سایز بالای بانک داده سرعت را به شدت کاهش می‌دهد. با کمینه کردن یک متریک (نسبت دورترین عضو مجموعه اصلی به نزدیک ترین عضو مجموعه انتخاب شده) می‌توان مجموعه‌ی اصلی با سایز ۱ درصد

	Name	Type	Params
0	model	PatchcoreModel	24.9 M
1	_transform	Compose	0
2	normalization_metrics	MinMax	0
3	image_threshold	F1AdaptiveThreshold	0
4	pixel_threshold	F1AdaptiveThreshold	0
5	image_metrics	AnomalibMetricCollection	0
6	pixel_metrics	AnomalibMetricCollection	0

```
Trainable params: 24.9 M
Non-trainable params: 0
Total params: 24.9 M
Total estimated model params size (MB): 99
```

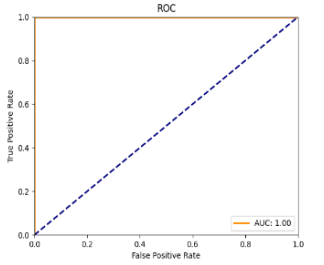
مجموعه‌ی اولیه ساخت که دقت بالا داشته به سرعت استنتاج نزدیک ۲۰۰ میلی‌ثانیه دست یابد.

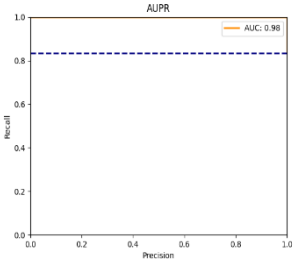
۳- الگوریتم اصلی جهت تشخیص و محلی سازی تصمیم

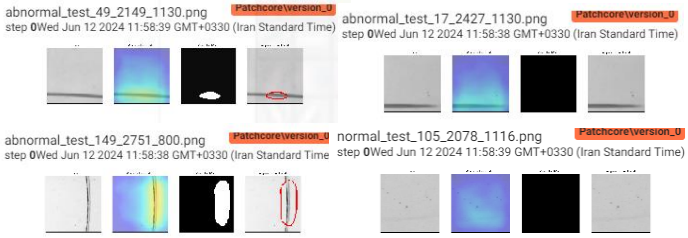
شامل استفاده از روش KNN در دو سطح پچ‌های محلی و پیکسل است. به صورت کلی این الگوریتم همانند مرحله‌ی ۲ ویژگی‌های میانی از عکس ورودی را استخراج کرده با بررسی فاصله‌ی دورترین عضو بین پچ‌های عکس ورودی و مجموعه کاهش داده شده، امتیاز ناهنجاری برای عکس در نظر میگیرد. در سطح پیکسل (به کمک ناحیه‌بندی) به صورت کلی فرآیندی مشابه با پچ‌های محلی رخ میدهد. در پایان به کمک امتیازهای جمع‌آوری شده، مدل امتیاز ناهنجاری کلی عکس ورودی را تعیین می‌کند که کلاسبندی به کمک آن رخ خواهد داد.

در تصاویر سمت چپ به نتایج حاصل از بررسی این مدل بر روی مجموعه داده فراهم شده پرداخته ایم، با توجه به سادگی الگوها و استفاده از وزن‌های شبکه ResNet50 آموزش داده شده از قبل، دستیابی به نتایج عالی دور از تصور نیست. سایز مجموعه اصلی ۱ درصد در نظر گرفته شده و از ۹ عضو جهت اجرای الگوریتم KNN استفاده کرده ایم.

Test metric	DataLoader 0
image_AUROC	1.0
image_F1Score	0.8888888955116272







الگوریتم PaDiM

این الگوریتم شامل سه مرحله‌ی کلی زیر است:

۱- ساخت فضای ثانویه

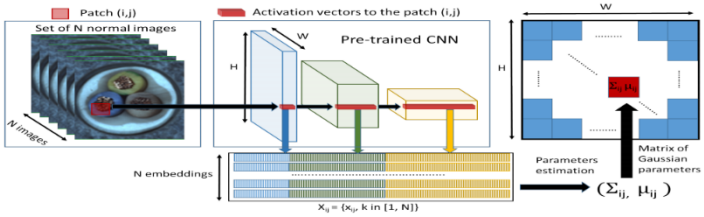
الگوریتم PaDiM با استفاده از خروجی لایه‌های سطح ۱ تا ۳ شبکه‌های از پیش تعیین شده همانند ResNet، از ویژگی‌های سطح پایین جهت تشکیل فضای ثانویه استفاده می‌کند، بدین صورت که با الحاق بردار ویژگی خروجی از هر لایه‌ی شبکه برای پچ‌های مکان‌های مختلف در عکس، بردارهای نمایش این داده‌ها در فضای ثانویه را می‌سازد.

۲- محاسبه‌ی توزیع پچ‌ها در فضای ثانویه

با درنظرگیری فرض استقلال پچ‌ها و داشتن توزیع نرمال، برای کلیه‌ی مکان‌های قرارگیری پچ‌ها، توزیعی نرمال تقریب زده می‌شود که میانگین و واریانس آن به کمک داده‌ها با استفاده از برآوردگر غیراریب محاسبه می‌شود.

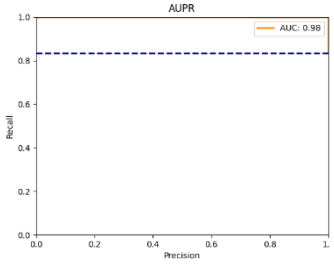
۳- تشخیص و کلاسبندی

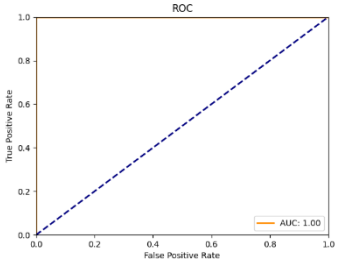
ابتدا نمایش تصویر ورودی طبق توضیحات قسمت اول در فضای ثانویه محاسبه می‌شود، سپس به کمک فاصله‌ی ماهالانوبیس، برای تمامی پچ‌های ساخته شده از عکس، فاصله‌ی بین بردار ویژگی پچ و توزیع نرمال پچ داده‌های عادی محاسبه شده و معیار ناهنجاری تابعی صعودی از این فاصله خواهد بود. جهت کاهش زمان اجرای الگوریتم از PCA (جهت کاهش فضای ثانویه) و استخراج تصادفی پچ ها استفاده شده است.



	Name	Type	Params
0	model	PadimModel	2.8 M
1	_transform	Compose	0
2	normalization_metrics	MinMax	0
3	image_threshold	F1AdaptiveThreshold	0
4	pixel_threshold	F1AdaptiveThreshold	0
5	image_metrics	AnomalibMetricCollection	0
6	pixel_metrics	AnomalibMetricCollection	0

```
Trainable params: 2.8 M
Non-trainable params: 0
Total params: 2.8 M
Total estimated model params size (MB): 11
```





Test metric	DataLoader 0
image_AUROC	1.0
image_F1Score	1.0

در روند آموزش PaDiM از ResNet18 از پیش آموزش دیده شده استفاده شده است.

الگوریتم FastFlow

این الگوریتم دارای دو مرحله‌ی کلی است:

۱- استخراج ویژگی‌های فضای ثانویه

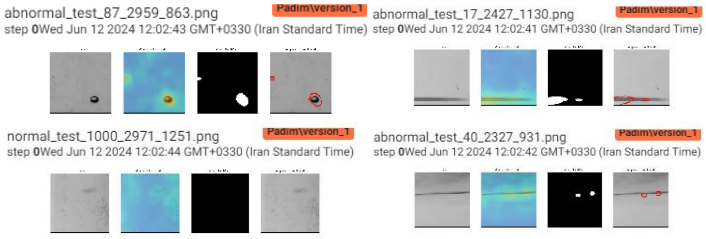
ابتدا بردار ویژگی‌های تصاویر به کمک شبکه‌های از پیش آموزش‌دادشده استخراج میگردد. بر خلاف دو مدل قبل، این مدل از ویژگی‌های استخراج شده از لایه‌ی آخر ResNet استفاده می‌کند.

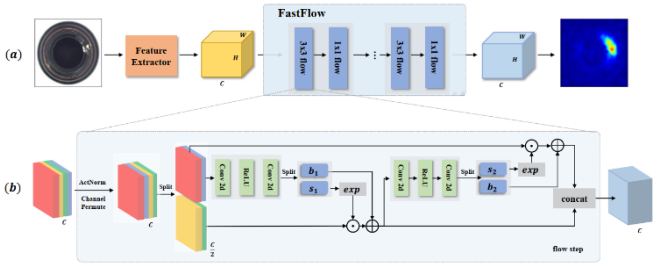
۲- تخمین درستنمایی در فضای ثانویه

الگوریتم اصلی جریان سریع در این قسمت رخ می‌دهد. این الگوریتم با فرض اینکه تبدیلی یک به یک و پوشا از توزیع اولیه‌ی داده در فضای ویژگی‌ها به توزیع نرمال در این فضا وجود دارد، مقدار Likelihood داده‌ی ورودی را به کمک Likelihood توزیع نرمال که توسط ماژول‌های FastFlow از روی دادگان آموزش در فضای ویژگی ساخته می‌شود، تقریب می‌زند. بنابراین این مقدار پیشامد می‌تواند معیاری برای ناهنجاری بوده در صورت کم‌بودن اشاره به خارج از توزیع بودن داده ورودی دارد.

انتخاب بهترین مدل

در این نوشتار به بررسی جزئی تنها سه مدل کفایت کرده‌ایم چرا که سایر روش‌ها مشابه با این روش، تلاش در یافتن داده‌های خارج از توزیع داده معمول دارند. با توجه به استفاده از شبکه‌های از پیش آموزش داده شده و استخراج ویژگی‌های گویا و از طرف دیگر ساده بودن مجموعه‌ی دادگان، در تلاش‌های گوناگون به مدل‌های با متریک ۱ برای AUROC و F1Score دست یافته ایم و لذا نمی‌توان صرفا طبق این معیارها مدل مناسب را مشخص کرد. از طرفی می‌توانیم اولیت را زمان آموزش، استنتاج و اندازه‌ی حافظه‌ی لازم مدل‌ها بدانیم و با این انتخاب مدل PaDiM دارای کمترین اندازه مدل است، مدل PatchCore سریع ترین زمان استنتاج و آموزش را دارد و می‌توانند به عنوان مدل‌های برتر در نظر گرفته شوند. الگوریتم FastFlow نیاز به یادگیری تابع تبدیل یک به یک و پوشا را دارد لذا زمان آموزش مدل بالاست و از طرف دیگر، حجم مدل نسبت به PatchCore پایین تر است. بنابراین در سناریوهای مختلف یکی از دومدل اول می‌تواند به عنوان مدل برگزیده انتخاب شود.





	Name	Type	Params
0	loss	FastflowLoss	0
1	_transform	Compose	0
2	normalization_metrics	MinMax	0
3	image_threshold	F1AdaptiveThreshold	0
4	pixel_threshold	F1AdaptiveThreshold	0
5	image_metrics	AnomalibMetricCollection	0
6	pixel_metrics	AnomalibMetricCollection	0
7	model	FastflowModel	7.0 M

```
Trainable params: 2.8 M
Non-trainable params: 4.2 M
Total params: 7.0 M
Total estimated model params size (MB): 27
```

Test metric	DataLoader 0
image_AUROC	1.0
image_F1Score	1.0