HW2-Paper

HANWEN LIAO
hl5361
hl5361@nyu.edu

1. README

- python notebook and dataset are in link: https://github.com/HanwenLiao/eng-ai-agents-hw/tree/main/assignments
- all my code is located in file: 02-assignment-2.ipynb

2. How my code works

2.1. Download and fetch dataset

Applying a Gaussian Blur helps in reducing image noise and detail, which simplifies the later stages of edge detection and contour finding.

2.2. Training models by anomalib

2.2.1 EfficientAD

```
data = MVTec(root="./datasets/MVTec",
             category=category,
             train_batch_size=1,
             eval_batch_size=32,
             num_workers=8)

model = EfficientAd()

engine = Engine(max_epochs=5)
engine.fit(datamodule=data, model=model)
engine.test(datamodule=data, model=model)

metrics=engine.trainer.callback_metrics
```

2.2.2 PatchCore

```
data = MVTec(root="./datasets/MVTec",
             category=category,
             train_batch_size=1,
             eval_batch_size=4,
             num_workers=2)

model = Patchcore(backbone="resnet18",
        layers=["layer2", "layer3", "layer4"],
        pre_trained=False,
        num_neighbors=3)
```

```
engine = Engine(max_epochs=3)
engine.fit(datamodule=data, model=model)
engine.test(datamodule=data, model=model)

metrics = engine.trainer.callback_metrics
```

2.3. AUROC Values and other scores

```
img_auroc = metrics.get("image_AUROC", torch.tensor(0.0)).item()
img_f1 = metrics.get("image_F1Score", torch.tensor(0.0)).item()
pxl_auroc = metrics.get("pixel_AUROC", torch.tensor(0.0)).item()
pxl_f1 = metrics.get("pixel_F1Score", torch.tensor(0.0)).item()
```

2.3.1 EfficientAD

Image-level AUROC and F1 Score are very high on Grid (0.9833) and Tile (0.9812), indicating that the model can distinguish normal and abnormal samples well. Leather (0.8655) performs slightly weaker, indicating that the ability to identify abnormalities in this category is relatively low. Pixel AUROC is good on Grid (0.9111) and Leather (0.9357), indicating that the model can Pixel F1 Score is generally low (the lowest is 0.4197), especially on Grid and Leather, indicating that the model is not accurate enough in locating abnormal areas.

Category	Image_AUROC	Image_F1Score	Pixel_AUROC	Pixel_F1Score
Grid	0.9833	0.9821	0.9111	0.4636
Tile	0.9812	0.9576	0.8303	0.5853
Leather	0.8655	0.8649	0.9357	0.4197

Table 1. Performance Metrics for Different Categories

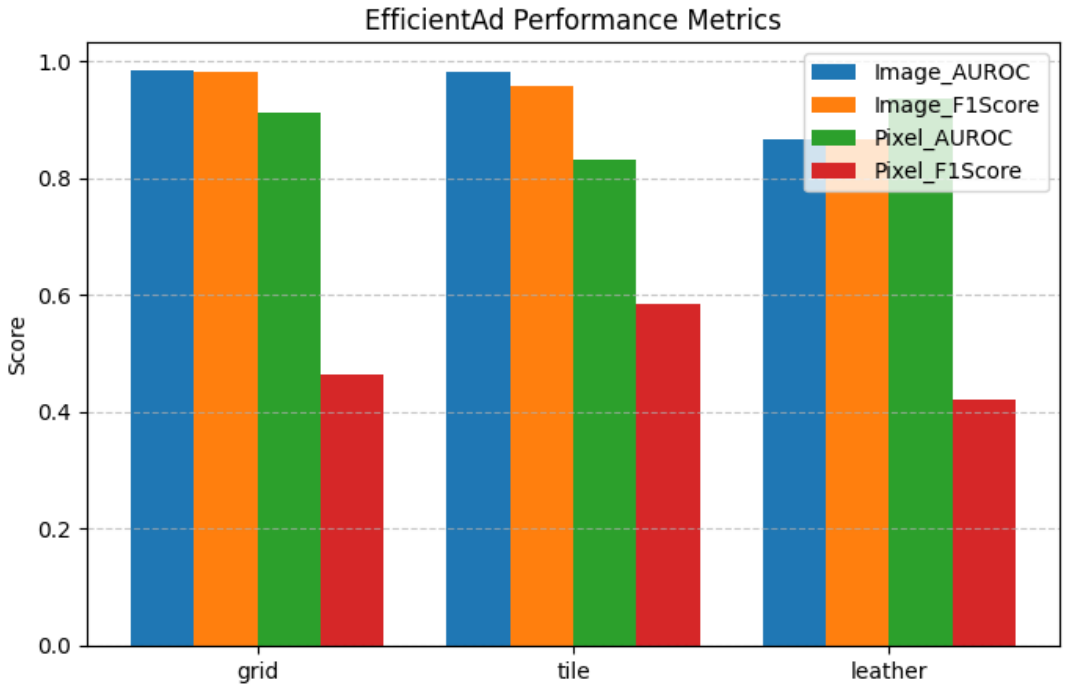

Figure 1

2.3.2 PatchCore

The Image AUROC of Grid (0.9031) and Tile (0.9466) is high, indicating that the model has strong overall anomaly detection capabilities for these two categories. The Image F1Score is above 0.86, indicating that the model is stable and accurate in image-level anomaly detection. The Pixel AUROC is highest on Leather (0.9635), indicating that the model can distinguish the abnormal regions of the Leather class well. The Pixel AUROC of Grid (0.7892) is low, which may be due to the small abnormal region or unclear features.

Category	Image_AUROC	Image_F1Score	Pixel_AUROC	Pixel_F1Score
Grid	0.9031	0.9333	0.7892	0.1386
Tile	0.9466	0.9308	0.8602	0.5741
Leather	0.8757	0.8634	0.9635	0.3619

Table 2. Performance Metrics for Different Categories

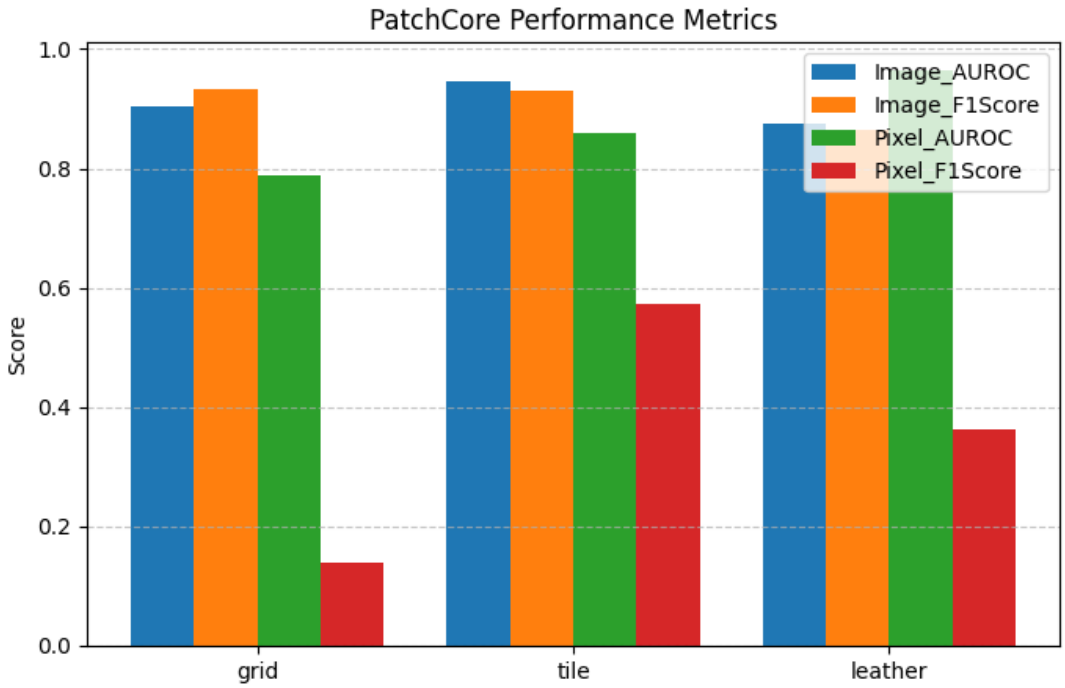

Figure 2

2.4. Abnormal heat map

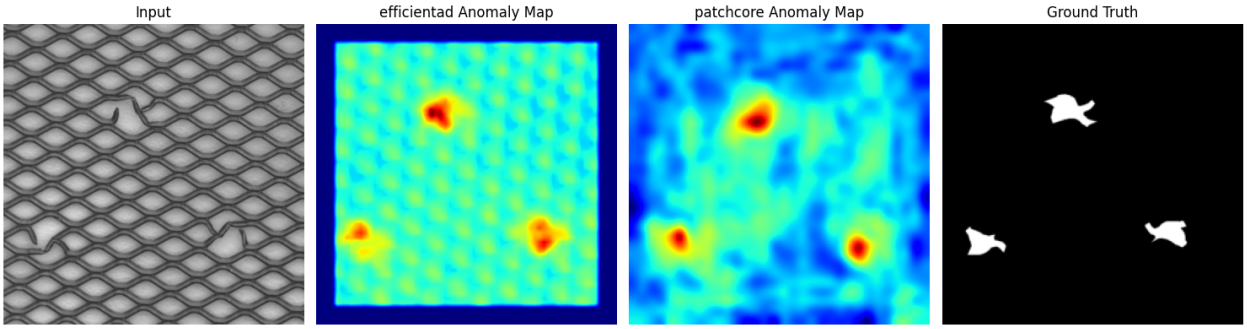

Figure 3

2.5. Similarity Search

- load model and preprocess the img

```
device = "cuda" if torch.cuda.is_available() else "cpu"
path = "/results/Patchcore/MVTec/leather/latest/weights/lightning/model.ckpt"
model = Patchcore.load_from_checkpoint(path).eval().to(device)

preprocess = transforms.Compose([
    transforms.Resize((256, 256)),
```

```
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
```

- extract features

```
def extract_features(model, img_path):
    img = Image.open(img_path).convert("RGB")
    tensor = preprocess(img).unsqueeze(0).to(device)

    with torch.no_grad():
        out = model(tensor)

    if "anomaly_map" not in out:
        raise KeyError(f"found {list(out.keys())}")

    fmap = out["anomaly_map"]
    features = F.adaptive_avg_pool2d(fmap, (1, 1)).view(-1).cpu().numpy()

    return np.pad(features,
    (0, max(0, 1024 - len(features))))[:1024] / np.linalg.norm(features)
```

- use Qdrant to save feature

```
client = QdrantClient(":memory:")
collection = "patchcore_leather"

if client.collection_exists(collection):
    client.delete_collection(collection)

client.create_collection(collection_name=collection,
vectors_config=VectorParams(size=1024, distance=Distance.COSINE))

os.makedirs("results", exist_ok=True)

points, img_id = [], 0
data_dir = "/content/drive/MyDrive/content/datasets/MVTec/leather/test/"
for root, _, files in os.walk(data_dir):
    for f in files:
        if f.endswith((".png", ".jpg")):
            img_path = os.path.join(root, f)
            vector = extract_features(model, img_path)
            points.append(PointStruct(id=img_id, vector=vector.tolist(),
            payload={"img": img_path, "abn": "good" not in img_path}))
            img_id += 1
client.upsert(collection_name=collection, points=points)
```

- query

2.5.1 EfficientAD

I input a Grid image classified as broken, and you can see that the five similar images successfully returned are all broken Grids.

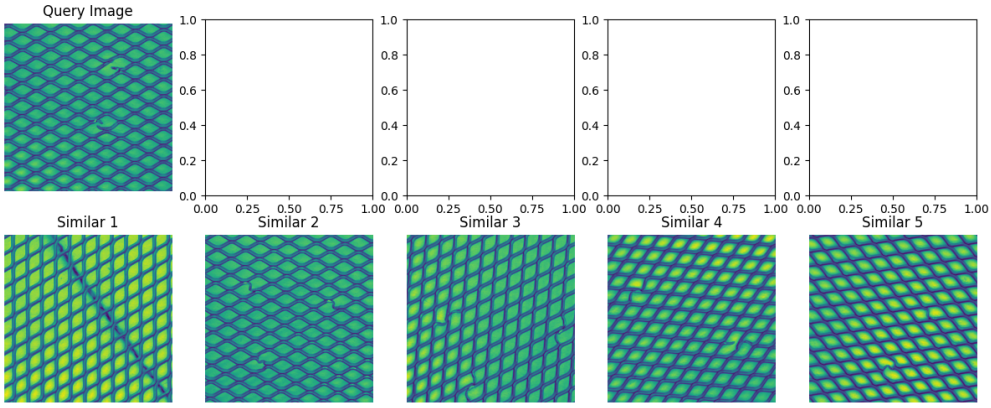

Figure 4

2.5.2 PatchCore

I input a Good Leather image, and the five returned images are all Good Leather.

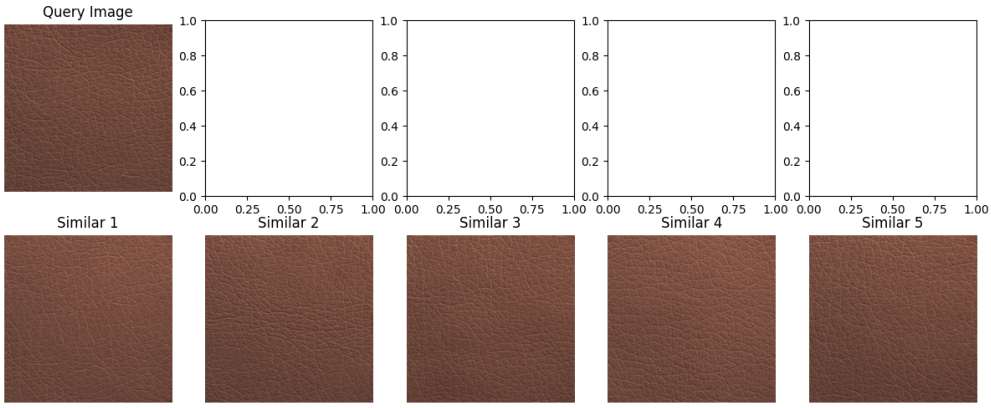

Figure 5

3. How these models work

3.1. EfficientAD

EfficientAD uses the pre-trained EfficientNet model to extract image features, and then models the feature distribution of normal samples through unsupervised learning. In the detection stage, new samples are compared with the normal distribution to calculate their anomaly scores.

3.2. PatchCore

The core principle of PatchCore is based on unsupervised anomaly detection. It uses pre-trained CNN to extract local image features and build a compact memory library to store the features of normal samples. During detection, the features of new samples are matched with the memory library for nearest neighbors (KNN nearest neighbor search).

4. Patch Description Network (PDN) Receptive Field Calculation

	Kernel	**Stride**	**Padding**		**Jump**	
Input	-	-	-	256×256	1	1
Conv1	4	1	3	256×256	1	$1 + (4 - 1) \times 1 = 4$
AvgPool1	2	2	1	128×128	2	$4 + (2 - 1) \times 1 = 5$
Conv2	4	1	3	128×128	2	$5 + (4 - 1) \times 2 = 11$
AvgPool2	2	2	1	64×64	4	$11 + (2 - 1) \times 2 = 13$
Conv3	3	1	1	64×64	4	$13 + (3 - 1) \times 4 = 21$
Conv4	4	1	0	61×61	4	$21 + (4 - 1) \times 4 = 33$