Scratch Detection from scratch !

Submitted by : Soham Chatterjee
Mail Id: babinchat02@gmail.com
Assignment : Mowito

Problem Statement

In this assigned task, we aim to build a robust model capable of identifying scratches on images containing text. The dataset consists of two classes: **good images** with clear, unsullied text and **bad images** where the text is obscured by scratches. The model should not only classify the images accurately but also provide a bounding box or mask highlighting the scratch in the bad images. The solution must address the following:

- Minimize the misclassification of good images as bad (**precision** of bad images).
- Ensure no/very less bad image is left undetected (**recall** of bad images).
- Allow the user to define a threshold for the size of the scratch to classify an image as bad.

Objectives

The primary objectives of this project are:

1. **Image Classification:** Develop a deep learning model to classify images as *good* (without scratches) or *bad* (with scratches).
2. **Scratch Localization:** Enhance the model to create a bounding box or mask around the scratch in bad images.
3. **Performance Metrics:** Evaluate the model based on:
    - **Precision:** Minimize the false positives of bad image classification.
    - **Recall:** Maximize the detection of bad images, ensuring no bad image is left undetected.
4. **Threshold Control:** Implement a user-defined threshold for scratch size to classify an image as bad.
5. **Data Expansion:** Explore and implement methods to expand the dataset of bad images beyond standard augmentation techniques.
6. **Model Generalization:** Investigate possibilities to extend the model to detect scratches on surfaces other than text, such as metallic surfaces or phone screens.

Introduction

Scratches over text-based surfaces degrade the usability and readability significantly. This affects applications pertaining to document analysis, labeling products, and digital archiving, to name a few. This automatically detecting and locating scratch can save time in various workflows, improve their efficiency, and reduce manual inspections to a great extent.

The present project deals with the development of a machine learning-based solution for classifying images of text into two classes: *good images*, where the text is clear and unobstructed, and *bad images*, where scratches obscure the text. Beyond classification, the model should localize scratches by generating bounding boxes or masks that enable precise detection and actionable insights.

This is a dataset of images containing three types of text, which introduces variety and makes the data realistic. From a user's point of view, we are also going to implement the code in such a way that it will allow for modification of the threshold in scratch size, so that the user can decide what a bad image is. Additional objectives include exploring data augmentation beyond the standard techniques and expanding the model to work on scratches over other surfaces, including metallic objects and phone screens.

We intend to provide a solution that can achieve high precision and recall of bad image classification, which also shows the ability for generalization and adaptation across different use cases.

Progress Report on Classification of Good and Bad Images

Our progress started with a simple CNN model that would classify the images as *good* or *bad* depending on whether they have scratches or not. The results were promising enough to go ahead and experiment with other complex architectures like AlexNet and ResNet. In these experiments, we could see that the custom CNN model outperformed the rest of the models with respect to recall for finding bad images. The simpler architecture generalized better for a few reasons:

- Dataset Complexity vs. Model Complexity: Our dataset was related to a binary classification task with relatively simple requirements. While AlexNet and ResNet are designed for complex multi-class problems, the simpler architecture of our custom CNN better suited the task, avoiding overfitting on the limited dataset.
- Transfer learning from models like AlexNet and ResNet employs pre-trained features on large-scale datasets such as ImageNet. These features did not fit very well in the specific distinction of scratches from clear text and hence did not perform better despite efforts towards fine tuning.
- Input Resolution and Feature Extraction Requirements: The task required the identification of small-scale disruptions such as scratches, which could be effectively captured by simpler models without relying on the deep hierarchical features emphasized by ResNet and AlexNet. Moreover, resizing images to match the input resolution of these complex models introduced distortions that may affect their performance.
- Training and Hyperparameter Optimization: Custom CNN contains lesser numbers of parameters, and light tuning, therefore, convergence was not an issue since minimal changes in its hyperparameter was needed, while AlexNet and ResNet required higher precision on the hyperparameter tunings for which were challenging to work out optimally for this dataset.
- Dataset Imbalance: There was some imbalance between good and bad image classes in our dataset. Simpler architectures showed better resistance to this imbalance by focusing more effectively on the minority class.

While the simpler CNN proved to be the most effective for this task, we can easily recognize that AlexNet and ResNet might give better results if advanced preprocessing, robust augmentation, and optimized transfer learning are performed. However, for our specific dataset and task requirements, the custom CNN remains the best option that achieves high recall in the classification of bad images with efficiency.

Results

Parameter	**Value**
Input Image Height	100
Input Image Width	100
CNN Architecture	2 Convolutional Layers, 2 MaxPooling Layers, 1 Fully Connected Layer
Convolutional Layer 1 Filters	32
Convolutional Layer 1 Kernel Size	(3, 3)
Convolutional Layer 2 Filters	64
Convolutional Layer 2 Kernel Size	(3, 3)
MaxPooling Size	(2, 2)
Fully Connected Layer Neurons	64
Output Layer Activation Function	Sigmoid
Loss Function	Binary Crossentropy
Optimizer	Adam
Batch Size	32
Epochs	20
Validation Split	0.2

Table 1: Important Parameters Used in the CNN Model

Layer (type)	Output Shape	Param #
conv2d_15 (Conv2D)	(None, 98, 98, 32)	896
max_pooling2d_15 (MaxPooling2D)	(None, 49, 49, 32)	0
conv2d_16 (Conv2D)	(None, 47, 47, 64)	18,496
max_pooling2d_16 (MaxPooling2D)	(None, 23, 23, 64)	0
flatten_6 (Flatten)	(None, 33856)	0
dense_14 (Dense)	(None, 64)	2,166,848
dense_15 (Dense)	(None, 1)	65

Total params: 6,558,917 (25.02 MB)
Trainable params: 2,186,305 (8.34 MB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 4,372,612 (16.68 MB)

Figure 1: Model architecture

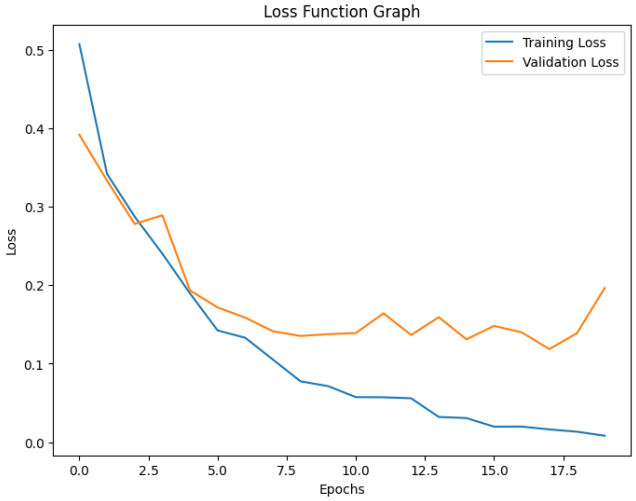

Figure 2: Loss function on training and validation

Classification Report:

	precision	recall	f1-score	support
Good Image	0.95	0.88	0.91	223
Bad Image	0.97	0.99	0.98	817
accuracy			0.96	1040
macro avg	0.96	0.93	0.94	1040
weighted avg	0.96	0.96	0.96	1040

Figure 3: Custom CNN Model : Classification Report on Test Data

In Figure 3 we see the model performed well on the test data , especiall the recall score. The recall for good images is comparatively low. If data augmentation techniques were applied uniformly across both classes, it might fail to address the specific characteristics of good images. Augmentations like adding noise or slight distortions may inadvertently cause good images to resemble bad images, thereby confusing the model and reducing its recall for the good image class.

Genral Remarks:

1. Changing the batch size did not change results much but lead to lower validation loss when larger batch sizes were used.
2. Using lower learning rates than 0.001 did not lead to good convergence of the loss function quickly.
3. The use of 3x3 filters in the convolutional layers is a common practice in CNNs, as it allows for detecting fine-grained features such as edges, textures, and small objects, which are essential in scratch detection. A 3x3 filter can effectively capture subtle variations in pixel intensity, which is crucial for identifying scratches, especially when they are thin and localized.
4. Batch normalization was applied, but it resulted in lower metrics on good images. The reasoning could be:
    - **Good Images (No Scratches):** The patterns are simpler, and their features may already be distinguishable without normalization. Applying normalization could dampen or alter critical features, reducing the model's ability to detect these patterns effectively.
    - **Bad Images (Scratches):** Scratches introduce more variability, making the features harder to separate. Normalization helps by aligning and scaling these features, making them easier for the model to classify.

Find the model weights at here

Progress Report on Prediction of Masks/Bounding Boxes

Initial Approach

The initial approach to predicting masks for identifying scratches on images involved exploring the concept of "masks" in image analysis. At first, I realized that for such tasks, the model architecture should be specifically designed for segmentation, as masks represent the parts of an image that correspond to certain objects or features. To model these masks, I explored different image segmentation techniques, particularly focusing on the distinction between semantic segmentation and instance segmentation. For the prediction of masks on scratched images, I determined that instance segmentation would be more suitable. This led me to investigate popular models like YOLO, U-Net, Mask R-CNN, and Faster R-CNN, which are commonly used for object detection and segmentation tasks.

Dataset Preparation and U-Net Model

I began by organizing my dataset for training and testing. I divided the directory into four subfolders:

- `train` - contains the training images.
- `train_masks` - contains the corresponding masks for the training images.
- `test` - contains the test images.
- `test_masks` - contains the corresponding masks for the test images.

With the dataset prepared, I implemented the U-Net architecture for semantic segmentation, which is well-suited for tasks like mask prediction. However, the performance was initially poor. The predicted masks on test images were not accurate, and the binary mask thresholds varied from image to image, likely due to different lighting conditions.

To address this, I applied a thresholding technique based on the 90th percentile to enhance the consistency of the predicted masks across varying conditions. The thresholding was implemented as follows:

```
# Ok so now percentile threshold on the tensor
pred_mask = pred_mask.squeeze(0).cpu().numpy()
threshold = np.percentile(pred_mask, 90)
pred_mask = (pred_mask > threshold).astype(np.float32)
```

Despite applying thresholding, the model performance remained unsatisfactory, and the results did not meet expectations.

```
model = UNet(in_channels=3, num_classes=1).to(device)
criterion = nn.BCEWithLogitsLoss()
optimizer = optim.Adam(model.parameters(), lr=0.0001)
```

Figure 4: Sample Prediction with Unet

Full code at: here

Switching to Mask R-CNN for Instance Segmentation

Realizing that instance segmentation might yield better results, I started exploring the Mask R-CNN architecture. I first attempted to use the Mask R-CNN implementation from `matterport/Mask_RCNN` on GitHub. However, I encountered multiple version compatibility issues due to differences in TensorFlow/Keras versions, specifically with the `keras.engine` module, which was no longer supported in recent TensorFlow versions.

Transition to PyTorch-based Mask R-CNN

Since the versioning issues in TensorFlow/Keras were proving difficult to resolve, and given that Google Colab provided better GPU support for PyTorch, I switched to using PyTorch-based models for instance segmentation. I used the following models from the PyTorch `torchvision` library:

```
from torchvision.models.detection import maskrcnn_resnet50_fpn
from torchvision.models.detection.faster_rcnn import FastRCNNPredictor
from torchvision.models.detection.mask_rcnn import MaskRCNNPredictor
```

Figure 5: The masks overlaid on the bad image.

Next, I converted the mask images as showin in Fig 5 into annotations in JSON format as shown in Fig 6, which is commonly used for instance segmentation tasks. The annotations contained the pixel-wise locations of each scratch in the image, as well as the associated labels.

Figure 6: annotations.json file preview

Model Overview

In this work, a Mask R-CNN model with a ResNet-50 backbone is employed for scratch detection. Mask R-CNN is a well-known deep learning architecture designed for object detection and instance segmentation. The model utilizes Region Proposal Networks (RPN) to propose regions of interest, followed by a fully connected layer for classifying the objects and predicting segmentation masks.

Model Architecture

The model is based on the Mask R-CNN architecture, which consists of the following main components shown explicitly in Fig 7:

- **Backbone**: A pre-trained ResNet-50 feature extractor is used to capture spatial features from input images.
- **Region Proposal Network (RPN)**: Generates potential object regions from the feature maps.
- **RoI Align**: Extracts fixed-size feature maps from the regions proposed by RPN.
- **Box and Mask Predictors**: Predict bounding boxes and segmentation masks for each detected object.

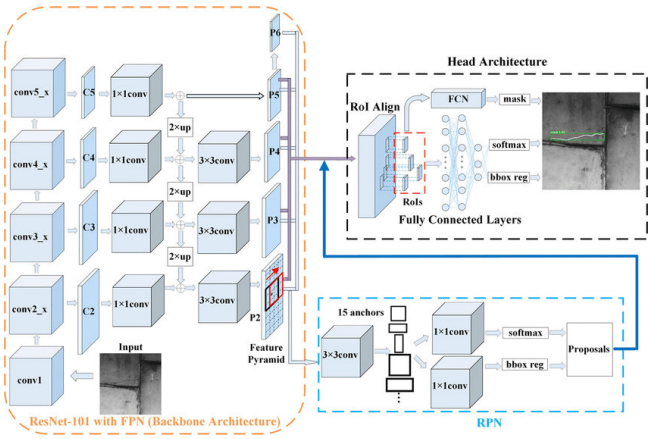

Figure 7: Enter Caption

Customization for Scratch Detection

The model was fine-tuned for the scratch detection task, which includes the following steps:

- **Class Modification**: The original model is customized to detect two classes: background and scratch.
- **Mask Prediction**: The mask predictor is replaced with a new one suited for the scratch class.

Training Setup

The model was trained using a standard stochastic gradient descent (SGD) optimizer with the following settings:

Hyperparameter	Value
Learning Rate	0.005
Momentum	0.9
Weight Decay	0.0005
Batch Size	2
Epochs	3

Table 2: Training Hyperparameters

Model Training and Results

The following code snippet shows the training for the model:

```
dataset = ScratchDataset(
    json_file='/content/annotations.json',
    img_dir='/content/images/UNet-PyTorch/data/train')

data_loader = DataLoader(
    dataset,
    batch_size=2,
    shuffle=True,
    collate_fn=lambda x: tuple(zip(*x)))

model = get_model_instance_segmentation(num_classes=2)
# Background + scratch
model.to(device)

params = [p for p in model.parameters() if p.requires_grad]
optimizer = torch.optim.SGD(params,
lr=0.005, momentum=0.9, weight_decay=0.0005)
```

```
num_epochs = 3
for epoch in range(num_epochs):
    loss = train_one_epoch(model, optimizer, data_loader, device)
    print(f"Epoch: {epoch}, Loss: {loss}")
```

Full code at: here

After preparing the data in the required format, I ran the Mask R-CNN model for two epochs using the pre-trained weights of the ResNet50 architecture. The results were more satisfactory than the U-Net approach, as the model was able to predict accurate masks on unseen images.

However, due to limited GPU resources on Google Colab, I was only able to train the model for two epochs, but the initial results showed promise. The masks were more precise, and the model was able to effectively detect scratches in a variety of lighting conditions. A sample prediction is shown in Fig 8

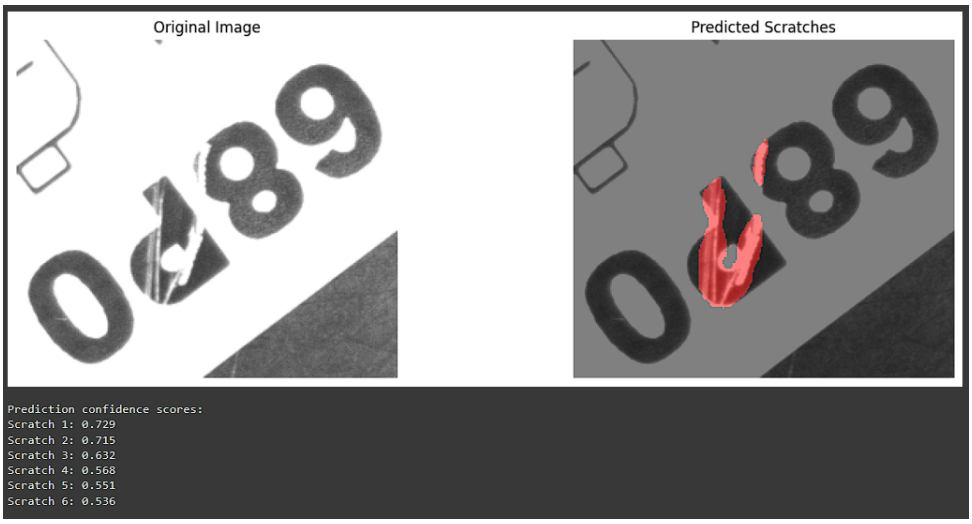

Figure 8: Sample Mask drawn on unseen image

General Remarks

The confidence scores, ranging from 0 to 1, indicate the model's certainty about detecting scratches in different areas of the image. These scores reflect the model's belief that the detected regions contain scratches, with higher values signifying higher confidence. For instance, in the case of a detection with scores as follows:

- **Score 0.729 (72.9%)**: The model is most confident about this scratch detection.
- **Score 0.715 (71.5%)**: The model is reasonably confident about this scratch detection.
- **Score 0.536 (53.6%)**: The model is least confident among these detections.

The presence of multiple scratches detected by the model can be attributed to several possible factors:

1. **Multiple Scratches**: The image may indeed contain multiple distinct scratches, and the model detects each one separately.
2. **Fragmentation of a Single Scratch**: The model might mistakenly fragment what should be one scratch into multiple smaller regions. This could be caused by various factors such as image noise, texture variations, or the model's inability to detect the full extent of a scratch as a single entity.

What I learnt

Through this process, I learned the importance of selecting the right architecture for the problem at hand. Initially, U-Net provided a good starting point, but switching to Mask R-CNN allowed for more accurate instance segmentation and better performance on unseen data. The implementation of a thresholding technique also helped mitigate lighting-related challenges in mask prediction. This approach, combining appropriate architecture selection and data preparation, led to significant improvements in the accuracy of the predicted masks.

Some more testing

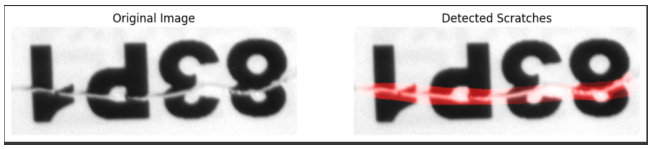

Figure 9: More test cases - 1

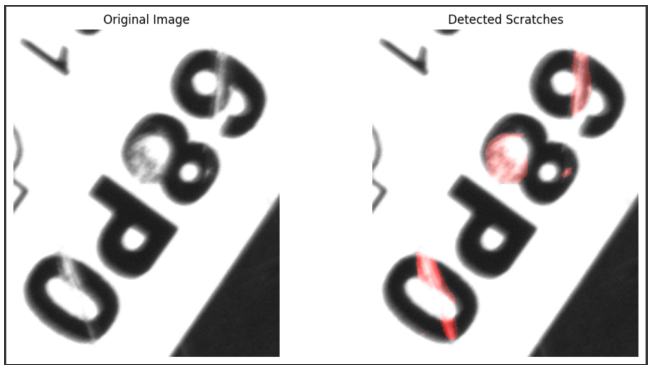

Figure 10: More test cases - 2

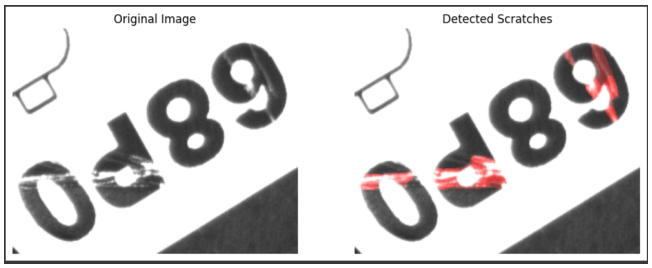

Figure 11: More test cases - 3

Summary of Approaches

Approach	**Outcome**
U-Net	Poor performance, inaccurate masks, thresholding applied to improve consistency
Mask R-CNN (PyTorch)	Improved results, accurate masks on unseen data, better handling of varying lighting conditions

Table 3: Summary of Different Approaches for Mask Prediction

Next tasks - Currently in progress

1. To extend the model to detect any kind of scratch. FOR THAT, we might need more training data like scratches on phones (some are available online on Kaggle). As shown in Fig. 12

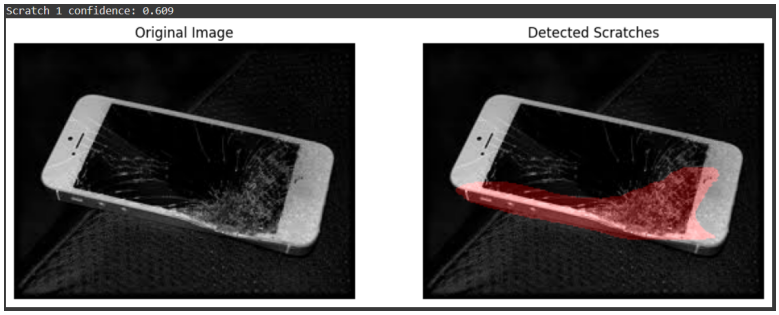

Figure 12: Sample scratch detection using our model trained only on the number plates. Certainly the model seems to understand the detection feature as a "scratch" but we need to train the model on the phone scratch images as well so that "scratch" is generalised more.

2. Figure out ways to increase the dataset size of bad images beyond the typical data augmentation techniques. **Solution : We can make synthetic data by adding random scratches to the good images to generate more data on bad images.**

Appendix

Github Link :
Click Here

References

1. **Implement Your Own Mask RCNN Model**. Available at: https://medium.com/analytics-vidhya/implement-your-own-mask-rcnn-model-65c994a0175d
2. **Image Segmentation with U-Net**. Available at: https://www.analyticsvidhya.com/blog/2022/10/image-segmentation-with-u-net/
3. **Mask R-CNN with OpenCV - PyImageSearch**. Available at: https://pyimagesearch.com/2018/11/19/mask-r-cnn-with-opencv/
4. **Instance Segmentation Using Mask R-CNN on Custom Dataset - YouTube**. Available at: https://www.youtube.com/watch?v=QP9Nl-nw890
5. **286 - Object Detection Using Mask R-CNN: End-to-End from Annotation to Prediction - YouTube**. Available at: https://www.youtube.com/watch?v=QntADriNHuk
6. **A Simple Guide to Mask R-CNN Custom Dataset Implementation (Computer Vision)**. Available at: https://medium.com/analytics-vidhya/a-simple-guide-to-maskrcnn-custom-dataset-implementation-27f7eab381f2
7. **What is Mask R-CNN? — Mask R-CNN — How Does Mask R-CNN Work? — Medium**. Available at: https://abhijeetpujara.medium.com/object-segmentation-with-mask-r-cnn-6e9c06cef81c
8. **Faster R-CNN in PyTorch - GitHub Repository**. Available at: https://github.com/jwyang/faster-rcnn.pytorch
9. **How to Use Mask R-CNN in Keras for Object Detection in Photographs**. Available at: https://machinelearningmastery.com/how-to-perform-object-detection-in-photographs-with-mask-r-cnn-in-keras/
10. **Hybrid Model Classification ResNet+VGG16+MobileNet**. Available at: https://www.kaggle.com/code/pratul007/hybrid-model-classification-resnet-vgg16-mobilenet