```
!pip install --upgrade pip
!pip install torch torchvision torchaudio
!pip install fsspec==2024.6.1
!pip install datasets==3.0.0
!pip install gcsfs==2024.6.0
!pip install jiwer
!pip install evaluate

# Imports
from google.colab import drive
import os, sys, itertools
import pandas as pd
from sklearn.model_selection import train_test_split
from PIL import Image

import torch
from torch.utils.data import Dataset

from datasets import load_dataset
import transformers
from transformers import Seq2SeqTrainingArguments, Seq2SeqTrainer
from transformers import VisionEncoderDecoderModel, TrOCRProcessor, default_data_
import evaluate
```

```
Requirement already satisfied: pip in /usr/local/lib/python3.10/dist-packages
Collecting pip
  Downloading pip-24.3.1-py3-none-any.whl.metadata (3.7 kB)
Downloading pip-24.3.1-py3-none-any.whl (1.8 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.8/1.8 MB 37.5 MB/s eta 0:00:00
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 24.1.2
    Uninstalling pip-24.1.2:
      Successfully uninstalled pip-24.1.2
Successfully installed pip-24.3.1
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: torchvision in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: torchaudio in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-pacl
Requirement already satisfied: typing-extensions>=4.8.0 in /usr/local/lib/pytl
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-pacl
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.10/dis
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.1
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/python
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/d
Collecting fsspec==2024.6.1
  Downloading fsspec-2024.6.1-py3-none-any.whl.metadata (11 kB)
Downloading fsspec-2024.6.1-py3-none-any.whl (177 kB)
Installing collected packages: fsspec
Installing collected packages: fsspec
  Attempting uninstall: fsspec
    Found existing installation: fsspec 2024.10.0
    Uninstalling fsspec-2024.10.0:
      Successfully uninstalled fsspec-2024.10.0
ERROR: pip's dependency resolver does not currently take into account all the
gcsfs 2024.10.0 requires fsspec==2024.10.0, but you have fsspec 2024.6.1 whicl
Successfully installed fsspec-2024.6.1
Collecting datasets==3.0.0
  Downloading datasets-3.0.0-py3-none-any.whl.metadata (19 kB)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-pacl
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-|
Requirement already satisfied: pyarrow>=15.0.0 in /usr/local/lib/python3.10/di
Collecting dill<0.3.9,>=0.3.0 (from datasets==3.0.0)
  Downloading dill-0.3.8-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: requests>=2.32.2 in /usr/local/lib/python3.10/(
Requirement already satisfied: tqdm>=4.66.3 in /usr/local/lib/python3.10/dist-
Collecting xxhash (from datasets==3.0.0)
  Downloading xxhash-3.5.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x8(
Collecting multiprocess (from datasets==3.0.0)
  Downloading multiprocess-0.70.17-py310-none-any.whl.metadata (7.2 kB)
Requirement already satisfied: fsspec<=2024.6.1,>=2023.1.0 in /usr/local/lib/|
Requirement already satisfied: aiohttp in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: huggingface-hub>=0.22.0 in /usr/local/lib/pytho
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-pa(
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-|
Requirement already satisfied: aiohappyeyeballs>=2.3.0 in /usr/local/lib/pytho
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.10/(
Requirement already satisfied: async-timeout<6.0,>=4.0 in /usr/local/lib/pytho
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dis
```

```
# Environment info
print("Python:".rjust(15), sys.version[0:6])
print("Pandas:".rjust(15), pd.__version__)
print("Transformers:".rjust(15), transformers.__version__)
print("Torch:".rjust(15), torch.__version__)
```

```
        Python: 3.10.1
        Pandas: 2.2.2
  Transformers: 4.46.3
         Torch: 2.5.1+cu121
```

```
# Mount Google Drive
drive.mount('/content/drive', force_remount=True)
path = '/content/drive/My Drive/CMPE 252 Project/cropped_plates/'
```

```
Mounted at /content/drive
```

```
# Dataset Preparation
file_names, texts = [], []
for file in os.listdir(path):
    if file.endswith(('.jpg', '.png')):
```

```
    if file.endswith(('.jpg', '.png')):
        file_names.append(file)
        texts.append(os.path.splitext(file)[0])

dataset = pd.DataFrame({'file_name': file_names, 'text': texts})
train_dataset, test_dataset = train_test_split(dataset, train_size=0.80, random_s
train_dataset.reset_index(drop=True, inplace=True)
test_dataset.reset_index(drop=True, inplace=True)
```

```
class License_Plates_OCR_Dataset(Dataset):
    def __init__(self, root_dir, df, processor, max_target_length=128):
        self.root_dir = root_dir
        self.df = df
        self.processor = processor
        self.max_target_length = max_target_length

    def __len__(self):
        return len(self.df)

    def __getitem__(self, idx):
        file_name = self.df['file_name'][idx]
        text = self.df['text'][idx]
        image = Image.open(self.root_dir + file_name).convert("RGB")
        pixel_values = self.processor(image, return_tensors="pt").pixel_values
        labels = self.processor.tokenizer(text, padding="max_length", max_length=
        labels = [label if label != self.processor.tokenizer.pad_token_id else -1
        return {"pixel_values": pixel_values.squeeze(), "labels": torch.tensor(la
```

Double-click (or enter) to edit

key: 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

Add token as git credential? (Y/n) n

```
!huggingface-cli login
```

```
    _|    _|  _|    _|    _|_|_|    _|_|_|  _|_|_|  _|      _|    _|_|_|
    _|    _|  _|    _|  _|        _|          _|    _|_|    _|  _|
    _|_|_|_|  _|    _|  _|  _|_|  _|  _|_|    _|    _|  _|  _|  _|  _|_|
    _|    _|  _|    _|  _|    _|  _|    _|    _|    _|    _|_|  _|    _|
    _|    _|    _|_|      _|_|_|    _|_|_|  _|_|_|  _|      _|    _|_|_|

    To log in, `huggingface_hub` requires a token generated from https://hugg
Enter your token (input will not be visible):
Add token as git credential? (Y/n) n
Token is valid (permission: fineGrained).
The token `testingLicensePlate` has been saved to /root/.cache/huggingface/st
Your token has been saved to /root/.cache/huggingface/token
```

```
Login successful.
The current active token is: `testingLicensePlate`
```

```
# Model Initialization
MODEL_CKPT = "microsoft/trocr-base-printed"
processor = TrOCRProcessor.from_pretrained(MODEL_CKPT)
train_ds = License_Plates_OCR_Dataset(path, train_dataset, processor)
test_ds = License_Plates_OCR_Dataset(path, test_dataset, processor)

model = VisionEncoderDecoderModel.from_pretrained(MODEL_CKPT)
model.config.decoder_start_token_id = processor.tokenizer.cls_token_id
model.config.pad_token_id = processor.tokenizer.pad_token_id
model.config.eos_token_id = processor.tokenizer.sep_token_id
model.config.max_length = 64
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_auth.py:94: Use
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tal
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access pu
  warnings.warn(
```

```
preprocessor_config.json: 100% 224/224 [00:00<00:00, 16.0kB/s]
tokenizer_config.json: 100% 1.12k/1.12k [00:00<00:00, 91.4kB/s]
vocab.json: 100% 899k/899k [00:00<00:00, 3.70MB/s]
merges.txt: 100% 456k/456k [00:00<00:00, 2.76MB/s]
special_tokens_map.json: 100% 772/772 [00:00<00:00, 60.6kB/s]
config.json: 100% 4.13k/4.13k [00:00<00:00, 311kB/s]
model.safetensors: 100% 1.33G/1.33G [00:06<00:00, 208MB/s]
```

```
Config of the encoder: <class 'transformers.models.vit.modeling_vit.ViTModel'>
  "attention_probs_dropout_prob": 0.0,
  "encoder_stride": 16,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.0,
  "hidden_size": 768,
  "image_size": 384,
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "layer_norm_eps": 1e-12,
  "model_type": "vit",
  "num_attention_heads": 12,
  "num_channels": 3,
  "num_hidden_layers": 12,
  "patch_size": 16,
  "qkv_bias": false,
  "transformers_version": "4.46.3"
}

Config of the decoder: <class 'transformers.models.trocr.modeling_trocr.TrOCRI
  "activation_dropout": 0.0,
  "activation_function": "gelu",
  "add_cross_attention": true,
  "attention_dropout": 0.0,
  "bos_token_id": 0,
  "classifier_dropout": 0.0,
  "cross_attention_hidden_size": 768,
  "d_model": 1024,
  "decoder_attention_heads": 16,
  "decoder_ffn_dim": 4096,
  "decoder_layerdrop": 0.0,
  "decoder_layers": 12,
  "decoder_start_token_id": 2,
  "dropout": 0.1,
  "eos_token_id": 2,
  "init_std": 0.02,
  "is_decoder": true,
  "layernorm_embedding": true,
  "max_position_embeddings": 512,
  "model_type": "trocr",
  "pad_token_id": 1,
  "scale_embedding": false,
  "transformers_version": "4.46.3",
  "use_cache": false,
  "use_learned_position_embeddings": true,
  "vocab_size": 50265
}
```

```
#Metrics
cer_metric = evaluate.load("cer")

#Beginning of Patch Attack
def overlay_patch(image_tensor, patch):
    patched_image = image_tensor.clone()
    patch_height, patch_width = patch.shape[1:]
    center_y = (patched_image.shape[1] - patch_height) // 2
    center_x = (patched_image.shape[2] - patch_width) // 2
    patched_image[:, center_y:center_y + patch_height, center_x:center_x + patch_wi
    return patched_image

def fast_gradient_sign_patch(image_tensor, true_label, processor, model, cer_metric
    #Computation set up
    device = model.device
    image_tensor = image_tensor.to(device)
    image_tensor = image_tensor.to(device)
    patch = torch.rand((3, *patch_size), device=device, requires_grad=True)
    optimizer = torch.optim.Adam([patch], lr=lr)
    # Tokenize true label
    true_ids = processor.tokenizer(true_label, return_tensors="pt").input_ids.to(de
    for epoch in range(epochs):
        optimizer.zero_grad()
        #Overlay patch on the image
        patched_image = overlay_patch(image_tensor, patch)
        #Forward pass through the model
        outputs = model(patched_image.unsqueeze(0), labels=true_ids)
        #Cross-entropy loss
        loss = outputs.loss
        loss.backward()
        optimizer.step()
        #Clamp patch values
        patch.data = torch.clamp(patch.data, 0, 1)
        #Computing CER
        predictions = processor.batch_decode(outputs.logits.argmax(dim=-1), skip_sp
        cer = cer_metric.compute(predictions=predictions, references=[true_label])
        print(f"Epoch {epoch + 1}/{epochs}, Loss: {loss.item():.4f}, CER: {cer:.4f}
    return patch
```

```
Downloading builder script: 100% ████████████████ 5.60k/5.60k [00:00<00:00, 417kB/
                                                   s]
```

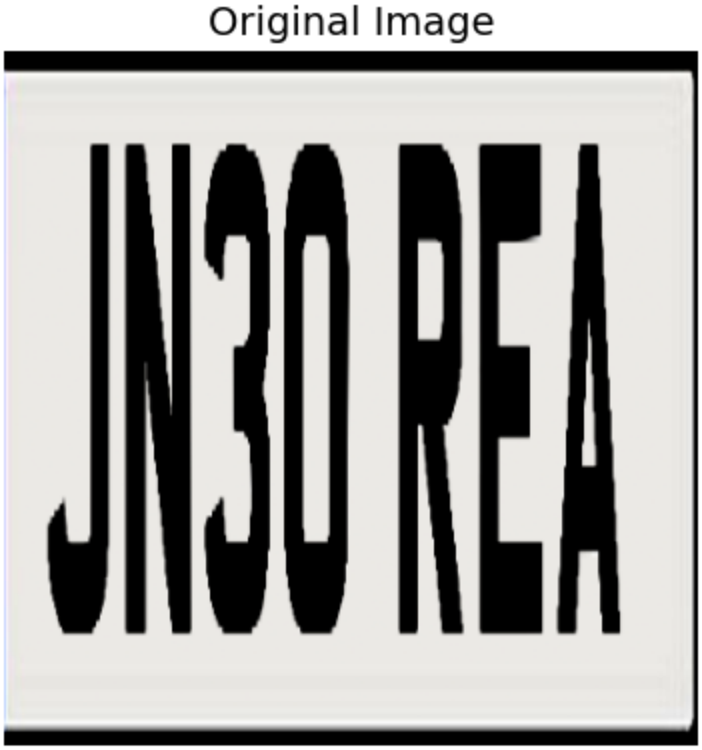
```
#Normalizing text by removing spaces
def normalize_text(text):
    return text.replace(" ", "")

#Evaluating test image
true_label = test_dataset.iloc[0]['text']
image_path = path + test_dataset.iloc[0]['file_name']
#Normalizing true label to remove spaces
normalized_true_label = normalize_text(true_label)
#Loading and preprocesing the image
image = Image.open(image_path).convert("RGB")
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
#Moving tensor to the same device
image_tensor = processor(image, return_tensors="pt").pixel_values[0].to(device)
#Ensuring model is on the same device
model = model.to(device)
#OCR on original image
original_image_tensor = image_tensor.unsqueeze(0)  # Add batch dimension
model.eval()  #Setting model to evaluation mode
with torch.no_grad():
    outputs = model.generate(original_image_tensor)
#Decode predictions
original_predictions = processor.batch_decode(outputs, skip_special_tokens=True)
normalized_predictions = [normalize_text(pred) for pred in original_predictions]
#Computing CER
```

```
#Computing CER
original_cer = cer_metric.compute(predictions=normalized_predictions, references=
print(f"True License Plate Label: {true_label}")
print(f"Predicted License Plate Label: {normalized_predictions}")
print(f"CER on Original Image: {original_cer:.4f}")

import matplotlib.pyplot as plt
def show_original_image(original_image_tensor, title="Original Image"):
    if len(original_image_tensor.shape) == 4:
        original_image_tensor = original_image_tensor.squeeze(0)
    #Tensor to NumPy array
    original_image_np = original_image_tensor.permute(1, 2, 0).cpu().detach().num
    #Plot
    plt.figure(figsize=(6, 4))
    plt.imshow(original_image_np)
    plt.title(title, fontsize=14)
    plt.axis("off")  # No axes
    plt.tight_layout()
    plt.show()
show_original_image(image_tensor, title="Original Image")
```

```
True License Plate Label: JN30REA
Predicted License Plate Label: ['JN30REA']
CER on Original Image: 0.0000
```



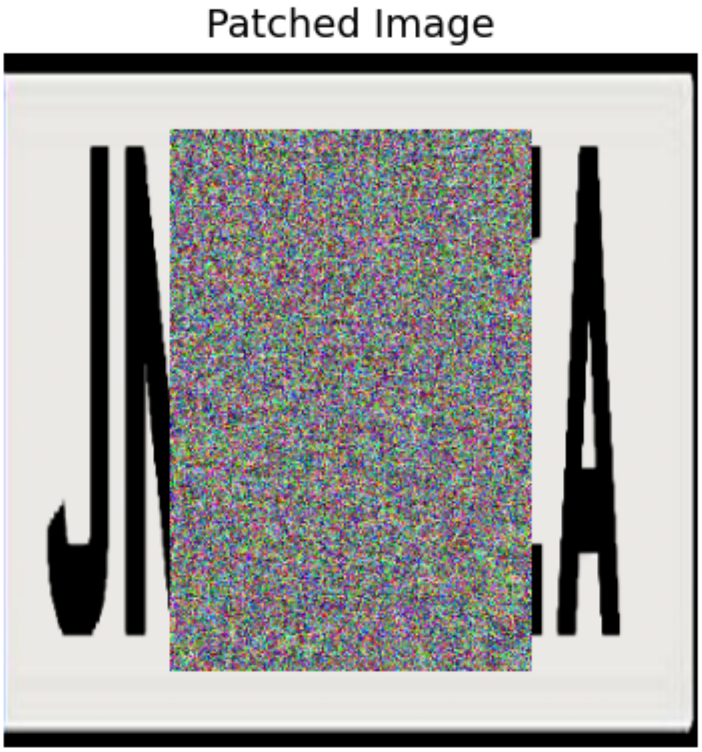
Big Patch Test

```
def normalize_text(text):
    return text.replace(" ", "")
```

```
#Evaluating test image
true_label = test_dataset.iloc[0]['text']
image_path = path + test_dataset.iloc[0]['file_name']
#Normalizing true label to remove spaces
normalized_true_label = normalize_text(true_label)
#Loading and preprocessing the image
image = Image.open(image_path).convert("RGB")
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
#Moving tensor to the same device
image_tensor = processor(image, return_tensors="pt").pixel_values[0].to(device)
#Ensuring model is on the same device
model = model.to(device)
#Adversarial Patch
adversarial_patch = fast_gradient_sign_patch(
    image_tensor, true_label, processor, model, cer_metric,
    patch_size=(300, 200), epochs=10, lr=0.01
)
#Applying the adversarial patch
patched_image = overlay_patch(image_tensor, adversarial_patch)
#Ensuring patched_image is on same device
patched_image = patched_image.unsqueeze(0).to(device)
#Evaluating model on patched image
model.eval()  #Setting model to evaluation mode
with torch.no_grad():
    outputs = model.generate(patched_image)
#Decode predictions
patched_predictions = processor.batch_decode(outputs, skip_special_tokens=True)
normalized_patched_predictions = [normalize_text(pred) for pred in patched_predic
#Computing CER
patched_cer = cer_metric.compute(predictions=normalized_patched_predictions, refe
print(f"True License Plate Label: {true_label}")
print(f"Predicted License Plate Label: {normalized_patched_predictions}")
print(f"CER on Patched Image: {patched_cer:.4f}")

import matplotlib.pyplot as plt
def show_patched_image(patched_image_tensor, title="Patched Image"):
    if len(patched_image_tensor.shape) == 4:
        patched_image_tensor = patched_image_tensor.squeeze(0)
    #Tensor to NumPy array
    patched_image_np = patched_image_tensor.permute(1, 2, 0).cpu().detach().numpy
    #Plot
    plt.figure(figsize=(6, 4))
    plt.imshow(patched_image_np)
    plt.title(title, fontsize=14)
    plt.axis("off")  #Remove axes
    plt.tight_layout()
    plt.show()
show_patched_image(patched_image, title="Patched Image")
```

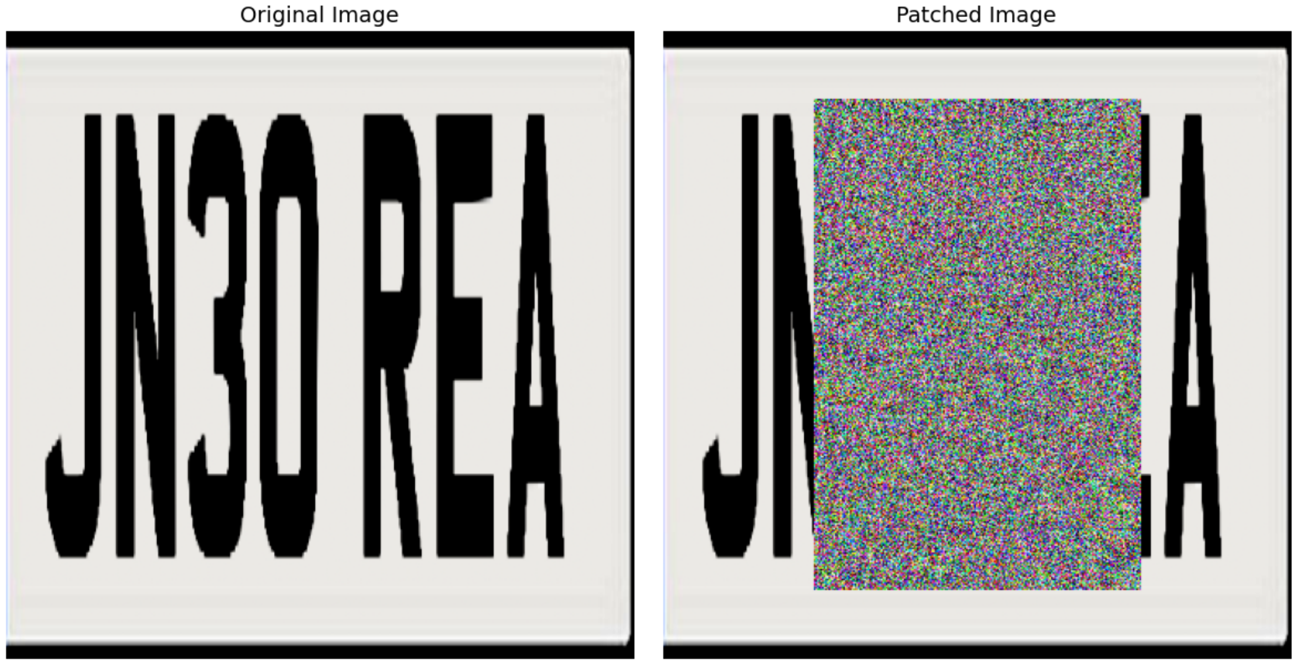
```
Epoch 1/10, Loss: 8.8039, CER: 0.7143
Epoch 2/10, Loss: 8.1672, CER: 0.8571
Epoch 3/10, Loss: 7.7497, CER: 0.8571
Epoch 4/10, Loss: 7.3782, CER: 0.7143
Epoch 5/10, Loss: 7.0035, CER: 0.7143
Epoch 6/10, Loss: 6.6491, CER: 0.5714
Epoch 7/10, Loss: 6.2217, CER: 0.5714
Epoch 8/10, Loss: 5.9306, CER: 0.7143
Epoch 9/10, Loss: 5.6402, CER: 0.5714
Epoch 10/10, Loss: 5.3764, CER: 0.5714
True License Plate Label: JN30REA
Predicted License Plate Label: ['JNJA']
CER on Patched Image: 0.5714
```



Showing Side by Side

```
def show_original_and_patched(original_image_tensor, patched_image_tensor, title1
    if len(patched_image_tensor.shape) == 4:
        patched_image_tensor = patched_image_tensor.squeeze(0)
    #Tensors to NumPy arrays
    original_image_np = original_image_tensor.permute(1, 2, 0).cpu().detach().num
    patched_image_np = patched_image_tensor.permute(1, 2, 0).cpu().detach().numpy
    #Plots
    fig, axes = plt.subplots(1, 2, figsize=(12, 6))
    #Original
    axes[0].imshow(original_image_np)
    axes[0].set_title(title1, fontsize=14)
    axes[0].axis("off")
    #Patched
    axes[1].imshow(patched_image_np)
    axes[1].imshow(patched_image_np)
    axes[1].set_title(title2, fontsize=14)
    axes[1].axis("off")
    plt.tight_layout()
    plt.show()

show_original_and_patched(image_tensor, patched_image)
```



Original Image

Patched Image

Average Test

```
import random
from torch.utils.data import DataLoader
def evaluate_license_plates(
    dataset, model, processor, cer_metric, patch_size=(300, 200), patch_epochs=10,
    #Sample 10 images from the dataset randomly
    sampled_indices = random.sample(range(len(dataset)), 10)
    sampled_data = [dataset[i] for i in sampled_indices]
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model = model.to(device)
    model.eval()
    patched_loss_sum, patched_cer_sum = 0, 0
    unpatched_loss_sum, unpatched_cer_sum = 0, 0
    for data in sampled_data:
        image_tensor = data["pixel_values"].to(device)
        #Taking out padding tokens before decoding
        true_label_ids = [id for id in data["labels"].tolist() if id != -100]
        true_label = processor.tokenizer.decode(true_label_ids, skip_special_tokens
        normalized_true_label = true_label.replace(" ", "")
        #Computing results for the unpatched image
        with torch.no_grad():
            outputs = model.generate(image_tensor.unsqueeze(0))
            predictions = processor.batch_decode(outputs, skip_special_tokens=True)
            normalized_predictions = [pred.replace(" ", "") for pred in predictions
            unpatched_cer = cer_metric.compute(
                predictions=normalized_predictions, references=[normalized_true_lab
            )
            #Calculating loss for unpatched image
            labels = processor.tokenizer(
                normalized_true_label, return_tensors="pt", padding=True, max_lengt
            ).input_ids.to(device)
            outputs = model(image_tensor.unsqueeze(0), labels=labels)
            unpatched_loss = outputs.loss.item()
            unpatched_loss_sum += unpatched_loss
            unpatched_cer_sum += unpatched_cer
        #Adversarial patch
        adversarial_patch = fast_gradient_sign_patch(
            image_tensor, normalized_true_label, processor, model, cer_metric,
            patch_size=patch_size, epochs=patch_epochs, lr=patch_lr
        )
        #Overlaying patch and evaluate
        patched_image = overlay_patch(image_tensor, adversarial_patch).unsqueeze(0)
        with torch.no_grad():
            outputs = model.generate(patched_image)
            predictions = processor.batch_decode(outputs, skip_special_tokens=True)
            normalized_predictions = [pred.replace(" ", "") for pred in predictions

            patched_cer = cer_metric.compute(
                predictions=normalized_predictions, references=[normalized_true_lab
            )
            #Calculating loss for patched image
            outputs = model(patched_image, labels=labels)
            patched_loss = outputs.loss.item()
            patched_loss_sum += patched_loss
            patched_cer_sum += patched_cer
    #Computing averages
    results = {
        "average_unpatched_loss": unpatched_loss_sum / 10,
        "average_unpatched_cer": unpatched_cer_sum / 10,
        "average_patched_loss": patched_loss_sum / 10,
        "average_patched_cer": patched_cer_sum / 10,
    }
    }
    return results
results = evaluate_license_plates(
    dataset=test_ds,
    model=model,
    processor=processor,
    cer_metric=cer_metric,
    patch_size=(300, 200),
    patch_epochs=10,
    patch_lr=0.01,
)
print("Evaluation Results:")
print(f"Average Unpatched Loss: {results['average_unpatched_loss']:.4f}")
print(f"Average Unpatched CER: {results['average_unpatched_cer']:.4f}")
print(f"Average Patched Loss: {results['average_patched_loss']:.4f}")
print(f"Average Patched CER: {results['average_patched_cer']:.4f}")
```

```
Epoch 1/10, Loss: 12.9482, CER: 1.4286
Epoch 2/10, Loss: 11.8461, CER: 1.4286
Epoch 3/10, Loss: 11.5001, CER: 0.8571
Epoch 4/10, Loss: 11.0169, CER: 1.0000
Epoch 5/10, Loss: 10.4940, CER: 1.0000
Epoch 6/10, Loss: 10.2354, CER: 1.0000
Epoch 7/10, Loss: 9.8858, CER: 1.0000
Epoch 8/10, Loss: 9.5758, CER: 1.0000
Epoch 9/10, Loss: 9.3143, CER: 1.0000
Epoch 10/10, Loss: 9.0356, CER: 1.0000
Epoch 1/10, Loss: 10.7484, CER: 0.8571
Epoch 2/10, Loss: 10.3589, CER: 0.7143
Epoch 3/10, Loss: 10.0162, CER: 0.8571
Epoch 4/10, Loss: 9.7111, CER: 0.8571
Epoch 5/10, Loss: 9.4011, CER: 0.8571
Epoch 6/10, Loss: 9.1245, CER: 0.7143
Epoch 7/10, Loss: 8.8336, CER: 0.8571
Epoch 8/10, Loss: 8.5134, CER: 0.7143
Epoch 9/10, Loss: 8.1494, CER: 0.7143
Epoch 10/10, Loss: 7.7863, CER: 0.7143
Epoch 1/10, Loss: 10.1406, CER: 0.8000
Epoch 2/10, Loss: 9.7473, CER: 1.1000
Epoch 3/10, Loss: 9.4276, CER: 1.0000
Epoch 4/10, Loss: 9.1135, CER: 1.0000
Epoch 5/10, Loss: 8.7465, CER: 1.1000
Epoch 6/10, Loss: 8.4170, CER: 1.3000
Epoch 7/10, Loss: 8.2221, CER: 1.1000
Epoch 8/10, Loss: 7.9302, CER: 1.1000
Epoch 9/10, Loss: 7.6454, CER: 0.7000
Epoch 10/10, Loss: 7.4467, CER: 0.7000
Epoch 1/10, Loss: 14.7177, CER: 0.7143
Epoch 2/10, Loss: 14.2191, CER: 0.7143
Epoch 3/10, Loss: 13.7471, CER: 0.7143
Epoch 4/10, Loss: 13.2894, CER: 0.7143
Epoch 5/10, Loss: 12.8679, CER: 1.0000
Epoch 6/10, Loss: 12.4475, CER: 1.0000
Epoch 7/10, Loss: 12.0261, CER: 1.0000
Epoch 8/10, Loss: 11.5676, CER: 1.0000
Epoch 8/10, Loss: 11.5676, CER: 1.0000
Epoch 9/10, Loss: 11.2032, CER: 1.0000
Epoch 10/10, Loss: 10.8408, CER: 1.0000
Epoch 1/10, Loss: 11.6774, CER: 0.7143
Epoch 2/10, Loss: 11.2447, CER: 0.5714
Epoch 3/10, Loss: 10.8362, CER: 0.5714
Epoch 4/10, Loss: 10.3961, CER: 0.5714
Epoch 5/10, Loss: 9.9598, CER: 0.7143
Epoch 6/10, Loss: 9.6459, CER: 0.8571
Epoch 7/10, Loss: 9.4258, CER: 0.8571
Epoch 8/10, Loss: 9.1212, CER: 0.8571
Epoch 9/10, Loss: 8.8588, CER: 0.8571
Epoch 10/10, Loss: 8.6077, CER: 0.8571
Epoch 1/10, Loss: 10.9899, CER: 0.8571
Epoch 2/10, Loss: 10.0310, CER: 0.7143
Epoch 3/10, Loss: 9.1487, CER: 0.8571
Epoch 4/10, Loss: 8.6085, CER: 0.8571
Epoch 5/10, Loss: 8.1950, CER: 0.8571
Epoch 6/10, Loss: 7.7470, CER: 0.8571
Epoch 7/10, Loss: 7.3084, CER: 0.8571
Epoch 8/10, Loss: 6.9092, CER: 0.8571
```

Small Patch Test

Small Patch Test

```
def normalize_text(text):
    return text.replace(" ", "")

#Evaluating test image
true_label = test_dataset.iloc[0]['text']
image_path = path + test_dataset.iloc[0]['file_name']
#Normalizing true label to remove spaces
normalized_true_label = normalize_text(true_label)
#Loading and preprocessing the image
image = Image.open(image_path).convert("RGB")
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
#Moving tensor to the same device
image_tensor = processor(image, return_tensors="pt").pixel_values[0].to(device)
#Ensuring model is on the same device
model = model.to(device)
#Adversarial Patch
adversarial_patch = fast_gradient_sign_patch(
    image_tensor, true_label, processor, model, cer_metric,
    patch_size=(60, 60), epochs=10, lr=0.01   #small patch test
)
#Applying the adversarial patch
patched_image = overlay_patch(image_tensor, adversarial_patch)
#Ensuring patched_image is on same device
patched_image = patched_image.unsqueeze(0).to(device)
#Evaluating model on patched image
```

```
model.eval()  #Setting model to evaluation mode
with torch.no_grad():
    outputs = model.generate(patched_image)
#Decode predictions
patched_predictions = processor.batch_decode(outputs, skip_special_tokens=True)
normalized_patched_predictions = [normalize_text(pred) for pred in patched_predicti
#Computing CER
patched_cer = cer_metric.compute(predictions=normalized_patched_predictions, refere
print(f"True License Plate Label: {true_label}")
print(f"Predicted License Plate Label: {normalized_patched_predictions}")
print(f"CER on Patched Image: {patched_cer:.4f}")

import matplotlib.pyplot as plt
def show_patched_image(patched_image_tensor, title="Patched Image"):
    if len(patched_image_tensor.shape) == 4:
        patched_image_tensor = patched_image_tensor.squeeze(0)
    #Tensor to NumPy array
    patched_image_np = patched_image_tensor.permute(1, 2, 0).cpu().detach().numpy()
    #Plot
    plt.figure(figsize=(6, 4))
    plt.imshow(patched_image_np)
    plt.title(title, fontsize=14)
    plt.axis("off")  #Remove axes
    plt.tight_layout()
    plt.show()
show_patched_image(patched_image, title="Patched Image")
```

```
Epoch 1/10, Loss: 8.6447, CER: 0.5714
Epoch 2/10, Loss: 8.5114, CER: 0.5714
Epoch 3/10, Loss: 8.4280, CER: 0.4286
Epoch 4/10, Loss: 8.3803, CER: 0.4286
Epoch 5/10, Loss: 8.3380, CER: 0.4286
Epoch 6/10, Loss: 8.2907, CER: 0.4286
Epoch 7/10, Loss: 8.2406, CER: 0.4286
Epoch 8/10, Loss: 8.2017, CER: 0.4286
Epoch 9/10, Loss: 8.1617, CER: 0.4286
Epoch 10/10, Loss: 8.1230, CER: 0.4286
True License Plate Label: JN30REA
Predicted License Plate Label: ['JN30REA']
CER on Patched Image: 0.0000
```