National College of Ireland

ScreenGuard: AI-Powered Image Authenticity Detection for Financial Transaction Fraud Prevention

MSc Research Project
MSc in Cyber Security (MSCCYB1_JAN25)

Satish Reddy Navuluri
Student ID: X23436913

School of Computing
National College of Ireland

Supervisor: Raza Ul Mustafa

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Satish Reddy Navuluri
Student ID:	X23436913
Programme:	Master of Science In Cyber Security 2025-2026 **Year:**
Module:	MSc (Research) Practicum/Internship Part 2 Information
Lecturer:	Raza Ul Mustafa
Submission Due Date:	28-01-2026
Project Title:	ScreenGuard: AI-Powered Image Authenticity Detection for Financial Transaction Fraud Prevention
Word Count:	664 **Page Count:** 09

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	N.Satish Reddy
Date:	28-01-2026

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

ScreenGuard: AI-Powered Image Authenticity Detection for Financial Transaction Fraud Prevention

Satish Reddy Navuluri
Student ID:X23436913

1 Introduction

This configuration manual provides instructions to set up, configure, and run the Document Tampering Detection System developed for the MSc project. While the system is framed toward financial transaction imagery, it is evaluated using a publicly available document-tampering dataset and therefore represents a reproducible proof-of-concept rather than a production-ready fintech deployment. The manual enables users to recreate the experimental environment, train the model if required, and deploy the Gradio-based demonstration interface.

2 System Requirements

2.1 Hardware Requirements

- Standard CPU (Intel/AMD)
- NVIDIA GPU with minimum 4GB VRAM for faster training
- At least 8GB RAM
- Storage: ~5GB free space for dataset, outputs and models

2.2 Software Requirements

- Operating System: Windows / macOS / Linux
- Python 3.10+
- Required Libraries:
    - TensorFlow
    - OpenCV
    - NumPy
    - Pillow
    - Pytesseract
    - Gradio
    - scikit-learn
- Tesseract OCR must be installed:

- Linux: sudo apt-get install tesseract-ocr
- Windows: Install from official Tesseract website, then add to PATH

3 Installation Steps

Step 1: Install Python and Dependencies

Install required libraries using:

pip install tensorflow opencv-python pytesseract pillow numpy scikit-learn gradio tqdm

Step 2: Install Tesseract OCR

- Linux:

sudo apt-get install tesseract-ocr

- Windows:

Install the Tesseract installer and ensure its installation folder is added to the PATH.

Step 3: Verify Installation

Run the following:

python -c "import tensorflow as tf; print(tf.__version__)"

python -c "import pytesseract; print(pytesseract.get_tesseract_version())"

4 Configuration Settings

The system parameters can be customised in a single configuration block or file. Common values include:

```
DATASET_ROOT = pathlib.Path("/content/drive/MyDrive/Test/images/training")

CLASS_NAMES = ["original", "forged"]  # 0 = original, 1 = forged
IMAGE_EXTS = {".jpg", ".jpeg", ".png", ".bmp", ".tif", ".tiff"}

IMG_HEIGHT = 224
IMG_WIDTH = 224
BATCH_SIZE = 16
EPOCHS = 20
RANDOM_SEED = 42
```

Figure 1: Configuration Settings

Only change the dataset path and weight path if your folder structure differs.

5 Preprocessing & Feature Extraction

The project uses multiple modalities, and each requires preprocessing:

RGB Image Processing

- Converts images to RGB format
- Resizes to 224×224 pixels
- Normalises pixel values

```
def load_rgb_image(path: pathlib.Path) -> np.ndarray:
    """Load RGB image, resize to fixed size, return uint8 array (H, W, 3)."""
    img = Image.open(path).convert("RGB")
    img = img.resize((IMG_WIDTH, IMG_HEIGHT))
    arr = np.array(img, dtype=np.uint8)
    return arr
```

Figure 2: RGB Image Processing

Error Level Analysis (ELA)

- Saves the image temporarily as JPEG
- Computes the pixel-level difference
- Enhances subtle compression artefacts that indicate tampering

```python
def compute_ela_image(path: pathlib.Path, quality: int = 95) -> np.ndarray:
    """
    Compute Error Level Analysis (ELA) image.
    Returns float32 array (H, W, 3) scaled to [0, 1].
    """
    image = Image.open(path).convert("RGB")
    image = image.resize((IMG_WIDTH, IMG_HEIGHT))

    # Save to a temporary JPEG file
    fd, tmp_path = tempfile.mkstemp(suffix=".jpg")
    os.close(fd)
    try:
        image.save(tmp_path, "JPEG", quality=quality)
        compressed = Image.open(tmp_path)

        ela_image = ImageChops.difference(image, compressed)

        extrema = ela_image.getextrema()
        max_diff = max([ex[1] for ex in extrema])
        if max_diff == 0:
            max_diff = 1
        scale = 255.0 / max_diff

        ela_image = Image.eval(ela_image, lambda px: int(px * scale))
        ela_array = np.array(ela_image, dtype=np.float32) / 255.0  # (H, W, 3)
        return ela_array
    finally:
        if os.path.exists(tmp_path):
            os.remove(tmp_path)
```

Figure 3: Error Level Analysis (ELA)

Edge Detection (Sobel Filter)

- Calculates gradients along x and y directions
- Produces an edge-magnitude map

```python
def compute_edge_map(path: pathlib.Path) -> np.ndarray:
    """
    Compute edge magnitude map using Sobel filters.
    Returns float32 array (H, W, 1) scaled to [0, 1].
    """
    img = cv2.imread(str(path), cv2.IMREAD_GRAYSCALE)
    if img is None:
        # Fallback: blank image, shouldn't happen if paths are valid
        img = np.zeros((IMG_HEIGHT, IMG_WIDTH), dtype=np.uint8)
    else:
        img = cv2.resize(img, (IMG_WIDTH, IMG_HEIGHT))

    sobelx = cv2.Sobel(img, cv2.CV_32F, 1, 0, ksize=3)
    sobely = cv2.Sobel(img, cv2.CV_32F, 0, 1, ksize=3)
    mag = np.sqrt(sobelx**2 + sobely**2)
    mag = mag / (mag.max() + 1e-8)
    mag = mag.astype(np.float32)
    mag = np.expand_dims(mag, axis=-1)  # (H, W, 1)
    return mag
```

Figure 4: Edge Detection

Metadata Features

Extracted fields include:

- Image width and height
- File size
- Aspect ratio
- Mean and standard deviation of pixel intensities
- DPI values

```python
def extract_metadata_features(path: pathlib.Path) -> np.ndarray:
    """
    Extract simple metadata features:
    [width, height, aspect_ratio, file_size_kb, mean_intensity, std_intensity, dpi_x, dpi_y]
    Returns float32 array of shape (8,).
    """
    img = Image.open(path).convert("L")
    w, h = img.size

    # File size in KB
    size_kb = path.stat().st_size / 1024.0

    # Intensity stats
    arr = np.array(img, dtype=np.float32) / 255.0
    mean_int = float(arr.mean())
    std_int = float(arr.std())
    aspect_ratio = w / (h + 1e-6)

    dpi_x = 0.0
    dpi_y = 0.0
    info = img.info
    if "dpi" in info:
        dpi = info["dpi"]
        if isinstance(dpi, (tuple, list)) and len(dpi) >= 2:
            dpi_x, dpi_y = float(dpi[0]), float(dpi[1])
        elif isinstance(dpi, (int, float)):
            dpi_x = float(dpi)
            dpi_y = float(dpi)

    return np.array(
        [w, h, aspect_ratio, size_kb, mean_int, std_int, dpi_x, dpi_y],
        dtype=np.float32,
    )
```

Figure 5: MetaData Features

OCR Text Extraction

- Converts image to grayscale
- Upscales for improved OCR accuracy
- Extracts text using Tesseract
- Returns [NO_TEXT] if no text is detected

```python
def extract_ocr_text(path: pathlib.Path) -> str:
    """
    Extract OCR text using Tesseract.
    Returns a single-line string; if nothing found, returns "[NO_TEXT]".
    """
    img = Image.open(path).convert("L")
    # Upscale a bit to help OCR
    img = img.resize((IMG_WIDTH * 2, IMG_HEIGHT * 2))
    text = pytesseract.image_to_string(img, lang="eng")
    text = text.replace("\n", " ").strip()
    if text == "":
        text = "[NO_TEXT]"
    return text
```

Figure 6: OCR Text Extraction

6 Model Setup

The model uses **four branches**:

1. **Image CNN branch** – extracts visual features
2. **Forensic CNN branch** – processes ELA + edge maps
3. **OCR branch** – TF-IDF vectorisation of extracted text
4. **Metadata branch** – numerical document attributes

These branches are fused and produce a final probability indicating the likelihood of tampering.

To load the saved model weights:

```
WEIGHTS_PATH = "/content/drive/MyDrive/document-tampering-detection-main/multimodal_tampering.weights.h5"

# after you define `model = ...` with the same layers:
model.load_weights(WEIGHTS_PATH)
print("Weights loaded.")
```

Figure 7: Load Model Weights

7 Training Instructions

Steps to Train the Model

1. Ensure dataset folders are correctly structured.
2. Open the training script or notebook.
3. Adjust configuration parameters if needed.
4. Train the model using:

```
# ==============================
# 8. Training
# ==============================
callbacks = [
    keras.callbacks.EarlyStopping(
        monitor="val_auc",
        mode="max",
        patience=5,
        restore_best_weights=True,
    )
]

history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=EPOCHS,
    callbacks=callbacks,
)
```

Figure 8: Training

Training Notes

- Training may take 10–30 minutes depending on hardware.
- Model checkpoints can be saved automatically after each epoch.

8 Evaluation Process

The evaluation stage measures model performance using:

- Accuracy
- AUC (Area Under Curve)
- Confusion Matrix
- Classification Report
- ROC and Precision–Recall curves

```
# ==============================
# 9. Evaluation
# ==============================
print("Validation performance:")
model.evaluate(val_ds)
```

```
Validation performance:
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step - accuracy: 0.7290 - auc: 0.8820 - loss: 0.4349
  [0.4511047601699829, 0.7160493731498718, 0.8662790656089783]
```

Figure 9: Evaluation

Visual graphs (loss, accuracy and ROC) are saved to the outputs/ folder.

9 Prediction

To classify a new document image:

```
test_image = "/content/1.jpeg"

prob, label = predict_tampering_with_label(model, test_image)

print(f"Tampering probability: {prob:.3f}")
print(f"Predicted label: {label}")
```

```
Probability(forged) = 0.415 | threshold = 0.50 | predicted: original
Tampering probability: 0.415
Predicted label: original
```

Figure 10: Prediction

- Output label: **Original** or **Forged**
- Probability closer to 1 = more likely forged

The threshold can be modified to adjust sensitivity.

10 Application Deployment (Gradio)

The application provides an intuitive interface for uploading documents.

To launch the Gradio app:

demo.launch()

The interface displays:

- Predicted class
- Probability of forgery

The user only needs to upload an image; the system handles all pre-processing.

11 Troubleshooting

Common Issues

- **Tesseract not installed:** Install Tesseract and confirm it is added to PATH.
- **Dataset path error:** Ensure folder structure:

dataset/original/
dataset/forged/

- **Image reading errors:** Check for corrupted or unsupported image formats.
- **GPU memory issues:** Lower batch size to 8 or 4.

References

Abdulqader, M.F., Dawod, A.Y. and Ablahd, A.Z., 2023. Detection of tamper forgery image in security digital mage. *Measurement: Sensors*, *27*, p.100746. Available on: https://doi.org/10.1016/j.measen.2023.100746

Ferreira, S., Antunes, M. and Correia, M.E., 2021. Exposing manipulated photos and videos in digital forensics analysis. *Journal of imaging*, *7*(7), p.102. Available on: https://doi.org/10.3390/jimaging7070102

Laxman, V., Ramesh, N., Prakash, S.K.J. and Aluvala, R., 2024. Emerging threats in digital payment and financial crime: A bibliometric review. *Journal of Digital Economy*, *3*, pp.205-222. Available on: https://doi.org/10.1016/j.jdec.2025.04.002

Saxena, A. and Tripathi, S.N., 2021. Exploring the security risks and safety measures of mobile payments in fintech environment in India. *International Journal of Management*, *12*(2), pp.408-417. Available on: https://iaeme.com/MasterAdmin/Journal_uploads/IJM/VOLUME_12_ISSUE_2/IJM_12_02_041.pdf