National College of Ireland

Configuration Manual

MSc Research Project
MSc Cybersecurity

Sharan Nagaraj Kumar
Student ID: x23269839

School of Computing
National College of Ireland

Supervisor: Dr. Michael Prior

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Sharan Nagaraj Kumar
Student ID:	x23269839
Programme:	MSc Cybersecurity
Year:	2024-25
Module:	MSc Research Project
Supervisor:	Dr. Michael Prior
Submission Due Date:	11/08/2025
Project Title:	From Deception to Detection: A Cybersecurity-Driven Approach to Deepfake Identification
Word Count:	806
Page Count:	9

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:	Sharan Nagaraj Kumar
Date:	3rd August 2025

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):	

Configuration Manual

Sharan Nagaraj Kumar
x23269839

1 Introduction

This configuration manual describes the process by which the research work was carried out from start to finish. This report is intended to explain about pre-requisites and technical aspects of the project. An overview of dataset creation, as we created a custom hybrid dataset using three benchmarked datasets and a web collection. Then follows the data pre-processing and augmentation steps to develop a uniform input to the model. Section 4 defines the model algorithms, followed by the hybrid model definition. The definition of the hybrid model is mentioned in Section 5, and screenshots of results are attached in Section 6. In addition to that, Robustness evaluation metrics are mentioned to measure the model's performance in the real world.

2 Dataset Creation and Setup

Initially, the ZIP files are extracted and saved in Google Drive for processing.

```
# Define zip file path and extract location
zip_path = "/content/Celeb DF (v2).zip"
extract_path = "/content/Celeb_DF_Unzipped"

# Create directory if it doesn't exist
os.makedirs(extract_path, exist_ok=True)

# Extract zip file
with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_path)

print(f"Extraction complete. Files are in: {extract_path}")
```

```
Extraction complete. Files are in: /content/Celeb_DF_Unzipped
```

Figure 1: Creating directory in Google Drive to store dataset

The below screenshot explains the pre-processing techniques applied to each image in the dataset and the labelling of images.

```python
# Define paths
ai_path = '/content/drive/MyDrive/Dataset/whole/fake-dataset'
real_path = '/content/drive/MyDrive/Dataset/whole/real-dataset'

# Parameters
img_size = (224, 224)
data = []
labels = []

# Load and preprocess images
def load_images_from_folder(folder_path, label):
    for filename in os.listdir(folder_path):
        img_path = os.path.join(folder_path, filename)
        try:
            img = Image.open(img_path).convert('RGB') # Converts to RGB
            img = img.resize(img_size) # resize the image
            img_array = np.array(img) / 255.0  # normalize pixel values to [0, 1]
            data.append(img_array)
            labels.append(label)
        except Exception as e:
            print(f"Error loading image {img_path}: {e}")

# Load AI-generated and real images
load_images_from_folder(ai_path, label=1)   # Label 1 for AI-generated
load_images_from_folder(real_path, label=0) # Label 0 for Real
```

Figure 2: Dataset Preparation and Labeling

After applying all processing, the images are converted into NumPy and stored in Google Drive.

```python
# Convert to numpy arrays
X = np.array(data)
y = np.array(labels)
np.save('/content/drive/MyDrive/Dataset/X.npy', X)
np.save('/content/drive/MyDrive/Dataset/y.npy', y)
# Print dataset info
print(f"Total samples: {len(X)}")
print(f"Shape of image data: {X.shape}")
print(f"Shape of labels: {y.shape}")
```

```
/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1043: UserWarn
  warnings.warn(
Total samples: 13185
Shape of image data: (13185, 224, 224, 3)
Shape of labels: (13185,)
```

Figure 3: Total Samples in the Dataset

3 Dataset Augmentation and Split

The transformers are defined for image processing.

```
])
train_transform = transforms.Compose([
    transforms.RandomResizedCrop(256, scale=(0.7, 1.0)),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomRotation(20),
    transforms.ColorJitter(brightness=0.4, contrast=0.4, saturation=0.4, hue=0.1),
    transforms.RandomGrayscale(p=0.2),
    transforms.GaussianBlur(kernel_size=3),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])
'''val_transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])'''
val_transform = transforms.Compose([
    transforms.Resize((256,256)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])
```

Figure 4: Transformers for Image Processing

Finally, the dataset is split into 70,15,15 for training, testing, and validation, respectively.

```
# Create full dataset with no transform (will assign per split)
full_dataset = AIDataset(root_ai=path_ai, root_real=path_real, transform=None)

# Split into train (70%), val (15%), test (15%)
train_size = int(0.7 * len(full_dataset))
val_size = int(0.15 * len(full_dataset))
test_size = len(full_dataset) - train_size - val_size
train_data, val_data, test_data = random_split(full_dataset, [train_size, val_size, test_size])

# Assign transforms to each split
train_data.dataset.transform = train_transform
val_data.dataset.transform = val_transform
test_data.dataset.transform = val_transform

# Create DataLoaders
batch_size = 32
train_loader = DataLoader(train_data, batch_size=batch_size, shuffle=True)
val_loader = DataLoader(val_data, batch_size=batch_size, shuffle=False)
test_loader = DataLoader(test_data, batch_size=batch_size, shuffle=False)
```

Figure 5: Data Split up for Training

```
Total samples: 13185
Train samples: 9229
Validation samples: 1977
Test samples: 1979
Shape of image batch: torch.Size([32, 3, 256, 256])
Shape of label batch: torch.Size([32])
```

Figure 6: Number of Samples in each Class set

4 Model Defenition

4.1 Convolutional Neural Network

CNN is used as the backbone of the model. This CNN model contains 4 layers. The model is referred to and developed with reference to TensorFlow (TensorFlow; 2025).

```python
# Define a simple CNN for binary classification
class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()

        self.conv_layers = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Dropout(0.25),

            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Dropout(0.25),

            nn.Conv2d(64, 128, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Dropout(0.25),

            nn.Conv2d(128, 256, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Dropout(0.25),
        )
```

Figure 7: Model Configuration of CNN

```
SimpleCNN(
  (conv_layers): Sequential(
    (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Dropout(p=0.25, inplace=False)
    (4): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (5): ReLU()
    (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (7): Dropout(p=0.25, inplace=False)
    (8): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU()
    (10): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (11): Dropout(p=0.25, inplace=False)
    (12): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU()
    (14): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (15): Dropout(p=0.25, inplace=False)
  )
  (fc_layers): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=65536, out_features=512, bias=True)
    (2): ReLU()
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=512, out_features=2, bias=True)
  )
)
```

Figure 8: Model configuration results

4.2 Vision Transformer

Pre-trained model imported and modified for our use case HuggingFace (2025).

VIT Model

```python
# Load pretrained ViT model
vit_model = timm.create_model('vit_base_patch16_224', pretrained=True)

# Modify the classifier head for binary classification
vit_model.head = nn.Linear(vit_model.head.in_features, 2)

# Freeze first few layers (patch embedding + positional embedding + early blocks)
for name, param in vit_model.named_parameters():
    if 'blocks.0' in name or 'blocks.1' in name or 'patch_embed' in name or 'pos_embed' in name:
        param.requires_grad = False

# Move model to device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
vit_model.to(device)

# Define loss function and optimizer
criterion_vit = nn.CrossEntropyLoss()
optimizer_vit = torch.optim.AdamW(vit_model.parameters(), lr=3e-5)

# Sanity check: print number of trainable parameters
trainable_params = sum(p.numel() for p in vit_model.parameters() if p.requires_grad)
print(f"Trainable parameters: {trainable_params}")
```

Figure 9: Model Configuration of ViT

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/setti
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
model.safetensors: 100% 346M/346M [00:00<00:00, 435MB/s]
Trainable parameters: 56706818
```

Figure 10: Model Configuration results

4.3 EfficientNet

Pre-trained model developed by Google trained based on ImageNet-1k (Face; 2025).

EfficientNet Model

```python
# Load pretrained EfficientNet-B0 model
effnet_model = timm.create_model('efficientnet_b0', pretrained=True)

# Replace the classifier head for binary classification
effnet_model.classifier = nn.Linear(effnet_model.classifier.in_features, 2)

# Move model to device
effnet_model.to(device)

# Define loss function and optimizer
criterion_effnet = nn.CrossEntropyLoss()
optimizer_effnet = torch.optim.Adam(effnet_model.parameters(), lr=1e-4)

# Sanity check: print number of trainable parameters
trainable_params_effnet = sum(p.numel() for p in effnet_model.parameters() if p.requires_grad)
print(f"Trainable parameters: {trainable_params_effnet}")
```

```
model.safetensors: 100% 21.4M/21.4M [00:00<00:00, 339MB/s]
Trainable parameters: 4010110
```

Figure 11: Model Configuration of EfficientNet and its results

5 Hybrid Model Configuration

The code below uses the model definition to create a hybrid model, which utilizes the first CNN, EfficientNet, and ViT.

Hybrid Model Construction (CNN + ViT + EfficientNet)

```python
# Custom Hybrid Model
class HybridModel(nn.Module):
    def __init__(self):
        super(HybridModel, self).__init__()

        # CNN Backbone (Simple CNN)
        self.cnn = nn.Sequential(
            nn.Conv2d(3, 16, 3, padding=1), nn.ReLU(), nn.MaxPool2d(2),  # 112x112
            nn.Conv2d(16, 32, 3, padding=1), nn.ReLU(), nn.MaxPool2d(2),  # 56x56
            nn.Conv2d(32, 64, 3, padding=1), nn.ReLU(), nn.AdaptiveAvgPool2d((1, 1))
        )
        self.cnn_fc = nn.Linear(64, 128)

        # EfficientNet-B0 (timm)
        self.effnet = timm.create_model('efficientnet_b0', pretrained=True, num_classes=0)
        self.effnet_fc = nn.Linear(self.effnet.num_features, 128)

        # Vision Transformer (HuggingFace)
        self.vit = ViTModel.from_pretrained('google/vit-base-patch16-224-in21k')
        self.vit_fc = nn.Linear(self.vit.config.hidden_size, 128)

        # Final Classifier after concatenation
        self.classifier = nn.Sequential(
            nn.Linear(128 * 3, 128),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(128, 2)  # Binary classification
        )
```

Figure 12: Hybird Model Defenition

```python
    def forward(self, x):
        # CNN path
        cnn_feat = self.cnn(x)
        cnn_feat = cnn_feat.view(cnn_feat.size(0), -1)
        cnn_feat = self.cnn_fc(cnn_feat)

        # EfficientNet path
        eff_feat = self.effnet(x)
        eff_feat = self.effnet_fc(eff_feat)

        # ViT path
        vit_inputs = {
            'pixel_values': x  # Assumes input has been transformed to ViT format
        }
        vit_out = self.vit(**vit_inputs)
        vit_feat = self.vit_fc(vit_out.last_hidden_state[:, 0, :])  # CLS token

        # Concatenate all features
        fused = torch.cat((cnn_feat, eff_feat, vit_feat), dim=1)
        output = self.classifier(fused)
        return output

# Instantiate and move to device
hybrid_model = HybridModel().to(device)

# Loss and optimizer
criterion_hybrid = nn.CrossEntropyLoss()
optimizer_hybrid = torch.optim.Adam(hybrid_model.parameters(), lr=1e-4)
```

Figure 13: Hybird Model Defenition

6 Model Results

The model is trained with 50 epochs, a batch size of 64. In every round, the best model is chosen and saved.

```python
# Setup training parameters
batch_size = 64
epochs = 50

# Use the correct variable names from earlier split
train_loader = DataLoader(train_data, batch_size=batch_size, shuffle=True)
val_loader = DataLoader(val_data, batch_size=batch_size)

# Choose and prepare model
current_model = model
criterion_current = nn.CrossEntropyLoss()
optimizer_current = torch.optim.Adam(current_model.parameters(), lr=1e-4)
scheduler_current = torch.optim.lr_scheduler.ReduceLROnPlateau(
    optimizer_current, mode='min', factor=0.5, patience=3, verbose=True
)

torch.save(hybrid_model.state_dict(), "/content/drive/MyDrive/Dataset/hybrid_model_weights.pth")


# Train model
trained_model = train_model(
    current_model,
    train_loader,
    val_loader,
    criterion_current,
    optimizer_current,
    scheduler_current,
    device,
    epochs=epochs
)
```

Figure 14: Model parameters for Training

7 Grad-CAM Heatmap Configuration and Result

The grad-CAM heatmap helps to visualize the areas in which the hybrid focuses to classify the images. It helps to fine-tune the model based on results.

```python
# Grad-CAM heatmap - Use the original image for visualization purposes
heatmap = generate_gradcam(hybrid_model, img_resized, target_class=pred_class, device=device, layer_name='cnn')

# Visualize - Use the original image for visualization purposes
show_gradcam_on_image(img, heatmap)
data_iter = iter(val_loader)
images, labels = next(data_iter)

img = images[0].unsqueeze(0).to(device)
label = labels[0].item()

# Move the model to the device before using it
hybrid_model.to(device)

# Resize the image to 224x224 for the ViT model
resize_transform = transforms.Resize((224, 224))
img_resized = resize_transform(img)

# Prediction
hybrid_model.eval()
with torch.no_grad():
    output = hybrid_model(img_resized) # Use the resized image
    probs = torch.softmax(output, dim=1)
    pred_class = torch.argmax(probs, dim=1).item()

print(f"Predicted class: {pred_class}")
print(f"True class: {label}")
print(f"Class probabilities: {probs.cpu().numpy()}")
```

Figure 15: Grad-CAM function

```
# Grad-CAM heatmap - Use the original image for visualization purposes
heatmap = generate_gradcam(hybrid_model, img_resized, target_class=pred_class, device=device, layer_name='cnn')

# Visualize - Use the original image for visualization purposes
show_gradcam_on_image(img, heatmap)
```

```
Predicted class: 0
True class: 0
Class probabilities: [[0.5037693 0.4962307]]
```

Figure 16: Printing Grad-CAM results

8 Robustness Evaluation Code

The figures that contain robustness evaluation code help to check how well the model performs in the real world by injecting three types of image degradation techniques.

```
hybrid_model.load_state_dict(torch.load("/content/drive/MyDrive/Dataset/hybrid_model_weights.pth"))
def apply_gaussian_blur(image_tensor, kernel_size=5):
    """
    Apply Gaussian blur to a tensor image.
    image_tensor: torch.Tensor of shape [3, H, W] with values in [0, 1]
    """
    img_np = image_tensor.permute(1, 2, 0).cpu().numpy()  # HWC
    img_np = (img_np * 255).astype(np.uint8)

    blurred = cv2.GaussianBlur(img_np, (kernel_size, kernel_size), 0)
    blurred = blurred.astype(np.float32) / 255.0
    blurred_tensor = torch.tensor(blurred).permute(2, 0, 1)
    return blurred_tensor
```

Figure 17: Gaussian Blur evaluation

```
def apply_jpeg_compression(image_tensor, quality=30):
    """
    Apply JPEG compression artifacts.
    quality: JPEG quality (lower means more compression)
    """
    img_np = image_tensor.permute(1, 2, 0).cpu().numpy()  # HWC
    img_np = (img_np * 255).astype(np.uint8)
    encode_param = [int(cv2.IMWRITE_JPEG_QUALITY), quality]
    _, encimg = cv2.imencode('.jpg', img_np, encode_param)
    decimg = cv2.imdecode(encimg, 1)  # BGR format
    decimg = cv2.cvtColor(decimg, cv2.COLOR_BGR2RGB)
    decimg = decimg.astype(np.float32) / 255.0
    decimg_tensor = torch.tensor(decimg).permute(2, 0, 1)
    return decimg_tensor

def apply_noise_injection(image_tensor, noise_std=0.05):
    """
    Add Gaussian noise to image tensor.
    noise_std: standard deviation of noise
    """
    noise = torch.randn_like(image_tensor) * noise_std
    noisy_img = image_tensor + noise
    noisy_img = torch.clamp(noisy_img, 0, 1)
    return noisy_img
```

Figure 18: JPEG Compression and Noise Injection

```python
# Function to evaluate model robustness
def evaluate_robustness(model, dataloader, device, perturbation_func, perturbation_name):
    model.eval()
    running_corrects = 0
    total = 0

    with torch.no_grad():
        for inputs, labels in dataloader:
            # Apply perturbation to each image in batch
            perturbed_inputs = torch.stack([perturbation_func(img) for img in inputs])
            perturbed_inputs = perturbed_inputs.to(device)
            labels = labels.to(device)

            outputs = model(perturbed_inputs)
            preds = torch.argmax(outputs, dim=1)
            running_corrects += torch.sum(preds == labels.data)
            total += labels.size(0)

    acc = running_corrects.double() / total
    print(f"Accuracy with {perturbation_name}: {acc:.4f}")
    return acc
```

Figure 19: Function for performing robustness evaluation

References

Face, H. (2025). google/efficientnet-b0 · Hugging Face — huggingface.co, `https://huggingface.co/google/efficientnet-b0`. [Accessed 24-07-2025].

HuggingFace (2025). Vision Transformer (ViT) — huggingface.co, `https://huggingface.co/docs/transformers/en/model_doc/vit`. [Accessed 24-07-2025].

TensorFlow (2025). Convolutional neural network (cnn) : tensorflow core. **URL:** *https://www.tensorflow.org/tutorials/images/cnn*