Configuration Manual

MSc Research Project
Programme Name

Anish Chinthulla
Student ID: x23113081

School of Computing
National College of Ireland

Supervisor: Professor. Hamilton Niculescu

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Chinthulla Anish……………………………………..

Student ID: ………x23113081……………………………………………..……

Programme:……MSc in Data Analytics………………… **Year:** …2023……………..

Module: ……MSc in Research Project……………………………………………….………

Lecturer: …Prof. Hamilton Niculescu………………………………………………………….………

Submission Due Date: …24/04/2025…………………………………………………………………………..………

Project Title: …… Deep Learning-Based Detection of Shoplifting Stages: using 3DCNN and LRCN…………..………

Word Count: ………500……… **Page Count:** …………………………………..…….………

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: ………

Date: ………

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

Deep Learning-Based Detection of Shoplifting Stages: using 3DCNN and LRCN

Anish Chinthulla
Student ID: x23113081

1 Project Overview

Deep Learning-Based Detection of Shoplifting Stages Using 3DCNN and LRCN".
The project investigates the effectiveness of two deep learning architectures—Long-Term Recurrent Convolutional Networks (LRCN) and Three-Dimensional Convolutional Neural Networks (3DCNN)—in detecting various stages of shoplifting behavior from surveillance footage. The models are trained and evaluated on real-world video data extracted from the UCF-Crime dataset to identify early indicators of suspicious activity.

2 Specification

2.1 Hardware Requirements

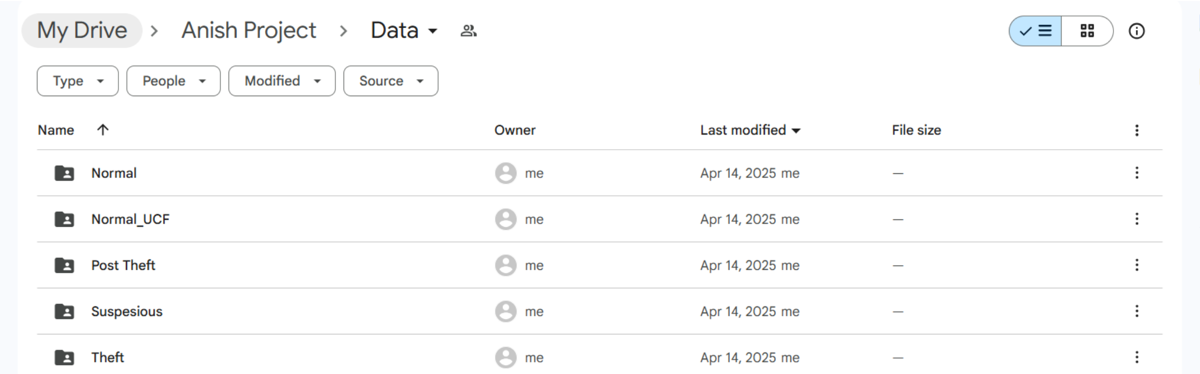
- **Processor:** Intel Core i7 or higher / AMD Ryzen equivalent
- **RAM:** Minimum 16GB
- **GPU:** NVIDIA CUDA-enabled GPU (RTX 2060 or better recommended)
- **Storage:** At least 50GB free disk space
- **Operating System:** Windows 10 / Ubuntu 20.04 LTS

2.2 Software Requirements

- **Programming Language:** Python (version 3.8+), selected for its extensive support in machine learning and computer vision tasks.
- **IDE:** Jupyter Notebook (via Anaconda distribution), widely used for interactive development and data analysis.
- **Cloud Platform:** Google Colab was utilized for training and testing deep learning models in a collaborative, cloud-based environment. It enabled GPU/TPU acceleration without the need for local computational resources, with Drive integration (drive.mount) for seamless data access and storage.
- **Libraries and Frameworks:**
    - **TensorFlow 2.x** and **Keras** for designing and training deep learning models.
    - **NumPy** and **OpenCV** for video processing and numerical computations.
    - **scikit-learn** for model evaluation and utility functions.
    - **Matplotlib** for visualizing performance metrics and training history.

3. Data Analysis and pre-processing

STEP 1-
The dataset was sourced from the UCF Crime Dataset. From this dataset, only the shoplifting-related data was extracted for further analysis. After segmenting the data into different categories — suspicious, theft, normal, and post-theft — the relevant segments were saved to Google Drive for storage and future use.



GOOGLE collab was used in the development of the code.

Mounting the drive into notebook

```
[ ]  from google.colab import drive
     drive.mount('/content/drive')
```

Imported libraries

1-importing-libraries

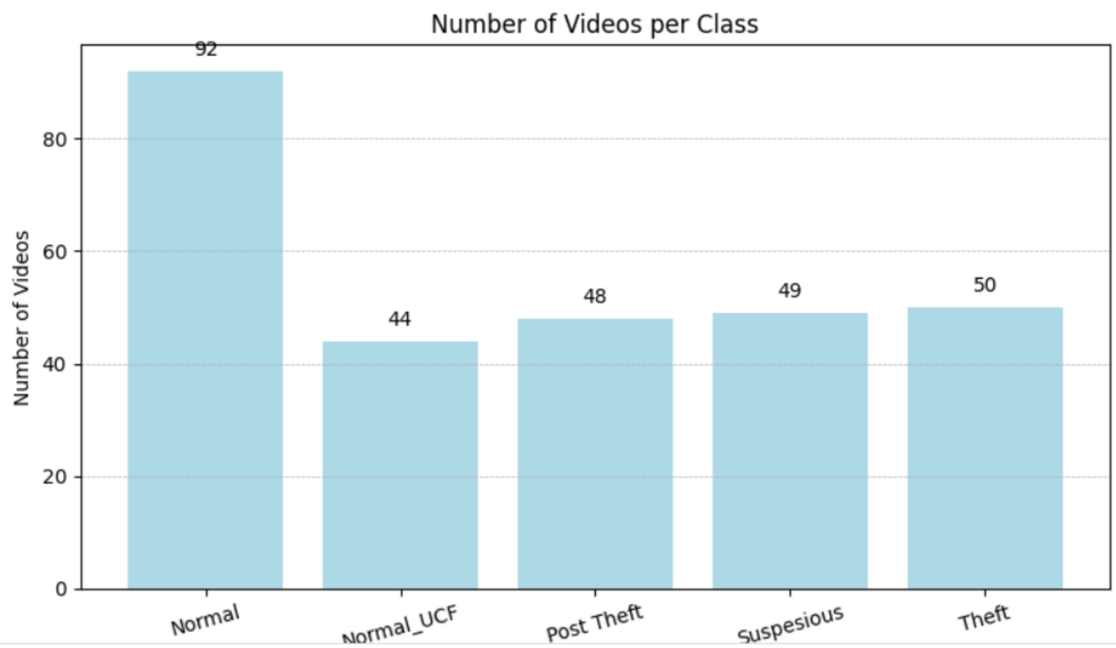
```
[ ]  import os
     import matplotlib.pyplot as plt
     import cv2
     import numpy as np
     import pandas as pd
```

Dataset path -

```
dataset_path = "/content/drive/MyDrive/Anish Project/Data"
dataset_path
```

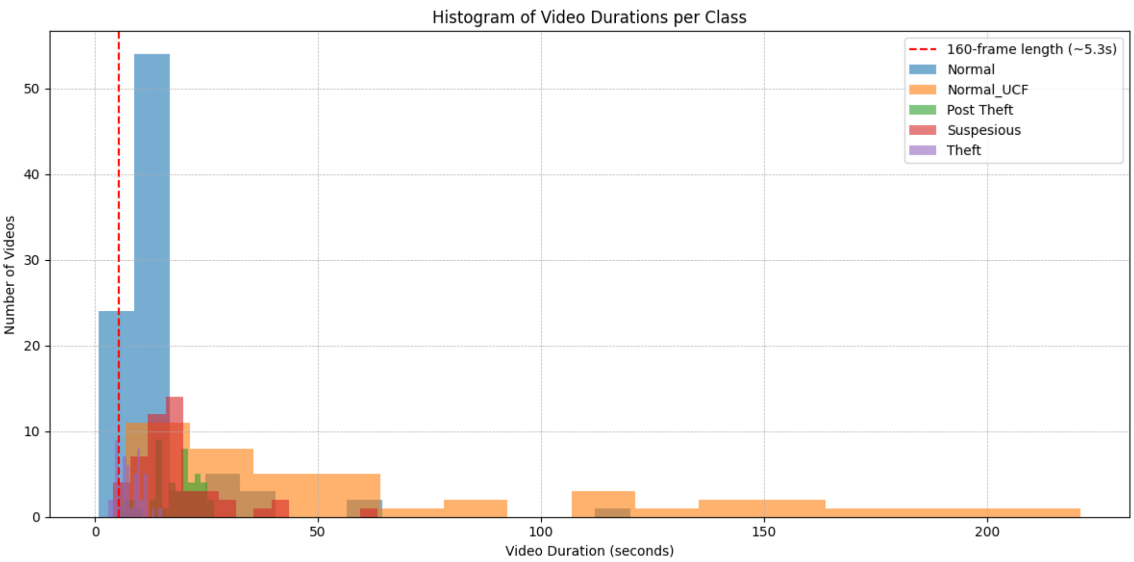
Step 2:- Number of Videos per Class

```
# Labels and values
labels = list(video_counts.keys())
counts = list(video_counts.values())

# Plot setup
plt.figure(figsize=(8, 5))
bars = plt.bar(labels, counts, color='lightblue')

# Add counts on top of each bar
for bar in bars:
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width() / 2, height + 2, str(int(height)),
             ha='center', va='bottom', fontsize=10)

# Final touches
plt.xlabel("Class")
plt.ylabel("Number of Videos")
plt.title("Number of Videos per Class")
plt.xticks(rotation=15)
plt.tight_layout()
plt.grid(axis='y', linestyle='--', linewidth=0.5)
plt.show()
```



Step 3.- Histogram for all the classes

```
[ ] durations_per_class = {}

    for class_name in sorted(os.listdir(dataset_path)):
        class_folder = os.path.join(dataset_path, class_name)
        if not os.path.isdir(class_folder):
            continue

        durations = []

        for video_file in os.listdir(class_folder):
            if not video_file.endswith(".mp4"):
                continue

            video_path = os.path.join(class_folder, video_file)
            cap = cv2.VideoCapture(video_path)

            fps = cap.get(cv2.CAP_PROP_FPS)
            frame_count = cap.get(cv2.CAP_PROP_FRAME_COUNT)

            if fps > 0:
                duration = frame_count / fps
                durations.append(duration)

            cap.release()

        durations_per_class[class_name] = durations

    # Plot histograms
    plt.figure(figsize=(12, 6))
    plt.axvline(x=5.3, color='red', linestyle='--', label='160-frame length (~5.3s)')

    for i, (class_name, durations) in enumerate(durations_per_class.items()):
        plt.hist(durations, bins=15, alpha=0.6, label=class_name)

    plt.xlabel("Video Duration (seconds)")
    plt.ylabel("Number of Videos")
    plt.title("Histogram of Video Durations per Class")
    plt.legend()
    plt.grid(True, linestyle='--', linewidth=0.5)
    plt.tight_layout()
    plt.show()
```

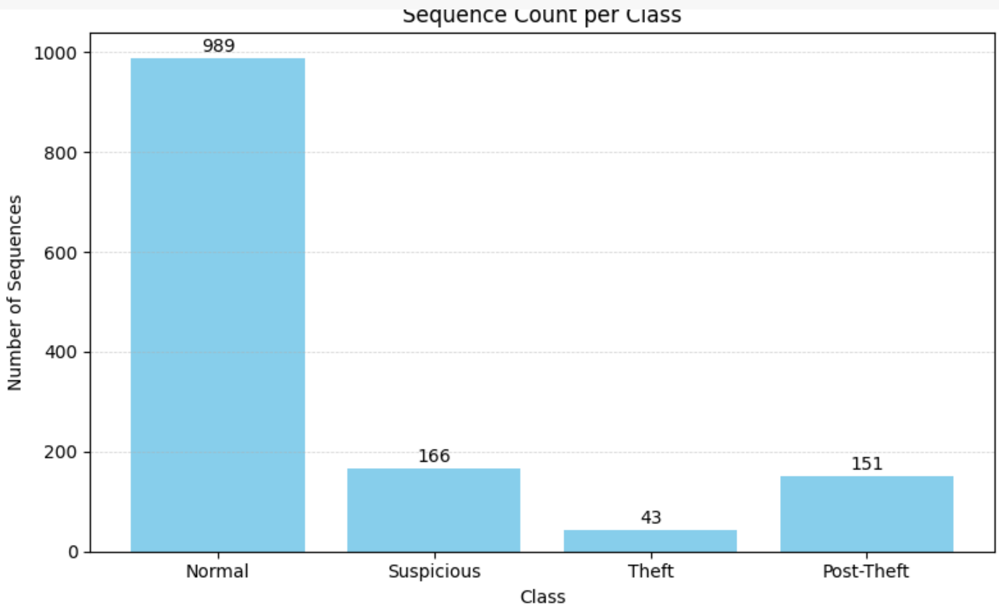


Step 5:- Reshaping the data into 120 frames and converting it into final shape of (120,64,64,1)

```
[ ] import os
    import cv2
    import numpy as np

    # Configuration
    VIDEO_ROOT = dataset_path
    SEQUENCE_LENGTH = 120
    FRAME_SIZE = (64, 64)  # Updated resolution

    # Label mapping: Normal_UCF + Normal = 0
    class_map = {
        "Normal": 0,
        "Normal_UCF": 0,
        "Suspesious": 1,
        "Theft": 2,
        "Post Theft": 3
    }

    X = []
    y = []

    # Process each class folder
    for class_name, label in class_map.items():
        folder_path = os.path.join(VIDEO_ROOT, class_name)
        if not os.path.isdir(folder_path):
            continue

        for video_file in os.listdir(folder_path):
            if not video_file.lower().endswith(".mp4"):
                continue

            video_path = os.path.join(folder_path, video_file)
            cap = cv2.VideoCapture(video_path)
            frames = []

            while True:
                ret, frame = cap.read()
                if not ret:
                    break

                gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
                resized = cv2.resize(gray, FRAME_SIZE)
                normalized = resized.astype("float32") / 255.0
                frames.append(normalized)

            cap.release()

            total = len(frames)
            if total < SEQUENCE_LENGTH:
                continue

            for i in range(0, total - SEQUENCE_LENGTH + 1, SEQUENCE_LENGTH):
                clip = frames[i:i + SEQUENCE_LENGTH]
                clip = np.array(clip).reshape(SEQUENCE_LENGTH, 64, 64, 1)
                X.append(clip)
                y.append(label)

            print(f"Processed {video_file} → {len(frames) // SEQUENCE_LENGTH} sequences")

    # Convert to NumPy arrays
    X = np.array(X)
    y = np.array(y)

    print("\nFinished sequence generation.")
    print(f"X shape: {X.shape}")
    print(f"y shape: {y.shape}")
```

Step 6 :- Number of frames per class after conversion.

```
[ ] import numpy as np
    import matplotlib.pyplot as plt

    # Class label mapping used earlier
    class_labels = {
        0: "Normal",
        1: "Suspicious",
        2: "Theft",
        3: "Post-Theft"
    }

    # Count each label in y
    unique, counts = np.unique(y, return_counts=True)

    # Prepare labels and values for plotting
    labels = [class_labels[i] for i in unique]
    values = counts

    # Print summary
    print("Number of sequences per class:")
    for label, count in zip(labels, values):
        print(f"{label}: {count}")

    # Plot
    plt.figure(figsize=(8, 5))
    bars = plt.bar(labels, values, color='skyblue')

    # Add labels on top of bars
    for bar, count in zip(bars, values):
        plt.text(bar.get_x() + bar.get_width() / 2, count + 5, str(count),
                 ha='center', va='bottom', fontsize=10)

    plt.title("Sequence Count per Class")
    plt.xlabel("Class")
    plt.ylabel("Number of Sequences")
    plt.grid(axis="y", linestyle="--", linewidth=0.5, alpha=0.6)
    plt.tight_layout()
    plt.show()
```



Step 7:- Test train split

Splitting Data for Training and Testing

```
[ ]  from sklearn.model_selection import train_test_split

     # Split the dataset with stratification
     X_train, X_test, y_train, y_test = train_test_split(
         X, y, test_size=0.2, random_state=42, stratify=y
     )

     # Display the shape of the new sets
     print("X_train:", X_train.shape)
     print("X_test: ", X_test.shape)
     print("y_train:", y_train.shape)
     print("y_test: ", y_test.shape)
```

```
X_train: (1079, 120, 64, 64, 1)
X_test:  (270, 120, 64, 64, 1)
y_train: (1079,)
y_test:  (270,)
```

4. Model

4.1 LRCN(Long-term Recurrent Convolutional Network)

Step 1. Model Architecture

```
[ ]  import os
     import tensorflow as tf
     from tensorflow.keras.models import Sequential
     from tensorflow.keras.layers import TimeDistributed, Conv2D, MaxPooling2D, BatchNormalization
     from tensorflow.keras.layers import Flatten, LSTM, Dense, Dropout
     from tensorflow.keras.optimizers import Adam

     # Input shape using 64x64 resized frames
     sequence_length = 120
     frame_height = 64
     frame_width = 64
     channels = 1
     input_shape = (sequence_length, frame_height, frame_width, channels)

     num_classes = 4  # Adjust to 3 if you merged theft + post-theft

     # Define the LRCN model
     model = Sequential()

     model.add(TimeDistributed(Conv2D(16, (3, 3), activation='relu'), input_shape=input_shape))
     model.add(TimeDistributed(MaxPooling2D(pool_size=(2, 2))))
     model.add(TimeDistributed(BatchNormalization()))

     model.add(TimeDistributed(Conv2D(32, (3, 3), activation='relu')))
     model.add(TimeDistributed(MaxPooling2D(pool_size=(2, 2))))
     model.add(TimeDistributed(BatchNormalization()))

     model.add(TimeDistributed(Flatten()))
     model.add(LSTM(128))  # LSTM kept as-is
     model.add(Dropout(0.5))
     model.add(Dense(num_classes, activation='softmax'))

     # Compile the model
     model.compile(
         loss='sparse_categorical_crossentropy',
         optimizer=Adam(learning_rate=0.0001),
         metrics=['accuracy']
     )

     model.summary()
```

Step 2 :- Model function and early stop and check point

```
[ ] from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

    # Path to save the best model weights
    checkpoint_path = os.path.join(os.getcwd(), 'checkpoint', 'LRCNClassification', 'checkpoint.weights.h5')
    os.makedirs(os.path.dirname(checkpoint_path), exist_ok=True)

    early_stopping = EarlyStopping(
        monitor='val_loss',
        patience=5,
        restore_best_weights=True,
        verbose=1
    )

    model_checkpoint = ModelCheckpoint(
        filepath=checkpoint_path,
        monitor='val_loss',
        save_best_only=True,
        save_weights_only=True,
        verbose=1
    )

    callbacks = [early_stopping, model_checkpoint]

    # Train the model
    history = model.fit(
        X_train, y_train,
        validation_data=(X_test, y_test),
        batch_size=4,
        epochs=50,
        callbacks=callbacks
    )
```

4.2 3D-CNN (Three-Dimensional Convolutional Neural Network)

Step 1:- Model Architecture

```
[ ] from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Conv3D, MaxPooling3D, BatchNormalization
    from tensorflow.keras.layers import Flatten, Dense, Dropout
    from tensorflow.keras.optimizers import Adam

    input_shape = (120, 64, 64, 1)
    num_classes = 4  # or 3 if you merged classes

    model = Sequential()

    model.add(Conv3D(32, kernel_size=(3, 3, 3), activation='relu', input_shape=input_shape))
    model.add(MaxPooling3D(pool_size=(1, 2, 2)))
    model.add(BatchNormalization())

    model.add(Conv3D(64, kernel_size=(3, 3, 3), activation='relu'))
    model.add(MaxPooling3D(pool_size=(2, 2, 2)))
    model.add(BatchNormalization())

    model.add(Conv3D(128, kernel_size=(3, 3, 3), activation='relu'))
    model.add(MaxPooling3D(pool_size=(2, 2, 2)))
    model.add(BatchNormalization())

    model.add(Flatten())
    model.add(Dense(256, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(num_classes, activation='softmax'))

    model.compile(
        loss='sparse_categorical_crossentropy',
        optimizer=Adam(learning_rate=0.0001),
        metrics=['accuracy']
    )

    model.summary()
```

Step 2: Model function and early stop and check point

```
[ ] from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
    import os

    checkpoint_path = os.path.join(os.getcwd(), 'checkpoint', '3DCNN', 'checkpoint.weights.h5')
    os.makedirs(os.path.dirname(checkpoint_path), exist_ok=True)

    early_stopping = EarlyStopping(
        monitor='val_loss',
        patience=5,
        restore_best_weights=True,
        verbose=1
    )

    model_checkpoint = ModelCheckpoint(
        filepath=checkpoint_path,
        monitor='val_loss',
        save_best_only=True,
        save_weights_only=True,
        verbose=1
    )

    callbacks = [early_stopping, model_checkpoint]

    history = model.fit(
        X_train, y_train,
        validation_data=(X_test, y_test),
        batch_size=4,
        epochs=50,
        callbacks=callbacks
    )
```

5. Evaluation

In this research , parameter used for evaluvation both models are same

1.Classification report

```python
import numpy as np
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
import matplotlib.pyplot as plt

# Define class labels
class_labels = ['Normal', 'Suspicious', 'Theft', 'Post-Theft']

# Predict class probabilities
print('Finding predicted classes and probabilities to build confusion matrix')
predicted_probs = model.predict(X_test, verbose=1)

# Get predicted class indices
predicted_classes = np.argmax(predicted_probs, axis=-1)  # shape: (num_samples,)

# Ensure y_test is flattened
y_test_flat = y_test.flatten()

# Generate confusion matrix
cm = confusion_matrix(y_test_flat, predicted_classes)

# Plot confusion matrix with labels
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=class_labels, yticklabels=class_labels)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.tight_layout()
plt.show()

# Classification report
print("\nClassification Report:\n")
print(classification_report(y_test_flat, predicted_classes, target_names=class_labels))
```

2.Accuracy

```python
[ ]  from sklearn.metrics import accuracy_score

     # Get final training and validation accuracy from history
     final_train_accuracy = history.history['accuracy'][-1]
     final_val_accuracy = history.history['val_accuracy'][-1]

     # Predict on test set
     predictions = model.predict(X_test)
     predicted_classes = np.argmax(predictions, axis=1)
     true_classes = y_test

     # Compute test accuracy using sklearn
     final_test_accuracy = accuracy_score(true_classes, predicted_classes)

     # Print all three
     print("Final Training Accuracy:       ", round(final_train_accuracy * 100, 2), "%")
     print("Final Validation Accuracy:     ", round(final_val_accuracy * 100, 2), "%")
     print("Test Set Accuracy (Final Eval):", round(final_test_accuracy * 100, 2), "%")
```

3. Training accuracy vs testing accuracy & Losses plots

```
[ ]
    # Plot training history
    pd.DataFrame(history.history).plot()
    plt.title('Accuracy and Loss over Epochs')
    plt.xlabel('Epoch')
    plt.ylabel('Value')
    plt.grid(True)
    plt.show()

    # Plot training vs validation accuracy
    plt.figure(figsize=(8, 7))
    plt.plot(history.history['accuracy'], label='train', color='tomato')
    plt.plot(history.history['val_accuracy'], label='test', color='skyblue')
    plt.title('Model Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend(loc='upper left')
    plt.grid(True)
    plt.show()

    plt.figure(figsize=(8, 7))
    plt.plot(history.history['loss'], label='train', color='tomato')
    plt.plot(history.history['val_loss'], label='test', color='skyblue')
    plt.title('Model Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend(loc='upper right')
    plt.grid(True)
    plt.show()
```