IMPLEMENTATION OF MACHINE LEARNING ALGORITHMS FOR EXAM MONITORING

Random Forest:

```
import os
import cv2
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Function to load and preprocess images
def load_images_from_folder(folder, label):
    images = []
    labels = []
    for filename in os.listdir(folder):
        img_path = os.path.join(folder, filename)
        img = cv2.imread(img_path)
        if img is not None:
            img = cv2.resize(img, (64, 64))  # Resize images to a fixed size
            img = img.astype('float32') / 255.0  # Normalize pixel values
            images.append(img)
            labels.append(label)  # Add label
    return images, labels

# Load images from "cheating" folder
cheating_images, cheating_labels = load_images_from_folder("C:/Users/ravit/OneDrive/Desktop/MINI project/Cheating", 0)

# Load images from "non cheating" folder
non_cheating_images, non_cheating_labels = load_images_from_folder("C:/Users/ravit/OneDrive/Desktop/MINI project/Non Cheating", 1

# Combine images and labels
images = cheating_images + non_cheating_images
labels = cheating_labels + non_cheating_labels
```

```
# Flatten images for Random Forest input
images = images.reshape(images.shape[0], -1)
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(images, labels, test_size=0.2, random_state=42)
# Build the Random Forest model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
# Train the model
rf_model.fit(X_train, y_train)
# Predict on the test set
y_pred = rf_model.predict(X_test)
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Test accuracy: {accuracy * 100:.2f}%')
# Function to make predictions
def classify_image(image_path):
    img = cv2.imread(image_path)
    img = cv2.resize(img, (64, 64))  # Resize to match the input shape of the model
    img = img.astype('float32') / 255.0  # Normalize pixel values
    img = img.flatten().reshape(1, -1)  # Flatten the image and add batch dimension

    prediction = rf_model.predict(img)
    if prediction[0] == 1:
        print("Non-Cheating")
    else:
        print("Cheating")

# Example usage
classify_image('_60001.png')
```

Artificial Neural Network:

```
import os
import cv2
import numpy as np
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D, Dropout
from tensorflow.keras.optimizers import Adam

# Function to load and preprocess images
def load_images_from_folder(folder, label):
    images = []
    labels = []
    for filename in os.listdir(folder):
        img_path = os.path.join(folder, filename)
        img = cv2.imread(img_path)
        if img is not None:
            img = cv2.resize(img, (64, 64))  # Resize images to a fixed size
            img = img.astype('float32') / 255.0  # Normalize pixel values
            images.append(img)
            labels.append(label)  # Add label
    return images, labels

# Load images from "cheating" folder
cheating_images, cheating_labels = load_images_from_folder("C:/Users/ravit/OneDrive/Desktop/MINI project/Cheating", 0)

# Load images from "non cheating" folder
non_cheating_images, non_cheating_labels = load_images_from_folder("C:/Users/ravit/OneDrive/Desktop/MINI project/Non Cheating", 1
```

```
# Convert lists to numpy arrays
images = np.array(images)
labels = np.array(labels)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(images, labels, test_size=0.2, random_state=42)
# Build the ANN model
model = Sequential([
    Conv2D(32, (3, 3), input_shape=(64, 64, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(32, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])
# Compile the model
model.compile(optimizer=Adam(learning_rate=0.001), loss='binary_crossentropy', metrics=['accuracy'])
# Train the model
history = model.fit(X_train, y_train, epochs=20, batch_size=32, validation_data=(X_test, y_test))
# Save the model
model.save('cheating_detection_model.h5')
# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test)
print(f'Test accuracy: {accuracy * 100:.2f}%')
# Function to make predictions
def classify_image(image_path):
    img = cv2.imread(image_path)
    img = cv2.resize(img, (64, 64))  # Resize to match the input shape of the model
    img = img.astype('float32') / 255.0  # Normalize pixel values
    img = np.expand_dims(img, axis=0)  # Add batch dimension

    prediction = model.predict(img)
    if prediction[0] > 0.5:
        print("Non-Cheating")
    else:
        print("Cheating")
# Example usage
classify_image('Copy of _61.png')
```

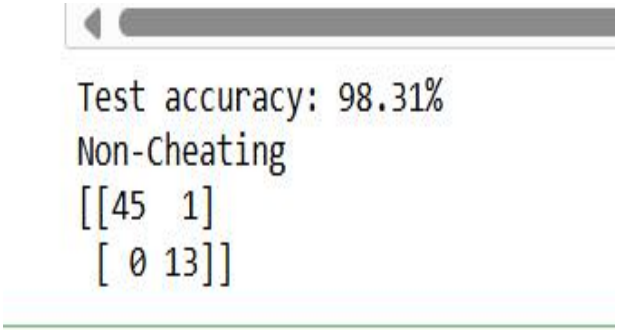
Decision Tree Algorithm

```
import os
import cv2
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Function to load and preprocess images
def load_images_from_folder(folder, label):
    images = []
    labels = []
    for filename in os.listdir(folder):
        img_path = os.path.join(folder, filename)
        img = cv2.imread(img_path)
        if img is not None:
            img = cv2.resize(img, (64, 64))  # Resize images to a fixed size
            img = img.astype('float32') / 255.0  # Normalize pixel values
            images.append(img)
            labels.append(label)  # Add label
    return images, labels

# Load images from "cheating" folder
cheating_images, cheating_labels = load_images_from_folder("C:/Users/ravit/OneDrive/Desktop/MINI project/Cheating", 0)

# Load images from "non cheating" folder
non_cheating_images, non_cheating_labels = load_images_from_folder("C:/Users/ravit/OneDrive/Desktop/MINI project/Non Cheating", 1

# Combine images and labels
images = cheating_images + non_cheating_images
labels = cheating_labels + non_cheating_labels

# Convert lists to numpy arrays
images = np.array(images)
labels = np.array(labels)

# Flatten images for Decision Tree input
images = images.reshape(images.shape[0], -1)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(images, labels, test_size=0.2, random_state=42)
```

```
# Build the Decision Tree model
dt_model = DecisionTreeClassifier(random_state=42)

# Train the model
dt_model.fit(X_train, y_train)

# Predict on the test set
y_pred = dt_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Test accuracy: {accuracy * 100:.2f}%')

# Function to make predictions
def classify_image(image_path):
    img = cv2.imread(image_path)
    img = cv2.resize(img, (64, 64))  # Resize to match the input shape of the model
    img = img.astype('float32') / 255.0  # Normalize pixel values
    img = img.flatten().reshape(1, -1)  # Flatten the image and add batch dimension

    prediction = dt_model.predict(img)
    if prediction[0] == 1:
        print("Non-Cheating")
    else:
        print("Cheating")

# Example usage
classify_image('_60001.png')
```

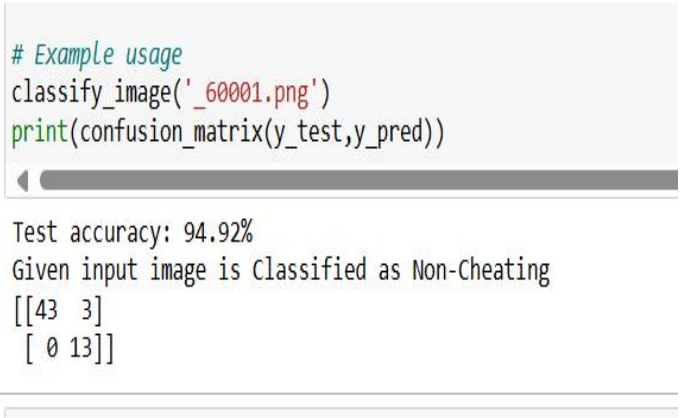
Results:

Random Forest:



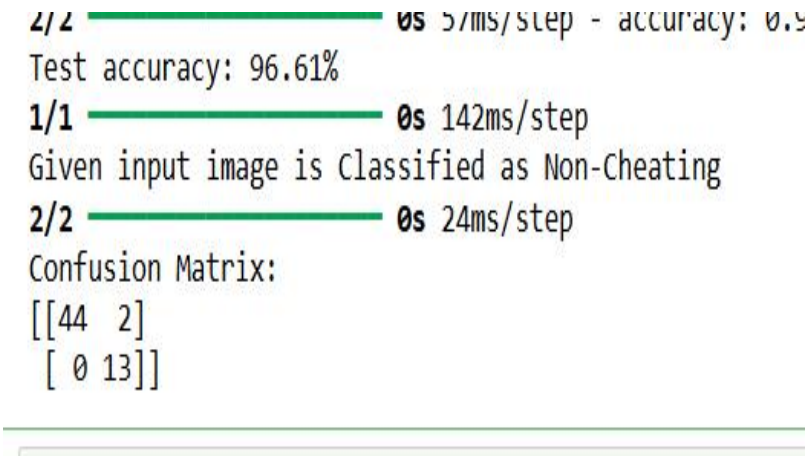
Confusion Matrix of Random Forest

Decision Tree:



Confusion Matrix of Decision Tree

Artificial Neural Networks:



Confusion Matrix of Artificial Neural Networks