Data Mining and Machine Learning in Cybersecurity
Lab 7
Malware Detection using Conv2D

Karthikeyan G
Roll No: CB.SC.P2.CYS24008

March 13, 2025

1 1. Loading and Preprocessing the Dataset

Listing 1: Python code for loading and preprocessing the dataset

```
import tensorflow as tf
from tensorflow.keras import layers, models
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.utils.class_weight import compute_class_weight
from sklearn.metrics import classification_report, confusion_matrix

# Load and preprocess the dataset
def load_and_preprocess_data(file_path):
    # Load CSV data
    data = pd.read_csv(file_path)

    # Normalize column names
    data.columns = data.columns.str.strip().str.lower()

    # Check if 'label' exists
    if 'label' not in data.columns:
        raise KeyError(f"'label' column not found. Available columns: {list(data
            .columns)}")

    # Convert labels to binary (0 = Benign, 1 = Malware)
    y = np.where(data['label'].str.lower() == "malware", 1, 0)

    # Drop the label column to get feature matrix
    X = data.drop(columns=['label']).values

    # Feature Scaling (Standardization)
    X = (X - np.mean(X, axis=0)) / (np.std(X, axis=0) + 1e-8)

    # Zero Padding & Reshape into Image Format
    img_size = 20
    X_padded = np.zeros((X.shape[0], img_size * img_size))
    X_padded[:, :X.shape[1]] = X
    X_padded = X_padded.reshape(-1, img_size, img_size, 1)

    # Split into train/test sets
```

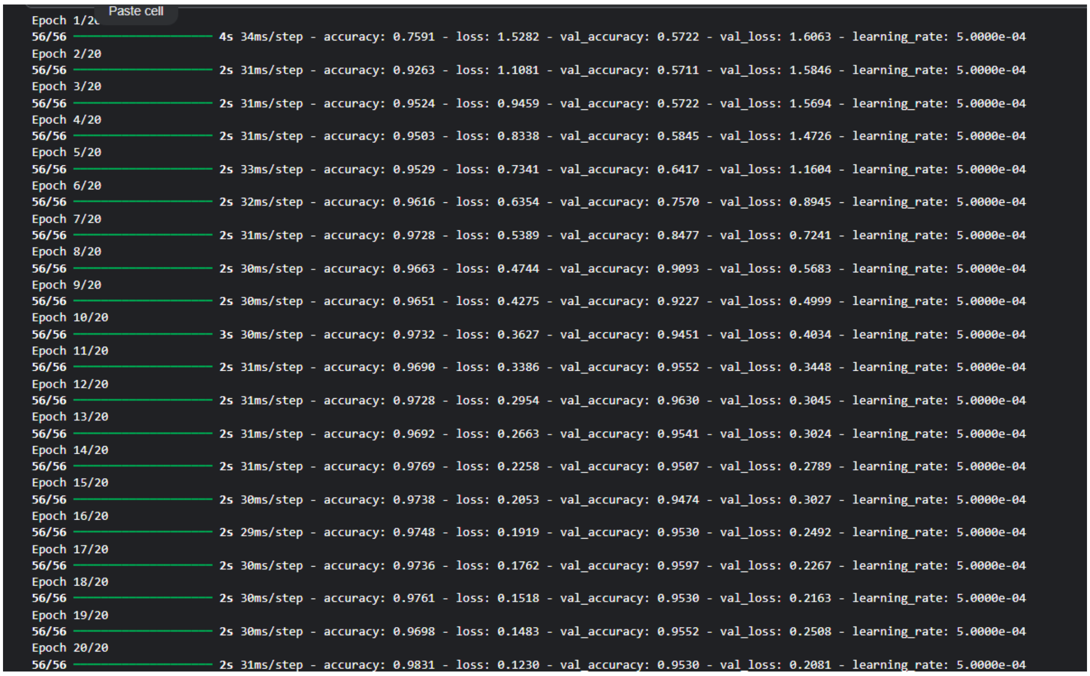
```
    X_train, X_test, y_train, y_test = train_test_split(
        X_padded, y, test_size=0.2, random_state=42, stratify=y, shuffle=True
    )

    return (X_train, y_train), (X_test, y_test)
```

2 2. Training the Conv2D Model

Listing 2: Defining and training the CNN model

```
# Load dataset
file_path = "/kaggle/input/android-malware-detection-dataset/
    Android_Malware_Benign.csv"
(x_train, y_train), (x_test, y_test) = load_and_preprocess_data(file_path)

# Compute class weights for handling imbalanced data
class_weights = compute_class_weight("balanced", classes=np.unique(y_train), y=
    y_train)
class_weight_dict = {i: class_weights[i] for i in range(len(class_weights))}

# Define the CNN Model with an explicit Input layer
model = models.Sequential([
    tf.keras.Input(shape=(20, 20, 1)),
    layers.Conv2D(32, (3, 3), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),

    layers.Flatten(),
    layers.Dense(128, activation='relu', kernel_regularizer=tf.keras.
        regularizers.l2(0.005)),
    layers.Dropout(0.3),
    layers.Dense(64, activation='relu'),
    layers.BatchNormalization(),
    layers.Dense(1, activation='sigmoid')
])

# Train the model
history = model.fit(x_train, y_train,
                    epochs=20,
                    batch_size=64,
                    validation_data=(x_test, y_test),
                    class_weight=class_weight_dict,
                    callbacks=[early_stopping, reduce_lr])

# Plot training & validation accuracy and loss over epochs
plt.figure(figsize=(12, 5))

# Accuracy plot
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], marker='o')
plt.plot(history.history['val_accuracy'], marker='o')
plt.title('Model Accuracy per Epoch')
plt.xlabel('Epoch')
```

```python
plt.ylabel('Accuracy')
plt.legend(['Train', 'Validation'], loc='lower right')

# Loss plot
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], marker='o')
plt.plot(history.history['val_loss'], marker='o')
plt.title('Model Loss per Epoch')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Validation'], loc='upper right')

plt.tight_layout()
plt.show()

# Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
print(f'\nTest accuracy: {test_acc:.4f}')

# Make predictions
y_pred = (model.predict(x_test) > 0.5).astype("int32")

# Print Classification Report
print("\nClassification Report:\n", classification_report(y_test, y_pred))

# Plot Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=["Benign", "Malware"], yticklabels=["Benign", "Malware"
                ])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()

# Malware Detection Summary
malware_detected = np.sum(y_pred)
total_samples = len(y_pred)
malware_percentage = (malware_detected / total_samples) * 100

print("\nMy Code Malware Detection Summary:")
print(f"Total test samples: {total_samples}")
print(f"Detected Malware Samples: {malware_detected} ({malware_percentage:.2f}%)
    ")

if malware_percentage > 50:
    print("\nHIGH ALERT: Significant malware activity detected! Immediate action
         recommended.")
elif malware_percentage > 20:
    print("\nMEDIUM ALERT: Malware presence detected. Further analysis advised."
        )
elif malware_percentage > 0:
    print("\nLOW ALERT: Some malware detected. Keep monitoring.")
else:
    print("\nSAFE: No malware detected. System appears secure.")
```



```
Epoch 1/2
56/56 ━━━━━━━━━━━━━━━━━━━━ 4s 34ms/step - accuracy: 0.7591 - loss: 1.5282 - val_accuracy: 0.5722 - val_loss: 1.6063 - learning_rate: 5.0000e-04
Epoch 2/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 2s 31ms/step - accuracy: 0.9263 - loss: 1.1081 - val_accuracy: 0.5711 - val_loss: 1.5846 - learning_rate: 5.0000e-04
Epoch 3/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 2s 31ms/step - accuracy: 0.9524 - loss: 0.9459 - val_accuracy: 0.5722 - val_loss: 1.5694 - learning_rate: 5.0000e-04
Epoch 4/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 2s 31ms/step - accuracy: 0.9503 - loss: 0.8338 - val_accuracy: 0.5845 - val_loss: 1.4726 - learning_rate: 5.0000e-04
Epoch 5/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 2s 33ms/step - accuracy: 0.9529 - loss: 0.7341 - val_accuracy: 0.6417 - val_loss: 1.1604 - learning_rate: 5.0000e-04
Epoch 6/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 2s 32ms/step - accuracy: 0.9616 - loss: 0.6354 - val_accuracy: 0.7570 - val_loss: 0.8945 - learning_rate: 5.0000e-04
Epoch 7/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 2s 31ms/step - accuracy: 0.9728 - loss: 0.5389 - val_accuracy: 0.8477 - val_loss: 0.7241 - learning_rate: 5.0000e-04
Epoch 8/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 2s 30ms/step - accuracy: 0.9663 - loss: 0.4744 - val_accuracy: 0.9093 - val_loss: 0.5683 - learning_rate: 5.0000e-04
Epoch 9/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 2s 30ms/step - accuracy: 0.9651 - loss: 0.4275 - val_accuracy: 0.9227 - val_loss: 0.4999 - learning_rate: 5.0000e-04
Epoch 10/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 3s 30ms/step - accuracy: 0.9732 - loss: 0.3627 - val_accuracy: 0.9451 - val_loss: 0.4034 - learning_rate: 5.0000e-04
Epoch 11/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 2s 31ms/step - accuracy: 0.9690 - loss: 0.3386 - val_accuracy: 0.9552 - val_loss: 0.3448 - learning_rate: 5.0000e-04
Epoch 12/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 2s 31ms/step - accuracy: 0.9728 - loss: 0.2954 - val_accuracy: 0.9630 - val_loss: 0.3045 - learning_rate: 5.0000e-04
Epoch 13/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 2s 31ms/step - accuracy: 0.9692 - loss: 0.2663 - val_accuracy: 0.9541 - val_loss: 0.3024 - learning_rate: 5.0000e-04
Epoch 14/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 2s 31ms/step - accuracy: 0.9769 - loss: 0.2258 - val_accuracy: 0.9507 - val_loss: 0.2789 - learning_rate: 5.0000e-04
Epoch 15/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 2s 30ms/step - accuracy: 0.9738 - loss: 0.2053 - val_accuracy: 0.9474 - val_loss: 0.3027 - learning_rate: 5.0000e-04
Epoch 16/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 2s 29ms/step - accuracy: 0.9748 - loss: 0.1919 - val_accuracy: 0.9530 - val_loss: 0.2492 - learning_rate: 5.0000e-04
Epoch 17/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 2s 30ms/step - accuracy: 0.9736 - loss: 0.1762 - val_accuracy: 0.9597 - val_loss: 0.2267 - learning_rate: 5.0000e-04
Epoch 18/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 2s 30ms/step - accuracy: 0.9761 - loss: 0.1518 - val_accuracy: 0.9530 - val_loss: 0.2163 - learning_rate: 5.0000e-04
Epoch 19/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 2s 30ms/step - accuracy: 0.9698 - loss: 0.1483 - val_accuracy: 0.9552 - val_loss: 0.2508 - learning_rate: 5.0000e-04
Epoch 20/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 2s 31ms/step - accuracy: 0.9831 - loss: 0.1230 - val_accuracy: 0.9530 - val_loss: 0.2081 - learning_rate: 5.0000e-04
```

Figure 1: Training accuracy and loss per epoch

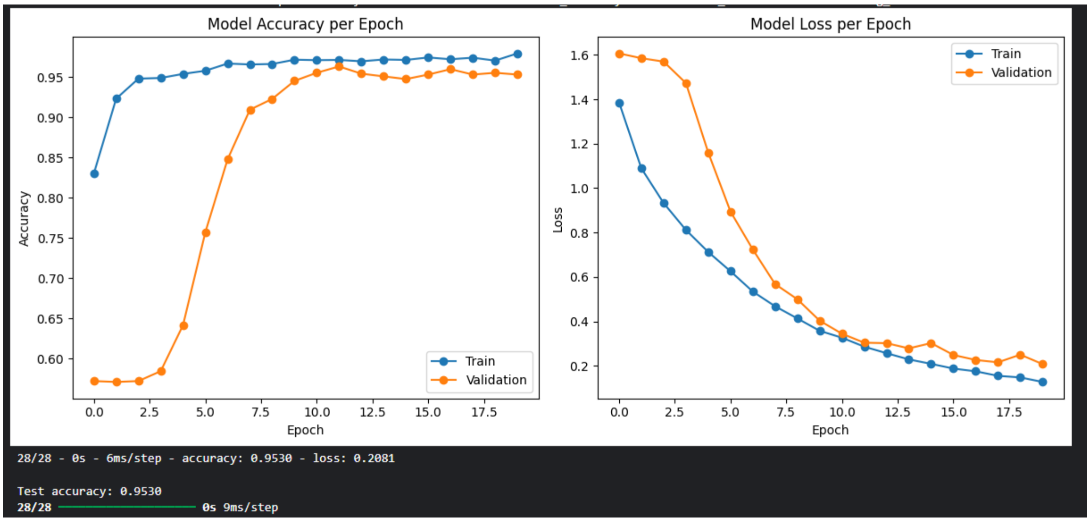
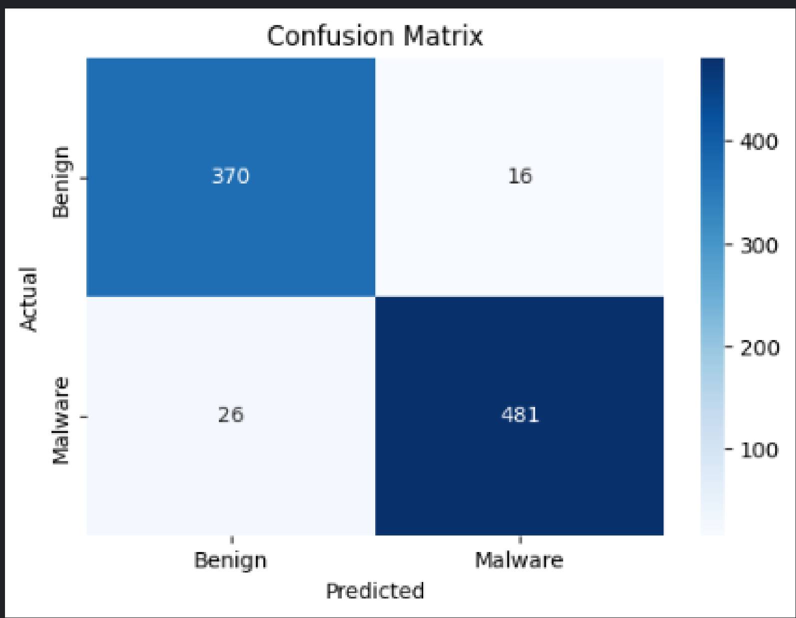


Figure 2: Validation accuracy and test accuracy trends

```
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.96      0.95       386
           1       0.97      0.95      0.96       507

    accuracy                           0.95       893
   macro avg       0.95      0.95      0.95       893
weighted avg       0.95      0.95      0.95       893
```




```
🔍 My Code Malware Detection Summary:
📌 Total test samples: 893
🚀 Detected Malware Samples: 497 (55.66%)

⚠️ HIGH ALERT: Significant malware activity detected! Immediate action recommended.
```

Figure 3: Confusion matrix of malware detection results