CNN Model Code

Code

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Conv2D, MaxPooling2D,
Flatten
from tensorflow.keras.optimizers import Adam
from art.attacks.evasion import FastGradientMethod, BasicIterativeMethod,
ProjectedGradientDescent
from art.estimators.classification import TensorFlowV2Classifier
from art.utils import to_categorical

# Load dataset
df =
pd.read_csv('C:/Users/siyua/OneDrive/Desktop/UNSW_NB15_training.csv')

# Drop rows with null value and duplicates
df = df.dropna().drop_duplicates()
df = df.drop(columns=['attack_cat','id'])

# Convert non-numeric columns to numeric
categorical_columns = ['proto', 'service', 'state']
label_encoders = {}
for col in categorical_columns:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le

# Separate features and target
X = df.drop(columns=['label'])
y = df['label']

# Normalize features
scaler = MinMaxScaler()
X_scaled = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)

# Feature importance using Random Forest
model = RandomForestClassifier(random_state=42)
```

```python
model.fit(X_scaled, y)
feature_importances = model.feature_importances_

# Select top 8 features
importance_df = pd.DataFrame({
    'Feature': X.columns,
    'Importance': feature_importances
}).sort_values(by='Importance', ascending=False)

top_features = importance_df['Feature'].head(8).tolist()
X_scaled = X_scaled[top_features]

# Reshape data for 2D CNN
X_scaled_reshaped = X_scaled.to_numpy().reshape(-1, 8, 1, 1)

# Split dataset
X_train, X_temp, y_train, y_temp = train_test_split(X_scaled_reshaped, y,
test_size=0.3, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp,
test_size=0.5, random_state=42)

# Convert labels to categorical
y_train_cat = to_categorical(y_train)
y_val_cat = to_categorical(y_val)
y_test_cat = to_categorical(y_test)

# Define 2D CNN model
def build_cnn_model(input_shape):
    model = Sequential([
        # First level
        Conv2D(32, (3, 1), activation='relu', input_shape=input_shape),
        MaxPooling2D(pool_size=(2, 1)),

        # Second level
        Conv2D(64, (3, 1), activation='relu'),
        Flatten(),

        # Full-connected layer
        Dense(128, activation='relu'),
        Dropout(0.5),
        Dense(64, activation='relu'),
        Dropout(0.5),
        Dense(2, activation='softmax')
    ])
```

```
    model.compile(optimizer=Adam(learning_rate=0.001),
loss='categorical_crossentropy', metrics=['accuracy'])
    return model

# Create CNN model
cnn_model = build_cnn_model(input_shape=(8, 1, 1))

# Train CNN model
cnn_model.fit(X_train, y_train_cat, epochs=20, batch_size=32,
validation_data=(X_val, y_val_cat))

# Evaluate baseline CNN
baseline_accuracy = cnn_model.evaluate(X_test, y_test_cat, verbose=0)[1]
print(f"Baseline CNN Accuracy: {baseline_accuracy:.2f}")

# Wrap CNN with ART
classifier = TensorFlowV2Classifier(
    model=cnn_model,
    nb_classes=2,
    input_shape=(8, 1, 1),
    loss_object=tf.keras.losses.CategoricalCrossentropy()
)

# Generate adversarial samples
def generate_adversarial_samples(classifier, X, method):
    if method == "FGSM":
        attack = FastGradientMethod(estimator=classifier, eps=0.2)
    elif method == "BIM":
        attack = BasicIterativeMethod(estimator=classifier, eps=0.2,
max_iter=10)
    elif method == "PGD":
        attack = ProjectedGradientDescent(estimator=classifier, eps=0.2,
max_iter=10)
    else:
        raise ValueError("Unknown attack method")
    return attack.generate(X)

# Test CNN on adversarial samples
adversarial_methods = ["FGSM", "BIM", "PGD"]
for method in adversarial_methods:
    X_test_adv = generate_adversarial_samples(classifier, X_test, method)
    y_pred_adv = np.argmax(classifier.predict(X_test_adv), axis=1)
    y_true = np.argmax(y_test_cat, axis=1)
    adv_accuracy = accuracy_score(y_true, y_pred_adv)
```

```python
    print(f"Accuracy on {method} adversarial samples: {adv_accuracy:.2f}")

# Adversarial training with min-max formulation
for method in adversarial_methods:
    X_train_adv = generate_adversarial_samples(classifier, X_train,
method)
    X_combined = np.vstack([X_train, X_train_adv])
    y_combined = np.vstack([y_train_cat, y_train_cat])
    cnn_model.fit(X_combined, y_combined, epochs=10, batch_size=32,
validation_data=(X_val, y_val_cat))

# Test robust CNN on adversarial samples
for method in adversarial_methods:
    X_test_adv = generate_adversarial_samples(classifier, X_test, method)
    y_pred_adv = np.argmax(classifier.predict(X_test_adv), axis=1)
    y_true = np.argmax(y_test_cat, axis=1)
    adv_accuracy = accuracy_score(y_true, y_pred_adv)
    print(f"Robust CNN Accuracy on {method} adversarial samples:
{adv_accuracy:.2f}")
```

Results

```
Epoch 1/20
3836/3836 ━━━━━━━━━━━━━━━━━━━━ 8s 2ms/step - accuracy: 0.9026 - loss: 0.2613 - val_accuracy: 0.9267 - val_loss: 0.1981
Epoch 2/20
3836/3836 ━━━━━━━━━━━━━━━━━━━━ 7s 2ms/step - accuracy: 0.9293 - loss: 0.1985 - val_accuracy: 0.9278 - val_loss: 0.1983
Epoch 3/20
3836/3836 ━━━━━━━━━━━━━━━━━━━━ 7s 2ms/step - accuracy: 0.9295 - loss: 0.1970 - val_accuracy: 0.9279 - val_loss: 0.1889
Epoch 4/20
3836/3836 ━━━━━━━━━━━━━━━━━━━━ 7s 2ms/step - accuracy: 0.9289 - loss: 0.1904 - val_accuracy: 0.9278 - val_loss: 0.1856
Epoch 5/20
3836/3836 ━━━━━━━━━━━━━━━━━━━━ 7s 2ms/step - accuracy: 0.9298 - loss: 0.1855 - val_accuracy: 0.9278 - val_loss: 0.1871
Epoch 6/20
3836/3836 ━━━━━━━━━━━━━━━━━━━━ 7s 2ms/step - accuracy: 0.9307 - loss: 0.1804 - val_accuracy: 0.9278 - val_loss: 0.1782
Epoch 7/20
3836/3836 ━━━━━━━━━━━━━━━━━━━━ 7s 2ms/step - accuracy: 0.9291 - loss: 0.1811 - val_accuracy: 0.9278 - val_loss: 0.1767
Epoch 8/20
3836/3836 ━━━━━━━━━━━━━━━━━━━━ 7s 2ms/step - accuracy: 0.9308 - loss: 0.1751 - val_accuracy: 0.9278 - val_loss: 0.1718
Epoch 9/20
3836/3836 ━━━━━━━━━━━━━━━━━━━━ 7s 2ms/step - accuracy: 0.9301 - loss: 0.1724 - val_accuracy: 0.9279 - val_loss: 0.1690
Epoch 10/20
3836/3836 ━━━━━━━━━━━━━━━━━━━━ 7s 2ms/step - accuracy: 0.9300 - loss: 0.1724 - val_accuracy: 0.9279 - val_loss: 0.1677
Epoch 11/20
3836/3836 ━━━━━━━━━━━━━━━━━━━━ 7s 2ms/step - accuracy: 0.9306 - loss: 0.1693 - val_accuracy: 0.9280 - val_loss: 0.1692
Epoch 12/20
3836/3836 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.9319 - loss: 0.1681 - val_accuracy: 0.9278 - val_loss: 0.1706
Epoch 13/20
3836/3836 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.9295 - loss: 0.1712 - val_accuracy: 0.9280 - val_loss: 0.1683
Epoch 14/20
3836/3836 ━━━━━━━━━━━━━━━━━━━━ 7s 2ms/step - accuracy: 0.9297 - loss: 0.1714 - val_accuracy: 0.9280 - val_loss: 0.1707
Epoch 15/20
3836/3836 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.9292 - loss: 0.1689 - val_accuracy: 0.9280 - val_loss: 0.1663
Epoch 16/20
3836/3836 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.9302 - loss: 0.1660 - val_accuracy: 0.9280 - val_loss: 0.1639
Epoch 17/20
3836/3836 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.9297 - loss: 0.1659 - val_accuracy: 0.9280 - val_loss: 0.1669
Epoch 18/20
3836/3836 ━━━━━━━━━━━━━━━━━━━━ 7s 2ms/step - accuracy: 0.9292 - loss: 0.1673 - val_accuracy: 0.9280 - val_loss: 0.1638
Epoch 19/20
3836/3836 ━━━━━━━━━━━━━━━━━━━━ 7s 2ms/step - accuracy: 0.9295 - loss: 0.1664 - val_accuracy: 0.9280 - val_loss: 0.1647
Epoch 20/20
3836/3836 ━━━━━━━━━━━━━━━━━━━━ 7s 2ms/step - accuracy: 0.9295 - loss: 0.1656 - val_accuracy: 0.9281 - val_loss: 0.1639
Baseline CNN Accuracy: 0.93
```

```
Accuracy on FGSM adversarial samples: 0.41
PGD - Batches: 822it [01:41,  7.76it/s]2024-11-30 17:38:09.
Accuracy on BIM adversarial samples: 0.19
PGD - Batches: 822it [01:41,  7.83it/s]2024-11-30 17:40:03.
Accuracy on PGD adversarial samples: 0.19
```

```
Robust CNN Accuracy on FGSM adversarial samples: 0.83
Robust CNN Accuracy on BIM adversarial samples: 0.66
Robust CNN Accuracy on PGD adversarial samples: 0.66
```

Results Visualization

Code

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

# Data
categories = ['FGSM', 'BIM', 'PGD']
baseline_accuracy = [0.93, 0.93, 0.93]
adversarial_accuracy = [0.41, 0.19, 0.19]
robust_accuracy = [0.83, 0.66, 0.66]

# Bar width and positions
bar_width = 0.25
x = np.arange(len(categories))

# Define custom pastel colors
custom_colors = ['#ff69b4', '#ffd700', '#add8e6']

# Plotting with new colors
plt.figure(figsize=(10, 6))
plt.bar(x - bar_width, baseline_accuracy, width=bar_width, label='Baseline
CNN Accuracy', color=custom_colors[0])
plt.bar(x, adversarial_accuracy, width=bar_width, label='Adversarial
Sample Accuracy', color=custom_colors[1])
plt.bar(x + bar_width, robust_accuracy, width=bar_width, label='Robust CNN
Accuracy', color=custom_colors[2])

# Labels and title
plt.xlabel('Attack Methods', fontsize=12)
plt.ylabel('Accuracy', fontsize=12)
plt.title('CNN Accuracy under Different Attack Scenarios', fontsize=14)
plt.xticks(x, categories)
```

```python
plt.ylim(0, 1)
plt.legend()

# Display plot
plt.tight_layout()
plt.show()
```

Results