T.C

İSTANBUL OKAN UNIVERSITY

FACULTY OF ENGINEERING AND NATURAL SCIENCES

DEPARTMENT OF SOFTWARE ENGINEERING

Artifical Intelligence Term Project

PREPARED BY:

Mustafa Doruk ÇİL

200218006

İSTANBUL

2025

TABLE OF CONTENTS

1) Aim of Project

This project aims to improvement of firewall actions with using deep learning. According to gained past logs of network traffic, firewall action shall be determined.

2) Information About Dataset

This dataset shows network traffic informations and related firewall actions. It includes 65,533 observations. Inputs are 11 features and outputs are 3 different class label. All features are in numerical form and outputs are in categorical form. Project data is stored in cvs file.

This dataset obtained from kaggle.com.

Dataset Link: https://www.kaggle.com/code/docxian/internet-firewall-analysis/input

Features

1) Source Port: The port number on the source (sender) machine sending to the traffic.

2) Destination Port: The port number on the destination (receiver) machine meant to receive the traffic.

3) NAT Source Port: If Network Address Translation (NAT) is used, this is the port number assigned by the NAT device for the outgoing traffic.

4) NAT Destination Port: If NAT is involved on the destination side, this is the translated port number for incoming traffic.

5) Bytes: Total number of bytes transferred during the connection or session. (inclues both sent and received bytes)

6) Bytes Sent: Number of bytes sent from the source machine to the destination.

7) Bytes Received: Number of bytes received from the destination back to the source.

8) Packets: Total number of data packets exchanged during the session. (includes both directions)

9) Elapsed Time (sec): Shows how long the connection/session lasted in seconds.

10) pkts_sent: Number of packets sent from the source to the destination.

11) pkts_received: Number of packets received from the destination by the source.

Class Labels

Allow: Shows that the requested network traffic is allowed by firewall.

Deny: Shows that the requested network traffic is denied by firewall.

Drop: Shows that firewall sliently discards the requested network traffic. This operation does not notify the sender.

3) Data Preprocessing

Dimension Reduction

As provided in section 2, some features are dependent to each other. So that we can apply dimension reduction technique. In this project Principal Component Analysis (PCA) dimension reduction technique is used for dimension reduction. 11 features translated to 4 features.

4) Information About Project Code

Programming Language

Python programming language is used in this project.

Used Python Libraries

Numpy: Used for providing matrix operations.

Pandas: Used for getting information from csv file and obtaining data preprocessing is required or not.

Scikit-Learn: Used for getting train test split and also getting confusion matrix for model evaluation.

Matplotlib: Used for drawing charts about the project.

Pytorch: Used for developing neural network architecture, training the model and evaluating the model.

Project Files

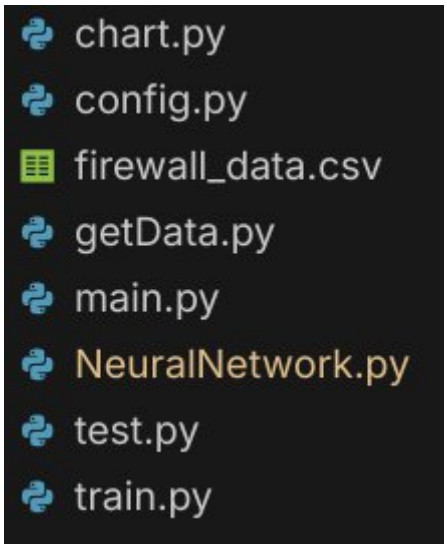


chart.py: Includes matplotlib chart draw functions.

config.py: Includes settings about dataset and hyperparameters.

firewall_data.csv: Includes dataset which is briefly explained in section 2.

getData.py: Includes data getting, splitting, preprocessing and dimension reduction functions.

main.py: Includes high level algorithm of the project.

NeuralNetwork.py: Includes Neural Network architecture.

test.py: Includes test function of the project.

train.py: Includes train and validation function of the project.

Project Steps and Codes

Step 1) Define Hyperparameters and Other Settings in config.py File

```
config_data = {
    "dataSet": "./firewall_data.csv",
    "numberOfIndependentVariables": 11,
    "reducedNumberOfDimensions": 4,
    "classLabel": "Action",
    "numberOfClassLabels": 3,
    "featureLabel": "Action",
    "trainTestSplitRatio": 0.3,
    "validationSplit": 0.2,
    "labelEncoder": {"allow": 0, "drop": 1, "deny": 2},
    "learningRate": 0.001,
    "weightDecay": 0.01,
    "numberOfEpochs": 20,
    "batchSize": 32,
    "dropOutRate": 0.2,
    "neuralNetworkLayers": [4, 9, 10, 3], # [input neurons, hidden layers, output neurons]
    "randomSeed": 36
}
```

Step 2) Get Dataset from CSV File

```
def get_data_set():
    dataSetPath = config_data["dataSet"]
    dataSet = pd.read_csv(dataSetPath)
    return dataSet
```

Step 3) Apply Label Encoding (Convert Categorical Data to Numerical Data)

```
def get_class_label(class_result):
    class_label = ""

    if class_result == 0:
        class_label = "allow"

    elif class_result == 1:
        class_label = "drop"

    elif class_result == 2:
        class_label = "deny"

    return class_label
```

Step 4) Apply Dimension Reduction with Principal Component Analysis (PCA)

```
def dimension_reduction(x):
    reduced_number_of_dimensions = config_data["reducedNumberOfDimensions"]
    pca = PCA(n_components=reduced_number_of_dimensions)
    X_reduced = pca.fit_transform(x)

    return X_reduced
```

Step 5) Get Train and Split Data

```
def get_splitted_train_test_data(x,y):
    trainTestSplitRatio = config_data["trainTestSplitRatio"]
    X_train, X_test, y_train, y_test = train_test_split(
        x,
        y,
        test_size=trainTestSplitRatio,
        random_state=0
    )
    return X_train, X_test, y_train, y_test
```

Step 6) Define Neural Network Architecture

```
import torch
import torch.nn as nn
from config import config_data

# Configuration
numberOfIndependentVariables = config_data["numberOfIndependentVariables"]
numberOfClassLabels = config_data["numberOfClassLabels"]
numberOfInputNeurons = config_data["neuralNetworkLayers"][0]
numberOfFirstHiddenLayerNeurons = config_data["neuralNetworkLayers"][1]
numberOfSecondHiddenLayerNeurons = config_data["neuralNetworkLayers"][2]
dropOutRate = config_data["dropOutRate"]
randomSeed = config_data["randomSeed"]
batchSize = config_data["batchSize"]
numberOfEpochs = config_data["numberOfEpochs"]
learningRate = config_data["learningRate"]
```

```
class NeuralNetwork(nn.Module):
    def __init__(self, input_features=numberOfInputNeurons, h1=numberOfFirstHiddenLayerNeurons,
                 h2=numberOfSecondHiddenLayerNeurons, output_features=numberOfClassLabels):
        super().__init__()
        self.function1 = nn.Sequential(
            nn.Linear(input_features, h1),
            nn.ReLU(inplace=True),
            nn.Dropout(dropOutRate)
        )
        self.function2 = nn.Sequential(
            nn.Linear(h1, h2),
            nn.ReLU(inplace=True),
            nn.Dropout(dropOutRate)
        )
        self.output = nn.Linear(h2, output_features)

    def forward(self, x):
        x = self.function1(x)
        x = self.function2(x)
        return self.output(x)

randomSeed = config_data["randomSeed"]
torch.manual_seed(randomSeed)
```

Step 7) Apply Training on Training Data, Get Train and Validation Results

```
import torch
import torch.nn as nn
from torch.utils.data import TensorDataset, DataLoader, random_split
import chart
from config import config_data

def train(neural_network, x_train, y_train):
    lossFunction = nn.CrossEntropyLoss()

    learningRate = config_data["learningRate"]
    batchSize = config_data["batchSize"]
    numberOfEpochs = config_data["numberOfEpochs"]
    validationSplit = config_data["validationSplit"]
    weightDecay = config_data["weightDecay"]
```

```
    optimizationFunction = torch.optim.Adam(
        neural_network.parameters(),
        lr=learningRate,
        weight_decay=weightDecay
    )

    full_dataset = TensorDataset(x_train, y_train)

    total_size = len(full_dataset)
    val_size = int(total_size * validationSplit)
    train_size = total_size - val_size

    train_dataset, val_dataset = random_split(full_dataset, [train_size, val_size])

    train_dataloader = DataLoader(train_dataset, batch_size=batchSize, shuffle=True)
    val_dataloader = DataLoader(val_dataset, batch_size=batchSize, shuffle=False)
```

```
    losses = []
    val_losses = []

    # training
    for epoch in range(numberOfEpochs):
        neural_network.train()
        epoch_loss = 0.0

        for batch_x, batch_y in train_dataloader:
            y_pred = neural_network(batch_x)
            loss = lossFunction(y_pred, batch_y)

            optimizationFunction.zero_grad()
            loss.backward()
            optimizationFunction.step()

            epoch_loss += loss.item()
```

```
    average_loss = epoch_loss / len(train_dataloader)
    losses.append(average_loss)

    # validation
    neural_network.eval()
    val_loss = 0.0

    with torch.no_grad():
        for batch_x, batch_y in val_dataloader:
            y_pred = neural_network(batch_x)
            loss = lossFunction(y_pred, batch_y)
            val_loss += loss.item()

    average_val_loss = val_loss / len(val_dataloader)
    val_losses.append(average_val_loss)
```

```
        print(f"Epoch: {epoch+1}/{numberOfEpochs}, "
              f"Training Loss: {average_loss:.4f}, Validation Loss: {average_val_loss:.4f}")

    chart.draw(numberOfEpochs, losses, "Training Loss", "Epochs", "Loss")
    chart.draw(numberOfEpochs, val_losses, "Validation Loss", "Epochs", "Loss")
```

Step 8) Apply Test, Print Confusion Matrix and Accuracy

```
from torch.utils.data import TensorDataset, DataLoader
from sklearn.metrics import confusion_matrix
from config import config_data
import torch
import torch.nn as nn
import getData

def test(neural_network, x_test, y_test):
    lossFunction = nn.CrossEntropyLoss()
    batchSize = config_data["batchSize"]

    neural_network.eval()

    test_dataset = TensorDataset(x_test, y_test)
    test_loader = DataLoader(test_dataset, batch_size=batchSize)

    total_loss = 0.0
    correct = 0
    all_preds = []
    all_labels = []
```

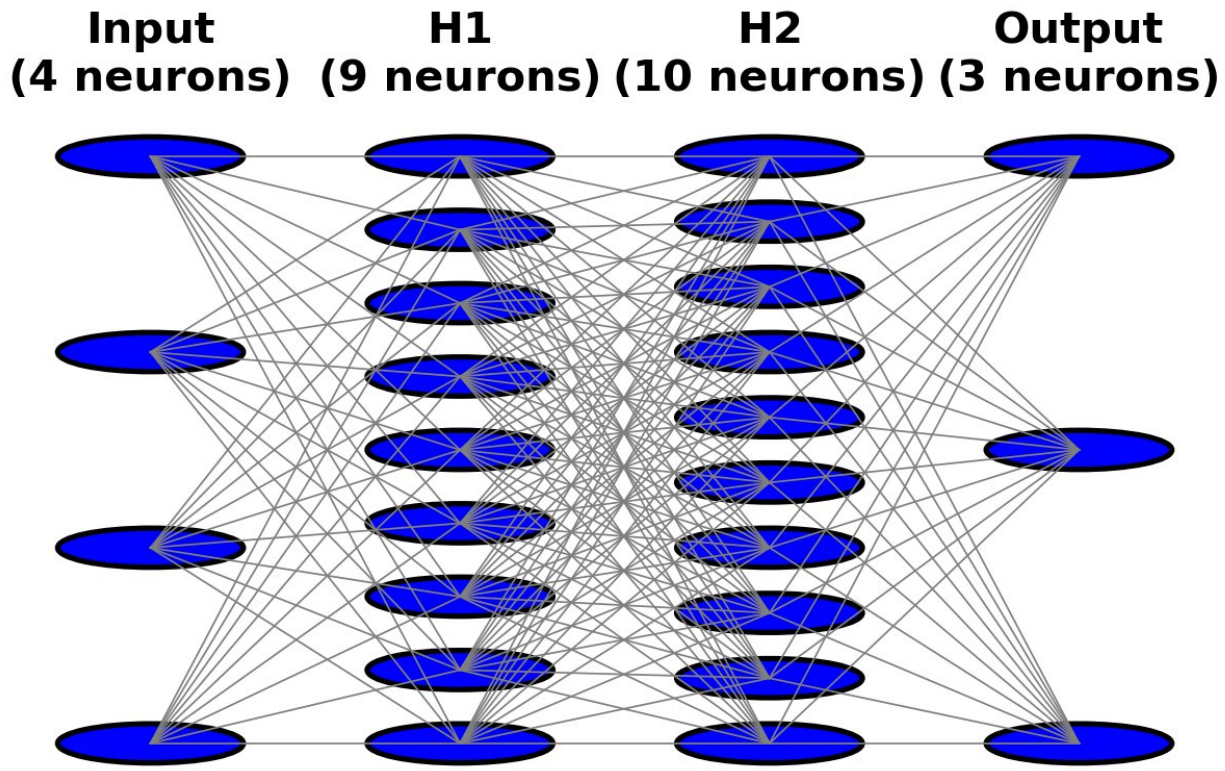
```
    for batch_idx, (inputs, labels) in enumerate(test_loader):
        outputs = neural_network(inputs)
        loss = lossFunction(outputs, labels)
        total_loss += loss.item()

        _, preds = torch.max(outputs, 1)

        all_preds.extend(preds.tolist())
        all_labels.extend(labels.tolist())

        for i in range(len(labels)):
            actual_class = labels[i].item()
            predicted_class = preds[i].item()

            actual_label = getData.get_class_label(actual_class)
            predicted_label = getData.get_class_label(predicted_class)

            print(f"""
                {batch_idx*batchSize + i + 1})
                  Actual Class: {actual_label} \t
                  Predicted Class: {predicted_label}""")
```

```
                if actual_class == predicted_class:
                    correct += 1

    # Calculate metrics
    avg_loss = total_loss / len(test_loader)
    print(f"\nAverage Test Loss: {avg_loss:.4f}")

    print("\nConfusion Matrix")
    print(confusion_matrix(all_labels, all_preds))

    total = len(y_test)
    print(f"\nCorrect Predictions: {correct}/{total}")
    print(f"Accuracy: {(correct/total)*100:.2f}%")
```

5) Neural Network Architecture

Neural Network includes 2 hidden layers. H1 refers to first hidden layer and includes 9 neurons,H2 refers to second hidden layer and includes 10 neurons. Output layers consists of 3 neuronseach of them are related to class labels.

Neural Network developed with Forward Architecture. At the end of the each layer ReLU (Rectified Linear Unit) activation function is used.



6) Hyperparameter Selection

Train Test Split Ratio: 0.3 (70% of the data will used for training 30% of the data will be used for testing)

Validation Split: 0.2 (20% of the training data will be used for validating the training)

Learning Rate: 0.001

Weight Decay: 0.01

Number of Epochs: 20

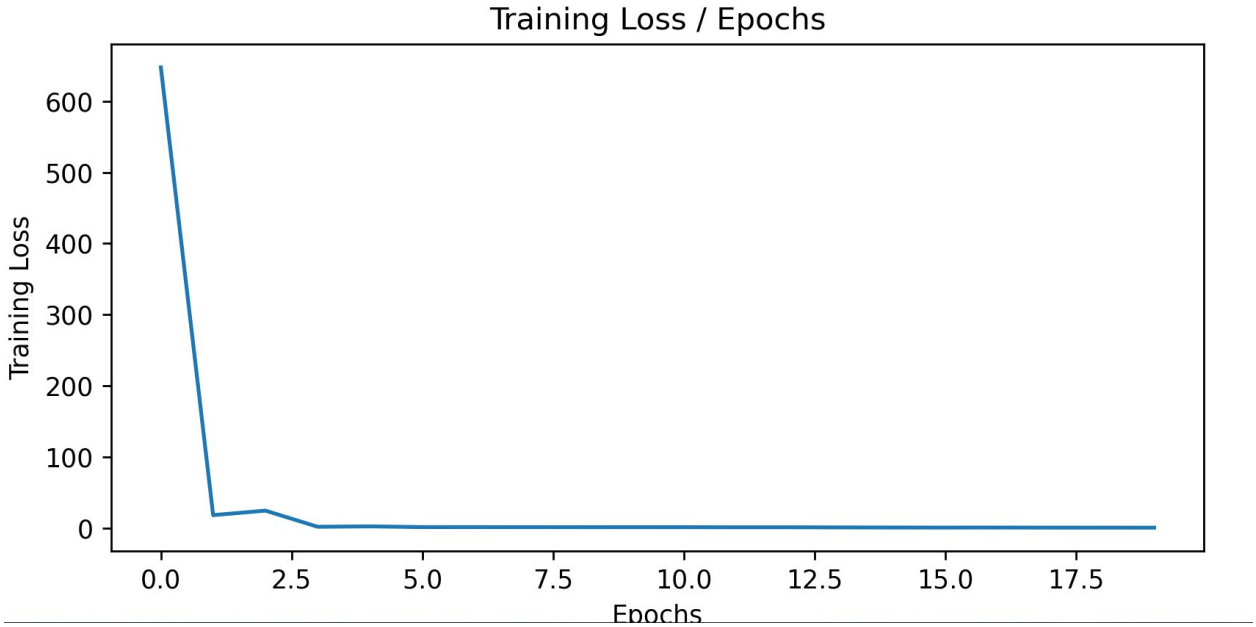
Batch Size: 32

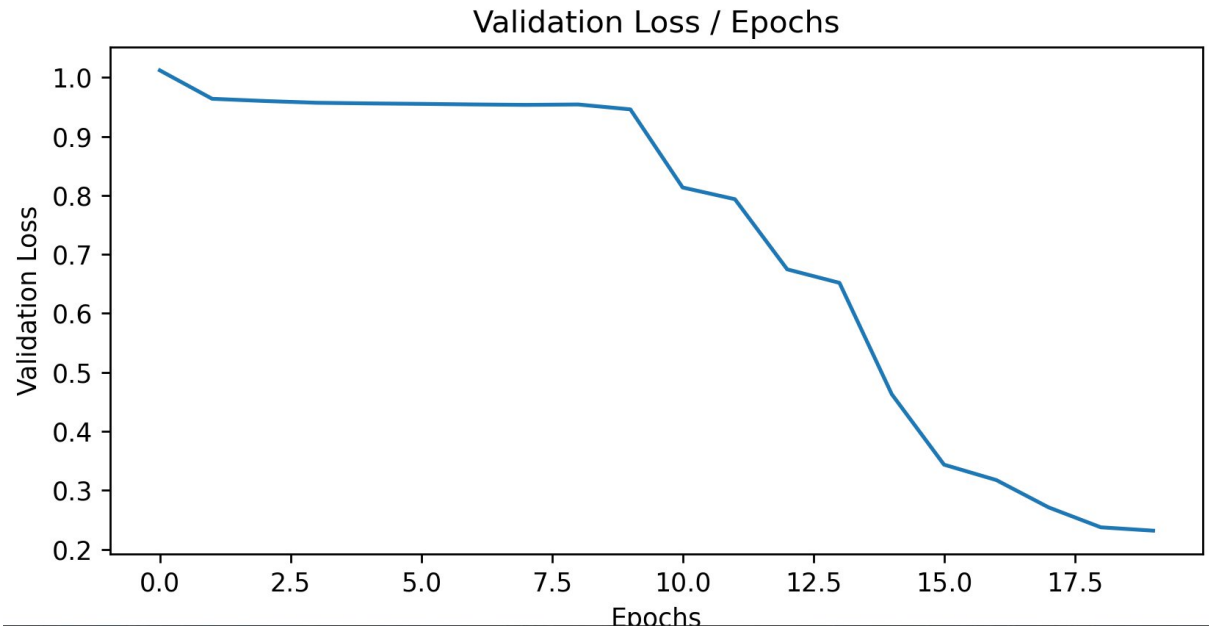
Drop Out Rate: 0.2

Optimization Method: Adam

7) Training, Validation and Test Errors

```
Epoch: 1/20, Training Loss: 647.5858, Validation Loss: 1.0121
Epoch: 2/20, Training Loss: 17.9258, Validation Loss: 0.9639
Epoch: 3/20, Training Loss: 24.2294, Validation Loss: 0.9602
Epoch: 4/20, Training Loss: 1.5483, Validation Loss: 0.9572
Epoch: 5/20, Training Loss: 2.1218, Validation Loss: 0.9561
Epoch: 6/20, Training Loss: 1.0658, Validation Loss: 0.9553
Epoch: 7/20, Training Loss: 1.0586, Validation Loss: 0.9543
Epoch: 8/20, Training Loss: 1.0346, Validation Loss: 0.9536
Epoch: 9/20, Training Loss: 0.9906, Validation Loss: 0.9543
Epoch: 10/20, Training Loss: 0.9471, Validation Loss: 0.9460
Epoch: 11/20, Training Loss: 0.9368, Validation Loss: 0.8135
Epoch: 12/20, Training Loss: 0.8551, Validation Loss: 0.7937
Epoch: 13/20, Training Loss: 0.8604, Validation Loss: 0.6747
Epoch: 14/20, Training Loss: 0.6408, Validation Loss: 0.6518
Epoch: 15/20, Training Loss: 0.5110, Validation Loss: 0.4630
Epoch: 16/20, Training Loss: 0.4026, Validation Loss: 0.3434
Epoch: 17/20, Training Loss: 0.4721, Validation Loss: 0.3173
Epoch: 18/20, Training Loss: 0.3409, Validation Loss: 0.2710
Epoch: 19/20, Training Loss: 0.2999, Validation Loss: 0.2372
Epoch: 20/20, Training Loss: 0.3062, Validation Loss: 0.2315
```





8) Confusion Matrix and Accuracy

```
Average Test Loss: 0.2882

Confusion Matrix
[[10881   323   125]
 [    3  3830    11]
 [    7   407  4073]]

Correct Predictions: 18784/19660
Accuracy: 95.54%
```

9) Project Github Link

https://github.com/zThyphon/Artifical-Intelligence-Term-Project