Network Intrusion Detection

Sofiane Alhoz

Overview

Goal :

Implement and evaluate the accuracy of different machine learning models for classification tasks using K-Fold Cross-Validation.

- Dataset
- For each model:
    - Perceptron
    - NN3
    - NN5
    - CNN2
    - CNN5

    We will tackle those points: Model view ⟶ Hyperparameter Tuning ⟶ K-Fold evaluation
- Conclusion

Dataset

Network intrusion detection involves identifying network flows that carry malicious traffic.
A flow is defined as a sequence of packets sharing the same attributes, such as:

Source IP, Destination IP, Source Port,

Destination Port, Protocol

Flow features refer to the statistical characteristics extracted from these flows.

76 features per file, 5 files

Total: 2830749 samples

...

Dataset

- Emulated Data
- Imbalanced data
- 76 features (flow statistics)

	Flow Type	Number of flows
1	BENIGN	2,273,097
2	DoS Hulk	231,073
3	PortScan	158,930
4	DDoS	128,027
5	DoS GoldenEye	10,293
6	FTP-Patator	7,938
7	SSH-Patator	5,897
8	DoS slowloris	5,796
9	DoS Slowhttptest	5,499
10	Bot	1,966
11	Web Attack Brute Force	1,507
12	Web Attack XSS	652
13	Infiltration	36
14	Web Attack Sql Injection	21
15	Heartbleed	11
	Total	**2,830,743**

Balance data:

balancing the data helps provide equal representation for all classes, preventing bias

```python
#We balance data as follows:
#1) oversample small classes so that their population/count is equal to mean_number_of_samples_per_class
#2) undersample large classes so that their count is equal to mean_number_of_samples_per_class
def balance_data(X,y,seed):
    np.random.seed(seed)
    unique,counts = np.unique(y,return_counts=True)
    mean_samples_per_class = int(round(np.mean(counts)))
    N,D = X.shape #(number of examples, number of features)
    new_X = np.empty((0,D))
    new_y = np.empty((0),dtype=int)
    for i,c in enumerate(unique):
        temp_x = X[y==c]
        indices = np.random.choice(temp_x.shape[0],mean_samples_per_class) # gets `mean_samples_per_class` indices of class `c`
        new_X = np.concatenate((new_X,temp_x[indices]),axis=0) # now we put new data into new_X
        temp_y = np.ones(mean_samples_per_class,dtype=int)*c
        new_y = np.concatenate((new_y,temp_y),axis=0)

    # in order to break class order in data we need shuffling
    indices = np.arange(new_y.shape[0])
    np.random.shuffle(indices)
    new_X =  new_X[indices,:]
    new_y = new_y[indices]
    return (new_X,new_y)
```

Normalize data:

normalizing the data standardizes the dataset, improving training stability and efficiency

```python
# normalization: Features are normalized to have a mean close to zero and comparable variations.
def normalize(data):
        data = data.astype(np.float32)

        eps = 1e-15

        mask = data==-1
        data[mask]=0
        mean_i = np.mean(data,axis=0)
        min_i = np.min(data,axis=0) #  to leave -1 (missing features) values as is and exclude in normilizing
        max_i = np.max(data,axis=0)

        r = max_i-min_i+eps
        data = (data-mean_i)/r  # zero centered

        #deal with missing features -1
        data[mask] = 0
        return data
```

Perceptron Classifier

```
class Perceptron(nn.Module):
    """
    Perceptron Regression Model
    """
    def __init__(self,input_dim,num_classes,device):
        super(Perceptron,self).__init__()
        self.classifier = nn.Linear(input_dim, num_classes).to(device)

    def forward(self,x):
        output= self.classifier(x)
        return output
```

Then I tried to identify the best combinations of hyperparameters from a predefined set of values, specifically for **learning rate** and **regularization strength**. These hyperparameters play a crucial role in determining the **convergence speed** and the model’s ability to **generalize without overfitting**. By testing different configurations, we can refine the model for better performance in later runs.

Perceptron Classifier

Configurations:

1. Batch_size = 1024
2. learning_rates = [,1e-4,1e-2,1e-1]
3. regularizations = [1e-6,1e-4,1e-3]
4. Epoch = 25

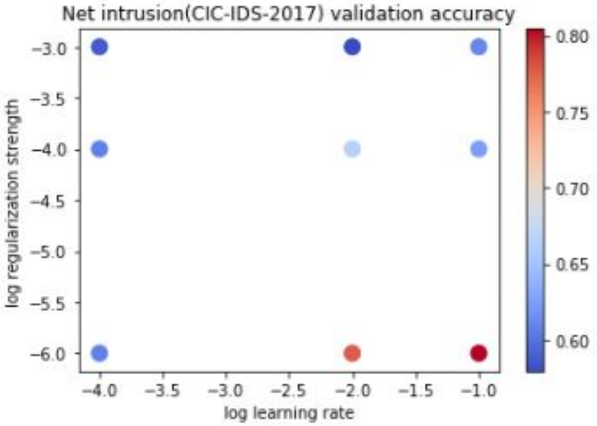
Training Accuracy on above hyperparameters:

```
[26]
...
    (0.0001, 1e-06) -> 60.83

    (0.0001, 0.0001) -> 60.38

    (0.0001, 0.01) -> 47.48

    (0.01, 1e-06) -> 80.50

    (0.01, 0.0001) -> 66.94

    (0.01, 0.01) -> 54.83

    (1.0, 1e-06) -> 78.96

    (1.0, 0.0001) -> 67.60

    (1.0, 0.01) -> 50.63
```

Results:

Best Learning rate is: [1e-1]

Best Regularization rate is : [1e-6]



KFold Evaluation

Split train 80% / test 20% might be problematic because attacks happens at different and precise moments

=> We need to use KFold Evaluation:

The model is trained **K times**, each time using **K-1 folds for training** and the **remaining fold for testing**.

5Fold Evaluation:

Itération 1: [Train | Train | Train | Train | Test]
Itération 2: [Train | Train | Train | Test | Train]
Itération 3: [Train | Train | Test | Train | Train]
Itération 4: [Train | Test | Train | Train | Train]
Itération 5: [Test | Train | Train | Train | Train]

This ensures that every data point is used for both training and testing, leading to **better generalization** and reducing the risk of **overfitting**.

```
Fold #0
INFO:models:Classifier ini
INFO:models:Starting train
DEBUG:models:Epoch [1/20],
DEBUG:models:Epoch [1/20],
DEBUG:models:Epoch [1/20],
DEBUG:models:Epoch [1/20],
DEBUG:models:Epoch [1/20],
DEBUG:models:Epoch [1/20],
DEBUG:models:Epoch [1/20],
DEBUG:models:Epoch [2/20],
DEBUG:models:Epoch [2/20],
DEBUG:models:Epoch [2/20],
DEBUG:models:Epoch [2/20],
DEBUG:models:Epoch [2/20],
DEBUG:models:Epoch [2/20],
DEBUG:models:Epoch [2/20],
DEBUG:models:Epoch [2/20],
DEBUG:models:Epoch [3/20],
DEBUG:models:Epoch [3/20],
DEBUG:models:Epoch [3/20],
DEBUG:models:Epoch [3/20],
DEBUG:models:Epoch [3/20],
DEBUG:models:Epoch [3/20],
DEBUG:models:Epoch [3/20],
DEBUG:models:Epoch [3/20],
...
DEBUG:models:Epoch [9/20],
DEBUG:models:Epoch [9/20],
DEBUG:models:Epoch [10/20]
WARNING:models:No improvem
```

Output is truncated. View as a sc

```
Loaded MachineLearningCVE/
balanced test acc:  73.825
```

Perceptron Classifier

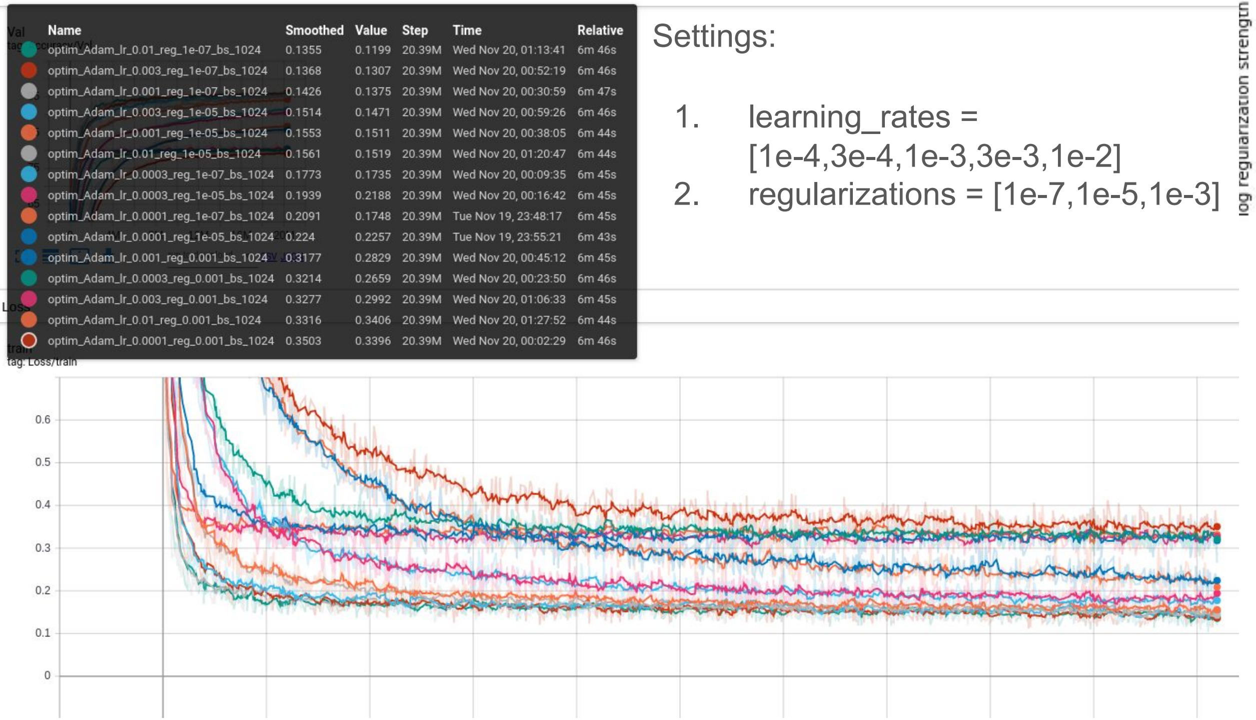
```
Fold #0
balanced test acc:  73.82532672957981
Fold #1
balanced test acc:  79.07317125426042
Fold #2
balanced test acc:  77.87223704729863
Fold #3
balanced test acc:  81.22136833071558
Fold #4
balanced test acc:  69.37087913280486
```

76.27

NN3 architecture

```
class Net3(nn.Module):
    """
    A neural network model consisting of multiple layers, used for classification tasks.

    Args:
    - input_dim (int): The dimensionality of the input data.
    - num_classes (int): The number of classes in the classification problem.
    - device (str): The device to be used for running the computations.

    Attributes:
    - input_dim (int): The dimensionality of the input data.
    - num_classes (int): The number of classes in the classification problem.
    - device (str): The device to be used for running the computations.

    Methods:
    - forward(x): Defines the forward pass of the model, taking in an input tensor x and returning the output tensor.

    """
    def __init__(self,input_dim,num_classes,device):
        super(Net3, self).__init__()
        # kernel
        print('building NN3')
        self.input_dim = input_dim
        self.num_classes = num_classes

        layers = []
        layers.append(nn.Dropout(p=0.1))
        layers.append(nn.Linear(self.input_dim, 128))
        layers.append(nn.BatchNorm1d(num_features=128))
        layers.append(nn.Dropout(p=0.3))
        layers.append(nn.Linear(128, 128))
        layers.append(nn.BatchNorm1d(num_features=128))
        layers.append(nn.Linear(128, self.num_classes))
        self.classifier = nn.Sequential(*layers).to(device)

    def forward(self, x):
        x = self.classifier(x)
        return x
```

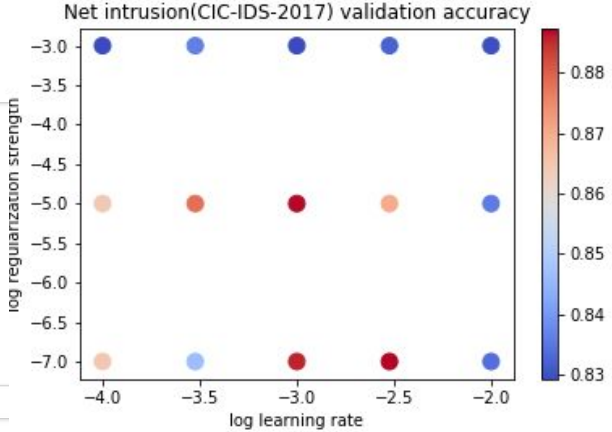
NN3: Hyperparameter Tuning 1st run



Name	Smoothed	Value	Step	Time	Relative
optim_Adam_lr_0.01_reg_1e-07_bs_1024	0.1355	0.1199	20.39M	Wed Nov 20, 01:13:41	6m 46s
optim_Adam_lr_0.003_reg_1e-07_bs_1024	0.1368	0.1307	20.39M	Wed Nov 20, 00:52:19	6m 46s
optim_Adam_lr_0.001_reg_1e-07_bs_1024	0.1426	0.1375	20.39M	Wed Nov 20, 00:30:59	6m 47s
optim_Adam_lr_0.003_reg_1e-05_bs_1024	0.1514	0.1471	20.39M	Wed Nov 20, 00:59:26	6m 46s
optim_Adam_lr_0.001_reg_1e-05_bs_1024	0.1553	0.1511	20.39M	Wed Nov 20, 00:38:05	6m 44s
optim_Adam_lr_0.01_reg_1e-05_bs_1024	0.1561	0.1519	20.39M	Wed Nov 20, 01:20:47	6m 44s
optim_Adam_lr_0.0003_reg_1e-07_bs_1024	0.1773	0.1735	20.39M	Wed Nov 20, 00:09:35	6m 45s
optim_Adam_lr_0.0003_reg_1e-05_bs_1024	0.1939	0.2188	20.39M	Wed Nov 20, 00:16:42	6m 45s
optim_Adam_lr_0.0001_reg_1e-07_bs_1024	0.2091	0.1748	20.39M	Tue Nov 19, 23:48:17	6m 45s
optim_Adam_lr_0.0001_reg_1e-05_bs_1024	0.224	0.2257	20.39M	Tue Nov 19, 23:55:21	6m 43s
optim_Adam_lr_0.001_reg_0.001_bs_1024	0.3177	0.2829	20.39M	Wed Nov 20, 00:45:12	6m 45s
optim_Adam_lr_0.0003_reg_0.001_bs_1024	0.3214	0.2659	20.39M	Wed Nov 20, 00:23:50	6m 46s
optim_Adam_lr_0.003_reg_0.001_bs_1024	0.3277	0.2992	20.39M	Wed Nov 20, 01:06:33	6m 45s
optim_Adam_lr_0.01_reg_0.001_bs_1024	0.3316	0.3406	20.39M	Wed Nov 20, 01:27:52	6m 44s
optim_Adam_lr_0.0001_reg_0.001_bs_1024	0.3503	0.3396	20.39M	Wed Nov 20, 00:02:29	6m 46s

Settings:

1. learning_rates = [1e-4,3e-4,1e-3,3e-3,1e-2]
2. regularizations = [1e-7,1e-5,1e-3]



Results:

1. Need more epochs:Because, as we can see for example in the loss curves, the model has not yet completed its learning process.
2. plot: Best val accuracy: 88.74
3. plot : Best parameters:(1e-3,1e-5)

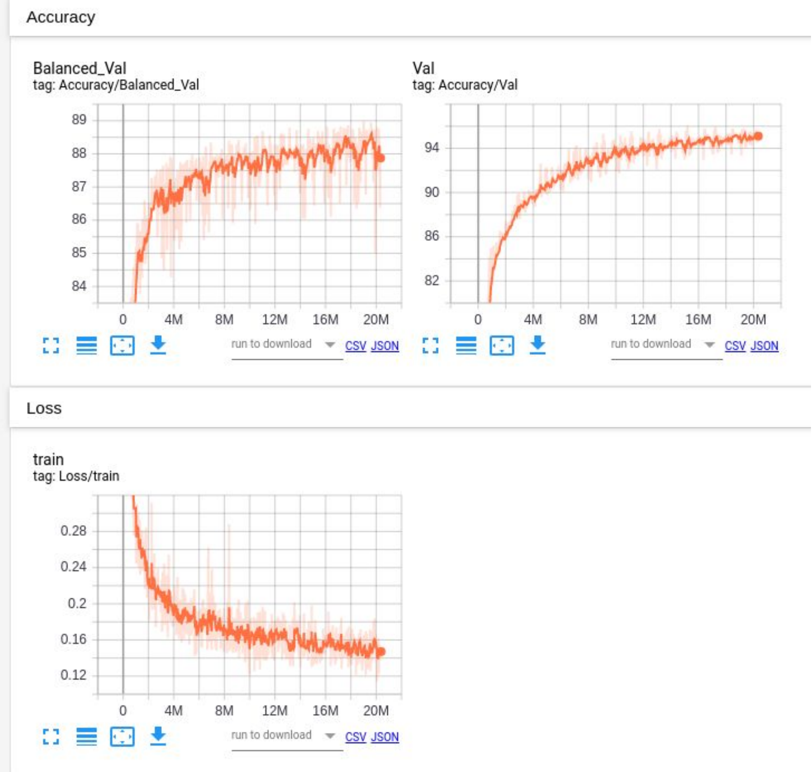
NN3: 2nd run with batchnorm

Settings:

1. learning_rates = [1e-3]
2. regularizations = [1e-5]
3. Num_epochs = 10
4. **Use of batchnorm**

Results:

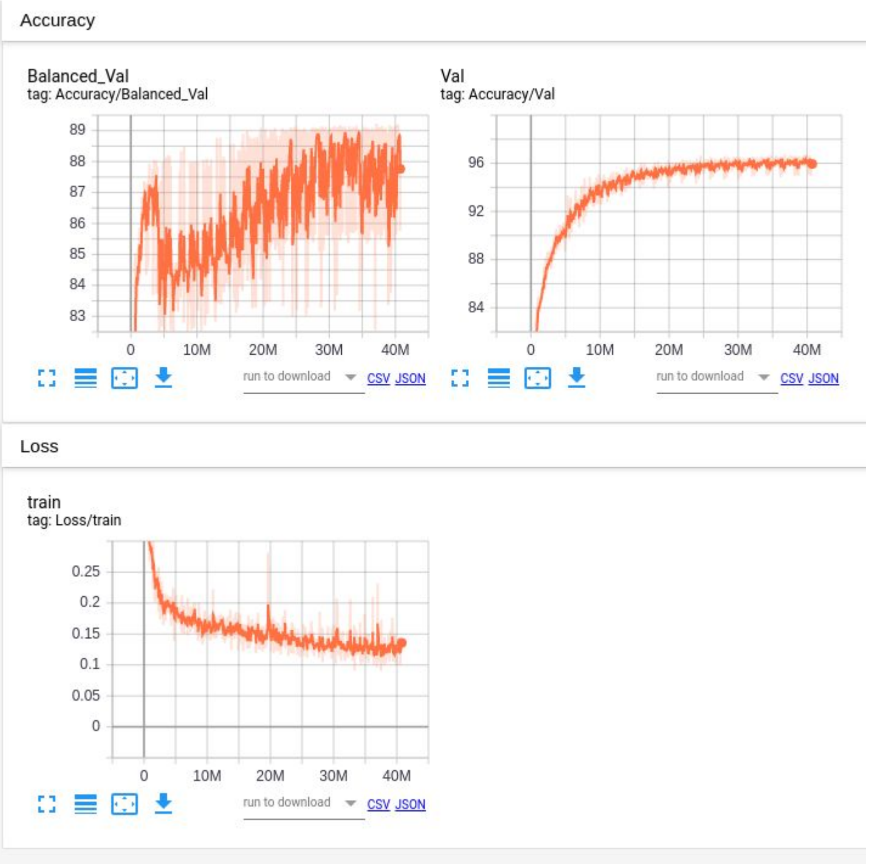
1. Best val accuracy:88.98, small improvement
2. Graph feedback: It is still learning. Need more epochs



NN3: 3rd run with batchnorm + dropout

Settings:

1. learning_rates = [1e-3]
2. regularizations = [1e-5]
3. Use of batchnorm
4. **Use of dropout (0.1,0.3)**
5. **num_epochs=20**



Results:

1. Best val accuracy:89.22, small improvement

NNN3: 5-Fold evaluation

Settings:

1. learning_rates = [1e-3]
2. regularizations = [1e-5]
3. Use batchnorm
4. Use dropout (0.1,0.3)
5. num_epochs=20

Results of the 5 fold::

1. Balanced Test acc: 85.61

```
---------------------------------------------
Fold #4
INFO:models:Classifier initialized with method nn3, input_dim 76, num_classes 15, num_
INFO:models:Starting training process...
building NN3
DEBUG:models:Epoch [1/20], Step [50/398], Loss: 1.0672
DEBUG:models:Epoch [1/20], Step [100/398], Loss: 0.6369
DEBUG:models:Epoch [1/20], Step [150/398], Loss: 0.5170
DEBUG:models:Epoch [1/20], Step [200/398], Loss: 0.4320
DEBUG:models:Epoch [1/20], Step [250/398], Loss: 0.4075
DEBUG:models:Epoch [1/20], Step [300/398], Loss: 0.3748
DEBUG:models:Epoch [1/20], Step [350/398], Loss: 0.3616
DEBUG:models:Epoch [2/20], Step [1/398], Loss: 0.3724
DEBUG:models:Epoch [2/20], Step [51/398], Loss: 0.3449
DEBUG:models:Epoch [2/20], Step [101/398], Loss: 0.3204
DEBUG:models:Epoch [2/20], Step [151/398], Loss: 0.3445
DEBUG:models:Epoch [2/20], Step [201/398], Loss: 0.3468
DEBUG:models:Epoch [2/20], Step [251/398], Loss: 0.3178
DEBUG:models:Epoch [2/20], Step [301/398], Loss: 0.3099
DEBUG:models:Epoch [2/20], Step [351/398], Loss: 0.2928
DEBUG:models:Epoch [3/20], Step [2/398], Loss: 0.3019
DEBUG:models:Epoch [3/20], Step [52/398], Loss: 0.2696
DEBUG:models:Epoch [3/20], Step [102/398], Loss: 0.2973
DEBUG:models:Epoch [3/20], Step [152/398], Loss: 0.2940
DEBUG:models:Epoch [3/20], Step [202/398], Loss: 0.2739
DEBUG:models:Epoch [3/20], Step [252/398], Loss: 0.2703
DEBUG:models:Epoch [3/20], Step [302/398], Loss: 0.2526
DEBUG:models:Epoch [3/20], Step [352/398], Loss: 0.2712
DEBUG:models:Epoch [4/20], Step [3/398], Loss: 0.2667
DEBUG:models:Epoch [4/20], Step [53/398], Loss: 0.2632
...
DEBUG:models:Epoch [7/20], Step [306/398], Loss: 0.2246
DEBUG:models:Epoch [7/20], Step [356/398], Loss: 0.2239
DEBUG:models:Epoch [8/20], Step [7/398], Loss: 0.2439
WARNING:models:No improvement in accuracy for 10 iterations.
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```
Loaded MachineLearningCVE/runs/nn3/Kfold-4th_run/optim_Adam_lr_0.001_reg_1e-05_bs_5120
balanced test acc:  86.24194198056313
85.73
```

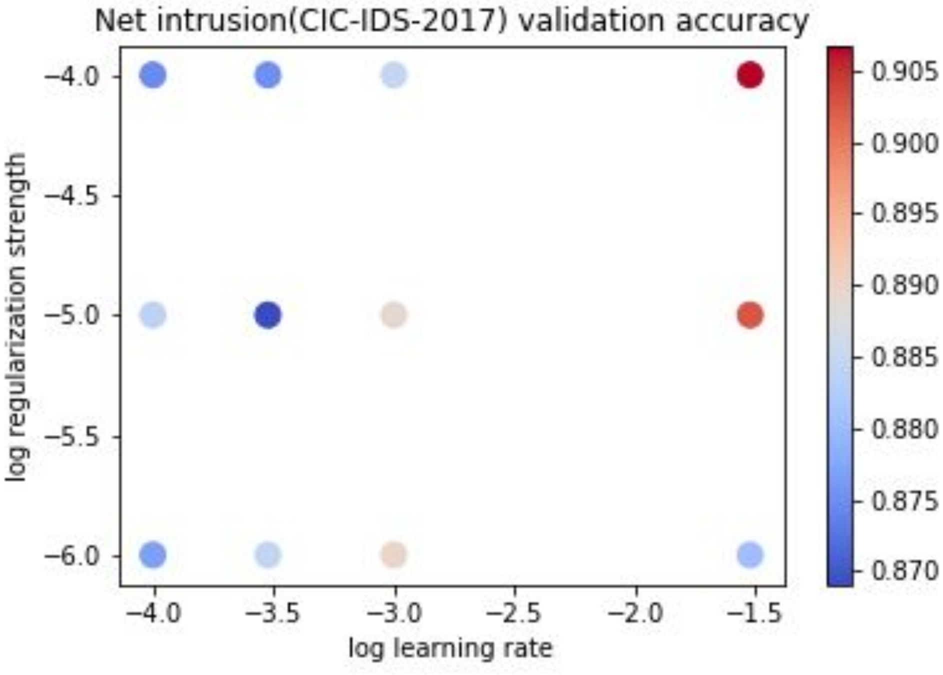
5 layer NN architecture

```python
class Net5(nn.Module):
    """
    A neural network model with 4 fully connected layers, using batch normalization and dropout for regularization.
    Args:
    - input_dim (int): the number of input features
    - num_classes (int): the number of output classes
    - device (torch.device): the device on which to run the model
    Attributes:
    - input_dim (int): the number of input features
    - num_classes (int): the number of output classes
    - model (nn.Sequential): the neural network architecture consisting of 4 fully connected layers
    """
    def __init__(self,input_dim,num_classes,device):
        super(Net5, self).__init__()
        # kernel
        self.input_dim = input_dim
        self.num_classes = num_classes
        layers = []
        layers.append(nn.Linear(input_dim,128))

        layers.append(nn.BatchNorm1d(128))
        layers.append(nn.ReLU(True))
        layers.append(nn.Linear(128,256))

        layers.append(nn.BatchNorm1d(256))
        layers.append(nn.Dropout(p=0.3))
        layers.append(nn.ReLU(True))
        layers.append(nn.Linear(256,256))

        layers.append(nn.BatchNorm1d(256))
        layers.append(nn.Dropout(p=0.4))
        layers.append(nn.ReLU(True))
        layers.append(nn.Linear(256,128))

        layers.append(nn.BatchNorm1d(128))
        layers.append(nn.Dropout(p=0.5))
        layers.append(nn.ReLU(True))
        layers.append(nn.Linear(128,num_classes))

        self.model = nn.Sequential(*layers).to(device)
    def forward(self, x):
        return self.model(x)
```

NN5: Hyperparameter Tuning: 1st run: Dropout higher layers

Configurations:

1. learning_rates = [3e-2,1e-3,3e-4,1e-4]
2. regularizations = [1e-6,1e-5,1e-4]
3. Batchnorm
4. Dropout prob: (0, 0, .3, .4, .5)
5. Epoch = 20



Results:

1. Val acc: 90.67

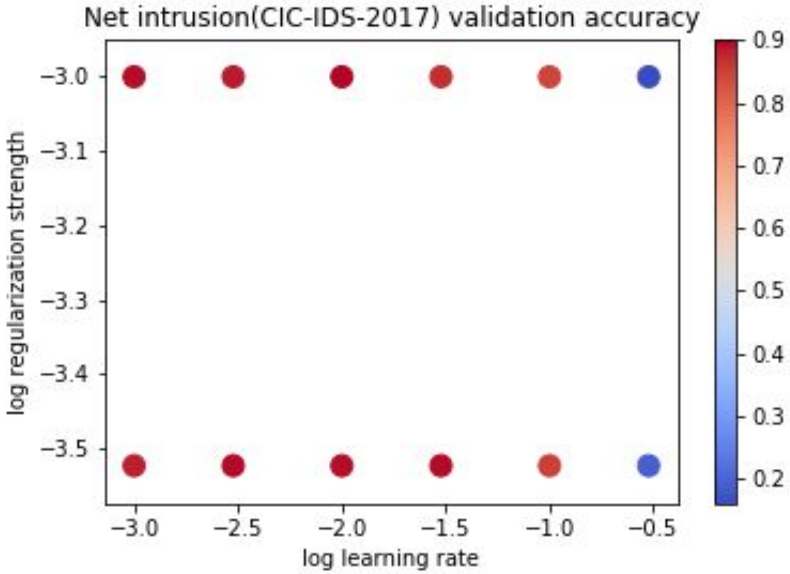
Feedback:

1. Bigger reg
2. Bigger lr
3. Larger epoch

NN5: Run 2: Dropout higher layers

Configurations:

1. learning_rates = [1e-3,3e-3,1e-2,3e-2,1e-1,3e-1]
2. regularizations = [3e-4,1e-3]
3. Batchnorm
4. Dropout prob: (0, 0, .3, .4, .5)
5. Epoch = 60



Results:

1. Val acc: 90.27

Feedback:

1. Search does not have to be dense
2. Bigger reg range
3. smaller lr
4. Conflicting to previous run, it could be because of small epoch number previously

NN5: Run 3: Dropout higher layers

Configurations:

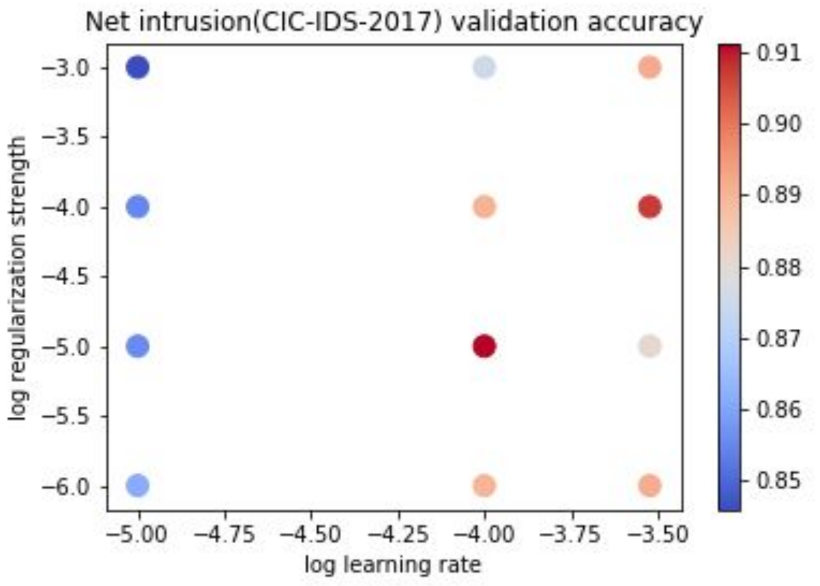
1. learning_rates = [1e-5,1e-4,3e-4]
2. regularizations = [1e-6,1e-5,1e-4,1e-3]
3. Batchnorm
4. Dropout prob: (0, 0, .3, .4, .5)
5. Epoch = 60

Results:

1. Val acc: 91.14,

Feedback:

1. Best param lr,reg = (1e-4,1e-5)



5-fold Evaluation: NN5

Settings

1. Best param lr,reg = (1e-4,1e-5)
2. Epochs = 60

Results:

1. 5-Fold Balanced Test acc: **85.63**

CNN2 architecture

```
class CNN2(nn.Module):
    """
    2-layer Convolutional Neural Network Model
    """
    def __init__(self,input_dim,num_classes,device):
        super(CNN2, self).__init__()
        # kernel
        self.input_dim = input_dim
        self.num_classes = num_classes

        conv_layers = []
        conv_layers.append(nn.Conv1d(in_channels=1,out_channels=64,kernel_size=3,padding=1)) # ;input_dim,64
        conv_layers.append(nn.BatchNorm1d(64))
        conv_layers.append(nn.ReLU(True))

        conv_layers.append(nn.Conv1d(in_channels=64,out_channels=128,kernel_size=3,padding=1)) #(input_dim,128)
        conv_layers.append(nn.BatchNorm1d(128))
        conv_layers.append(nn.ReLU(True))

        self.conv = nn.Sequential(*conv_layers).to(device)

        fc_layers = []
        fc_layers.append(nn.Linear(input_dim*128,num_classes))
        self.classifier = nn.Sequential(*fc_layers).to(device)

    def forward(self, x):
        batch_size, D = x.shape
        x = x.view(batch_size,1,D)

        x = self.conv(x)
        x = torch.flatten(x,1)
        x = self.classifier(x)
        return x
```

CNN2: run 1

Config::

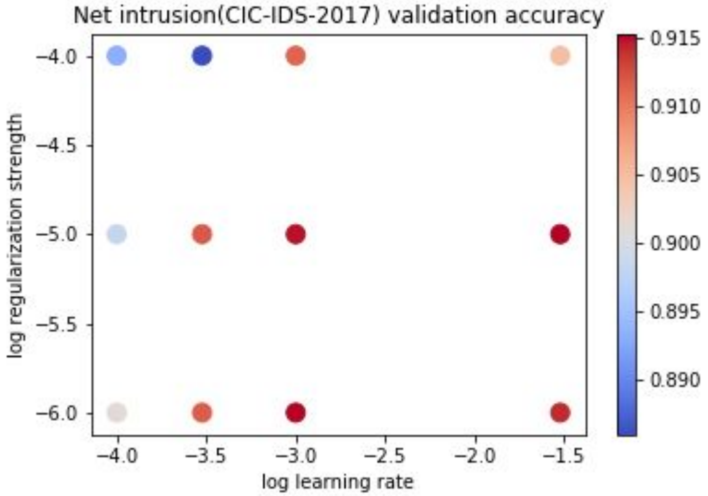
1. num_epochs = 60
2. learning_rates = [3e-2,1e-3,3e-4,1e-4]
3. regularizations = [1e-6,1e-5,1e-4]

acc:

1. Val:91.53

Feedback:

1. Lr 1e-3 is best, maybe need smaller reg



CNN2: run 2

Configs:

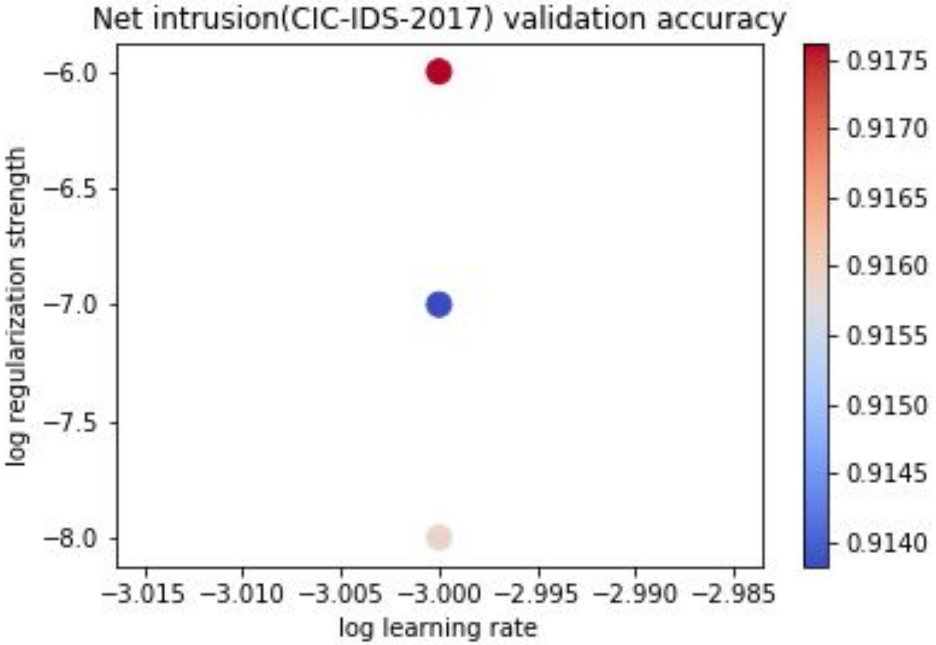
1. num_epochs = 60
2. learning_rates = [1e-3]
3. regularizations = [1e-8,1e-7,1e-6]

acc:

1. Val:91.76

Feedback:

1. Best reg is 1e-6



5-fold Evaluation: CNN2

Settings:

1. num_epochs = 60
2. learning_rates = [1e-3]
3. regularizations = [1e-6]

Results:

1. **5-Fold** acc: 88.14%

CNN5 architecture

```python
class CNN5(nn.Module):
    """
    5-layer Convolutional Neural Network Model
    """
    def __init__(self,input_dim,num_classes,device):
        super(CNN5, self).__init__()
        # kernel
        self.input_dim = input_dim
        self.num_classes = num_classes

        conv_layers = []
        conv_layers.append(nn.Conv1d(in_channels=1,out_channels=64,kernel_size=3,padding=1)) # ;input_dim,64
        conv_layers.append(nn.BatchNorm1d(64))
        conv_layers.append(nn.ReLU(True))

        conv_layers.append(nn.Conv1d(in_channels=64,out_channels=128,kernel_size=3,padding=1)) #(input_dim,128)
        conv_layers.append(nn.BatchNorm1d(128))
        conv_layers.append(nn.ReLU(True))

        conv_layers.append(nn.Conv1d(in_channels=128,out_channels=256,kernel_size=3,padding=1)) #(input_dim,128)
        conv_layers.append(nn.BatchNorm1d(256))
        conv_layers.append(nn.ReLU(True))

        conv_layers.append(nn.Conv1d(in_channels=256,out_channels=256,kernel_size=3,padding=1)) #(input_dim,128)
        conv_layers.append(nn.BatchNorm1d(256))
        conv_layers.append(nn.ReLU(True))

        conv_layers.append(nn.Conv1d(in_channels=256,out_channels=128,kernel_size=3,padding=1)) #(input_dim,128)
        conv_layers.append(nn.BatchNorm1d(128))
        conv_layers.append(nn.ReLU(True))

        self.conv = nn.Sequential(*conv_layers).to(device)
        fc_layers = []
        fc_layers.append(nn.Linear(input_dim*128,num_classes))
        self.classifier = nn.Sequential(*fc_layers).to(device)

    def forward(self, x):
        batch_size, D = x.shape
        x = x.view(batch_size,1,D)
        x = self.conv(x)
        x = torch.flatten(x,1)
        x = self.classifier(x)
        return x
```

5-fold Evaluation: CNN5

Settings:

1. num_epochs = 60
2. learning_rates = [1e-3]
3. regularizations = [1e-6]

Results:

1. **5-Fold** acc: 88.18%

Conclusion

Classifier	5-fold Accuracy
Perceptron	76.27
NN3	85.73
NN5	85.61
CNN2	88.14
CNN5	88.18

CNN5 achieved the best results

The Perceptron results confirm that **linear classification is insufficient** for complex intrusion detection tasks.

Also, the results shows that depth increase doesn’t make a big difference, contrary to the change from NN to CNN, which makes a big difference.