Prepocessing the data

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remoun
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Double-click (or enter) to edit

```
cols="""duration,
protocol_type,
service,
flag,
src_bytes,
dst_bytes,
land,
wrong_fragment,
urgent,
hot,
num_failed_logins,
logged_in,
num_compromised,
root_shell,
su_attempted,
num_root,
num_file_creations,
num_shells,
num_access_files,
num_outbound_cmds,
is_host_login,
is_guest_login,
count,
srv_count,
serror_rate,
srv_serror_rate,
rerror_rate,
srv_rerror_rate,
same_srv_rate,
diff_srv_rate,
srv_diff_host_rate,
dst_host_count,
dst_host_srv_count,
dst_host_same_srv_rate,
dst_host_diff_srv_rate,
dst_host_same_src_port_rate,
dst_host_srv_diff_host_rate,
dst_host_serror_rate,
dst_host_srv_serror_rate,
dst_host_rerror_rate,
dst_host_srv_rerror_rate"""

columns=[]
for c in cols.split(','):
    if(c.strip()):
       columns.append(c.strip())

columns.append('target')
print(columns)
print(len(columns))
```

```
['duration', 'protocol_type', 'service', 'flag', 'src_bytes', 'dst_bytes', 'land', 'wrong_fragment', 'urgent', 'hot', 'n
42
```

```
attacks_types = {
    'normal': 'normal',
    'back': 'dos',
    'buffer_overflow': 'u2r',
    'ftp_write': 'r2l',
    'guess_passwd': 'r2l',
    'imap': 'r2l',
    'ipsweep': 'probe',
    'land': 'dos',
    'loadmodule': 'u2r',
    'multihop': 'r2l',
    'neptune': 'dos',
    'nmap': 'probe',
    'perl': 'u2r',
    'phf': 'r2l',
    'pod': 'dos',
    'portsweep': 'probe',
    'rootkit': 'u2r',
    'satan': 'probe',
    'smurf': 'dos',
    'spy': 'r2l',
    'teardrop': 'dos',
    'warezclient': 'r2l',
    'warezmaster': 'r2l',
}
```

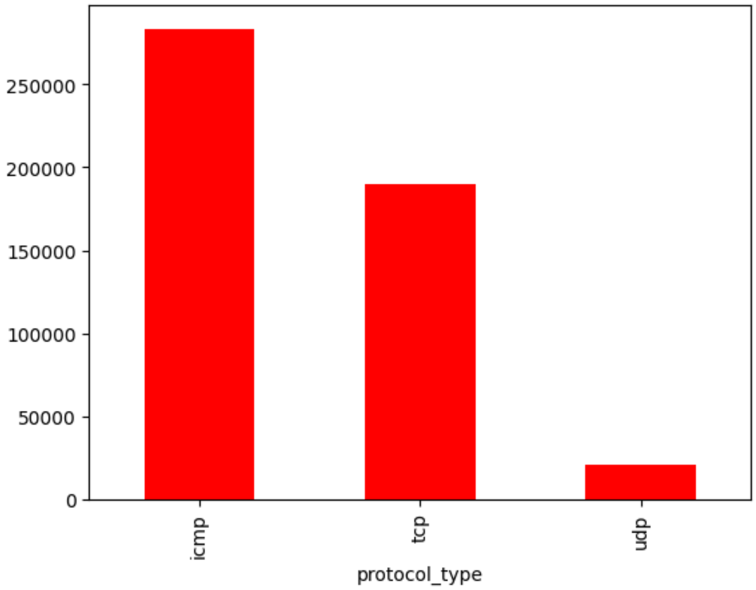
```
path = "../content/kddcup.data_10_percent.gz"
df = pd.read_csv(path,names=columns)

#Adding Attack Type column
df['Attack Type'] = df.target.apply(lambda r:attacks_types[r[:-1]])

df.head()

df['Attack Type'].value_counts()
```

```
Attack Type
dos       391458
normal     97278
probe       4107
r2l         1126
u2r           52
Name: count, dtype: int64
```

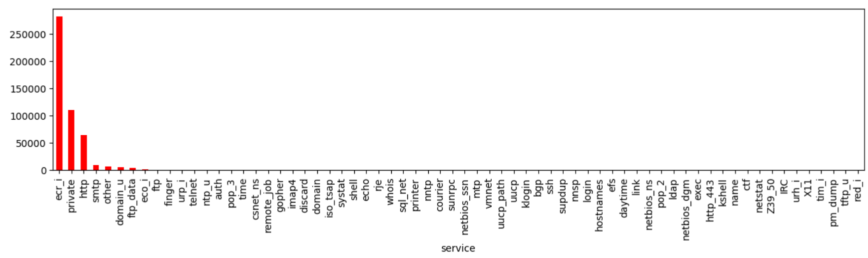
Visualizing the data

```
def bar_graph(feature):
    df[feature].value_counts().plot(kind="bar",color="red")
```

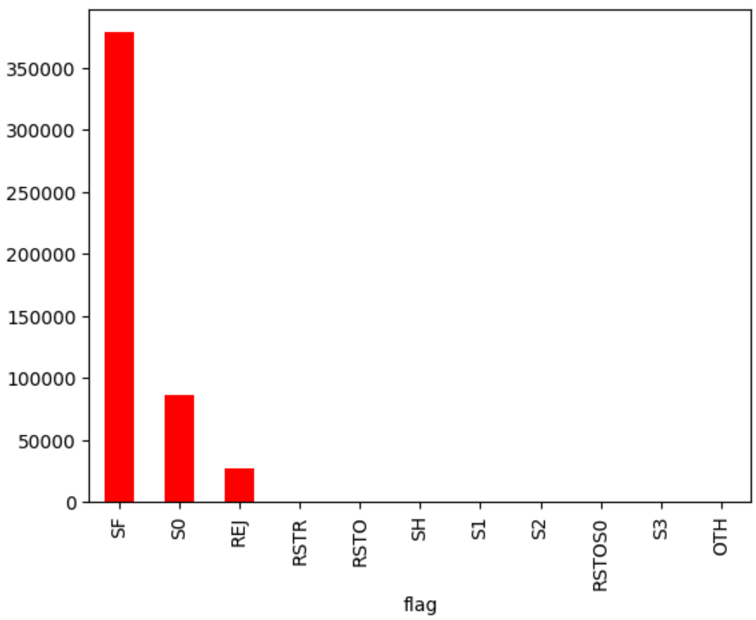
```
bar_graph('protocol_type')
```



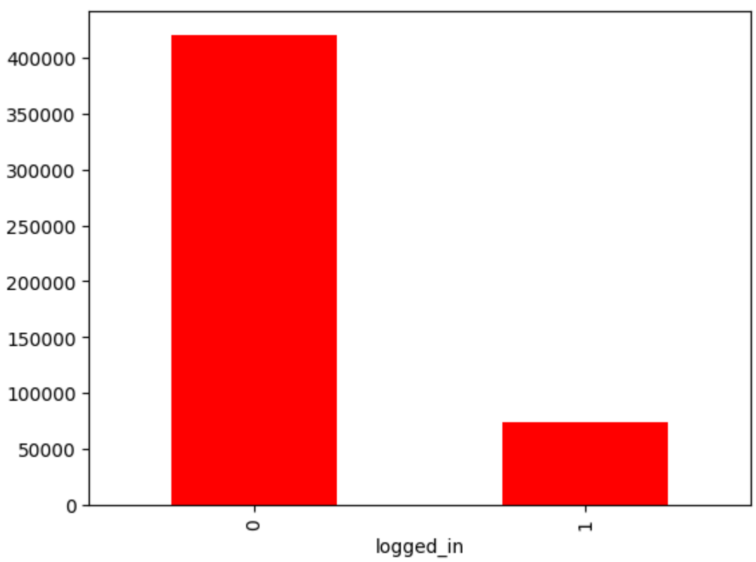
```
plt.figure(figsize=(15,3))
bar_graph('service')
```



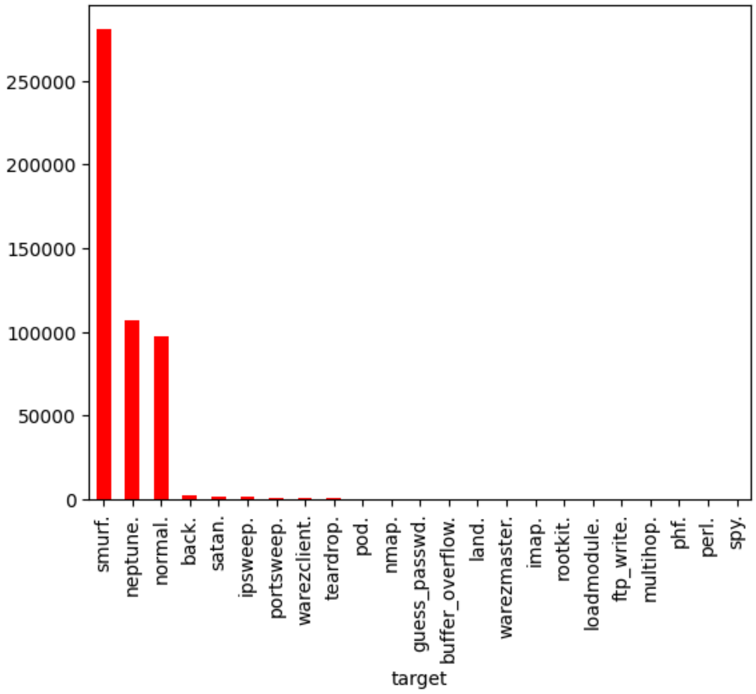

```
bar_graph('flag')
```



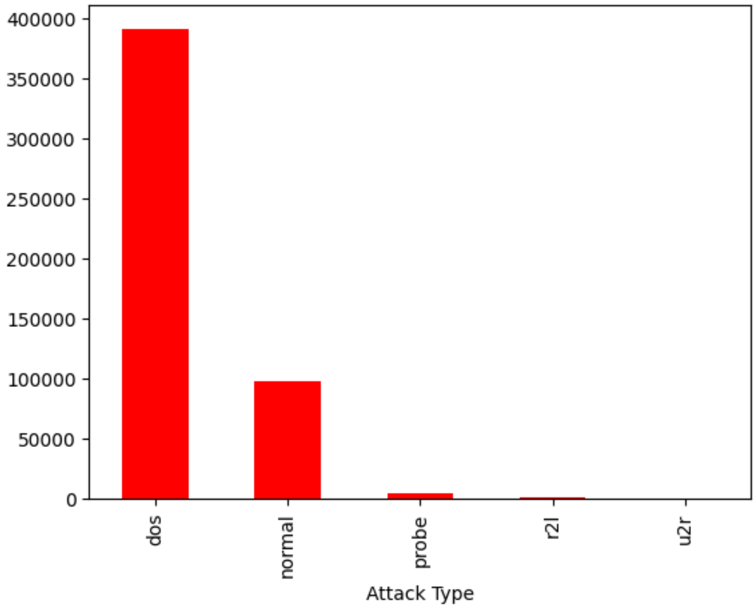

```
bar_graph('logged_in')
```




```
bar_graph('target')
```




```
bar_graph('Attack Type')
```



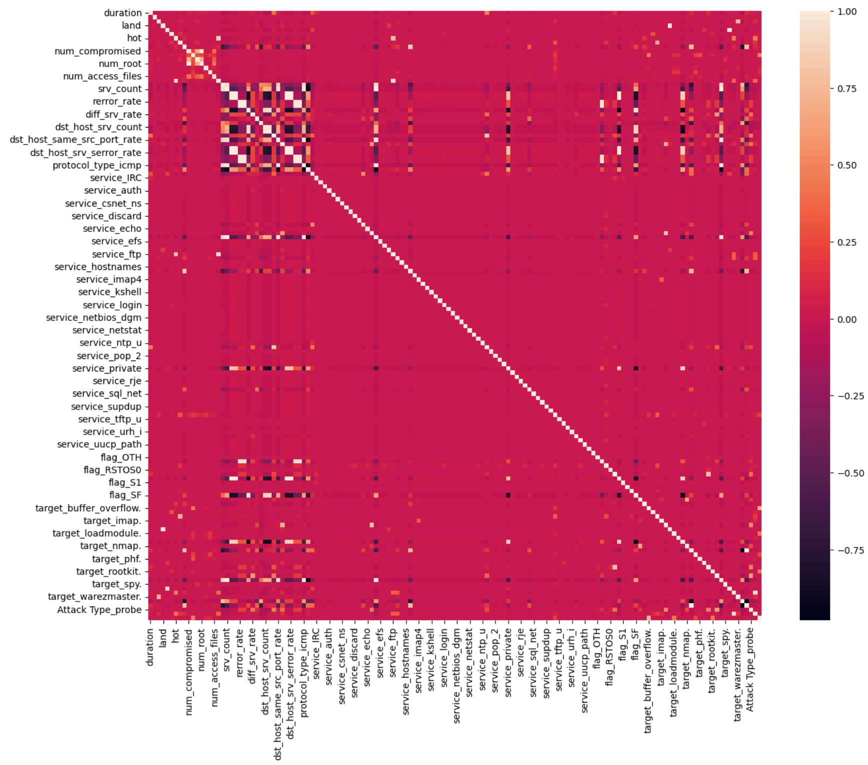

```
df.columns
```

```
Index(['duration', 'protocol_type', 'service', 'flag', 'src_bytes',
       'dst_bytes', 'land', 'wrong_fragment', 'urgent', 'hot',
       'num_failed_logins', 'logged_in', 'num_compromised', 'root_shell',
       'su_attempted', 'num_root', 'num_file_creations', 'num_shells',
       'num_access_files', 'num_outbound_cmds', 'is_host_login',
       'is_guest_login', 'count', 'srv_count', 'serror_rate',
       'srv_serror_rate', 'rerror_rate', 'srv_rerror_rate', 'same_srv_rate',
       'diff_srv_rate', 'srv_diff_host_rate', 'dst_host_count',
       'dst_host_srv_count', 'dst_host_same_srv_rate',
       'dst_host_diff_srv_rate', 'dst_host_same_src_port_rate',
       'dst_host_srv_diff_host_rate', 'dst_host_serror_rate',
       'dst_host_srv_serror_rate', 'dst_host_rerror_rate',
       'dst_host_srv_rerror_rate', 'target', 'Attack Type'],
      dtype='object')
```

Removing highly correlated columns

- List item
- List item

```
df = df.dropna(axis = 'columns') # drop columns with NaN

df = df[[col for col in df if df[col].nunique() > 1]] # keep columns where there are more than 1 unique values

# corr = df.corr()
# Convert categorical variables into numerical representations using one-hot encoding
df_encoded = pd.get_dummies(df)

# Calculate the correlation matrix
corr_matrix = df_encoded.corr()

plt.figure(figsize=(15,12))

sns.heatmap(corr_matrix)

plt.show()
```




```
df.head()
```

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fr
0	0	tcp	http	SF	181	5450	0	
1	0	tcp	http	SF	239	486	0	
2	0	tcp	http	SF	235	1337	0	
3	0	tcp	http	SF	219	1337	0	
4	0	tcp	http	SF	217	2032	0	

5 rows × 41 columns

```
#This variable is highly correlated with num_compromised and should be ignored for analysis.
#(Correlation = 0.9938277978738366)
df.drop('num_root',axis = 1,inplace = True)

#This variable is highly correlated with serror_rate and should be ignored for analysis.
#(Correlation = 0.9983615072725952)
df.drop('srv_serror_rate',axis = 1,inplace = True)

#This variable is highly correlated with rerror_rate and should be ignored for analysis.
#(Correlation = 0.9947309539817937)
df.drop('srv_rerror_rate',axis = 1, inplace=True)

#This variable is highly correlated with srv_serror_rate and should be ignored for analysis.
#(Correlation = 0.9993041091850098)
df.drop('dst_host_srv_serror_rate',axis = 1, inplace=True)

#This variable is highly correlated with rerror_rate and should be ignored for analysis.
#(Correlation = 0.9869947924956001)
df.drop('dst_host_serror_rate',axis = 1, inplace=True)

#This variable is highly correlated with srv_rerror_rate and should be ignored for analysis.
#(Correlation = 0.9821663427308375)
df.drop('dst_host_rerror_rate',axis = 1, inplace=True)

#This variable is highly correlated with rerror_rate and should be ignored for analysis.
#(Correlation = 0.9851995540751249)
df.drop('dst_host_srv_rerror_rate',axis = 1, inplace=True)

#This variable is highly correlated with dst_host_srv_count and should be ignored for analysis.
#(Correlation = 0.9736854572953938)
df.drop('dst_host_same_srv_rate',axis = 1, inplace=True)
```

```
df.head()
```

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fr
0	0	tcp	http	SF	181	5450	0	
1	0	tcp	http	SF	239	486	0	
2	0	tcp	http	SF	235	1337	0	
3	0	tcp	http	SF	219	1337	0	
4	0	tcp	http	SF	217	2032	0	

5 rows × 33 columns

Label encoding the features

```
#protocol_type feature mapping
pmap = {'icmp':0,'tcp':1,'udp':2}
df['protocol_type'] = df['protocol_type'].map(pmap)
```

```
#flag feature mapping
fmap = {'SF':0,'S0':1,'REJ':2,'RSTR':3,'RSTO':4,'SH':5 ,'S1':6 ,'S2':7,'RSTOS0':8,'S3':9 ,'OTH':10}
df['flag'] = df['flag'].map(fmap)
```

```
#attack type feature mapping
amap = {'dos':0,'normal':1,'probe':2,'r2l':3,'u2r':4}
df['Attack Type'] = df['Attack Type'].map(amap)
```

```
df.drop('service',axis = 1,inplace= True)
```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import accuracy_score
```

```
import tensorflow as tf
from keras.models import Sequential, Model
from keras.layers import Dense, Conv1D, MaxPooling1D, Flatten, Dropout, Input, Concatenate, Add
```

```
df = df.drop(['target',], axis=1)
print(df.shape)

# Target variable and train set
Y = df[['Attack Type']]
X = df.drop(['Attack Type',], axis=1)

sc = MinMaxScaler()
X = sc.fit_transform(X)

# Split test and train data
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.33,
                                                    random_state=42)
print(X_train.shape, X_test.shape)
print(Y_train.shape, Y_test.shape)
```

```
(494021, 31)
(330994, 30) (163027, 30)
(330994, 1) (163027, 1)
```

```
df.to_csv("ids.csv", index=False)
```

```
pd.read_csv("ids.csv")
```

	duration	protocol_type	flag	src_bytes	dst_bytes	land	wrong_fragme
0	0	1	0	181	5450	0	
1	0	1	0	239	486	0	
2	0	1	0	235	1337	0	
3	0	1	0	219	1337	0	
4	0	1	0	217	2032	0	
...	...	...	...	...	...	...	
494016	0	1	0	310	1881	0	
494017	0	1	0	282	2286	0	
494018	0	1	0	203	1200	0	
494019	0	1	0	291	1200	0	
494020	0	1	0	219	1234	0	

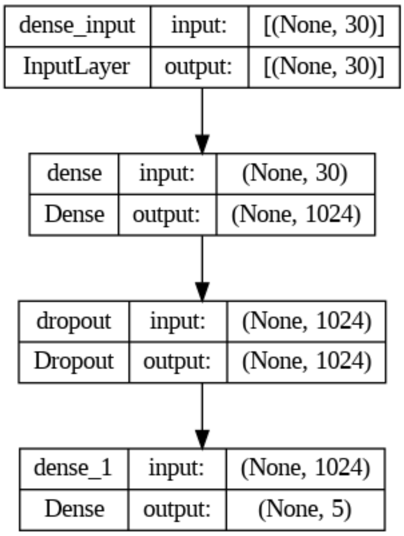
494021 rows × 31 columns

Shallow Neural Network

```
shallow_model = Sequential([
    Dense(1024, input_dim=30, activation='relu'),
    Dropout(0.01),
    Dense(5, activation='softmax')
])
```

```
shallow_model.compile(loss ='sparse_categorical_crossentropy', optimizer = 'adam', metrics = ['accuracy'])
```

```
tf.keras.utils.plot_model(shallow_model, to_file="shallow_model.png", show_shapes=True)
```




```
shallow_model.fit(X_train, Y_train.values.ravel(), epochs=10, batch_size=32)
```

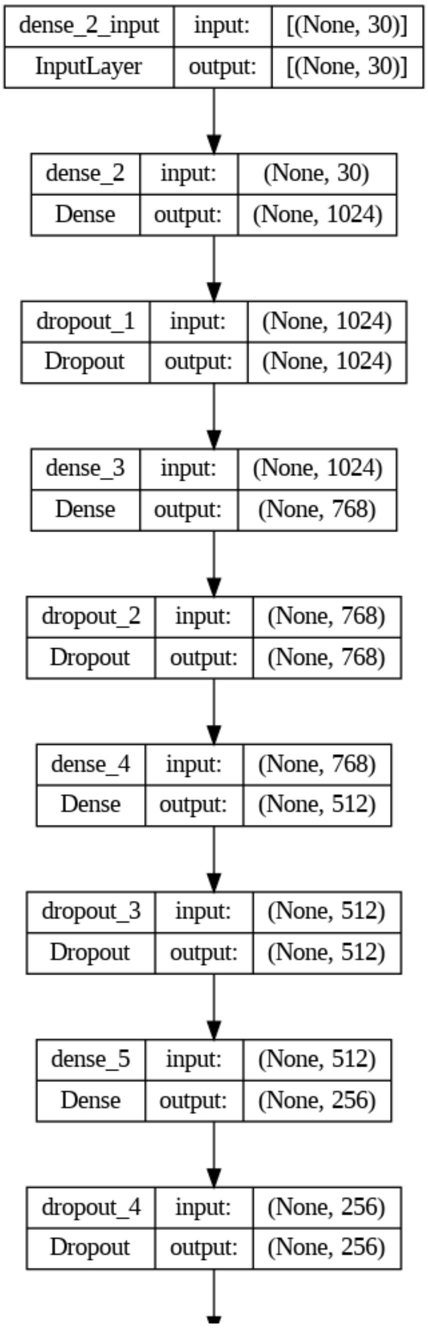
```
Epoch 1/10
10344/10344 [==============================] - 31s 3ms/step - loss: 0.0133 - accuracy: 0.9964
Epoch 2/10
10344/10344 [==============================] - 30s 3ms/step - loss: 0.0047 - accuracy: 0.9987
Epoch 3/10
10344/10344 [==============================] - 30s 3ms/step - loss: 0.0040 - accuracy: 0.9989
Epoch 4/10
10344/10344 [==============================] - 29s 3ms/step - loss: 0.0038 - accuracy: 0.9990
Epoch 5/10
10344/10344 [==============================] - 30s 3ms/step - loss: 0.0036 - accuracy: 0.9990
Epoch 6/10
10344/10344 [==============================] - 31s 3ms/step - loss: 0.0035 - accuracy: 0.9991
Epoch 7/10
10344/10344 [==============================] - 29s 3ms/step - loss: 0.0033 - accuracy: 0.9991
Epoch 8/10
10344/10344 [==============================] - 29s 3ms/step - loss: 0.0031 - accuracy: 0.9992
Epoch 9/10
10344/10344 [==============================] - 29s 3ms/step - loss: 0.0032 - accuracy: 0.9992
Epoch 10/10
10344/10344 [==============================] - 34s 3ms/step - loss: 0.0031 - accuracy: 0.9992
<keras.src.callbacks.History at 0x7cee8d6272e0>
```

Deep Neural Network

```
deep_model = Sequential([
    Dense(1024, input_dim=30, activation='relu'),
    Dropout(0.01),
    Dense(768, activation='relu'),
    Dropout(0.01),
    Dense(512, activation='relu'),
    Dropout(0.01),
    Dense(256, activation='relu'),
    Dropout(0.01),
    Dense(128, activation='relu'),
    Dropout(0.01),
    Dense(5, activation='softmax')
])
```

```
deep_model.compile(loss ='sparse_categorical_crossentropy', optimizer = 'adam', metrics = ['accuracy'])
```

```
tf.keras.utils.plot_model(deep_model, to_file="deep_model.png", show_shapes=True)
```

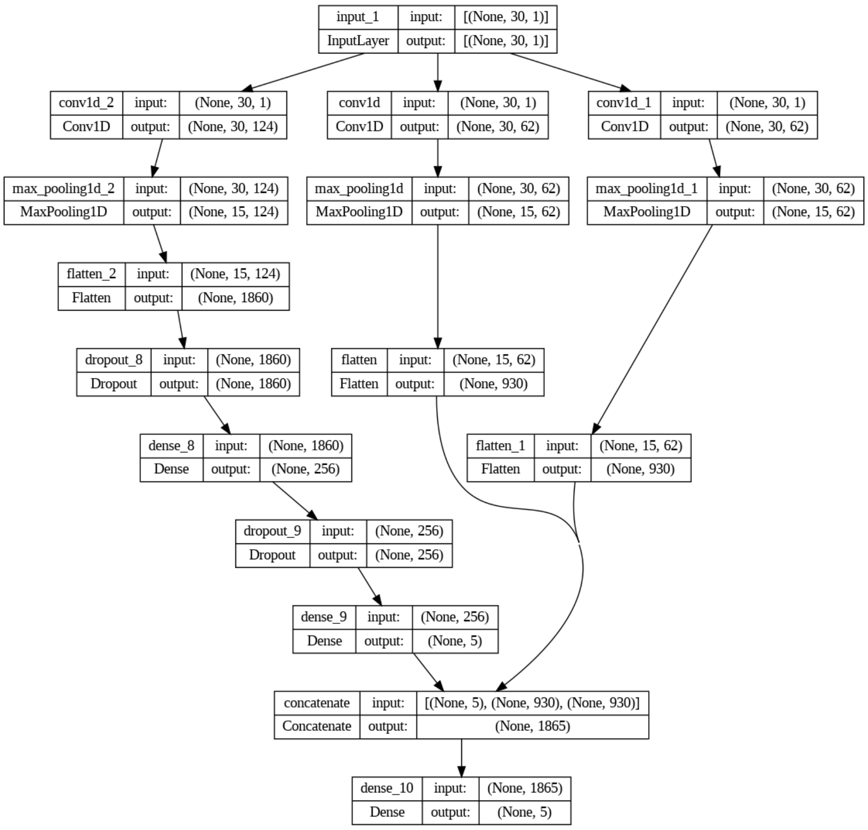



```
deep_model.fit(X_train, Y_train.values.ravel(), epochs=10, batch_size=32)
```

```
Epoch 1/10
10344/10344 [==============================] - 254s 24ms/step - loss: 0.0161 - accuracy: 0.9963
Epoch 2/10
10344/10344 [==============================] - 249s 24ms/step - loss: 0.0083 - accuracy: 0.9981
Epoch 3/10
10344/10344 [==============================] - 247s 24ms/step - loss: 0.0077 - accuracy: 0.9984
Epoch 4/10
10344/10344 [==============================] - 248s 24ms/step - loss: 0.0072 - accuracy: 0.9986
Epoch 5/10
10344/10344 [==============================] - 249s 24ms/step - loss: 0.0063 - accuracy: 0.9987
Epoch 6/10
10344/10344 [==============================] - 243s 24ms/step - loss: 0.0064 - accuracy: 0.9987
Epoch 7/10
10344/10344 [==============================] - 242s 23ms/step - loss: 0.0067 - accuracy: 0.9987
Epoch 8/10
10344/10344 [==============================] - 244s 24ms/step - loss: 0.0060 - accuracy: 0.9987
Epoch 9/10
10344/10344 [==============================] - 244s 24ms/step - loss: 0.0071 - accuracy: 0.9988
Epoch 10/10
10344/10344 [==============================] - 245s 24ms/step - loss: 0.0057 - accuracy: 0.9989
<keras.src.callbacks.History at 0x7cee8d592290>
```

Convolutional Neural Network

```
# cnn_model = Sequential([
#     Conv1D(64, 3, padding="same", activation="relu", input_shape=(30,1)),
#     MaxPooling1D(pool_size=(2)),
#     Flatten(),
#     Dense(128, activation="relu"),
#     Dropout(0.5),
#     Dense(5, activation="softmax")
# ])

inputs = Input(shape=(30, 1))
y = Conv1D(62, 3, padding="same", activation="relu", input_shape=(30,1))(inputs)
y = MaxPooling1D(pool_size=(2))(y)
y1 = Flatten()(y)

y = Dropout(0.5)(y)
y = Conv1D(62, 3, padding="same", activation="relu", input_shape=(30,1))(inputs)
y = MaxPooling1D(pool_size=(2))(y)
y2 = Flatten()(y)

y = Dropout(0.5)(y)
y = Conv1D(124, 3, padding="same", activation="relu", input_shape=(30,1))(inputs)
y = MaxPooling1D(pool_size=(2))(y)
y = Flatten()(y)
y = Dropout(0.5)(y)
y = Dense(256, activation="relu")(y)
y = Dropout(0.5)(y)
y = Dense(5, activation='softmax')(y)

y = Concatenate()([y, y1, y2])

outputs = Dense(5, activation='softmax')(y)
cnn_model = Model(inputs=inputs, outputs=outputs)


cnn_model.compile(loss ='sparse_categorical_crossentropy', optimizer = 'adam', metrics = ['accuracy'])


tf.keras.utils.plot_model(cnn_model, to_file="cnn_model.png", show_shapes=True)
```




```
cnn_model.fit(X_train.reshape((-1,30,1)), Y_train.values.ravel(), epochs=10, batch_size=32)
```

```
Epoch 1/10
10344/10344 [==============================] - 135s 13ms/step - loss: 0.0243 - accuracy: 0.9937
Epoch 2/10
10344/10344 [==============================] - 135s 13ms/step - loss: 0.0090 - accuracy: 0.9977
Epoch 3/10
10344/10344 [==============================] - 136s 13ms/step - loss: 0.0076 - accuracy: 0.9980
Epoch 4/10
10344/10344 [==============================] - 134s 13ms/step - loss: 0.0070 - accuracy: 0.9982
Epoch 5/10
10344/10344 [==============================] - 138s 13ms/step - loss: 0.0065 - accuracy: 0.9983
Epoch 6/10
10344/10344 [==============================] - 135s 13ms/step - loss: 0.0062 - accuracy: 0.9984
Epoch 7/10
10344/10344 [==============================] - 132s 13ms/step - loss: 0.0060 - accuracy: 0.9984
Epoch 8/10
10344/10344 [==============================] - 136s 13ms/step - loss: 0.0058 - accuracy: 0.9985
Epoch 9/10
10344/10344 [==============================] - 134s 13ms/step - loss: 0.0056 - accuracy: 0.9986
Epoch 10/10
10344/10344 [==============================] - 132s 13ms/step - loss: 0.0054 - accuracy: 0.9986
<keras.src.callbacks.History at 0x7cee1d904b20>
```

Testing the neural network

```
shallow_preds_train = shallow_model.predict(X_train)
shallow_test = shallow_model.predict(X_test)
```

```
10344/10344 [==============================] - 16s 2ms/step
5095/5095 [==============================] - 7s 1ms/step
```

```
deep_preds_train = deep_model.predict(X_train)
deep_test = deep_model.predict(X_test)
```

```
10344/10344 [==============================] - 53s 5ms/step
5095/5095 [==============================] - 25s 5ms/step
```

```
cnn_preds_train = cnn_model.predict(X_train.reshape((-1,30,1)))
cnn_test = cnn_model.predict(X_test.reshape((-1,30,1)))
```

```
10344/10344 [==============================] - 41s 4ms/step
5095/5095 [==============================] - 20s 4ms/step
```

```
print("SHALLOW NEURAL NETWORK")
print("Training Accuracy:", accuracy_score(Y_train, np.argmax(shallow_preds_train, axis=1)))
print("Testing Accuracy:", accuracy_score(Y_test, np.argmax(shallow_test, axis=1)))
```

```
SHALLOW NEURAL NETWORK
Training Accuracy: 0.9993655474117356
Testing Accuracy: 0.9991841842148846
```

```
print("DEEP NEURAL NETWORK")
print("Training Accuracy:", accuracy_score(Y_train, np.argmax(deep_preds_train, axis=1)))
print("Testing Accuracy:", accuracy_score(Y_test, np.argmax(deep_test, axis=1)))
```

```
DEEP NEURAL NETWORK
Training Accuracy: 0.9993021021529092
Testing Accuracy: 0.9991596484017985
```

```
print("CONVOLUTIONAL NEURAL NETWORK")
print("Training Accuracy:", accuracy_score(Y_train, np.argmax(cnn_preds_train, axis=1)))
print("Testing Accuracy:", accuracy_score(Y_test, np.argmax(cnn_test, axis=1)))
```

```
CONVOLUTIONAL NEURAL NETWORK
Training Accuracy: 0.9987703704598875
Testing Accuracy: 0.9986321284204457
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report

# Function to print metrics for train and test sets
def print_metrics(model_name, y_true_train, y_pred_train, y_true_test, y_pred_test):
    print(f"\nMetrics for {model_name} Model:")

    # Train set
    print("\nTrain Set:")
    print("Accuracy:", accuracy_score(y_true_train, y_pred_train))
    print("Precision:", precision_score(y_true_train, y_pred_train, average='weighted'))
    print("Recall:", recall_score(y_true_train, y_pred_train, average='weighted'))
    print("F1-Score:", f1_score(y_true_train, y_pred_train, average='weighted'))
    print("Classification Report:\n", classification_report(y_true_train, y_pred_train))

    # Test set
    print("\nTest Set:")
    print("Accuracy:", accuracy_score(y_true_test, y_pred_test))
    print("Precision:", precision_score(y_true_test, y_pred_test, average='weighted'))
    print("Recall:", recall_score(y_true_test, y_pred_test, average='weighted'))
    print("F1-Score:", f1_score(y_true_test, y_pred_test, average='weighted'))
    print("Classification Report:\n", classification_report(y_true_test, y_pred_test))

# Predictions from the models
shallow_preds_train_labels = np.argmax(shallow_preds_train, axis=1)
shallow_test_labels = np.argmax(shallow_test, axis=1)

deep_preds_train_labels = np.argmax(deep_preds_train, axis=1)
deep_test_labels = np.argmax(deep_test, axis=1)

cnn_preds_train_labels = np.argmax(cnn_preds_train, axis=1)
cnn_test_labels = np.argmax(cnn_test, axis=1)

# Print metrics for each model
print_metrics("Shallow Neural Network", Y_train, shallow_preds_train_labels, Y_test, shallow_test_labels)
print_metrics("Deep Neural Network", Y_train, deep_preds_train_labels, Y_test, deep_test_labels)
print_metrics("Convolutional Neural Network", Y_train, cnn_preds_train_labels, Y_test, cnn_test_labels)
```

```
Metrics for Shallow Neural Network Model:

Train Set:
Accuracy: 0.9993655474117356
Precision: 0.9993663423272089
Recall: 0.9993655474117356
F1-Score: 0.9993651285758409
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    262352
           1       1.00      1.00      1.00     65111
           2       0.99      0.99      0.99      2759
           3       0.92      0.93      0.92       739
           4       0.82      0.70      0.75        33

    accuracy                           1.00    330994
   macro avg       0.95      0.92      0.93    330994
weighted avg       1.00      1.00      1.00    330994


Test Set:
Accuracy: 0.9991841842148846
Precision: 0.9991774408740962
Recall: 0.9991841842148846
F1-Score: 0.999179830762075
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    129106
           1       1.00      1.00      1.00     32167
           2       0.99      0.98      0.98      1348
           3       0.92      0.91      0.92       387
           4       0.75      0.63      0.69        19

    accuracy                           1.00    163027
   macro avg       0.93      0.90      0.92    163027
weighted avg       1.00      1.00      1.00    163027


Metrics for Deep Neural Network Model:

Train Set:
Accuracy: 0.9993021021529092
Precision: 0.9992978951133856
Recall: 0.9993021021529092
F1-Score: 0.9992956849953479
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    262352
           1       1.00      1.00      1.00     65111
           2       1.00      0.98      0.99      2759
           3       0.90      0.94      0.92       739
           4       0.70      0.42      0.53        33

    accuracy                           1.00    330994
```

More info

```
df.columns
```

```
Index(['duration', 'protocol_type', 'flag', 'src_bytes', 'dst_bytes', 'land',
       'wrong_fragment', 'urgent', 'hot', 'num_failed_logins', 'logged_in',
       'num_compromised', 'root_shell', 'su_attempted', 'num_file_creations',
       'num_shells', 'num_access_files', 'is_guest_login', 'count',
```