Data entry and cleaning

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from keras.utils import to_categorical
# Load the CSV file
data_df = pd.read_csv('/content/drive/MyDrive/Scenario-B-merged_5s.csv')
data_df.columns = data_df.columns.str.strip()

# Clean the data (to remove the infinite and NaN values from the dataset)
data_df = data_df.replace([np.inf, -np.inf], np.nan)
data_df.dropna(inplace=True)

# Filter labels of interest
desired_labels = ['VOIP', 'VIDEO', 'FILE-TRANSFER', 'CHAT', 'BROWSING']
data_df = data_df[data_df['label'].isin(desired_labels)]
data_df.shape
```

```
(10845, 29)
```

Functions for Data pre-processing

```
# Define a function to extract numerical features
def extract_numerical_features(df):
    numerical_features = df.select_dtypes(include=[np.number])
    #print("These are numerical features \n",numerical_features.head(1))
    return numerical_features

# Group packets into Packet Blocks
def aggregate_packets(df, block_size):
    grouped = df.groupby(['Source IP', 'Source Port', 'Destination IP', 'Destination Port', 'Protocol'])
    #groupby command return (name, group)
    #name contains the values of 'Source IP', 'Source Port', 'Destination IP', 'Destination Port', 'Protocol'
    #group is a subset of the dataset that stores the value corresponding to the name
    packet_blocks = []
    labels = []

    # Lists to store the first five entries of name and group
    name_list = []
    group_list = []

    for name, group in grouped:
        # Store the first five names and groups
        if len(name_list) < 5:
            name_list.append(name)
            group_list.append(group.head(2))  # Store only the first few rows of the group

        packets = extract_numerical_features(group).values
        for i in range(0, len(packets), block_size): #range(Starting_point, ending_point, intercept)
            block = packets[i:i + block_size]
            if len(block) == block_size:
                packet_blocks.append(block.flatten())
                labels.append(group['label'].iloc[0])

    # Print the first five names and groups
    print("Top 5 Names:")
    for n in name_list:
        print(n)

    print("\nTop 5 Groups:")
    for g in group_list:
        print(g)
    print(len(labels))
    return np.array(packet_blocks), np.array(labels)
```

Printing the values of top 5 entries in the group and name

```
K = 50  # It is the aggregation degree thus can be used as the block size
packet_blocks, labels = aggregate_packets(data_df, K)
```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x7c2246b08d00>
Top 5 Names:
('10.0.2.15', 33071, '195.154.126.78', 443, 6)
('10.0.2.15', 33308, '195.154.126.78', 443, 6)
('10.0.2.15', 33827, '82.161.239.177', 110, 6)
('10.0.2.15', 34328, '198.52.200.39', 443, 6)
('10.0.2.15', 34361, '5.9.123.81', 9001, 6)

Top 5 Groups:
      Source IP  Source Port  Destination IP  Destination Port  Protocol  \
3095  10.0.2.15        33071  195.154.126.78               443         6
3113  10.0.2.15        33071  195.154.126.78               443         6

      Flow Duration  Flow Bytes/s  Flow Packets/s  Flow IAT Mean  \
3095        1107532  17962.460678       34.310521   29933.297297
3113         391763  13564.323328       30.630764   35614.818182

      Flow IAT Std  ...  Bwd IAT Min  Active Mean  Active Std  Active Max  \
3095  62747.732625  ...            5            0           0           0
3113  60733.911802  ...          192            0           0           0

      Active Min  Idle Mean  Idle Std  Idle Max  Idle Min     label
3095           0          0         0         0         0  BROWSING
3113           0          0         0         0         0  BROWSING

[2 rows x 29 columns]
      Source IP  Source Port  Destination IP  Destination Port  Protocol  \
3636  10.0.2.15        33308  195.154.126.78               443         6

      Flow Duration   Flow Bytes/s  Flow Packets/s  Flow IAT Mean  \
3636        2555398  272331.355037      318.932706    3139.309582

      Flow IAT Std  ...  Bwd IAT Min  Active Mean  Active Std  Active Max  \
3636  19768.540085  ...            0            0           0           0

      Active Min  Idle Mean  Idle Std  Idle Max  Idle Min     label
3636           0          0         0         0         0  BROWSING

[1 rows x 29 columns]
      Source IP  Source Port  Destination IP  Destination Port  Protocol  \
1310  10.0.2.15        33827  82.161.239.177               110         6
1311  10.0.2.15        33827  82.161.239.177               110         6

      Flow Duration  Flow Bytes/s  Flow Packets/s  Flow IAT Mean  \
1310        4957659  40023.123817       62.327804   16096.295455
1311        4971111  31849.821901       56.928924   17628.053191

      Flow IAT Std  ...  Bwd IAT Min  Active Mean  Active Std  Active Max  \
1310  42445.675395  ...            0            0           0           0
1311  52439.582796  ...            8            0           0           0

      Active Min  Idle Mean  Idle Std  Idle Max  Idle Min     label
1310           0          0         0         0         0  BROWSING
1311           0          0         0         0         0  BROWSING

[2 rows x 29 columns]
      Source IP  Source Port Destination IP  Destination Port  Protocol  \
3638  10.0.2.15        34328  198.52.200.39               443         6
```

Print the shape and the 1st value of the labels and packet_blocks

```
print(labels[10])
print(labels.shape)
print(packet_blocks.shape)
#as each iteration is flattened thus block_size * column_number here 50 * 26 = 1300 (not counting source port, destination po
print(packet_blocks[10])
#this shows that One packet block has one label
```

```
FILE-TRANSFER
(201,)
(201, 1300)
[3.6629e+04 4.4300e+02 6.0000e+00 ... 0.0000e+00 0.0000e+00 0.0000e+00]
```

```
# Normalize the features in between (0,1)
scaler = MinMaxScaler()
packet_blocks_scaled = scaler.fit_transform(packet_blocks)
packet_blocks_scaled.shape
```

```
(201, 1300)
```

```
# Calculate the number of features
total_features = packet_blocks_scaled.shape[1]
total_features
```

```
1300
```

```
# Ideally, M and N should be such that M * N = total_features
M = 3000//K
N = 60
M,N
```

```
(60, 60)
```

```
# ideally M * N is at least equal to total_features
packet_blocks_padded = np.pad(packet_blocks_scaled, ((0, 0), (0, M * N - total_features)), 'constant')
#np.pad(array, pad_width, mode) here "constant" means 0
#(0, 0) for the first axis: No padding is applied to the rows (first axis).
#(0, M * N - total_features) for the second axis: This pads the columns (second axis) to ensure each block has exactly
# M * N features, filling with zeros if needed.


# Reshape the features into M * N dimensions
packet_images = packet_blocks_padded.reshape((-1, M, N))
#After reshaping, the total number of elements must remain the same. So, NumPy will calculate the size of the -1 dimension a
#inferred_dimension = total_Elements / (M * N)


# Convert labels to categorical format
labels_encoded = pd.factorize(labels)[0]
#factorize converts the character labels to numeric values
#.factorize() returns two labels one teh original and other the converted and we want the converted thus [0]

labels_categorical = to_categorical(labels_encoded)
#to_categorical is used for one-hot encoding (transforms categorical labels into binary vectors)

labels_categorical.shape,labels_encoded.shape
```

```
((201, 5), (201,))
```

```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(packet_images, labels_categorical, test_size=0.2, random_state=42)

# Save the preprocessed data
np.save('X_train.npy', X_train)
np.save('X_test.npy', X_test)
np.save('y_train.npy', y_train)
np.save('y_test.npy', y_test)
```

Retreating the saved files

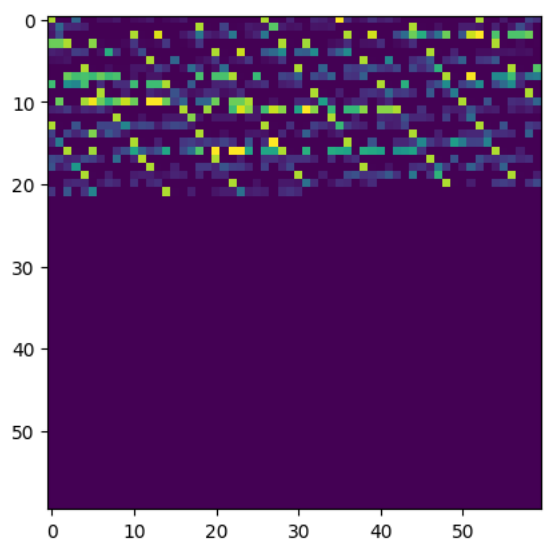
```
# Model and training
import numpy as np
from keras.models import Sequential
from keras.layers import Conv1D, MaxPooling1D, Flatten, Dense, Dropout
from keras.optimizers import Adam
from keras.callbacks import ModelCheckpoint, EarlyStopping

# Load the preprocessed data
X_train = np.load('X_train.npy')
X_test = np.load('X_test.npy')
y_train = np.load('y_train.npy')
y_test = np.load('y_test.npy')
X_train.shape
```

```
(160, 60, 60)
```

Sample image and data stored in the column

```
import matplotlib.pyplot as plt
plt.imshow(X_train[20].reshape((M, N)))
plt.show()
```




```
X_train.shape,X_train[1]
```

```
((160, 60, 60),
 array([[6.68033140e-01, 6.05135474e-03, 0.00000000e+00, ...,
         1.01985518e-03, 3.19718481e-02, 1.54241928e-02],
        [2.09813753e-02, 1.76245211e-01, 2.85406024e-02, ...,
         4.61162129e-04, 1.46029412e-02, 1.17024715e-02],
        [3.10596345e-02, 4.09237124e-04, 0.00000000e+00, ...,
         0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
        ...,
        [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, ...,
         0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
        [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, ...,
         0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
        [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, ...,
         0.00000000e+00, 0.00000000e+00, 0.00000000e+00]]))
```

Model creation

```
# Define the 1D CNN model
def create_cnn_model(input_shape, num_classes):
    model = Sequential()

    # First Convolutional Layer
    model.add(Conv1D(5, kernel_size=6, strides=1, padding='same', activation='relu', input_shape=input_shape))
    model.add(MaxPooling1D(pool_size=3))

    # Second Convolutional Layer
    model.add(Conv1D(10, kernel_size=5, strides=1, padding='same', activation='relu'))
    model.add(MaxPooling1D(pool_size=3))

    # Flatten and Dense Layers
    model.add(Flatten())
    model.add(Dense(64, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(num_classes, activation='softmax'))

    return model
#Convulation layer: filters=5: Number of output filters in the convolution.
#kernel_size This filter slides over the input data and performs a dot product with the input values within the kernel's ler
#strides specifies the step size with which the filter moves across the input data. It determines how many elements the filt
#Activation function applied after the convolution (Rectified Linear Unit).
# parameter (in Convolution layer) = (kernel_size * kernel_size * number of input channel +1) * number of output channel
# parameter (in dense layer) = (input_no._of_neurons +1) * output_no._of_nuerons
input_shape = (M, N)  # Adjust the input shape accordingly
num_classes = y_train.shape[1]  # Number of classes

model = create_cnn_model(input_shape, num_classes)
model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
input_shape,num_classes
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `in
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 60, 5)	1,805
max_pooling1d (MaxPooling1D)	(None, 20, 5)	0
conv1d_1 (Conv1D)	(None, 20, 10)	260
max_pooling1d_1 (MaxPooling1D)	(None, 6, 10)	0
flatten (Flatten)	(None, 60)	0
dense (Dense)	(None, 64)	3,904
dropout (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 5)	325

Total params: 6,294 (24.59 KB)
Trainable params: 6,294 (24.59 KB)
Non-trainable params: 0 (0.00 B)

```
((60, 60), 5)
```

Model training

```
# Train the model
history = model.fit(X_train, y_train,
                    batch_size=32,
                    epochs=50,
                    validation_split=0.2)

# Evaluate on the test set
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f'Test accuracy: {test_acc:.4f}')
```

```
Epoch 23/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.6865 - loss: 0.7884 - val_accuracy: 0.7500 - val_loss: 0.6614
```

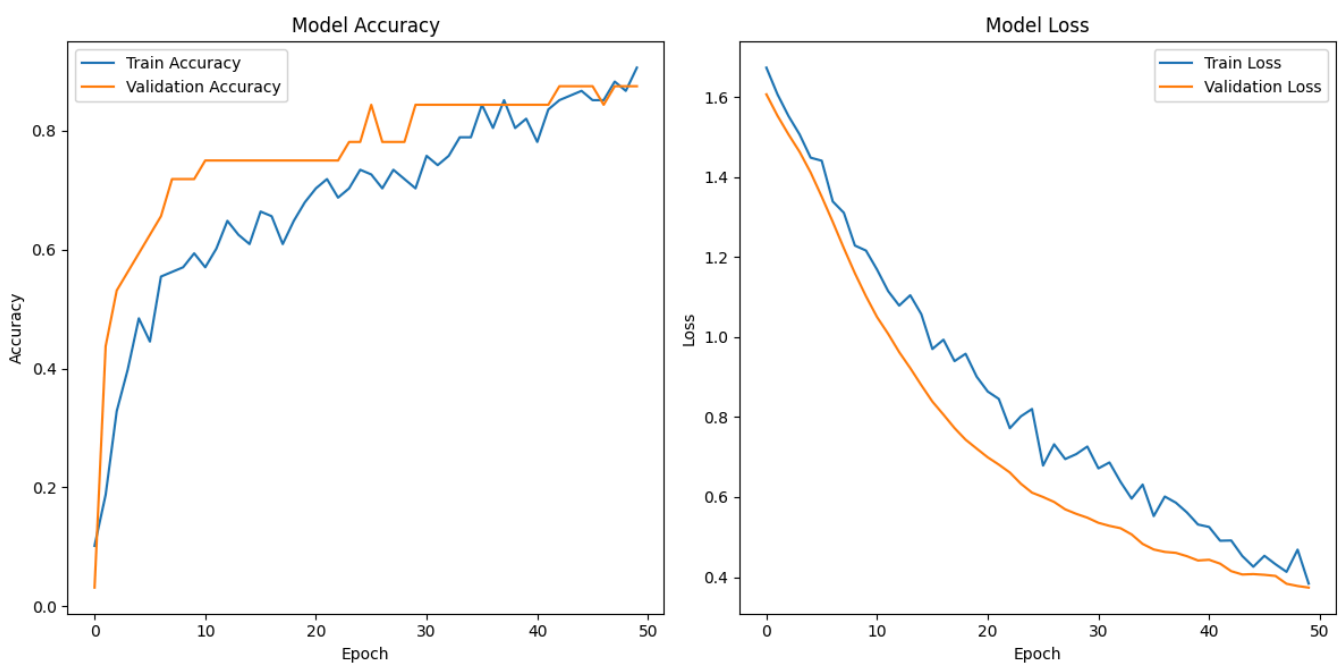
```
Epoch 46/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.8698 - loss: 0.4308 - val_accuracy: 0.8750 - val_loss: 0.4058
Epoch 47/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - accuracy: 0.8448 - loss: 0.4198 - val_accuracy: 0.8438 - val_loss: 0.4030
Epoch 48/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.8865 - loss: 0.4097 - val_accuracy: 0.8750 - val_loss: 0.3835
Epoch 49/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.8615 - loss: 0.4721 - val_accuracy: 0.8750 - val_loss: 0.3778
Epoch 50/50
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.8740 - loss: 0.4394 - val_accuracy: 0.8750 - val_loss: 0.3739
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.7274 - loss: 0.8538
Test accuracy: 0.7317
```

```
# Plot training & validation accuracy values
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='best')

# Plot training & validation loss values
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(loc='best')

plt.tight_layout()
plt.show()
```



Start coding or generate with AI.