In [1]:

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import tensorflow
import keras
import time
from sklearn.preprocessing import MinMaxScaler
from keras.utils import np_utils
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam, SGD
from sklearn.metrics import precision_score, recall_score, f1_score, roc_auc_score, accuracy_score, classification_report, confusion_matrix, plot_confusion_matrix
from sklearn.decomposition import PCA
import warnings
warnings.filterwarnings("ignore")
```

In [2]:

```
np.random.seed(1)
tensorflow.random.set_seed(1)
RS=42
```

In [3]:

```
## Loading Original Data
## (https://www.kaggle.com/datasets/fanbyprinciple/iot-device-identificatio
n?select=iot_device_train.csv)
original_dataset = pd.read_csv('../input/iot-device/iot_device_data.csv')
original_dataset
```

Out[3]:

	ack	ack_A	ack_B	bytes	bytes_A	bytes_A_B_ratio	bytes_B	ds_field_A	ds_field_
0	9	5	5	1213	743	0.713924	668	0	0
1	9	5	5	1213	743	1.806874	668	0	0
2	9	5	5	1213	743	0.103124	668	0	0
3	9	5	5	1213	743	1.806874	668	0	0
4	9	5	5	1213	743	1.806874	668	0	0
...	...	...	...	...	...	...	...	...	...
1895	264	116	148	212053	202036	20.169310	10017	0	0
1896	24	13	11	7749	5364	2.249056	2385	0	0
1897	20	9	11	7091	5336	3.040454	1755	0	0
1898	30	14	16	7882	5789	2.765885	2093	0	0
1899	294	147	147	209972	197919	16.420724	12053	0	0

1900 rows × 298 columns

In [4]:

```
## Shape: (Total number of data records, Number of features)
original_dataset.shape
```

Out[4]:

```
(1900, 298)
```

In [5]:

```
## Different features are on different scale, so normalization is needed.
original_dataset.describe(include='all')
```

Out[5]:

	ack	ack_A	ack_B	bytes	bytes_A	b
count	1900.000000	1900.000000	1900.000000	1.900000e+03	1.900000e+03	1
unique	NaN	NaN	NaN	NaN	NaN	N
top	NaN	NaN	NaN	NaN	NaN	N
freq	NaN	NaN	NaN	NaN	NaN	N
mean	227.773158	81.211579	146.878947	1.498682e+05	1.421069e+05	1
std	4461.164912	1125.546944	3415.513067	4.937287e+06	4.870776e+06	3
min	0.000000	0.000000	0.000000	6.000000e+01	0.000000e+00	0
25%	0.000000	0.000000	0.000000	2.400000e+02	0.000000e+00	0
50%	9.000000	5.000000	5.000000	1.213000e+03	7.430000e+02	0
75%	15.000000	7.000000	8.000000	2.411000e+03	1.160000e+03	1
max	184378.000000	39265.000000	145113.000000	2.137146e+08	2.108638e+08	7

11 rows × 298 columns

In [6]:

```
## Available Features
list(original_dataset.columns)
```

Out[6]:

```
['ack',
 'ack_A',
 'ack_B',
 'bytes',
 'bytes_A',
 'bytes_A_B_ratio',
 'bytes_B',
 'ds_field_A',
 'ds_field_B',
 'duration',
 'http_GET',
 'http_POST',
 'http_bytes_avg',
 'http_bytes_entropy',
 'http_bytes_firstQ',
 'http_bytes_max',
 'http_bytes_median',
 'http_bytes_min',
 'http_bytes_stdev',
 'http_bytes_sum',
 'http_bytes_thirdQ',
 'http_bytes_var',
 'http_cookie_count',
 'http_cookie_values_avg',
 'http_cookie_values_entropy',
 'http_cookie_values_firstQ',
 'http_cookie_values_max',
 'http_cookie_values_median',
 'http_cookie_values_min',
 'http_cookie_values_stdev',
 'http_cookie_values_sum',
 'http_cookie_values_thirdQ',
 'http_cookie_values_var',
 'http_count_host',
 'http_count_req_content_type',
 'http_count_resp_code',
 'http_count_resp_content_type',
 'http_count_transactions',
 'http_count_user_agents',
 'http_dom_host_alexaRank',
```

```
'http_dom_resp_code',
'http_has_location',
'http_has_referrer',
'http_has_req_content_type',
'http_has_resp_content_type',
'http_has_user_agent',
'http_inter_arrivel_avg',
'http_inter_arrivel_entropy',
'http_inter_arrivel_firstQ',
'http_inter_arrivel_max',
'http_inter_arrivel_median',
'http_inter_arrivel_min',
'http_inter_arrivel_stdev',
'http_inter_arrivel_sum',
'http_inter_arrivel_thirdQ',
'http_inter_arrivel_var',
'http_req_bytes_avg',
'http_req_bytes_entropy',
'http_req_bytes_firstQ',
'http_req_bytes_max',
'http_req_bytes_median',
'http_req_bytes_min',
'http_req_bytes_stdev',
'http_req_bytes_sum',
'http_req_bytes_thirdQ',
'http_req_bytes_var',
'http_resp_bytes_avg',
'http_resp_bytes_entropy',
'http_resp_bytes_firstQ',
'http_resp_bytes_max',
'http_resp_bytes_median',
'http_resp_bytes_min',
'http_resp_bytes_stdev',
'http_resp_bytes_sum',
'http_resp_bytes_thirdQ',
'http_resp_bytes_var',
'http_time_avg',
'http_time_entropy',
'http_time_firstQ',
'http_time_max',
'http_time_median',
```

```
 'http_time_min',
 'http_time_stdev',
 'http_time_sum',
 'http_time_thirdQ',
 'http_time_var',
 'packet_inter_arrivel_A_avg',
 'packet_inter_arrivel_A_entropy',
 'packet_inter_arrivel_A_firstQ',
 'packet_inter_arrivel_A_max',
 'packet_inter_arrivel_A_median',
 'packet_inter_arrivel_A_min',
 'packet_inter_arrivel_A_stdev',
 'packet_inter_arrivel_A_sum',
 'packet_inter_arrivel_A_thirdQ',
 'packet_inter_arrivel_A_var',
 'packet_inter_arrivel_B_avg',
 'packet_inter_arrivel_B_entropy',
 'packet_inter_arrivel_B_firstQ',
 'packet_inter_arrivel_B_max',
 'packet_inter_arrivel_B_median',
 'packet_inter_arrivel_B_min',
 'packet_inter_arrivel_B_stdev',
 'packet_inter_arrivel_B_sum',
 'packet_inter_arrivel_B_thirdQ',
 'packet_inter_arrivel_B_var',
 'packet_inter_arrivel_avg',
 'packet_inter_arrivel_entropy',
 'packet_inter_arrivel_firstQ',
 'packet_inter_arrivel_max',
 'packet_inter_arrivel_median',
 'packet_inter_arrivel_min',
 'packet_inter_arrivel_stdev',
 'packet_inter_arrivel_sum',
 'packet_inter_arrivel_thirdQ',
 'packet_inter_arrivel_var',
 'packet_size_A_avg',
 'packet_size_A_entropy',
 'packet_size_A_firstQ',
 'packet_size_A_max',
 'packet_size_A_median',
 'packet_size_A_min',
```

```
 'packet_size_A_stdev',
 'packet_size_A_sum',
 'packet_size_A_thirdQ',
 'packet_size_A_var',
 'packet_size_B_avg',
 'packet_size_B_entropy',
 'packet_size_B_firstQ',
 'packet_size_B_max',
 'packet_size_B_median',
 'packet_size_B_min',
 'packet_size_B_stdev',
 'packet_size_B_sum',
 'packet_size_B_thirdQ',
 'packet_size_B_var',
 'packet_size_avg',
 'packet_size_entropy',
 'packet_size_firstQ',
 'packet_size_max',
 'packet_size_median',
 'packet_size_min',
 'packet_size_stdev',
 'packet_size_sum',
 'packet_size_thirdQ',
 'packet_size_var',
 'packets',
 'packets_A',
 'packets_A_B_ratio',
 'packets_B',
 'push',
 'push_A',
 'push_B',
 'reset',
 'reset_A',
 'reset_B',
 'ssl_count_certificates',
 'ssl_count_client_cipher_algs',
 'ssl_count_client_ciphersuites',
 'ssl_count_client_compressions',
 'ssl_count_client_elliptic_curves',
 'ssl_count_client_key_exchange_algs',
 'ssl_count_client_mac_algs',
```

```
'ssl_count_server_ciphersuite',
'ssl_count_server_compression',
'ssl_count_server_elliptic_curve',
'ssl_count_server_name',
'ssl_count_transactions',
'ssl_count_version',
'ssl_dom_server_ciphersuite',
'ssl_dom_server_name_alexaRank',
'ssl_dom_version',
'ssl_handshake_duration_avg',
'ssl_handshake_duration_entropy',
'ssl_handshake_duration_firstQ',
'ssl_handshake_duration_max',
'ssl_handshake_duration_median',
'ssl_handshake_duration_min',
'ssl_handshake_duration_stdev',
'ssl_handshake_duration_sum',
'ssl_handshake_duration_thirdQ',
'ssl_handshake_duration_var',
'ssl_ratio_client_elliptic_curves',
'ssl_ratio_server_name',
'ssl_req_bytes_avg',
'ssl_req_bytes_entropy',
'ssl_req_bytes_firstQ',
'ssl_req_bytes_max',
'ssl_req_bytes_median',
'ssl_req_bytes_min',
'ssl_req_bytes_stdev',
'ssl_req_bytes_sum',
'ssl_req_bytes_thirdQ',
'ssl_req_bytes_var',
'ssl_resp_bytes_avg',
'ssl_resp_bytes_entropy',
'ssl_resp_bytes_firstQ',
'ssl_resp_bytes_max',
'ssl_resp_bytes_median',
'ssl_resp_bytes_min',
'ssl_resp_bytes_stdev',
'ssl_resp_bytes_sum',
'ssl_resp_bytes_thirdQ',
'ssl_resp_bytes_var',
```

```
'tcp_analysis_duplicate_ack',
'tcp_analysis_keep_alive',
'tcp_analysis_lost_segment',
'tcp_analysis_out_of_order',
'tcp_analysis_retransmission',
'tcp_analysis_reused_ports',
'ttl_A_avg',
'ttl_A_entropy',
'ttl_A_firstQ',
'ttl_A_max',
'ttl_A_median',
'ttl_A_min',
'ttl_A_stdev',
'ttl_A_sum',
'ttl_A_thirdQ',
'ttl_A_var',
'ttl_B_avg',
'ttl_B_entropy',
'ttl_B_firstQ',
'ttl_B_max',
'ttl_B_median',
'ttl_B_min',
'ttl_B_stdev',
'ttl_B_sum',
'ttl_B_thirdQ',
'ttl_B_var',
'ttl_avg',
'ttl_entropy',
'ttl_firstQ',
'ttl_max',
'ttl_median',
'ttl_min',
'ttl_stdev',
'ttl_sum',
'ttl_thirdQ',
'ttl_var',
'is_ssl',
'is_http',
'is_g_http',
'is_cdn_http',
'is_img_http',
```

```
            'is_ad_http',
            'is_numeric_url_http',
            'is_numeric_url_with_port_http',
            'is_tv_http',
            'is_cloud_http',
            'B_is_system_port',
            'B_is_user_port',
            'B_is_dynamic_and_or_private_port',
            'B_port_is_11095',
            'B_port_is_1900',
            'B_port_is_5222',
            'B_port_is_5223',
            'B_port_is_5228',
            'B_port_is_54975',
            'B_port_is_80',
            'B_port_is_8080',
            'B_port_is_8280',
            'B_port_is_9543',
            'B_port_is_else',
            'subdomain_is_99sets',
            'subdomain_is_ccc',
            'subdomain_is_else',
            'subdomain_is_feeds',
            'subdomain_is_h10141.www1',
            'subdomain_is_img',
            'subdomain_is_unresolved',
            'subdomain_is_whp.aus1.cold.extweb',
            'subdomain_is_whp.hou9.cold.extweb',
            'subdomain_is_www',
            'subdomain_is_www.cloud',
            'domain_is_dlink',
            'domain_is_else',
            'domain_is_epicurious',
            'domain_is_google',
            'domain_is_hp',
            'domain_is_hpeprint',
            'domain_is_livecdn',
            'domain_is_mako',
            'domain_is_proteussensor',
            'domain_is_samsung',
            'domain_is_unresolved',
```

```
 'suffix_is_biz',
 'suffix_is_cloudfront.net',
 'suffix_is_co.il',
 'suffix_is_com',
 'suffix_is_com.sg',
 'suffix_is_else',
 'suffix_is_empty_char_value',
 'suffix_is_googleapis.com',
 'suffix_is_net',
 'suffix_is_org',
 'suffix_is_unresolved',
 'device_category']
```

In [7]:

```
## Distribution of Different IoT Device Categories
original_dataset['device_category'].value_counts()
```

Out[7]:

```
security_camera    200
TV                 200
smoke_detector     200
thermostat         200
watch              200
baby_monitor       200
motion_sensor      200
lights             200
socket             200
water_sensor       100
Name: device_category, dtype: int64
```

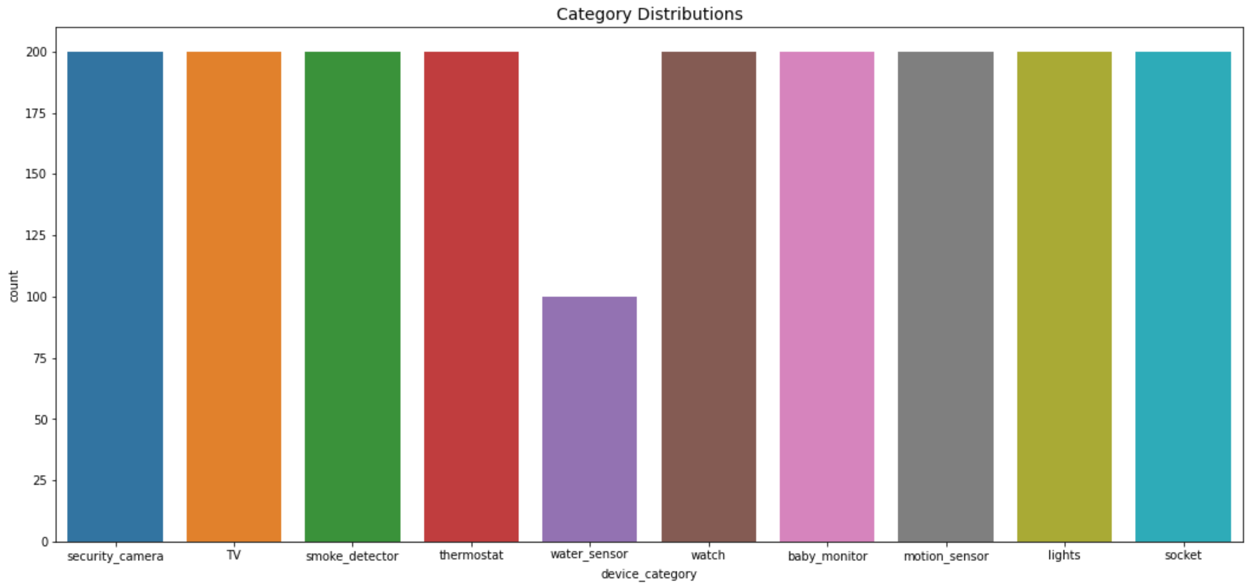
In [8]:

```
np.unique(original_dataset['device_category'])
```

Out[8]:

```
array(['TV', 'baby_monitor', 'lights', 'motion_sensor', 'security_came
ra',
       'smoke_detector', 'socket', 'thermostat', 'watch', 'water_senso
r'],
      dtype=object)
```

In [9]:
```
plt.figure(figsize=(18,8))
sns.countplot(x='device_category', data=original_dataset)
plt.title('Category Distributions', fontsize=14)
plt.show()
```



In [10]:

```
## Replace Categories name with integer values
IoT_device_dataset = original_dataset.replace({'device_category': {'TV':0
,
                                                                   'baby_m
onitor':1,
                                                                   'smoke_
detector':2,
                                                                   'socke
t':3,
                                                                   'watch'
:4,
                                                                   'water_
sensor':5,
                                                                   'light
s':6,
                                                                   'thermo
stat':7,
                                                                   'motion
_sensor': 8,
                                                                   'securi
ty_camera':9}
                                              })
shuffled_iot_device_dataset = IoT_device_dataset.sample(frac=1).reset_ind
ex(drop=True)
shuffled_iot_device_dataset
```

Out[10]:

	ack	ack_A	ack_B	bytes	bytes_A	bytes_A_B_ratio	bytes_B	ds_field_A	ds_field_B
0	28	13	15	13381	2579	0.238752	10802	0	0
1	0	0	0	240	0	0.000000	240	0	64
2	0	0	0	240	0	0.000000	240	0	64
3	0	0	0	240	0	0.000000	240	0	64
4	38	20	18	14730	7814	1.129844	6916	0	0
...	...	...	...	...	...	...	...	...	...
1895	0	0	0	240	0	0.000000	240	0	64
1896	14	6	8	4937	4212	5.809647	725	0	0
1897	38	20	18	14826	7814	1.114375	7012	0	0
1898	9	5	5	1213	743	0.244601	668	0	0
1899	38	20	18	14714	7814	1.132464	6900	0	0

1900 rows × 298 columns

In [11]:

```
## Segregating Data and Output Catgeory Labels
data = shuffled_iot_device_dataset.drop(['device_category'], axis = 1)
output_labels = shuffled_iot_device_dataset['device_category'].to_frame()
```

In [12]:

```
# from imblearn.over_sampling import SMOTE

# balancer = SMOTE(random_state=42, sampling_strategy='minority')
# oversampled_data, oversampled_labels = balancer.fit_resample(data, output
_labels)
```

In [13]:

```
# oversampled_full_dataset = pd.concat([oversampled_data, oversampled_label
s], axis = 1)
# oversampled_full_dataset
```

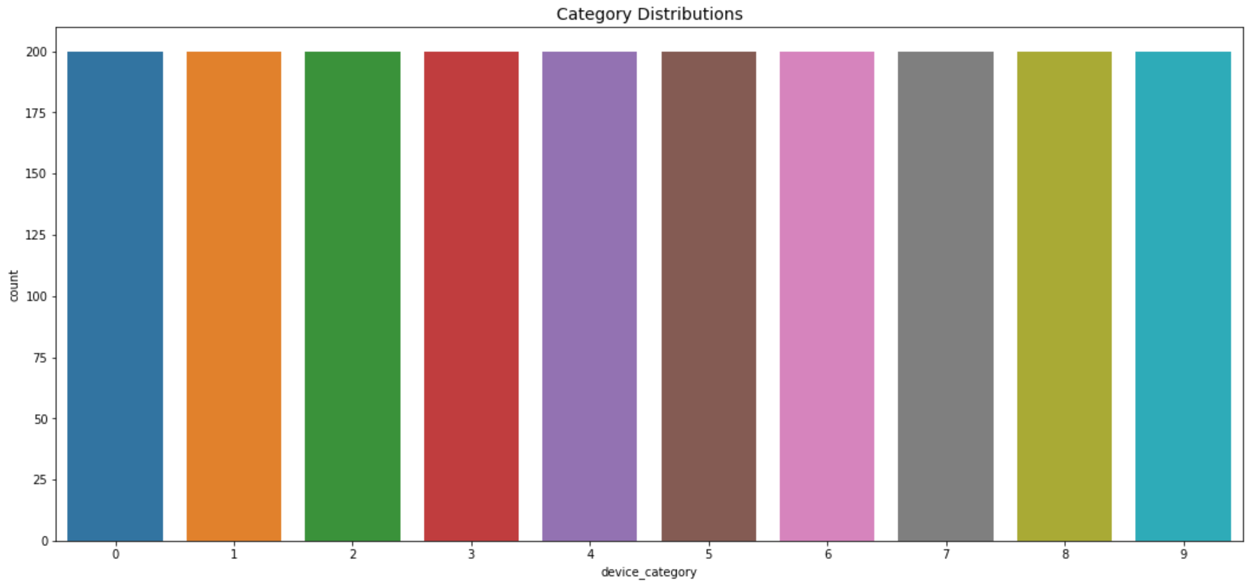
Out[13]:

	ack	ack_A	ack_B	bytes	bytes_A	bytes_A_B_ratio	bytes_B	ds_field_A	ds_field_B
0	28	13	15	13381	2579	0.238752	10802	0	0
1	0	0	0	240	0	0.000000	240	0	64
2	0	0	0	240	0	0.000000	240	0	64
3	0	0	0	240	0	0.000000	240	0	64
4	38	20	18	14730	7814	1.129844	6916	0	0
...	...	...	...	...	...	...	...	...	...
1995	0	0	0	240	0	0.000000	240	0	64
1996	0	0	0	240	0	0.000000	240	0	64
1997	0	0	0	240	0	0.000000	240	0	64
1998	9	5	5	1213	743	1.712060	668	0	0
1999	0	0	0	240	0	0.000000	240	0	64

2000 rows × 298 columns

In [15]:

```
# plt.figure(figsize=(18,8))
# sns.countplot(x='device_category', data=oversampled_full_dataset)
# plt.title('Category Distributions', fontsize=14)
# plt.show()
```



In [16]:

```
# ## Splitting Dataset into Train and Test
# train_data, test_data, train_labels, test_labels = train_test_split(overs
ampled_data, oversampled_labels, test_size= 0.2, random_state=42)
```

In [14]:

```
## Splitting Dataset into Train and Test
train_data, test_data, train_labels, test_labels = train_test_split(data,
output_labels, test_size= 0.2, random_state=42)
```

In [15]:

```
## Training Data and Labels
print(train_data.shape)
print(train_labels.shape)
```

```
(1520, 297)
(1520, 1)
```

In [16]:
```
## testing Data and Labels
print(test_data.shape)
print(test_labels.shape)
```

```
(380, 297)
(380, 1)
```

Data Preprocessing

Normalisation

In [17]:

```
# Applying Min-Max Normalization
scaler = MinMaxScaler()

## Normalisation on train data
scaled_train_data = scaler.fit_transform(train_data)
normalized_train_data = pd.DataFrame(scaled_train_data, columns = train_d
ata.columns)
normalized_train_data
```

Out[17]:

	ack	ack_A	ack_B	bytes	bytes_A	bytes_A_B_ratio	bytes_B	c
0	0.000364	0.000405	0.000315	0.000047	0.000032	0.001686	0.000820	0
1	0.004127	0.005178	0.002390	0.003534	0.000250	0.001177	0.103792	0
2	0.000364	0.000243	0.000315	0.000047	0.000026	0.014019	0.000820	0
3	0.000364	0.000405	0.000315	0.000047	0.000032	0.017805	0.000820	0
4	0.001537	0.001618	0.001132	0.000627	0.000339	0.017650	0.009333	0
...	...	...	...	...	...	...	...	..
1515	0.000000	0.000000	0.000000	0.000005	0.000000	0.000000	0.000243	0
1516	0.000040	0.000081	0.000000	0.000000	0.000003	0.015766	0.000000	0
1517	0.000000	0.000000	0.000000	0.000005	0.000000	0.000000	0.000243	0
1518	0.001092	0.001376	0.000629	0.000302	0.000070	0.004608	0.007404	0
1519	0.001740	0.001618	0.001447	0.000843	0.000704	0.070652	0.004800	0

1520 rows × 297 columns

In [18]:

```
## Normalisation on test data
scaled_test_data = scaler.transform(test_data)
normalized_test_data = pd.DataFrame(scaled_test_data, columns = test_data
.columns)
normalized_test_data
```

Out[18]:

	ack	ack_A	ack_B	bytes	bytes_A	bytes_A_B_ratio	bytes_B	ds
0	0.000890	0.000971	0.000629	0.000446	0.000247	0.018498	0.006464	0.(
1	0.001821	0.001376	0.001761	0.000054	0.000028	0.014163	0.000901	0.(
2	0.001011	0.001052	0.000755	0.000415	0.000053	0.002246	0.011526	0.(
3	0.000000	0.000000	0.000000	0.000005	0.000000	0.000000	0.000243	0.(
4	0.000364	0.000405	0.000315	0.000047	0.000032	0.028487	0.000820	0.(
...	...	...	...	...	...	...	...	...
375	0.000607	0.000647	0.000440	0.000115	0.000042	0.008348	0.002387	0.(
376	0.000485	0.000485	0.000377	0.000061	0.000028	0.011261	0.001137	0.(
377	0.000364	0.000405	0.000315	0.000047	0.000032	0.015805	0.000820	0.(
378	0.000000	0.000000	0.000000	0.000005	0.000000	0.000000	0.000243	0.(
379	0.009184	0.009467	0.006919	0.002666	0.000535	0.003874	0.067533	0.(

380 rows × 297 columns

Missing Value Check

In [19]:
```
## on train data (No Missin values as all 297 columns have 1520 records)
normalized_train_data.describe().iloc[0,:].value_counts()
```
Out[19]:
```
1520.0    297
Name: count, dtype: int64
```

In [20]:
```
## on test data (No Missin values as all 297 columns have 380 records)
normalized_test_data.describe().iloc[0,:].value_counts()
```
Out[20]:
```
380.0    297
Name: count, dtype: int64
```

Check for NaN's

In [21]:
```
## check for NaN's in train data (No Nan's are present in the data as well)
set(list(normalized_train_data.isna().sum()))
```
Out[21]:
```
{0}
```

In [22]:
```
## check for NaN's in test data (No Nan's are present in the data as well)
set(list(normalized_test_data.isna().sum()))
```
Out[22]:
```
{0}
```

Dimensionality Reduction using PCA

In [29]:

```
## PCA on Train data with 100 principle components
pca = PCA(n_components=100)
pca_train_data = pca.fit_transform(normalized_train_data)
pca_train_data_df = pd.DataFrame(data = pca_train_data)
pca_train_data_df
```

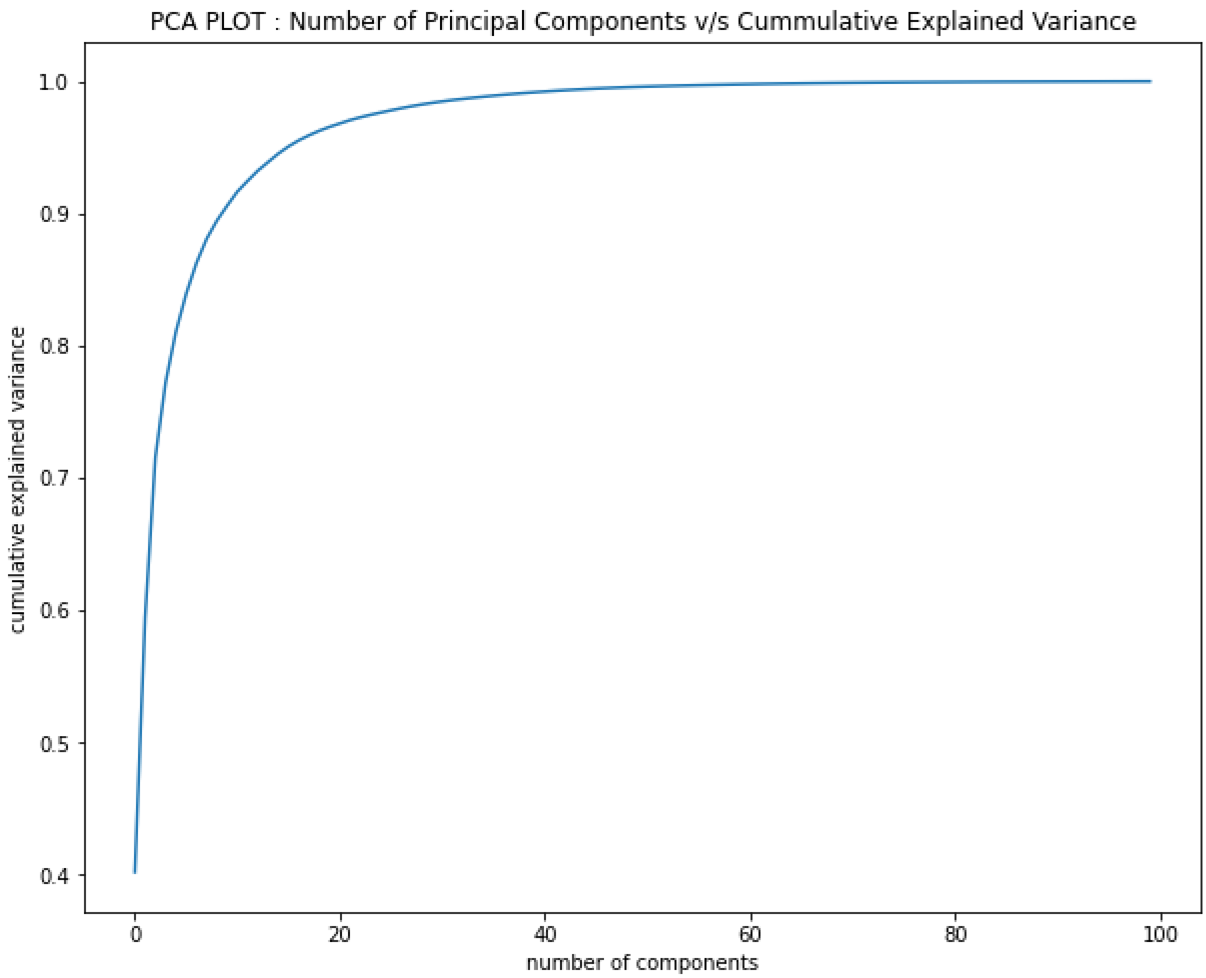
Out[29]:

	0	1	2	3	4	5	6
0	0.730723	0.334485	-1.602227	0.792602	-0.855409	0.567965	0.052991
1	3.971868	-1.403770	1.917217	0.831664	-0.382580	0.974598	-0.089277
2	-0.974354	3.427019	1.593951	0.328361	-0.086042	-0.086963	0.028725
3	-1.267939	-0.081727	-0.197027	1.436464	0.585704	-0.617177	-0.191118
4	3.593161	-1.029513	0.958960	1.160786	-1.373294	-0.244275	-0.443705
...	...	...	...	...	...	...	...
1515	-2.764082	-1.208451	0.277930	-0.345539	-0.560544	-0.300277	0.003780
1516	0.204716	0.098055	-1.182011	-0.888119	1.198058	0.966092	-0.400854
1517	-2.741367	-1.281360	0.697635	-0.539893	0.170775	0.525007	0.079502
1518	3.145956	-0.947389	0.620843	0.142516	-0.631295	-0.435939	0.339229
1519	3.415415	-1.039432	0.994208	-0.608370	0.447814	-0.627860	-0.214160

1520 rows × 100 columns

In [31]:

```
plt.figure(figsize= (10,8))
plt.plot(sorted(pca.explained_variance_ratio_.cumsum()))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance');
plt.title('PCA PLOT : Number of Principal Components v/s Cummulative Expl
ained Variance');
```



In [51]:
```
## PCA on Test Data with 100 principle components
pca_test_data = pca.transform(normalized_test_data)
pca_test_data_df = pd.DataFrame(data = pca_test_data)
# print('Explained variation per principal component: {}'.format(pca.explai
ned_variance_ratio_))
pca_test_data_df
```

Out[51]:

	0	1	2	3	4	5	6
0	3.472587	-1.096878	1.148159	-0.638872	0.502043	-0.498727	-0.079283
1	-0.916997	3.755897	1.895829	0.747434	-0.257603	-0.213008	2.248197
2	3.543257	-1.121608	1.363254	-0.770928	0.827523	0.032399	0.600176
3	-2.718372	-1.249895	0.678412	-0.512176	0.192241	0.506805	0.050189
4	0.046470	3.325332	0.435577	-1.469099	0.003612	0.218240	-1.994153
...	...	...	...	...	...	...	...
375	2.923064	-0.918469	0.942193	-0.793925	0.614170	0.357629	0.195342
376	0.445789	4.169480	0.567475	-1.526351	-0.243617	0.125420	0.180114
377	0.808986	0.417074	-1.625024	0.965179	-0.811076	0.440363	0.125915
378	-2.735025	-1.274956	0.693326	-0.531163	0.176822	0.514632	0.081860
379	-1.059270	-0.108549	0.372497	1.722217	1.801037	0.142878	-0.068969

380 rows × 100 columns

Binarizing Labels

In [52]:

```
## Binarized Training Labels
binarized_train_labels = np_utils.to_categorical(train_labels, num_classe
s = 10)
binarized_train_labels

## Binarized Testing Labels
binarized_test_labels = np_utils.to_categorical(test_labels, num_classes
= 10)
binarized_test_labels
```

Out[52]:

```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 1., 0.],
       [0., 0., 0., ..., 1., 0., 0.],
       ...,
       [0., 1., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 1., ..., 0., 0., 0.]], dtype=float32)
```

Model 1: NN Model without PCA

In [53]:

```
## Build Model 1
model = keras.Sequential()
model.add(Dense(512, activation='relu', input_shape=(297,)))
model.add(Dropout(rate = 0.1))
model.add(Dense(256, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dropout(rate = 0.1))
model.add(Dense(64, activation='relu'))
model.add(Dense(10, activation='softmax')) ## Number of Neurons in Last la
yer == Number of Categories present in Dataset
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_10 (Dense)             (None, 512)               152576
_________________________________________________________________
dropout_4 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_11 (Dense)             (None, 256)               131328
_________________________________________________________________
dense_12 (Dense)             (None, 128)               32896
_________________________________________________________________
dropout_5 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_13 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_14 (Dense)             (None, 10)                650
=================================================================
Total params: 325,706
Trainable params: 325,706
Non-trainable params: 0
_________________________________________________________________
```

In [54]:
```
## Compile Model 1
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=
['accuracy'])
```

In [55]:

```
## Fit Training Data on Neural Network
start_time_1 = time.time()
model_hist = model.fit(normalized_train_data, binarized_train_labels,
                       epochs=50,
                       batch_size= 8,
                       validation_split=0.2,
                       verbose=2)
computational_time_1 = time.time() - start_time_1
```

```
Epoch 1/50
152/152 - 1s - loss: 1.1505 - accuracy: 0.5493 - val_loss: 0.7186 - va
l_accuracy: 0.7171
Epoch 2/50
152/152 - 0s - loss: 0.6836 - accuracy: 0.7105 - val_loss: 0.5485 - va
l_accuracy: 0.7632
Epoch 3/50
152/152 - 0s - loss: 0.5377 - accuracy: 0.7590 - val_loss: 0.5277 - va
l_accuracy: 0.7303
Epoch 4/50
152/152 - 0s - loss: 0.4640 - accuracy: 0.8010 - val_loss: 0.6016 - va
l_accuracy: 0.7566
Epoch 5/50
152/152 - 0s - loss: 0.4118 - accuracy: 0.7952 - val_loss: 0.5811 - va
l_accuracy: 0.7664
Epoch 6/50
152/152 - 0s - loss: 0.4002 - accuracy: 0.8150 - val_loss: 0.4874 - va
l_accuracy: 0.7993
Epoch 7/50
152/152 - 0s - loss: 0.3471 - accuracy: 0.8396 - val_loss: 0.4812 - va
l_accuracy: 0.7928
Epoch 8/50
152/152 - 0s - loss: 0.3283 - accuracy: 0.8298 - val_loss: 0.4747 - va
l_accuracy: 0.7961
Epoch 9/50
152/152 - 0s - loss: 0.3565 - accuracy: 0.8191 - val_loss: 0.4938 - va
l_accuracy: 0.8125
Epoch 10/50
152/152 - 0s - loss: 0.3719 - accuracy: 0.8150 - val_loss: 0.4572 - va
l_accuracy: 0.7895
Epoch 11/50
152/152 - 0s - loss: 0.3561 - accuracy: 0.8248 - val_loss: 0.5574 - va
l_accuracy: 0.8125
Epoch 12/50
152/152 - 0s - loss: 0.3501 - accuracy: 0.8215 - val_loss: 0.4506 - va
l_accuracy: 0.8322
Epoch 13/50
152/152 - 0s - loss: 0.3231 - accuracy: 0.8380 - val_loss: 0.4617 - va
l_accuracy: 0.8289
Epoch 14/50
```

```
152/152 - 0s - loss: 0.3055 - accuracy: 0.8405 - val_loss: 0.4633 - va
l_accuracy: 0.8158
Epoch 15/50
152/152 - 0s - loss: 0.3158 - accuracy: 0.8347 - val_loss: 0.4696 - va
l_accuracy: 0.8191
Epoch 16/50
152/152 - 0s - loss: 0.2918 - accuracy: 0.8528 - val_loss: 0.4909 - va
l_accuracy: 0.8026
Epoch 17/50
152/152 - 0s - loss: 0.3185 - accuracy: 0.8314 - val_loss: 0.5206 - va
l_accuracy: 0.7961
Epoch 18/50
152/152 - 0s - loss: 0.2960 - accuracy: 0.8413 - val_loss: 0.4600 - va
l_accuracy: 0.8224
Epoch 19/50
152/152 - 1s - loss: 0.2951 - accuracy: 0.8569 - val_loss: 0.4669 - va
l_accuracy: 0.8191
Epoch 20/50
152/152 - 0s - loss: 0.2906 - accuracy: 0.8553 - val_loss: 0.4218 - va
l_accuracy: 0.8322
Epoch 21/50
152/152 - 0s - loss: 0.2809 - accuracy: 0.8512 - val_loss: 0.4674 - va
l_accuracy: 0.8257
Epoch 22/50
152/152 - 0s - loss: 0.2642 - accuracy: 0.8577 - val_loss: 0.5001 - va
l_accuracy: 0.8224
Epoch 23/50
152/152 - 0s - loss: 0.2948 - accuracy: 0.8487 - val_loss: 0.5026 - va
l_accuracy: 0.8289
Epoch 24/50
152/152 - 0s - loss: 0.3213 - accuracy: 0.8462 - val_loss: 0.5003 - va
l_accuracy: 0.8092
Epoch 25/50
152/152 - 0s - loss: 0.3076 - accuracy: 0.8520 - val_loss: 0.6039 - va
l_accuracy: 0.7829
Epoch 26/50
152/152 - 0s - loss: 0.3086 - accuracy: 0.8438 - val_loss: 0.4997 - va
l_accuracy: 0.8289
Epoch 27/50
152/152 - 0s - loss: 0.2798 - accuracy: 0.8577 - val_loss: 0.4746 - va
l_accuracy: 0.8191
```

```
Epoch 28/50
152/152 - 0s - loss: 0.3201 - accuracy: 0.8586 - val_loss: 0.4675 - va
l_accuracy: 0.8158
Epoch 29/50
152/152 - 0s - loss: 0.2961 - accuracy: 0.8512 - val_loss: 0.4493 - va
l_accuracy: 0.8092
Epoch 30/50
152/152 - 0s - loss: 0.2865 - accuracy: 0.8610 - val_loss: 0.4107 - va
l_accuracy: 0.8257
Epoch 31/50
152/152 - 0s - loss: 0.2615 - accuracy: 0.8627 - val_loss: 0.5473 - va
l_accuracy: 0.8158
Epoch 32/50
152/152 - 0s - loss: 0.2870 - accuracy: 0.8446 - val_loss: 0.5079 - va
l_accuracy: 0.8487
Epoch 33/50
152/152 - 0s - loss: 0.2696 - accuracy: 0.8618 - val_loss: 0.4673 - va
l_accuracy: 0.8257
Epoch 34/50
152/152 - 0s - loss: 0.2948 - accuracy: 0.8503 - val_loss: 0.4305 - va
l_accuracy: 0.8158
Epoch 35/50
152/152 - 0s - loss: 0.2674 - accuracy: 0.8618 - val_loss: 0.5123 - va
l_accuracy: 0.8553
Epoch 36/50
152/152 - 0s - loss: 0.2938 - accuracy: 0.8503 - val_loss: 0.4085 - va
l_accuracy: 0.8388
Epoch 37/50
152/152 - 0s - loss: 0.2668 - accuracy: 0.8544 - val_loss: 0.4202 - va
l_accuracy: 0.8355
Epoch 38/50
152/152 - 0s - loss: 0.2734 - accuracy: 0.8536 - val_loss: 0.4456 - va
l_accuracy: 0.8257
Epoch 39/50
152/152 - 0s - loss: 0.2701 - accuracy: 0.8602 - val_loss: 0.5016 - va
l_accuracy: 0.8388
Epoch 40/50
152/152 - 0s - loss: 0.2730 - accuracy: 0.8544 - val_loss: 0.5114 - va
l_accuracy: 0.8520
Epoch 41/50
152/152 - 0s - loss: 0.2921 - accuracy: 0.8618 - val_loss: 0.4734 - va
```

```
l_accuracy: 0.8388
Epoch 42/50
152/152 - 0s - loss: 0.2590 - accuracy: 0.8668 - val_loss: 0.4591 - va
l_accuracy: 0.8553
Epoch 43/50
152/152 - 0s - loss: 0.2764 - accuracy: 0.8520 - val_loss: 0.5775 - va
l_accuracy: 0.8289
Epoch 44/50
152/152 - 0s - loss: 0.2904 - accuracy: 0.8586 - val_loss: 0.5110 - va
l_accuracy: 0.8125
Epoch 45/50
152/152 - 0s - loss: 0.3049 - accuracy: 0.8495 - val_loss: 0.5463 - va
l_accuracy: 0.8191
Epoch 46/50
152/152 - 0s - loss: 0.2819 - accuracy: 0.8618 - val_loss: 0.5410 - va
l_accuracy: 0.8289
Epoch 47/50
152/152 - 0s - loss: 0.2831 - accuracy: 0.8561 - val_loss: 0.4882 - va
l_accuracy: 0.8355
Epoch 48/50
152/152 - 0s - loss: 0.2657 - accuracy: 0.8561 - val_loss: 0.5330 - va
l_accuracy: 0.8553
Epoch 49/50
152/152 - 0s - loss: 0.3204 - accuracy: 0.8487 - val_loss: 0.5704 - va
l_accuracy: 0.8257
Epoch 50/50
152/152 - 0s - loss: 0.2979 - accuracy: 0.8569 - val_loss: 0.5510 - va
l_accuracy: 0.8092
```

In [56]:

```
## validtion Accuracy vs Epochs
epochs_1 = list(range(0,51))
val_acc_list_1 = model_hist.history['val_accuracy']
val_loss_list_1 = model_hist.history['val_loss']
```

In [57]:

```
# Predicting the Test set results
pred_labels_1 = model.predict(normalized_test_data)
pred_labels_1
```

Out[57]:

```
array([[1.88408914e-04, 1.37231029e-06, 4.27967696e-11, ...,
        9.99272525e-01, 2.16762626e-08, 7.70998270e-07],
       [6.19398630e-13, 3.06843138e-22, 1.09105019e-27, ...,
        3.23153403e-14, 1.00000000e+00, 7.64641153e-16],
       [2.35903873e-21, 1.89048307e-24, 0.00000000e+00, ...,
        1.00000000e+00, 8.35992297e-34, 6.35128925e-26],
       ...,
       [4.09944696e-05, 9.99936342e-01, 1.09017790e-08, ...,
        5.64928087e-08, 8.64806116e-10, 1.79170820e-05],
       [1.00533954e-08, 5.58873892e-10, 9.02292584e-07, ...,
        1.83712629e-07, 8.70593431e-05, 2.53592897e-10],
       [8.57450334e-21, 7.84091994e-16, 1.00000000e+00, ...,
        1.67106183e-13, 3.87360674e-17, 5.38070588e-22]], dtype=float3
2)
```

In [58]:

```
pred_labels_list_1 = []
for i in pred_labels_1:
    pred_labels_list_1.append(np.argmax(i))
```

In [59]:

```
pred_labels_df_1 = pd.DataFrame(pred_labels_list_1, columns=['pred_label_1'])
pred_labels_df_1
```

Out[59]:

	pred_label_1
0	7
1	8
2	7
3	6
4	0
...	...
375	7
376	9
377	1
378	3
379	2

380 rows × 1 columns

In [60]:

```
## Confusion Matrix
model_1_cm = confusion_matrix(test_labels, pred_labels_df_1)
model_1_cm
```

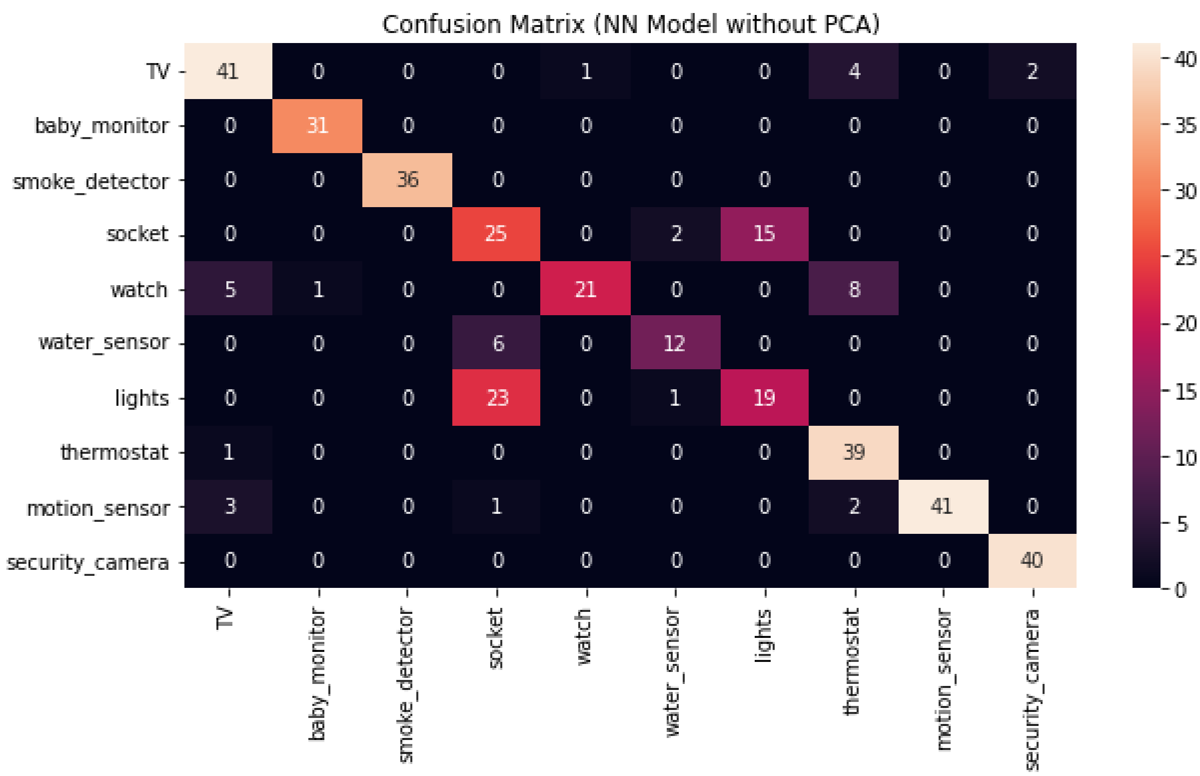
Out[60]:

```
array([[41,  0,  0,  0,  1,  0,  0,  4,  0,  2],
       [ 0, 31,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0, 36,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0, 25,  0,  2, 15,  0,  0,  0],
       [ 5,  1,  0,  0, 21,  0,  0,  8,  0,  0],
       [ 0,  0,  0,  6,  0, 12,  0,  0,  0,  0],
       [ 0,  0,  0, 23,  0,  1, 19,  0,  0,  0],
       [ 1,  0,  0,  0,  0,  0,  0, 39,  0,  0],
       [ 3,  0,  0,  1,  0,  0,  0,  2, 41,  0],
       [ 0,  0,  0,  0,  0,  0,  0,  0,  0, 40]])
```

In [61]:

```
## Classification Report
class_names = ['TV', 'baby_monitor', 'smoke_detector','socket',
               'watch',
               'water_sensor',
               'lights',
               'thermostat',
               'motion_sensor',
               'security_camera']

plt.figure(figsize= (10,5))
c1 = sns.heatmap(model_1_cm, annot=True, fmt="d",  xticklabels=class_name
s, yticklabels=class_names)
c1 = c1.set_title("Confusion Matrix (NN Model without PCA)")
```



In [62]:
```
## Classification Report
print(classification_report(test_labels, pred_labels_df_1, target_names=c
lass_names))
print(f"Computational Time by Model 1: {computational_time_1} seconds")
```

```
                 precision    recall  f1-score   support

              TV       0.82      0.85      0.84        48
     baby_monitor       0.97      1.00      0.98        31
   smoke_detector       1.00      1.00      1.00        36
           socket       0.45      0.60      0.52        42
            watch       0.95      0.60      0.74        35
     water_sensor       0.80      0.67      0.73        18
           lights       0.56      0.44      0.49        43
       thermostat       0.74      0.97      0.84        40
    motion_sensor       1.00      0.87      0.93        47
  security_camera       0.95      1.00      0.98        40

        accuracy                           0.80       380
       macro avg       0.82      0.80      0.80       380
    weighted avg       0.82      0.80      0.80       380

Computational Time by Model 1: 17.601972579956055 seconds
```

In [63]:
```
## Evaluation on Test Data
scores = model.evaluate(normalized_test_data, binarized_test_labels)
print(f'Test Accuracy: {scores[1]*100} % && Test Loss: {scores[0]}')
```

```
12/12 [==============================] - 0s 3ms/step - loss: 1.1500 -
accuracy: 0.8026
Test Accuracy: 80.26315569877625 % && Test Loss: 1.1500059366226196
```

Model 2: NN Model with PCA

In [75]:

```
## Building Model 2 with PCA extracted features
model2 = keras.Sequential()
model2.add(Dense(512, activation='relu', input_shape=(100,))) ## input sh
ape is same as PCA extracted features
model2.add(Dropout(rate = 0.25))
model2.add(Dense(256, activation='relu'))
model2.add(Dense(128, activation='relu'))
model2.add(Dropout(rate = 0.25))
model2.add(Dense(64, activation='relu'))
model2.add(Dense(10, activation='softmax')) ## Number of Neurons in Last l
ayer == Number of Categories present in Dataset
model2.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_20 (Dense)             (None, 512)               51712     
_________________________________________________________________
dropout_8 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_21 (Dense)             (None, 256)               131328    
_________________________________________________________________
dense_22 (Dense)             (None, 128)               32896     
_________________________________________________________________
dropout_9 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_23 (Dense)             (None, 64)                8256      
_________________________________________________________________
dense_24 (Dense)             (None, 10)                650       
=================================================================
Total params: 224,842
Trainable params: 224,842
Non-trainable params: 0
_________________________________________________________________
```

In [76]:
```
## Compile Model 2
model2.compile(optimizer='adam', loss='categorical_crossentropy', metrics
=['accuracy'])
```

In [77]:

```
## Fit PCA Training Data on Neural Network
start_time_2 = time.time()
model2_hist = model2.fit(pca_train_data_df, binarized_train_labels,
                         epochs=50,
                         batch_size= 8,
                         validation_split=0.2,
                         verbose=2)
computational_time_2 = time.time() - start_time_2
```

```
Epoch 1/50
152/152 - 1s - loss: 1.1001 - accuracy: 0.5839 - val_loss: 0.5854 - va
l_accuracy: 0.7171
Epoch 2/50
152/152 - 0s - loss: 0.5081 - accuracy: 0.7837 - val_loss: 0.4300 - va
l_accuracy: 0.7763
Epoch 3/50
152/152 - 0s - loss: 0.4077 - accuracy: 0.8191 - val_loss: 0.4048 - va
l_accuracy: 0.8026
Epoch 4/50
152/152 - 0s - loss: 0.3555 - accuracy: 0.8298 - val_loss: 0.4178 - va
l_accuracy: 0.8092
Epoch 5/50
152/152 - 0s - loss: 0.3477 - accuracy: 0.8166 - val_loss: 0.4416 - va
l_accuracy: 0.8191
Epoch 6/50
152/152 - 0s - loss: 0.3264 - accuracy: 0.8396 - val_loss: 0.4092 - va
l_accuracy: 0.8158
Epoch 7/50
152/152 - 0s - loss: 0.3170 - accuracy: 0.8421 - val_loss: 0.3787 - va
l_accuracy: 0.8289
Epoch 8/50
152/152 - 0s - loss: 0.2955 - accuracy: 0.8396 - val_loss: 0.4125 - va
l_accuracy: 0.8125
Epoch 9/50
152/152 - 0s - loss: 0.3010 - accuracy: 0.8363 - val_loss: 0.4339 - va
l_accuracy: 0.8125
Epoch 10/50
152/152 - 0s - loss: 0.3089 - accuracy: 0.8421 - val_loss: 0.3938 - va
l_accuracy: 0.8191
Epoch 11/50
152/152 - 0s - loss: 0.2933 - accuracy: 0.8429 - val_loss: 0.4050 - va
l_accuracy: 0.8224
Epoch 12/50
152/152 - 0s - loss: 0.2776 - accuracy: 0.8528 - val_loss: 0.4321 - va
l_accuracy: 0.8158
Epoch 13/50
152/152 - 0s - loss: 0.2832 - accuracy: 0.8503 - val_loss: 0.4158 - va
l_accuracy: 0.8421
Epoch 14/50
```

```
152/152 - 0s - loss: 0.2950 - accuracy: 0.8512 - val_loss: 0.4437 - va
l_accuracy: 0.8322
Epoch 15/50
152/152 - 0s - loss: 0.2765 - accuracy: 0.8594 - val_loss: 0.4417 - va
l_accuracy: 0.8322
Epoch 16/50
152/152 - 0s - loss: 0.2868 - accuracy: 0.8528 - val_loss: 0.4520 - va
l_accuracy: 0.8322
Epoch 17/50
152/152 - 0s - loss: 0.2912 - accuracy: 0.8610 - val_loss: 0.4501 - va
l_accuracy: 0.8520
Epoch 18/50
152/152 - 0s - loss: 0.2652 - accuracy: 0.8651 - val_loss: 0.4161 - va
l_accuracy: 0.8553
Epoch 19/50
152/152 - 0s - loss: 0.2557 - accuracy: 0.8643 - val_loss: 0.4426 - va
l_accuracy: 0.8322
Epoch 20/50
152/152 - 0s - loss: 0.2548 - accuracy: 0.8676 - val_loss: 0.4800 - va
l_accuracy: 0.8553
Epoch 21/50
152/152 - 0s - loss: 0.2609 - accuracy: 0.8594 - val_loss: 0.4980 - va
l_accuracy: 0.8553
Epoch 22/50
152/152 - 0s - loss: 0.2665 - accuracy: 0.8627 - val_loss: 0.5077 - va
l_accuracy: 0.8421
Epoch 23/50
152/152 - 0s - loss: 0.2919 - accuracy: 0.8627 - val_loss: 0.4362 - va
l_accuracy: 0.8289
Epoch 24/50
152/152 - 0s - loss: 0.2668 - accuracy: 0.8594 - val_loss: 0.4201 - va
l_accuracy: 0.8355
Epoch 25/50
152/152 - 0s - loss: 0.2633 - accuracy: 0.8610 - val_loss: 0.4970 - va
l_accuracy: 0.8355
Epoch 26/50
152/152 - 0s - loss: 0.2475 - accuracy: 0.8668 - val_loss: 0.4804 - va
l_accuracy: 0.8684
Epoch 27/50
152/152 - 0s - loss: 0.2649 - accuracy: 0.8610 - val_loss: 0.4937 - va
l_accuracy: 0.8553
```

```
Epoch 28/50
152/152 - 0s - loss: 0.2681 - accuracy: 0.8635 - val_loss: 0.4639 - va
l_accuracy: 0.8257
Epoch 29/50
152/152 - 0s - loss: 0.2553 - accuracy: 0.8618 - val_loss: 0.4603 - va
l_accuracy: 0.8421
Epoch 30/50
152/152 - 0s - loss: 0.2428 - accuracy: 0.8734 - val_loss: 0.4986 - va
l_accuracy: 0.8553
Epoch 31/50
152/152 - 1s - loss: 0.2517 - accuracy: 0.8668 - val_loss: 0.5172 - va
l_accuracy: 0.8289
Epoch 32/50
152/152 - 0s - loss: 0.2614 - accuracy: 0.8668 - val_loss: 0.4777 - va
l_accuracy: 0.8651
Epoch 33/50
152/152 - 0s - loss: 0.2663 - accuracy: 0.8635 - val_loss: 0.4860 - va
l_accuracy: 0.8618
Epoch 34/50
152/152 - 0s - loss: 0.2586 - accuracy: 0.8709 - val_loss: 0.4922 - va
l_accuracy: 0.8289
Epoch 35/50
152/152 - 0s - loss: 0.2821 - accuracy: 0.8676 - val_loss: 0.6354 - va
l_accuracy: 0.8553
Epoch 36/50
152/152 - 0s - loss: 0.2751 - accuracy: 0.8602 - val_loss: 0.5387 - va
l_accuracy: 0.8586
Epoch 37/50
152/152 - 0s - loss: 0.2541 - accuracy: 0.8594 - val_loss: 0.5448 - va
l_accuracy: 0.8618
Epoch 38/50
152/152 - 0s - loss: 0.2543 - accuracy: 0.8725 - val_loss: 0.5183 - va
l_accuracy: 0.8388
Epoch 39/50
152/152 - 0s - loss: 0.2553 - accuracy: 0.8717 - val_loss: 0.5576 - va
l_accuracy: 0.8421
Epoch 40/50
152/152 - 0s - loss: 0.2471 - accuracy: 0.8676 - val_loss: 0.4760 - va
l_accuracy: 0.8618
Epoch 41/50
152/152 - 0s - loss: 0.2858 - accuracy: 0.8692 - val_loss: 0.4529 - va
```

```
l_accuracy: 0.8618
Epoch 42/50
152/152 - 0s - loss: 0.2489 - accuracy: 0.8684 - val_loss: 0.4914 - va
l_accuracy: 0.8586
Epoch 43/50
152/152 - 0s - loss: 0.2454 - accuracy: 0.8709 - val_loss: 0.5147 - va
l_accuracy: 0.8454
Epoch 44/50
152/152 - 0s - loss: 0.2512 - accuracy: 0.8725 - val_loss: 0.6844 - va
l_accuracy: 0.8520
Epoch 45/50
152/152 - 0s - loss: 0.2657 - accuracy: 0.8676 - val_loss: 0.5048 - va
l_accuracy: 0.8520
Epoch 46/50
152/152 - 0s - loss: 0.2888 - accuracy: 0.8676 - val_loss: 0.5618 - va
l_accuracy: 0.8487
Epoch 47/50
152/152 - 1s - loss: 0.2435 - accuracy: 0.8783 - val_loss: 0.5465 - va
l_accuracy: 0.8618
Epoch 48/50
152/152 - 0s - loss: 0.2577 - accuracy: 0.8676 - val_loss: 0.5599 - va
l_accuracy: 0.8750
Epoch 49/50
152/152 - 0s - loss: 0.2407 - accuracy: 0.8758 - val_loss: 0.5659 - va
l_accuracy: 0.8651
Epoch 50/50
152/152 - 0s - loss: 0.2373 - accuracy: 0.8775 - val_loss: 0.5766 - va
l_accuracy: 0.8487
```

In [78]:

```
## validtion Accuracy vs Epochs
epochs = list(range(0,51))
val_acc2_list = model2_hist.history['val_accuracy']
val_loss2_list = model2_hist.history['val_loss']
```

In [79]:

```
# Predicting the Test set results
pred_labels_2 = model2.predict(pca_test_data_df)
pred_labels_2
```

Out[79]:

```
array([[3.1030449e-14, 3.8727879e-20, 5.8227278e-29, ..., 3.9028641e-1
3,
        3.7697266e-32, 3.5818726e-24],
       [5.6803744e-14, 1.5594415e-16, 8.3492881e-13, ..., 1.8104543e-1
6,
        1.0000000e+00, 9.8025383e-17],
       [2.7458778e-16, 1.7792040e-19, 5.5705274e-14, ..., 1.0000000e+0
0,
        4.3650569e-18, 2.7057520e-16],
       ...,
       [1.5263764e-09, 1.0000000e+00, 2.0637056e-10, ..., 3.2200960e-1
4,
        1.8849373e-10, 1.1423950e-08],
       [7.6234907e-11, 4.2779367e-09, 2.1513248e-08, ..., 2.3533402e-0
9,
        5.3162166e-05, 2.6893229e-08],
       [0.0000000e+00, 1.3092337e-33, 1.0000000e+00, ..., 1.0025112e-3
4,
        1.6918529e-25, 6.8233339e-32]], dtype=float32)
```

In [80]:

```
pred_labels_list_2 = []
for i in pred_labels_2:
    pred_labels_list_2.append(np.argmax(i))
```

In [81]:

```
pred2_labels_df = pd.DataFrame(pred_labels_list_2, columns=['pred_label_
2'])
```

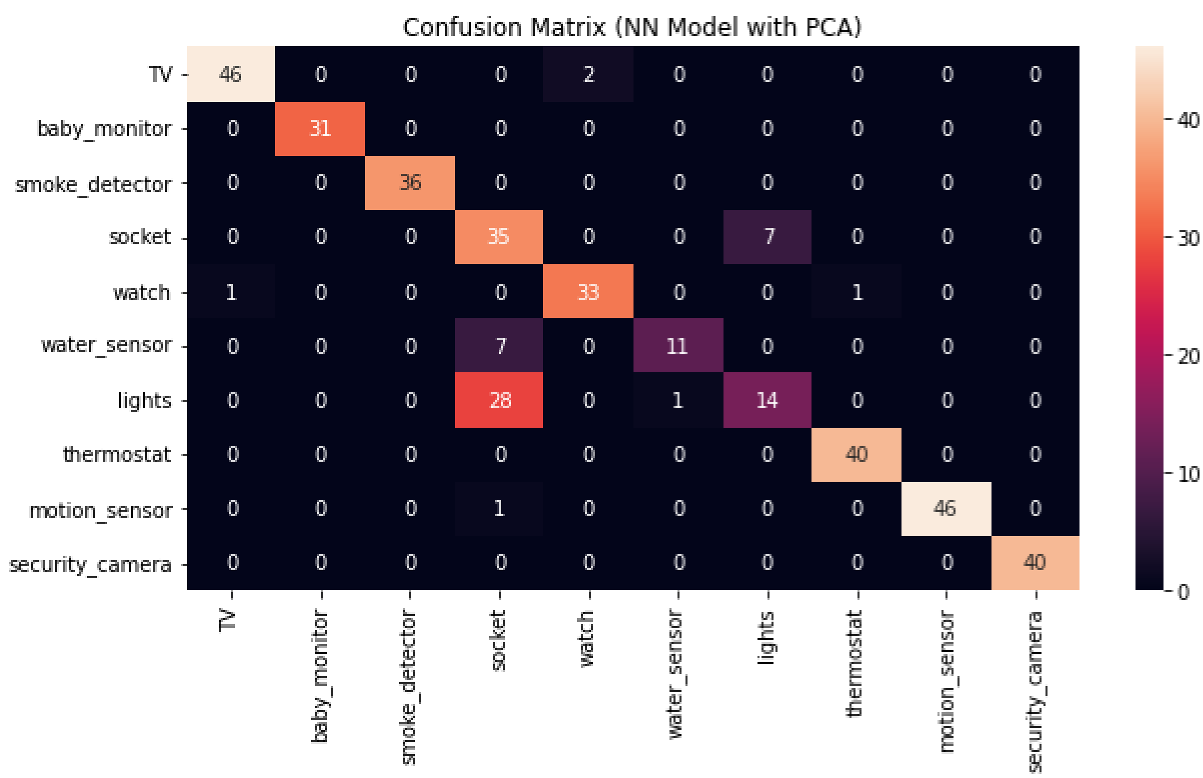
In [82]:
```
## Confusion Matrix
model_2_cm = confusion_matrix(test_labels, pred2_labels_df)
model_2_cm
```

Out[82]:
```
array([[46,  0,  0,  0,  2,  0,  0,  0,  0,  0],
       [ 0, 31,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0, 36,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0, 35,  0,  0,  7,  0,  0,  0],
       [ 1,  0,  0,  0, 33,  0,  0,  1,  0,  0],
       [ 0,  0,  0,  7,  0, 11,  0,  0,  0,  0],
       [ 0,  0,  0, 28,  0,  1, 14,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0,  0, 40,  0,  0],
       [ 0,  0,  0,  1,  0,  0,  0,  0, 46,  0],
       [ 0,  0,  0,  0,  0,  0,  0,  0,  0, 40]])
```

In [83]:

```
## Classification Report
class_names = ['TV', 'baby_monitor', 'smoke_detector','socket',
               'watch',
               'water_sensor',
               'lights',
               'thermostat',
               'motion_sensor',
               'security_camera']

plt.figure(figsize= (10,5))
s2 = sns.heatmap(model_2_cm, annot=True, fmt="d",  xticklabels=class_name
s, yticklabels=class_names)
s2 = s2.set_title("Confusion Matrix (NN Model with PCA)")
```



In [84]:

```
## Classification Report
print(classification_report(test_labels, pred2_labels_df, target_names=cl
ass_names))
print(f"Computational Time by Model 2: {computational_time_2} seconds")
```

```
                 precision    recall  f1-score   support

             TV       0.98      0.96      0.97        48
    baby_monitor       1.00      1.00      1.00        31
  smoke_detector       1.00      1.00      1.00        36
          socket       0.49      0.83      0.62        42
           watch       0.94      0.94      0.94        35
    water_sensor       0.92      0.61      0.73        18
          lights       0.67      0.33      0.44        43
      thermostat       0.98      1.00      0.99        40
   motion_sensor       1.00      0.98      0.99        47
 security_camera       1.00      1.00      1.00        40

        accuracy                           0.87       380
       macro avg       0.90      0.86      0.87       380
    weighted avg       0.89      0.87      0.87       380

Computational Time by Model 2: 20.881667613983154 seconds
```

In [85]:

```
## Evaluation of Model 2
scores2 = model2.evaluate(pca_test_data_df, binarized_test_labels)
print(f'Test Accuracy: {scores2[1]*100} % && Test Loss: {scores2[0]}')
```

```
12/12 [==============================] - 0s 2ms/step - loss: 0.3638 -
accuracy: 0.8737
Test Accuracy: 87.36842274665833 % && Test Loss: 0.36377066373825073
```

Model 3: NN Model with SGD Optimizer

In [86]:

```
## Building Model 3 with PCA extracted features
model3 = keras.Sequential()
model3.add(Dense(512, activation='relu', input_shape=(100,))) ## input sh
ape is same as PCA extracted features
model3.add(Dropout(rate = 0.25))
model3.add(Dense(256, activation='relu'))
model3.add(Dense(128, activation='relu'))
model3.add(Dropout(rate = 0.25))
model3.add(Dense(64, activation='relu'))
model3.add(Dense(10, activation='softmax')) ## Number of Neurons in Last l
ayer == Number of Categories present in Dataset
model3.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_25 (Dense)             (None, 512)               51712
_________________________________________________________________
dropout_10 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_26 (Dense)             (None, 256)               131328
_________________________________________________________________
dense_27 (Dense)             (None, 128)               32896
_________________________________________________________________
dropout_11 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_28 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_29 (Dense)             (None, 10)                650
=================================================================
Total params: 224,842
Trainable params: 224,842
Non-trainable params: 0
_________________________________________________________________
```

In [87]:
```
## Compile Model 3
model3.compile(optimizer='SGD', loss='categorical_crossentropy', metrics=
['accuracy'])
```

In [88]:

```
## Fit PCA Training Data on Neural Network
start_time_3 = time.time()
model3_hist = model3.fit(pca_train_data_df, binarized_train_labels,
                         epochs=50,
                         batch_size= 8,
                         validation_split=0.2,
                         verbose=2)
computational_time_3 = time.time() - start_time_3
```

```
Epoch 1/50
152/152 - 1s - loss: 2.1464 - accuracy: 0.2155 - val_loss: 1.9068 - va
l_accuracy: 0.3520
Epoch 2/50
152/152 - 0s - loss: 1.7185 - accuracy: 0.4145 - val_loss: 1.4631 - va
l_accuracy: 0.5164
Epoch 3/50
152/152 - 0s - loss: 1.3269 - accuracy: 0.5255 - val_loss: 1.1462 - va
l_accuracy: 0.5691
Epoch 4/50
152/152 - 0s - loss: 1.1062 - accuracy: 0.5831 - val_loss: 0.9991 - va
l_accuracy: 0.6184
Epoch 5/50
152/152 - 0s - loss: 0.9889 - accuracy: 0.6069 - val_loss: 0.9498 - va
l_accuracy: 0.6414
Epoch 6/50
152/152 - 0s - loss: 0.8986 - accuracy: 0.6291 - val_loss: 0.8407 - va
l_accuracy: 0.6678
Epoch 7/50
152/152 - 0s - loss: 0.8376 - accuracy: 0.6653 - val_loss: 0.7918 - va
l_accuracy: 0.6809
Epoch 8/50
152/152 - 0s - loss: 0.7804 - accuracy: 0.6941 - val_loss: 0.7392 - va
l_accuracy: 0.7138
Epoch 9/50
152/152 - 0s - loss: 0.7252 - accuracy: 0.6900 - val_loss: 0.7082 - va
l_accuracy: 0.7039
Epoch 10/50
152/152 - 0s - loss: 0.6676 - accuracy: 0.7130 - val_loss: 0.6608 - va
l_accuracy: 0.7171
Epoch 11/50
152/152 - 0s - loss: 0.6365 - accuracy: 0.7377 - val_loss: 0.6172 - va
l_accuracy: 0.7401
Epoch 12/50
152/152 - 0s - loss: 0.6017 - accuracy: 0.7385 - val_loss: 0.6199 - va
l_accuracy: 0.7138
Epoch 13/50
152/152 - 0s - loss: 0.5592 - accuracy: 0.7632 - val_loss: 0.5785 - va
l_accuracy: 0.7500
Epoch 14/50
```

```
152/152 - 0s - loss: 0.5431 - accuracy: 0.7730 - val_loss: 0.5733 - va
l_accuracy: 0.7336
Epoch 15/50
152/152 - 0s - loss: 0.5314 - accuracy: 0.7599 - val_loss: 0.5773 - va
l_accuracy: 0.7303
Epoch 16/50
152/152 - 0s - loss: 0.5200 - accuracy: 0.7689 - val_loss: 0.5403 - va
l_accuracy: 0.7467
Epoch 17/50
152/152 - 0s - loss: 0.5061 - accuracy: 0.7730 - val_loss: 0.5380 - va
l_accuracy: 0.7763
Epoch 18/50
152/152 - 0s - loss: 0.4724 - accuracy: 0.7936 - val_loss: 0.5078 - va
l_accuracy: 0.7500
Epoch 19/50
152/152 - 0s - loss: 0.4562 - accuracy: 0.7854 - val_loss: 0.5158 - va
l_accuracy: 0.7599
Epoch 20/50
152/152 - 0s - loss: 0.4555 - accuracy: 0.7845 - val_loss: 0.4846 - va
l_accuracy: 0.7664
Epoch 21/50
152/152 - 0s - loss: 0.4551 - accuracy: 0.7870 - val_loss: 0.4684 - va
l_accuracy: 0.7862
Epoch 22/50
152/152 - 0s - loss: 0.4472 - accuracy: 0.7870 - val_loss: 0.4851 - va
l_accuracy: 0.7500
Epoch 23/50
152/152 - 0s - loss: 0.4288 - accuracy: 0.7895 - val_loss: 0.4574 - va
l_accuracy: 0.7895
Epoch 24/50
152/152 - 0s - loss: 0.4164 - accuracy: 0.8092 - val_loss: 0.4523 - va
l_accuracy: 0.8026
Epoch 25/50
152/152 - 0s - loss: 0.4146 - accuracy: 0.8059 - val_loss: 0.4496 - va
l_accuracy: 0.7763
Epoch 26/50
152/152 - 0s - loss: 0.4034 - accuracy: 0.8092 - val_loss: 0.4656 - va
l_accuracy: 0.7664
Epoch 27/50
152/152 - 0s - loss: 0.4011 - accuracy: 0.8133 - val_loss: 0.4642 - va
l_accuracy: 0.7763
```

```
Epoch 28/50
152/152 - 0s - loss: 0.3842 - accuracy: 0.8125 - val_loss: 0.4336 - va
l_accuracy: 0.7928
Epoch 29/50
152/152 - 0s - loss: 0.3910 - accuracy: 0.8092 - val_loss: 0.4555 - va
l_accuracy: 0.7796
Epoch 30/50
152/152 - 0s - loss: 0.3762 - accuracy: 0.8166 - val_loss: 0.4580 - va
l_accuracy: 0.7763
Epoch 31/50
152/152 - 0s - loss: 0.3751 - accuracy: 0.8084 - val_loss: 0.4321 - va
l_accuracy: 0.7763
Epoch 32/50
152/152 - 0s - loss: 0.3638 - accuracy: 0.8232 - val_loss: 0.4300 - va
l_accuracy: 0.7928
Epoch 33/50
152/152 - 0s - loss: 0.3693 - accuracy: 0.8125 - val_loss: 0.4265 - va
l_accuracy: 0.7993
Epoch 34/50
152/152 - 0s - loss: 0.3622 - accuracy: 0.8166 - val_loss: 0.4315 - va
l_accuracy: 0.7928
Epoch 35/50
152/152 - 0s - loss: 0.3592 - accuracy: 0.8183 - val_loss: 0.4167 - va
l_accuracy: 0.7993
Epoch 36/50
152/152 - 0s - loss: 0.3486 - accuracy: 0.8240 - val_loss: 0.4091 - va
l_accuracy: 0.7961
Epoch 37/50
152/152 - 0s - loss: 0.3498 - accuracy: 0.8174 - val_loss: 0.4012 - va
l_accuracy: 0.8026
Epoch 38/50
152/152 - 0s - loss: 0.3501 - accuracy: 0.8322 - val_loss: 0.4110 - va
l_accuracy: 0.7961
Epoch 39/50
152/152 - 0s - loss: 0.3358 - accuracy: 0.8273 - val_loss: 0.4013 - va
l_accuracy: 0.8026
Epoch 40/50
152/152 - 0s - loss: 0.3364 - accuracy: 0.8265 - val_loss: 0.4132 - va
l_accuracy: 0.8125
Epoch 41/50
152/152 - 0s - loss: 0.3472 - accuracy: 0.8281 - val_loss: 0.4192 - va
```

```
l_accuracy: 0.7993
Epoch 42/50
152/152 - 0s - loss: 0.3359 - accuracy: 0.8372 - val_loss: 0.4032 - va
l_accuracy: 0.8059
Epoch 43/50
152/152 - 0s - loss: 0.3321 - accuracy: 0.8339 - val_loss: 0.4012 - va
l_accuracy: 0.8026
Epoch 44/50
152/152 - 0s - loss: 0.3266 - accuracy: 0.8322 - val_loss: 0.3992 - va
l_accuracy: 0.7993
Epoch 45/50
152/152 - 0s - loss: 0.3254 - accuracy: 0.8380 - val_loss: 0.4248 - va
l_accuracy: 0.7961
Epoch 46/50
152/152 - 0s - loss: 0.3327 - accuracy: 0.8339 - val_loss: 0.4020 - va
l_accuracy: 0.8191
Epoch 47/50
152/152 - 0s - loss: 0.3217 - accuracy: 0.8273 - val_loss: 0.4017 - va
l_accuracy: 0.7928
Epoch 48/50
152/152 - 0s - loss: 0.3200 - accuracy: 0.8141 - val_loss: 0.3945 - va
l_accuracy: 0.8059
Epoch 49/50
152/152 - 0s - loss: 0.3146 - accuracy: 0.8388 - val_loss: 0.3883 - va
l_accuracy: 0.8125
Epoch 50/50
152/152 - 0s - loss: 0.3153 - accuracy: 0.8289 - val_loss: 0.4042 - va
l_accuracy: 0.8125
```

In [89]:

```
## validtion Accuracy vs Epochs
epochs = list(range(0,51))
val_acc3_list = model3_hist.history['val_accuracy']
val_loss3_list = model3_hist.history['val_loss']
```

In [90]:

```
# Predicting the Test set results
pred_labels_3 = model3.predict(pca_test_data_df)
pred_labels_list_3 = []
for i in pred_labels_3:
    pred_labels_list_3.append(np.argmax(i))
pred3_labels_df = pd.DataFrame(pred_labels_list_3, columns=['pred_label_
3'])
```

In [91]:

```
## Confusion Matrix
model_3_cm = confusion_matrix(test_labels, pred3_labels_df)
model_3_cm
```

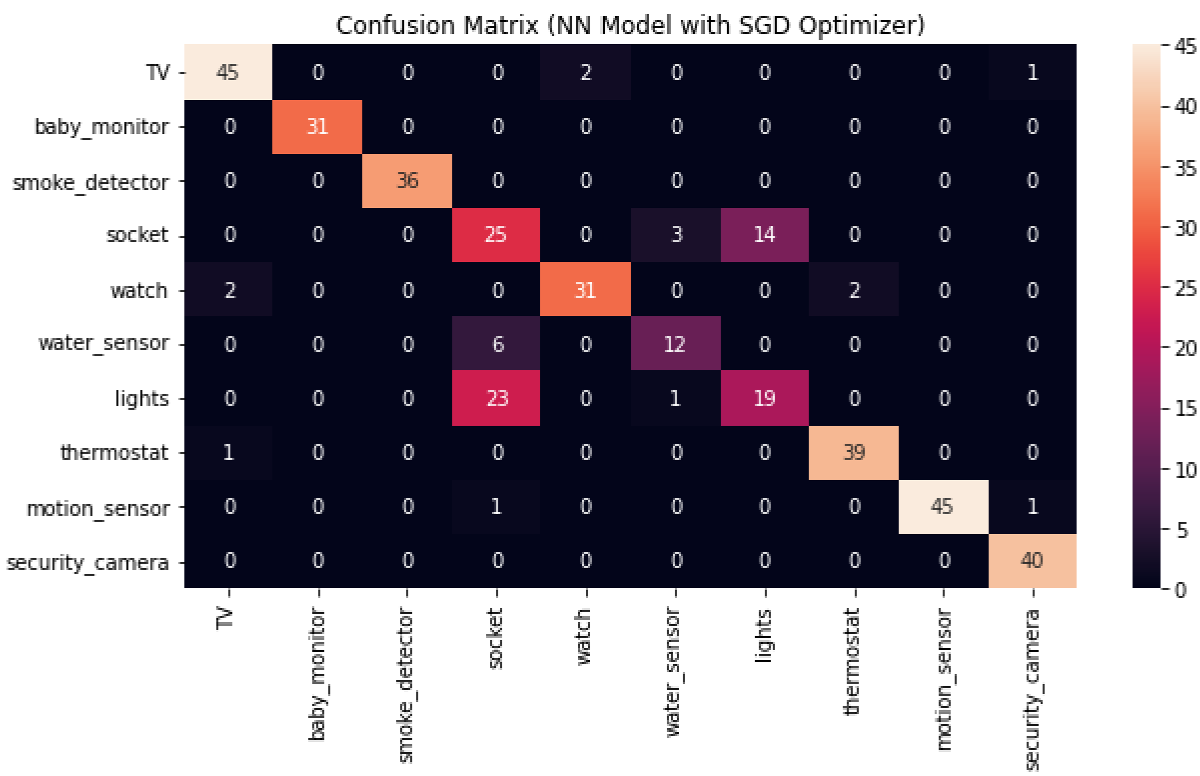
Out[91]:

```
array([[45,  0,  0,  0,  2,  0,  0,  0,  0,  1],
       [ 0, 31,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0, 36,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0, 25,  0,  3, 14,  0,  0,  0],
       [ 2,  0,  0,  0, 31,  0,  0,  2,  0,  0],
       [ 0,  0,  0,  6,  0, 12,  0,  0,  0,  0],
       [ 0,  0,  0, 23,  0,  1, 19,  0,  0,  0],
       [ 1,  0,  0,  0,  0,  0,  0, 39,  0,  0],
       [ 0,  0,  0,  1,  0,  0,  0,  0, 45,  1],
       [ 0,  0,  0,  0,  0,  0,  0,  0,  0, 40]])
```

In [92]:

```
## Classification Report
class_names = ['TV', 'baby_monitor', 'smoke_detector','socket',
               'watch',
               'water_sensor',
               'lights',
               'thermostat',
               'motion_sensor',
               'security_camera']

plt.figure(figsize= (10,5))
s3 = sns.heatmap(model_3_cm, annot=True, fmt="d",  xticklabels=class_name
s, yticklabels=class_names)
s3 = s3.set_title("Confusion Matrix (NN Model with SGD Optimizer)")
```



In [93]:
```
## Classification Report
print(classification_report(test_labels, pred3_labels_df, target_names=class_names))
print(f"Computational Time by Model 3: {computational_time_3} seconds")
```

```
                 precision    recall  f1-score   support

             TV       0.94      0.94      0.94        48
    baby_monitor       1.00      1.00      1.00        31
  smoke_detector       1.00      1.00      1.00        36
          socket       0.45      0.60      0.52        42
           watch       0.94      0.89      0.91        35
    water_sensor       0.75      0.67      0.71        18
          lights       0.58      0.44      0.50        43
      thermostat       0.95      0.97      0.96        40
   motion_sensor       1.00      0.96      0.98        47
 security_camera       0.95      1.00      0.98        40

        accuracy                           0.85       380
       macro avg       0.86      0.85      0.85       380
    weighted avg       0.86      0.85      0.85       380

Computational Time by Model 3: 20.826194524765015 seconds
```

In [94]:
```
## Evaluation of Model 3
scores3 = model3.evaluate(pca_test_data_df, binarized_test_labels)
print(f'Test Accuracy: {scores3[1]*100} % && Test Loss: {scores3[0]}')
```

```
12/12 [==============================] - 0s 3ms/step - loss: 0.3167 - accuracy: 0.8500
Test Accuracy: 85.00000238418579 % && Test Loss: 0.31669941544532776
```

Model 4: NN Model with ADAM Optimizer and PCA extracted features - Our Framework Model

In [95]:
```
## Building Model 4 with PCA extracted features
model4 = keras.Sequential()
model4.add(Dense(512, activation='relu', input_shape=(100,))) ## input sh
ape is same as PCA extracted features
model4.add(Dropout(rate = 0.25))
model4.add(Dense(256, activation='relu'))
model4.add(Dense(128, activation='relu'))
model4.add(Dropout(rate = 0.25))
model4.add(Dense(64, activation='relu'))
model4.add(Dense(10, activation='softmax')) ## Number of Neurons in Last l
ayer == Number of Categories present in Dataset
model4.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_30 (Dense)             (None, 512)               51712     
_________________________________________________________________
dropout_12 (Dropout)         (None, 512)               0         
_________________________________________________________________
dense_31 (Dense)             (None, 256)               131328    
_________________________________________________________________
dense_32 (Dense)             (None, 128)               32896     
_________________________________________________________________
dropout_13 (Dropout)         (None, 128)               0         
_________________________________________________________________
dense_33 (Dense)             (None, 64)                8256      
_________________________________________________________________
dense_34 (Dense)             (None, 10)                650       
=================================================================
Total params: 224,842
Trainable params: 224,842
Non-trainable params: 0
_________________________________________________________________
```

In [96]:
```
## Compile Model 4
model4.compile(optimizer='adam', loss='categorical_crossentropy', metrics
=['accuracy'])
```

In [97]:
```
## Fit PCA Training Data on Neural Network
start_time_4 = time.time()
model4_hist = model4.fit(pca_train_data_df, binarized_train_labels,
                      epochs=50,
                      batch_size= 8,
                      validation_split=0.2,
                      verbose=2)
computational_time_4 = time.time() - start_time_4
```

```
Epoch 1/50
152/152 - 1s - loss: 1.1200 - accuracy: 0.5773 - val_loss: 0.6230 - va
l_accuracy: 0.7105
Epoch 2/50
152/152 - 0s - loss: 0.5249 - accuracy: 0.7623 - val_loss: 0.4265 - va
l_accuracy: 0.7730
Epoch 3/50
152/152 - 0s - loss: 0.4173 - accuracy: 0.7993 - val_loss: 0.3867 - va
l_accuracy: 0.8026
Epoch 4/50
152/152 - 0s - loss: 0.3656 - accuracy: 0.8248 - val_loss: 0.4368 - va
l_accuracy: 0.8059
Epoch 5/50
152/152 - 0s - loss: 0.3442 - accuracy: 0.8183 - val_loss: 0.4888 - va
l_accuracy: 0.7928
Epoch 6/50
152/152 - 0s - loss: 0.3173 - accuracy: 0.8396 - val_loss: 0.3871 - va
l_accuracy: 0.8191
Epoch 7/50
152/152 - 0s - loss: 0.3111 - accuracy: 0.8396 - val_loss: 0.4127 - va
l_accuracy: 0.8191
Epoch 8/50
152/152 - 0s - loss: 0.2972 - accuracy: 0.8429 - val_loss: 0.4615 - va
l_accuracy: 0.8125
Epoch 9/50
152/152 - 0s - loss: 0.2919 - accuracy: 0.8421 - val_loss: 0.4662 - va
l_accuracy: 0.8059
Epoch 10/50
152/152 - 0s - loss: 0.2983 - accuracy: 0.8405 - val_loss: 0.4511 - va
l_accuracy: 0.8191
Epoch 11/50
152/152 - 0s - loss: 0.3067 - accuracy: 0.8339 - val_loss: 0.4492 - va
l_accuracy: 0.8454
Epoch 12/50
152/152 - 0s - loss: 0.2862 - accuracy: 0.8421 - val_loss: 0.5358 - va
l_accuracy: 0.8059
Epoch 13/50
152/152 - 0s - loss: 0.3304 - accuracy: 0.8273 - val_loss: 0.4265 - va
l_accuracy: 0.8520
Epoch 14/50
```

```
152/152 - 0s - loss: 0.2870 - accuracy: 0.8487 - val_loss: 0.5180 - va
l_accuracy: 0.8487
Epoch 15/50
152/152 - 1s - loss: 0.2851 - accuracy: 0.8610 - val_loss: 0.4545 - va
l_accuracy: 0.8158
Epoch 16/50
152/152 - 1s - loss: 0.2952 - accuracy: 0.8495 - val_loss: 0.5620 - va
l_accuracy: 0.8289
Epoch 17/50
152/152 - 0s - loss: 0.2847 - accuracy: 0.8388 - val_loss: 0.5627 - va
l_accuracy: 0.8257
Epoch 18/50
152/152 - 0s - loss: 0.2730 - accuracy: 0.8701 - val_loss: 0.4309 - va
l_accuracy: 0.8421
Epoch 19/50
152/152 - 0s - loss: 0.2629 - accuracy: 0.8692 - val_loss: 0.4394 - va
l_accuracy: 0.8520
Epoch 20/50
152/152 - 0s - loss: 0.2609 - accuracy: 0.8717 - val_loss: 0.4283 - va
l_accuracy: 0.8553
Epoch 21/50
152/152 - 0s - loss: 0.2559 - accuracy: 0.8717 - val_loss: 0.4763 - va
l_accuracy: 0.8289
Epoch 22/50
152/152 - 0s - loss: 0.2503 - accuracy: 0.8569 - val_loss: 0.5086 - va
l_accuracy: 0.8355
Epoch 23/50
152/152 - 0s - loss: 0.2543 - accuracy: 0.8594 - val_loss: 0.4546 - va
l_accuracy: 0.8388
Epoch 24/50
152/152 - 0s - loss: 0.2569 - accuracy: 0.8734 - val_loss: 0.4560 - va
l_accuracy: 0.8289
Epoch 25/50
152/152 - 0s - loss: 0.2760 - accuracy: 0.8594 - val_loss: 0.5001 - va
l_accuracy: 0.8322
Epoch 26/50
152/152 - 0s - loss: 0.2513 - accuracy: 0.8692 - val_loss: 0.4988 - va
l_accuracy: 0.8421
Epoch 27/50
152/152 - 0s - loss: 0.2534 - accuracy: 0.8586 - val_loss: 0.5295 - va
l_accuracy: 0.8289
```

```
Epoch 28/50
152/152 - 0s - loss: 0.2460 - accuracy: 0.8692 - val_loss: 0.5428 - va
l_accuracy: 0.8257
Epoch 29/50
152/152 - 0s - loss: 0.2670 - accuracy: 0.8586 - val_loss: 0.5392 - va
l_accuracy: 0.8158
Epoch 30/50
152/152 - 0s - loss: 0.2555 - accuracy: 0.8750 - val_loss: 0.5407 - va
l_accuracy: 0.8355
Epoch 31/50
152/152 - 0s - loss: 0.2598 - accuracy: 0.8643 - val_loss: 0.5076 - va
l_accuracy: 0.8355
Epoch 32/50
152/152 - 0s - loss: 0.2835 - accuracy: 0.8503 - val_loss: 0.5012 - va
l_accuracy: 0.8553
Epoch 33/50
152/152 - 0s - loss: 0.2757 - accuracy: 0.8668 - val_loss: 0.4627 - va
l_accuracy: 0.8421
Epoch 34/50
152/152 - 0s - loss: 0.2671 - accuracy: 0.8668 - val_loss: 0.6797 - va
l_accuracy: 0.8322
Epoch 35/50
152/152 - 0s - loss: 0.2591 - accuracy: 0.8635 - val_loss: 0.6990 - va
l_accuracy: 0.8421
Epoch 36/50
152/152 - 0s - loss: 0.2628 - accuracy: 0.8676 - val_loss: 0.6286 - va
l_accuracy: 0.8355
Epoch 37/50
152/152 - 0s - loss: 0.2480 - accuracy: 0.8734 - val_loss: 0.6293 - va
l_accuracy: 0.8684
Epoch 38/50
152/152 - 0s - loss: 0.2525 - accuracy: 0.8684 - val_loss: 0.6749 - va
l_accuracy: 0.8355
Epoch 39/50
152/152 - 0s - loss: 0.2444 - accuracy: 0.8725 - val_loss: 0.6279 - va
l_accuracy: 0.8553
Epoch 40/50
152/152 - 0s - loss: 0.2534 - accuracy: 0.8692 - val_loss: 0.5993 - va
l_accuracy: 0.8520
Epoch 41/50
152/152 - 0s - loss: 0.2385 - accuracy: 0.8775 - val_loss: 0.6320 - va
```

```
l_accuracy: 0.8454
Epoch 42/50
152/152 - 0s - loss: 0.2423 - accuracy: 0.8676 - val_loss: 0.6469 - va
l_accuracy: 0.8520
Epoch 43/50
152/152 - 0s - loss: 0.2488 - accuracy: 0.8668 - val_loss: 0.6364 - va
l_accuracy: 0.8388
Epoch 44/50
152/152 - 0s - loss: 0.2437 - accuracy: 0.8717 - val_loss: 0.6706 - va
l_accuracy: 0.8322
Epoch 45/50
152/152 - 0s - loss: 0.2428 - accuracy: 0.8750 - val_loss: 0.6876 - va
l_accuracy: 0.8388
Epoch 46/50
152/152 - 0s - loss: 0.2486 - accuracy: 0.8684 - val_loss: 0.8912 - va
l_accuracy: 0.8355
Epoch 47/50
152/152 - 0s - loss: 0.2500 - accuracy: 0.8758 - val_loss: 0.7929 - va
l_accuracy: 0.8487
Epoch 48/50
152/152 - 0s - loss: 0.2424 - accuracy: 0.8709 - val_loss: 0.8102 - va
l_accuracy: 0.8520
Epoch 49/50
152/152 - 0s - loss: 0.2392 - accuracy: 0.8832 - val_loss: 0.8711 - va
l_accuracy: 0.8553
Epoch 50/50
152/152 - 0s - loss: 0.2383 - accuracy: 0.8799 - val_loss: 0.9331 - va
l_accuracy: 0.8454
```

In [98]:

```
## validtion Accuracy vs Epochs
epochs = list(range(0,51))
val_acc4_list = model4_hist.history['val_accuracy']
val_loss4_list = model4_hist.history['val_loss']
```

In [99]:

```
# Predicting the Test set results
pred_labels_4 = model4.predict(pca_test_data_df)
pred_labels_list_4 = []
for i in pred_labels_4:
    pred_labels_list_4.append(np.argmax(i))
pred4_labels_df = pd.DataFrame(pred_labels_list_4, columns=['pred_label_
4'])
```

In [100]:

```
## Confusion Matrix
model_4_cm = confusion_matrix(test_labels, pred4_labels_df)
model_4_cm
```

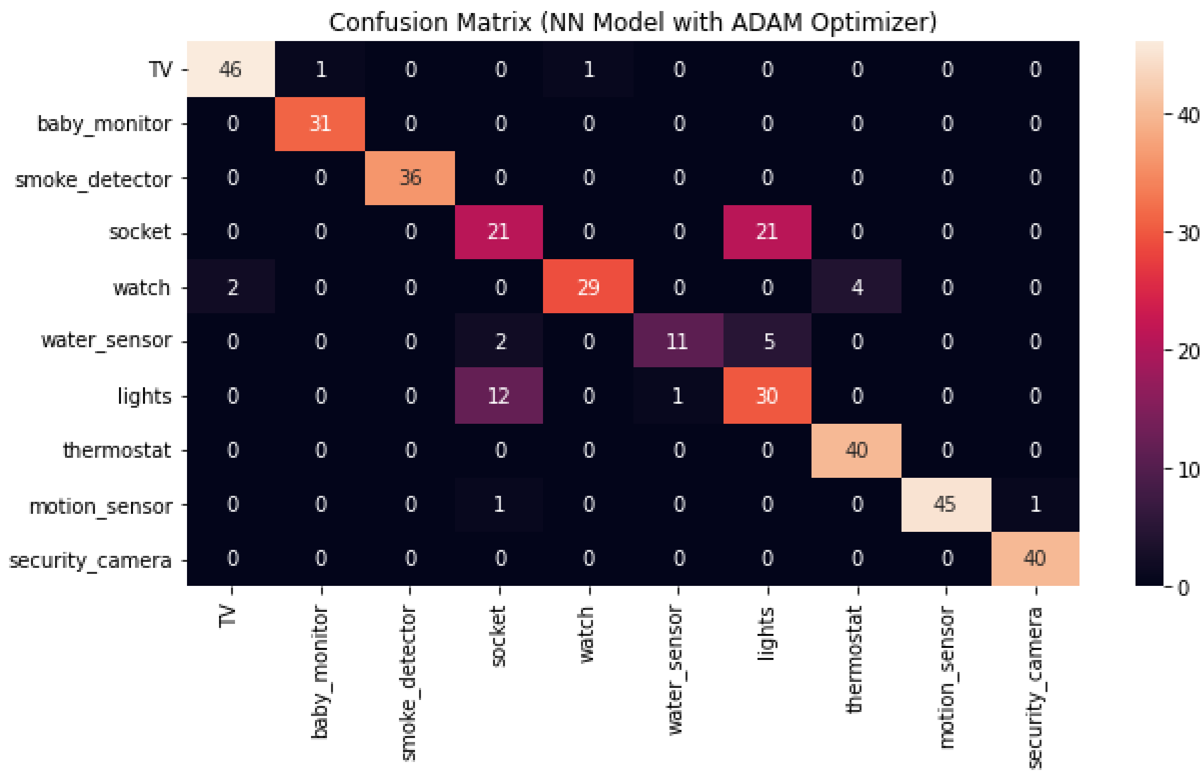
Out[100]:

```
array([[46,  1,  0,  0,  1,  0,  0,  0,  0,  0],
       [ 0, 31,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0, 36,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0, 21,  0,  0, 21,  0,  0,  0],
       [ 2,  0,  0,  0, 29,  0,  0,  4,  0,  0],
       [ 0,  0,  0,  2,  0, 11,  5,  0,  0,  0],
       [ 0,  0,  0, 12,  0,  1, 30,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0,  0, 40,  0,  0],
       [ 0,  0,  0,  1,  0,  0,  0,  0, 45,  1],
       [ 0,  0,  0,  0,  0,  0,  0,  0,  0, 40]])
```

In [101]:

```
## Classification Report
class_names = ['TV', 'baby_monitor', 'smoke_detector','socket',
               'watch',
               'water_sensor',
               'lights',
               'thermostat',
               'motion_sensor',
               'security_camera']


plt.figure(figsize= (10,5))
s4 = sns.heatmap(model_4_cm, annot=True, fmt="d",  xticklabels=class_name
s, yticklabels=class_names)
s4 = s4.set_title("Confusion Matrix (NN Model with ADAM Optimizer)")
```



In [102]:

```
## Classification Report
print(classification_report(test_labels, pred4_labels_df, target_names=class_names))
print(f"Computational Time by Model 4: {computational_time_4} seconds")
```

```
                 precision    recall  f1-score   support

             TV       0.96      0.96      0.96        48
    baby_monitor       0.97      1.00      0.98        31
  smoke_detector       1.00      1.00      1.00        36
          socket       0.58      0.50      0.54        42
           watch       0.97      0.83      0.89        35
    water_sensor       0.92      0.61      0.73        18
          lights       0.54      0.70      0.61        43
      thermostat       0.91      1.00      0.95        40
   motion_sensor       1.00      0.96      0.98        47
security_camera       0.98      1.00      0.99        40

        accuracy                           0.87       380
       macro avg       0.88      0.86      0.86       380
    weighted avg       0.87      0.87      0.87       380

Computational Time by Model 4: 20.896550178527832 seconds
```

In [103]:

```
## Evaluation of Model 4
scores4 = model4.evaluate(pca_test_data_df, binarized_test_labels)
print(f'Test Accuracy: {scores4[1]*100} % && Test Loss: {scores4[0]}')
```

```
12/12 [==============================] - 0s 2ms/step - loss: 0.4966 - accuracy: 0.8658
Test Accuracy: 86.57894730567932 % && Test Loss: 0.4965648055076599
```

In [ ]:

Additional Note: After comparing all the above models, we have finalised Model 2 as our final proposed framework that provides the highest accuracy of 87.4% among all other implemented models.