In [1]:
```
import tensorflow as tf
print(tf.__version__)
```

```
2.3.0
```

The purpose of this project was to find a truly interesting

data set that would be fairly challenging and implement a

solution using Tensorflow 2 and using generators

```
U.S. Energy Information Administration projects a 28% increase
in world energy consumption by 2040.

Driven Data Challenge: predict the type of electrical appliances
from a combination of spectrograms of current
and voltage measurements

These spectrograms were generated from current and voltage measurements
sampled at 30 kHz from 11 different appliance types present in more than
60 households in Pittsburgh, Pennsylvania, USA. Data collection took place
during the summer of 2013, and winter of 2014. Each appliance type is
represented by dozens of different instances of varying make/models.
For each appliance, plug load measurements were post-processed to extract
a two-second-long window of measurements of current and voltage. Some
observations have this window capturing the start-up transient
state (turning appliance on) and for others it capture parts of
steady-state operation (appliance is up and running). Each appliance
in the training and test datasets has two spectrograms: one for current,
and one for voltage (one image for each spectrogram).
```

In [2]:
```
#We will import the following tools based on our architectural decisions.
#The data are ping files.
#We will use mp_image to sort through the images to visualize along with
#the os library.
#We'll figure out how the images look, data types, how many there are
#We'll use the Pandas library to create dataframes
#Keras with a Tensorflow back end (with Tensorflow 2 you can technically
#see it as Keras doing the high level and tensorflow
#taking care of lower level features for great flexibility)
#this should provide an effective way to normalize the images, one_hot encode
#the labels and iterate through any dataframes we created
#Tensorflow 2 will also provide us with powerful tools to
#create a convolutional neural network to classify the images,
#we will construct the model using both the Sequential and Model options



import numpy as np
import numpy.random as nr
import matplotlib.pyplot as plt
import keras
import keras.utils.np_utils as ku
import keras.models as models
import keras.layers as layers
from keras import regularizers
from keras.layers import Dropout
import pandas as pd
import os
from matplotlib import image as mp_image
%matplotlib inline
```

In [3]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.
rect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&scope=email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.com%2fau
ve%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.readonly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly&response_type=code
s://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3aietf%3awg%
3a2.0%3aoob&scope=email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.go
s.com%2fauth%2fdrive.photos.readonly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly&response_type=code)

Enter your authorization code:
··········
Mounted at /content/drive
```

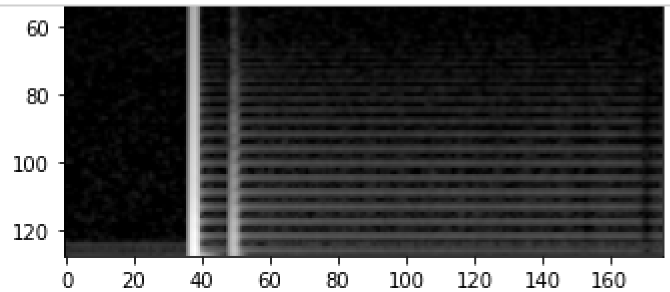
1. ETL

In [4]:
```
#Download training folder and place in the same folder as this notebook
image_folder='/content/drive/My Drive/train'
```

Visualization, data type, data size

In [71]:

```
#Use os library to create an iteration loop to look through the images using image_folder above
#using matplotlib for plotting and visualizing the images.
#At the same time that each image comes up denote its filename and shape so we can identify them
#Show each image with the information so we can get an idea of the entire data set

fig = plt.figure()
%matplotlib inline
file_names = os.listdir(image_folder)
img_num = 0
for file_name in file_names:
    file_path = os.path.join(image_folder, file_name)
    # Open the file using the matplotlib.image library
    image = mp_image.imread(file_path)
    # Add the image to the figure (which will have 1 row, a column
    #for each filename, and a position based on its index in the file_names list)
    a=fig.add_subplot(1, len(file_names), file_names.index(file_name)+1)
    # Add the image to the plot
    image_plot = plt.imshow(image)
    # Add a caption with the file name
    a.set_title(file_name)
    # Show filenames
    print(file_name)
    # Show image shape
    print(image.shape)
    # Show images
    plt.show()
# Show the plot
```




```
1691_c.png
(128, 176, 4)
```

In [6]:
```
#The images look very interesting. Definitely need a convolutional
#neural network to classify this stuff
#Images are size 128X176 and 4 dimensions
#How many images are in this folder? Let's check:
len(file_names)
```

Out[6]:
```
1152
```

Dataframes/Data Wrangling

In [7]:
```
#Note that the file names are in this format XXXX_v.png or
#XXXX_c.png for current and voltage
#Let's take a look at the image labels to see how we tie them
#to the images file names for classification
df = pd.read_csv('/content/drive/My Drive/train_labels.csv')
df
```

Out[7]:

	id	appliance
0	1000	4
1	1001	9
2	1002	4
3	1003	9
4	1004	6
...	...	...
571	1571	3
572	1572	3
573	1573	7
574	1574	3
575	1575	1

576 rows × 2 columns

In [8]:
```python
#There will be one image set for voltage,
#let's create a column that adds _v.png to the id name
df['voltage'] = df['id'].astype(str) + "_v.png"
df['voltage']
```

Out[8]:
```
0      1000_v.png
1      1001_v.png
2      1002_v.png
3      1003_v.png
4      1004_v.png
          ...
571    1571_v.png
572    1572_v.png
573    1573_v.png
574    1574_v.png
575    1575_v.png
Name: voltage, Length: 576, dtype: object
```

In [9]:
```python
#There will be one image set for current,
#let's create a column that adds _c.png to the id name
df['current'] = df['id'].astype(str)+"_c.png"
df['current']
```

Out[9]:
```
0      1000_c.png
1      1001_c.png
2      1002_c.png
3      1003_c.png
4      1004_c.png
          ...
571    1571_c.png
572    1572_c.png
573    1573_c.png
574    1574_c.png
575    1575_c.png
Name: current, Length: 576, dtype: object
```

In [10]:
```
#Let's check our dataframe with the added voltage and current columns
df
```

Out[10]:

	id	appliance	voltage	current
0	1000	4	1000_v.png	1000_c.png
1	1001	9	1001_v.png	1001_c.png
2	1002	4	1002_v.png	1002_c.png
3	1003	9	1003_v.png	1003_c.png
4	1004	6	1004_v.png	1004_c.png
...	...	...	...	...
571	1571	3	1571_v.png	1571_c.png
572	1572	3	1572_v.png	1572_c.png
573	1573	7	1573_v.png	1573_c.png
574	1574	3	1574_v.png	1574_c.png
575	1575	1	1575_v.png	1575_c.png

576 rows × 4 columns

In [11]:
```
# An appliance code ties to an appliance name,
#Let's create a dictionary that ties them together

ApplianceCategories = {0: 'Heater',
1: 'Fridge',
2: 'Hairdryer',
3: 'Microwave',
4: 'Air Conditioner',
5: 'Vacuum',
6: 'Incandescent Light Bulb',
7: 'Laptop',
8: 'Compact Fluorescent Lamp',
9: 'Fan',
10: 'Washing Machine'}
```

In [12]:
```
#Let's see how many counts of each appliance category are avaliable
df['Appliance_Categories'] = [ApplianceCategories[x] for x in df['appliance']]
df['Appliance_Categories'].value_counts()
```

Out[12]:
```
Microwave                  124
Air Conditioner            113
Fan                         78
Laptop                      59
Hairdryer                   45
Incandescent Light Bulb     35
Fridge                      34
Washing Machine             33
Compact Fluorescent Lamp    31
Heater                      15
Vacuum                       9
Name: Appliance_Categories, dtype: int64
```

In [13]:
```
# Let's make a column called Categories
#just to make it shorter and easier to use downstream
df['Categories'] = df['Appliance_Categories'][:,]
df['Categories']
```

Out[13]:
```
0              Air Conditioner
1                          Fan
2              Air Conditioner
3                          Fan
4      Incandescent Light Bulb
                ...
571                  Microwave
572                  Microwave
573                     Laptop
574                  Microwave
575                     Fridge
Name: Categories, Length: 576, dtype: object
```

In [14]:
```
#Check the dataframe
df
```

Out[14]:

	id	appliance	voltage	current	Appliance_Categories	Categories
0	1000	4	1000_v.png	1000_c.png	Air Conditioner	Air Conditioner
1	1001	9	1001_v.png	1001_c.png	Fan	Fan
2	1002	4	1002_v.png	1002_c.png	Air Conditioner	Air Conditioner
3	1003	9	1003_v.png	1003_c.png	Fan	Fan
4	1004	6	1004_v.png	1004_c.png	Incandescent Light Bulb	Incandescent Light Bulb
...	...	...	...	...	...	...
571	1571	3	1571_v.png	1571_c.png	Microwave	Microwave
572	1572	3	1572_v.png	1572_c.png	Microwave	Microwave
573	1573	7	1573_v.png	1573_c.png	Laptop	Laptop
574	1574	3	1574_v.png	1574_c.png	Microwave	Microwave
575	1575	1	1575_v.png	1575_c.png	Fridge	Fridge

576 rows × 6 columns

In [15]:
```python
#If we create a feature label set, we actually have two sets of features file names
#one for voltage and one current
X = df[['voltage','current','appliance']]
X
```

Out[15]:

	voltage	current	appliance
0	1000_v.png	1000_c.png	4
1	1001_v.png	1001_c.png	9
2	1002_v.png	1002_c.png	4
3	1003_v.png	1003_c.png	9
4	1004_v.png	1004_c.png	6
...	...	...	...
571	1571_v.png	1571_c.png	3
572	1572_v.png	1572_c.png	3
573	1573_v.png	1573_c.png	7
574	1574_v.png	1574_c.png	3
575	1575_v.png	1575_c.png	1

576 rows × 3 columns

In [16]:
```
#We'll create one df for voltage
X1=df[['voltage','Categories']]
X1
```

Out[16]:

	voltage	Categories
0	1000_v.png	Air Conditioner
1	1001_v.png	Fan
2	1002_v.png	Air Conditioner
3	1003_v.png	Fan
4	1004_v.png	Incandescent Light Bulb
...	...	...
571	1571_v.png	Microwave
572	1572_v.png	Microwave
573	1573_v.png	Laptop
574	1574_v.png	Microwave
575	1575_v.png	Fridge

576 rows × 2 columns

In [17]:
```
#And we'll create on df for current
X2=df[['current','Categories']]
X2
```

Out[17]:

	current	Categories
0	1000_c.png	Air Conditioner
1	1001_c.png	Fan
2	1002_c.png	Air Conditioner
3	1003_c.png	Fan
4	1004_c.png	Incandescent Light Bulb
...	...	...
571	1571_c.png	Microwave
572	1572_c.png	Microwave
573	1573_c.png	Laptop
574	1574_c.png	Microwave
575	1575_c.png	Fridge

576 rows × 2 columns

Scaling /Normalization /Data Augmentation, Train/Test split

Two sets of features, one for voltage and one for current

In [18]:
```python
#We will rescale/normalize the data using Keras ImageDataGenerator
#we are going to feed it to two image generators separately, either
#current or voltage, we will try to maximize the size of the
#training set as we split both voltage and current images, each set
#will have a validation split of 3/36 note that we started with a
#larger testing set but once you maximize the accuracy of a smaller
#training set (say 70%) it is beneficial to use more data in training
#especially when using deep neural networks and considering that some
#categories were very scarce in data, I also wanted to
#note, image augmentation did not work well for me, so I avoided it
#and concentrated on feature extraction by building a deeper network.
#To make the number of images divided by the batch size a whole number,
#we will make the batch size 32 which is the default in Keras but it
#seems to be a good size for this data. We are using train generators
#or this task as we've already defined the validation split for each
#generator in the ImageDataGenerator constructor. We will use the
#flow_from_data_frame option so that the train_generator can define
#the data either by current or voltage based on the dataframes
#we created. We will use Categories as labels as this column is numerical,
#we have defined it as the keys for our dictionary were the string names
#of the appliances are defined

from keras.preprocessing.image import ImageDataGenerator



train_datagen = ImageDataGenerator(
        rescale=1./255,
        #width_shift_range=0.5,
        #height_shift_range=0.5,
        #zoom_range=0.4,
        fill_mode='nearest',
        validation_split=3/36)


train_generator_1 = train_datagen.flow_from_dataframe(
        dataframe= X1,
        directory='/content/drive/My Drive/train',
        x_col="voltage",
        y_col="Categories",
        target_size=(128, 176),
        batch_size=32,
        color_mode='grayscale', #don't forget to convert to grayscale so we work with 1 dimension
        class_mode='categorical',subset='training') # set as training data

validation_generator_1 = train_datagen.flow_from_dataframe(
        dataframe=X1,
```

```
        directory='/content/drive/My Drive/train',
        x_col= "voltage",
        y_col="Categories",
        target_size=(128, 176),
        batch_size=32,
        color_mode='grayscale',
        class_mode='categorical',subset='validation') # set as validation data


train_generator_2 = train_datagen.flow_from_dataframe(
        dataframe=X2,
        directory='/content/drive/My Drive/train',
        x_col="current",
        y_col="Categories",
        target_size=(128, 176),
        batch_size=32,
        color_mode='grayscale',
        class_mode='categorical',subset='training') # set as training data

validation_generator_2 = train_datagen.flow_from_dataframe(
        dataframe=X2,
        directory='/content/drive/My Drive/train',
        x_col="current",
        y_col="Categories",
        target_size=(128, 176),
        batch_size=32,
        color_mode='grayscale',
        class_mode='categorical',subset='validation') # set as validation data

def format_gen_outputs(generator_1, generator_2):
    x1 = generator_1[0]
    x2 = generator_2[0]
    y1 = generator_1[1]
    return [x1, x2], y1

combo_train = map(format_gen_outputs, train_generator_1, train_generator_2)

combo_val = map(format_gen_outputs, validation_generator_1, validation_generator_2)
```

```
Found 528 validated image filenames belonging to 11 classes.
Found 48 validated image filenames belonging to 11 classes.
Found 528 validated image filenames belonging to 11 classes.
Found 48 validated image filenames belonging to 11 classes.
```

2. Model Creation, Training and Testing

In [19]:
```
from keras import backend as K
```

Two models, one for voltage and one for current

In [20]:

```python
# Now we define a CNN classifier network
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Activation, Flatten, Dense, BatchNormalization
from keras import optimizers
from keras import regularizers

#Note the choices here on the initial iterations started with one block of Conv2D,
#MaxPooling2D and Dropout with 32 filters, doubling the filters with each block added,
#however,
#before focusing on adding blocks it was noted that there were excessive and large
#oscillations and thus BatchNormalization was added (first before dropout),
#improvements were noted and then blocks continued to be added until a point of
#diminishing returns at this point it was noted that dropout was also needed due
#to high variance, the oscillations had improved but it the variance was still high
#overfitting to training data, and so dropout needed to also be tailored to benefit
#this particular model and an initial dropout of 0.5 throughout was reduced,
#the final pattern being more regularization on the deeper layers on the network
#and less on the more shallow layers, it was noted that excessive regularization
#had a negative effect on gradient descent and it was difficult to continue making
#progress on minimizing the loss


# Define the model as a sequence of layers
model_1 = Sequential()

model_1.add(Conv2D(32, kernel_size=3, activation = 'relu', padding = 'SAME', name= 'conv_1',
                   input_shape = train_generator_1.image_shape))
model_1.add(BatchNormalization(axis=-1))
model_1.add(MaxPooling2D(pool_size = (2,2), name = 'pool_1'))
model_1.add(Dropout(0.2))
model_1.add(Conv2D(64, kernel_size=3, activation = 'relu', padding = 'SAME', name= 'conv_2'))
model_1.add(BatchNormalization(axis=-1))
model_1.add(MaxPooling2D(pool_size = (2,2), name = 'pool_2'))
model_1.add(Dropout(0.2))
model_1.add(Conv2D(128, kernel_size=3, activation = 'relu', padding = 'SAME', name= 'conv_3'))
model_1.add(BatchNormalization(axis=-1))
model_1.add(MaxPooling2D(pool_size = (2,2), name = 'pool_3'))
model_1.add(Dropout(0.3))
model_1.add(Conv2D(256, kernel_size=3, activation = 'relu', padding = 'SAME', name= 'conv_4'))
model_1.add(BatchNormalization(axis=-1))
model_1.add(MaxPooling2D(pool_size = (2,2), name = 'pool_4'))
model_1.add(Dropout(0.3))
#model_1.add(Flatten(name = 'flatten'))
```

```python
print(model_1.summary())
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv_1 (Conv2D)              (None, 128, 176, 32)      320       
_________________________________________________________________
batch_normalization (BatchNo (None, 128, 176, 32)      128       
_________________________________________________________________
pool_1 (MaxPooling2D)        (None, 64, 88, 32)        0         
_________________________________________________________________
dropout (Dropout)            (None, 64, 88, 32)        0         
_________________________________________________________________
conv_2 (Conv2D)              (None, 64, 88, 64)        18496     
_________________________________________________________________
batch_normalization_1 (Batch (None, 64, 88, 64)        256       
_________________________________________________________________
pool_2 (MaxPooling2D)        (None, 32, 44, 64)        0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 32, 44, 64)        0         
_________________________________________________________________
conv_3 (Conv2D)              (None, 32, 44, 128)       73856     
_________________________________________________________________
batch_normalization_2 (Batch (None, 32, 44, 128)       512       
_________________________________________________________________
pool_3 (MaxPooling2D)        (None, 16, 22, 128)       0         
_________________________________________________________________
dropout_2 (Dropout)          (None, 16, 22, 128)       0         
_________________________________________________________________
conv_4 (Conv2D)              (None, 16, 22, 256)       295168    
_________________________________________________________________
batch_normalization_3 (Batch (None, 16, 22, 256)       1024      
_________________________________________________________________
pool_4 (MaxPooling2D)        (None, 8, 11, 256)        0         
_________________________________________________________________
dropout_3 (Dropout)          (None, 8, 11, 256)        0         
=================================================================
Total params: 389,760
Trainable params: 388,800
Non-trainable params: 960
_________________________________________________________________
None
```

In [21]:
```python
# Now we define a CNN classifier network
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Activation, Flatten, Dense
from keras import optimizers
from keras import regularizers

model_2 = Sequential()

model_2.add(Conv2D(32, kernel_size=3, activation = 'relu', padding = 'SAME', name= 'conv_5',
                   input_shape = train_generator_2.image_shape))
model_2.add(BatchNormalization(axis=-1))
model_2.add(MaxPooling2D(pool_size = (2,2), name = 'pool_5'))
model_2.add(Dropout(0.2))
model_2.add(Conv2D(64, kernel_size=3, activation = 'relu', padding = 'SAME', name= 'conv_6'))
model_2.add(BatchNormalization(axis=-1))
model_2.add(MaxPooling2D(pool_size = (2,2), name = 'pool_6'))
model_2.add(Dropout(0.2))
model_2.add(Conv2D(128, kernel_size=3, activation = 'relu', padding = 'SAME', name= 'conv_7'))
model_2.add(BatchNormalization(axis=-1))
model_2.add(MaxPooling2D(pool_size = (2,2), name = 'pool_7'))
model_2.add(Dropout(0.3))
model_2.add(Conv2D(256, kernel_size=3, activation = 'relu', padding = 'SAME', name= 'conv_8'))
model_2.add(BatchNormalization(axis=-1))
model_2.add(MaxPooling2D(pool_size = (2,2), name = 'pool_8'))
model_2.add(Dropout(0.3))
#model_2.add(Flatten(name = 'flatten_2'))




print(model_2.summary())
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv_5 (Conv2D)              (None, 128, 176, 32)      320
_________________________________________________________________
batch_normalization_4 (Batch (None, 128, 176, 32)      128
_________________________________________________________________
pool_5 (MaxPooling2D)        (None, 64, 88, 32)        0
_________________________________________________________________
dropout_4 (Dropout)          (None, 64, 88, 32)        0
_________________________________________________________________
conv_6 (Conv2D)              (None, 64, 88, 64)        18496
```

```
batch_normalization_5 (Batch (None, 64, 88, 64)         256
_________________________________________________________________
pool_6 (MaxPooling2D)        (None, 32, 44, 64)         0
_________________________________________________________________
dropout_5 (Dropout)          (None, 32, 44, 64)         0
_________________________________________________________________
conv_7 (Conv2D)              (None, 32, 44, 128)        73856
_________________________________________________________________
batch_normalization_6 (Batch (None, 32, 44, 128)        512
_________________________________________________________________
pool_7 (MaxPooling2D)        (None, 16, 22, 128)        0
_________________________________________________________________
dropout_6 (Dropout)          (None, 16, 22, 128)        0
_________________________________________________________________
conv_8 (Conv2D)              (None, 16, 22, 256)        295168
_________________________________________________________________
batch_normalization_7 (Batch (None, 16, 22, 256)        1024
_________________________________________________________________
pool_8 (MaxPooling2D)        (None, 8, 11, 256)         0
_________________________________________________________________
dropout_7 (Dropout)          (None, 8, 11, 256)         0
=================================================================
Total params: 389,760
Trainable params: 388,800
Non-trainable params: 960
_________________________________________________________________
None
```

Merge Models into one output

In [22]:

```python
from keras.layers import *

#Note at this point Model was chosen instead of Sequential
#to have the flexibility of custom inputs and outputs

mergedOut = Add()([model_1.output,model_2.output])
#Add() -> finalizes the merge layer that sums the inputs of the two models
#The second parentheses "calls" the layer with the output tensors of the two models
#it will demand that both model1 and model2 have the same output shape
```

In [23]:
```
#Here we'll combine the two models, or we can refer to them as
#two separate columns of layers feeding into these Flatten, Dense,
#and Dense layers below that compose the classifier portion, which is
#common in practice and on the literature. Here, we improved gradient
#descent by increasing the size of the Dense layer immediately after
#the Flatten layer, starting with 256 and doubling it's size
#while keeping hyperparameters and architecture constant,
#the best size was chosen right before
#obtaining dimishing returns.

mergedOut = Flatten(name='Flatten')(mergedOut)
mergedOut = Dense(1024, activation = 'relu', name = 'dense_1')(mergedOut)#, kernel_regularizer=tf.keras.regularizers.l2())(mergedOut)
mergedOut = BatchNormalization(axis=-1)(mergedOut)
mergedOut = Dropout(0.4)(mergedOut)
mergedOut = Dense(11, activation='softmax',name='dense_2')(mergedOut)
```

In [24]:
```
from keras.models import Model
#Create the dual input model

model = Model([model_1.input, model_2.input], mergedOut)
```

```
In [25]: print(model.summary())
```

```
Model: "functional_1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
conv_1_input (InputLayer)       [(None, 128, 176, 1) 0
__________________________________________________________________________________________________
conv_5_input (InputLayer)       [(None, 128, 176, 1) 0
__________________________________________________________________________________________________
conv_1 (Conv2D)                 (None, 128, 176, 32) 320         conv_1_input[0][0]
__________________________________________________________________________________________________
conv_5 (Conv2D)                 (None, 128, 176, 32) 320         conv_5_input[0][0]
__________________________________________________________________________________________________
batch_normalization (BatchNorma (None, 128, 176, 32) 128         conv_1[0][0]
__________________________________________________________________________________________________
batch_normalization_4 (BatchNor (None, 128, 176, 32) 128         conv_5[0][0]
__________________________________________________________________________________________________
pool_1 (MaxPooling2D)           (None, 64, 88, 32)   0           batch_normalization[0][0]
__________________________________________________________________________________________________
pool_5 (MaxPooling2D)           (None, 64, 88, 32)   0           batch_normalization_4[0][0]
__________________________________________________________________________________________________
dropout (Dropout)               (None, 64, 88, 32)   0           pool_1[0][0]
__________________________________________________________________________________________________
dropout_4 (Dropout)             (None, 64, 88, 32)   0           pool_5[0][0]
__________________________________________________________________________________________________
conv_2 (Conv2D)                 (None, 64, 88, 64)   18496       dropout[0][0]
__________________________________________________________________________________________________
conv_6 (Conv2D)                 (None, 64, 88, 64)   18496       dropout_4[0][0]
__________________________________________________________________________________________________
batch_normalization_1 (BatchNor (None, 64, 88, 64)   256         conv_2[0][0]
__________________________________________________________________________________________________
batch_normalization_5 (BatchNor (None, 64, 88, 64)   256         conv_6[0][0]
__________________________________________________________________________________________________
pool_2 (MaxPooling2D)           (None, 32, 44, 64)   0           batch_normalization_1[0][0]
__________________________________________________________________________________________________
pool_6 (MaxPooling2D)           (None, 32, 44, 64)   0           batch_normalization_5[0][0]
__________________________________________________________________________________________________
dropout_1 (Dropout)             (None, 32, 44, 64)   0           pool_2[0][0]
__________________________________________________________________________________________________
dropout_5 (Dropout)             (None, 32, 44, 64)   0           pool_6[0][0]
__________________________________________________________________________________________________
conv_3 (Conv2D)                 (None, 32, 44, 128)  73856       dropout_1[0][0]
__________________________________________________________________________________________________
conv_7 (Conv2D)                 (None, 32, 44, 128)  73856       dropout_5[0][0]
__________________________________________________________________________________________________
batch_normalization_2 (BatchNor (None, 32, 44, 128)  512         conv_3[0][0]
```

```
batch_normalization_6 (BatchNor (None, 32, 44, 128)  512         conv_7[0][0]
__________________________________________________________________________________________________
pool_3 (MaxPooling2D)           (None, 16, 22, 128)  0           batch_normalization_2[0][0]
__________________________________________________________________________________________________
pool_7 (MaxPooling2D)           (None, 16, 22, 128)  0           batch_normalization_6[0][0]
__________________________________________________________________________________________________
dropout_2 (Dropout)             (None, 16, 22, 128)  0           pool_3[0][0]
__________________________________________________________________________________________________
dropout_6 (Dropout)             (None, 16, 22, 128)  0           pool_7[0][0]
__________________________________________________________________________________________________
conv_4 (Conv2D)                 (None, 16, 22, 256)  295168      dropout_2[0][0]
__________________________________________________________________________________________________
conv_8 (Conv2D)                 (None, 16, 22, 256)  295168      dropout_6[0][0]
__________________________________________________________________________________________________
batch_normalization_3 (BatchNor (None, 16, 22, 256)  1024        conv_4[0][0]
__________________________________________________________________________________________________
batch_normalization_7 (BatchNor (None, 16, 22, 256)  1024        conv_8[0][0]
__________________________________________________________________________________________________
pool_4 (MaxPooling2D)           (None, 8, 11, 256)   0           batch_normalization_3[0][0]
__________________________________________________________________________________________________
pool_8 (MaxPooling2D)           (None, 8, 11, 256)   0           batch_normalization_7[0][0]
__________________________________________________________________________________________________
dropout_3 (Dropout)             (None, 8, 11, 256)   0           pool_4[0][0]
__________________________________________________________________________________________________
dropout_7 (Dropout)             (None, 8, 11, 256)   0           pool_8[0][0]
__________________________________________________________________________________________________
add (Add)                       (None, 8, 11, 256)   0           dropout_3[0][0]
                                                                 dropout_7[0][0]
__________________________________________________________________________________________________
Flatten (Flatten)               (None, 22528)        0           add[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 1024)         23069696    Flatten[0][0]
__________________________________________________________________________________________________
batch_normalization_8 (BatchNor (None, 1024)         4096        dense_1[0][0]
__________________________________________________________________________________________________
dropout_8 (Dropout)             (None, 1024)         0           batch_normalization_8[0][0]
__________________________________________________________________________________________________
dense_2 (Dense)                 (None, 11)           11275       dropout_8[0][0]
==================================================================================================
Total params: 23,864,587
Trainable params: 23,860,619
Non-trainable params: 3,968
__________________________________________________________________________________________________
None
```

In [26]:
```python
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
```

In [27]:
```python
#Even with the batch normalization it was noted that there were still oscillations
#even though overall the loss was decreasing, it was a necessity to save the best
#weights (highest validation accuracy)

def get_checkpoint_best_only_conv():
    """
    This function returns a ModelCheckpoint object that:
    - saves only the weights that generate the highest validation (testing) accuracy
    - saves into a directory called 'checkpoints_best_only' inside the current working directory
    - generates a file called 'checkpoints_best_only/checkpoint'
    """
    checkpoint_best_path_conv = 'checkpoints_best_only_conv/checkpoint'

    checkpoint_best = ModelCheckpoint(filepath=checkpoint_best_path_conv,
                                      save_weights_only=True,
                                      save_freq='epoch',
                                      monitor='val_accuracy',
                                      save_best_only=True,
                                      verbose=1)
    return checkpoint_best
```

In [28]:
```python
#Due to very gradual but steady increases it was needed to set the patience quite
#high on EarlyStopping callback (Monitoring the minimum validation loss)

def get_early_stopping():
    """
    This function should return an EarlyStopping callback that stops training when
    the validation (testing) accuracy has not improved in the last 3 epochs.

    """
    return tf.keras.callbacks.EarlyStopping(monitor='val_loss', mode="min", patience=50)
```

In [29]:
```python
#Best results acheived with Adam
opt = optimizers.Adam(1e-3)
#opt = optimizers.Nadam(learning_rate=0.002, beta_1=0.9, beta_2=0.999)
#opt = optimizers.RMSprop()
#opt = optimizers.SGD(learning_rate=0.0001, momentum=0.9, nesterov=True)
```

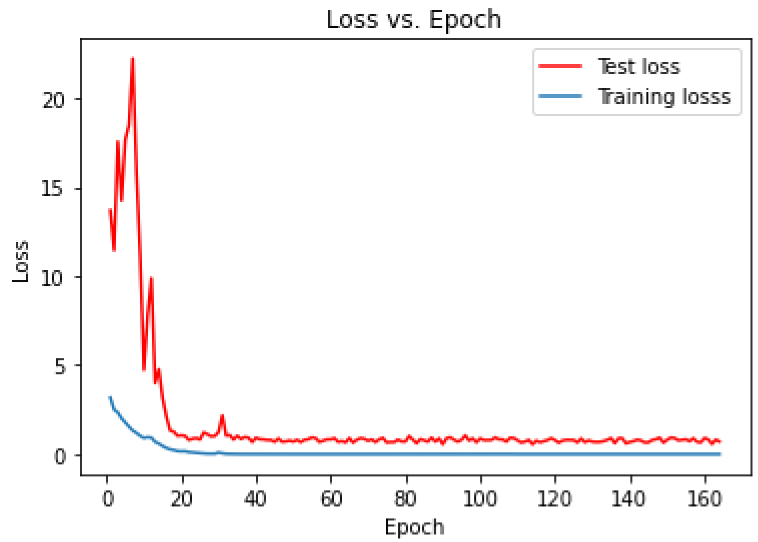
In [30]:
```
## Compile the model

model.compile(optimizer = opt, loss = 'categorical_crossentropy',
                metrics = ['accuracy'])
```

In [31]:
```
#ReduceOnPlateau added to callbacks as in the initial iterations the loss kept oscillating
#back and forth without improvement constantly, parameters were set to work well with the
#Adam optimizer
checkpoint_best_only = get_checkpoint_best_only_conv()
early_stopping = get_early_stopping()
reduce_on_plateau = tf.keras.callbacks.ReduceLROnPlateau(
                              monitor='val_loss',
                              factor=0.1, patience=10,
                              verbose=1, mode='auto', min_delta=0.0001,
                              cooldown=0, min_lr=0)

callbacks = [checkpoint_best_only, early_stopping, reduce_on_plateau]
```

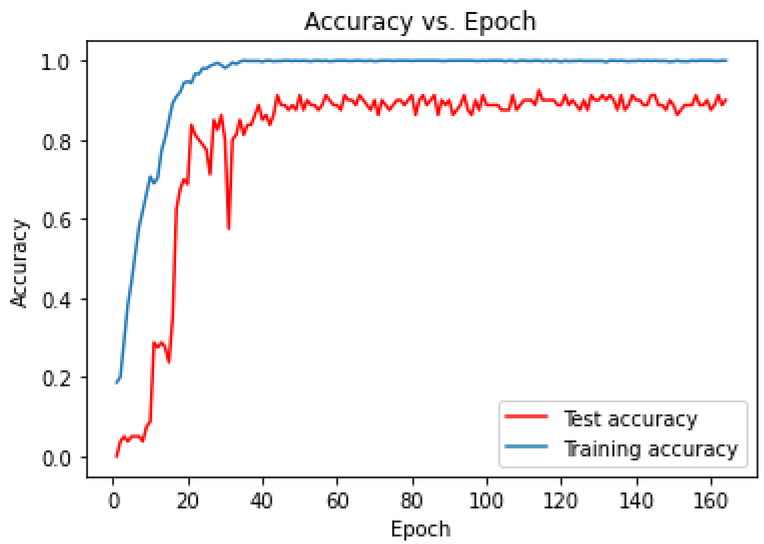
In [32]:
```python
# Train the model
num_epochs = 1000
history = model.fit(
    combo_train,
    steps_per_epoch = 1056 // 32,
    validation_data = combo_val,
    validation_steps = 96 // 32,
    epochs = num_epochs, callbacks=callbacks)
```

```
Epoch 141/1000
33/33 [==============================] - ETA: 0s - loss: 0.0044 - accuracy: 1.0000
Epoch 00141: val_accuracy did not improve from 0.92500
33/33 [==============================] - 4s 134ms/step - loss: 0.0044 - accuracy: 1.0000 - val_loss: 0.7349 - val_accuracy: 0.9000
Epoch 142/1000
33/33 [==============================] - ETA: 0s - loss: 0.0068 - accuracy: 0.9990
Epoch 00142: val_accuracy did not improve from 0.92500
33/33 [==============================] - 4s 134ms/step - loss: 0.0068 - accuracy: 0.9990 - val_loss: 0.8142 - val_accuracy: 0.8875
Epoch 143/1000
33/33 [==============================] - ETA: 0s - loss: 0.0042 - accuracy: 1.0000
Epoch 00143: val_accuracy did not improve from 0.92500
33/33 [==============================] - 4s 134ms/step - loss: 0.0042 - accuracy: 1.0000 - val_loss: 0.7737 - val_accuracy: 0.8875
Epoch 144/1000
33/33 [==============================] - ETA: 0s - loss: 0.0058 - accuracy: 0.9990
Epoch 00144: val_accuracy did not improve from 0.92500

Epoch 00144: ReduceLROnPlateau reducing learning rate to 1.000000082740371e-12.
33/33 [==============================] - 4s 136ms/step - loss: 0.0058 - accuracy: 0.9990 - val_loss: 0.6595 - val_accuracy: 0.9125
Epoch 145/1000
33/33 [==============================] - ETA: 0s - loss: 0.0044 - accuracy: 1.0000
```

In [33]:
```
def plot_loss(history):
    '''Function to plot the loss vs. epoch'''
    train_loss = history.history['loss']
    test_loss = history.history['val_loss']
    x = list(range(1, len(test_loss) + 1))
    plt.plot(x, test_loss, color = 'red', label = 'Test loss')
    plt.plot(x, train_loss, label = 'Training losss')
    plt.legend()
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.title('Loss vs. Epoch')

plot_loss(history)
```



In [34]:
```
def plot_accuracy(history):
    train_acc = history.history['accuracy']
    test_acc = history.history['val_accuracy']
    x = list(range(1, len(test_acc) + 1))
    plt.plot(x, test_acc, color = 'red', label = 'Test accuracy')
    plt.plot(x, train_acc, label = 'Training accuracy')
    plt.legend()
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.title('Accuracy vs. Epoch')

plot_accuracy(history)
```



In [35]:
```
df['Categories'].unique()
```

Out[35]:
```
array(['Air Conditioner', 'Fan', 'Incandescent Light Bulb', 'Microwave',
       'Compact Fluorescent Lamp', 'Hairdryer', 'Laptop',
       'Washing Machine', 'Fridge', 'Heater', 'Vacuum'], dtype=object)
```

In [36]:
```
classes = ['Air Conditioner', 'Fan', 'Incandescent Light Bulb', 'Microwave',
       'Compact Fluorescent Lamp', 'Hairdryer', 'Laptop',
       'Washing Machine', 'Fridge', 'Heater', 'Vacuum']
```

In [37]:
```
import numpy as np
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
%matplotlib inline

print("Generating predictions from validation data...")
# Get the image and label arrays for the first batch of validation data

combo_val = map(format_gen_outputs, validation_generator_1, validation_generator_2)
x_test_1 = validation_generator_1[0][0]
x_test_2 = validation_generator_2[0][0]
y_test = validation_generator_1[0][1]

# Use the model to predict the class
class_probabilities = model.predict([x_test_1,x_test_2])

# The model returns a probability value for each class
# The one with the highest probability is the predicted class
predictions = np.argmax(class_probabilities, axis=1)

# The actual labels are hot encoded (e.g. [0 1 0], so get the one with the value 1
true_labels = np.argmax(y_test, axis=1)

print (predictions)#predictions on top
print(true_labels)#true labels at the bottom
```

```
Generating predictions from validation data...
[ 0  1  8  2  7  8  8  0  8  0  0  7  0  2  5  8  2  0  2  8  0  0  8  8
  7  6  0  2  8  0 10  4]
[10  1  8  4  7  8  8  0  8  0  0  1  0  2  5  8  2  0  2  8  0  0  8  8
  7  6  0  2  8  0 10  4]
```

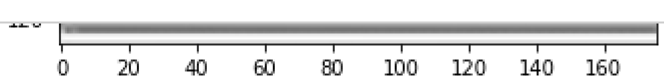
In [38]:

```
from keras.models import load_model

modelFileName = 'spectogram-classifier.h5'

model.save(modelFileName) # saves the trained model
print("Model saved.")

del model  # deletes the existing model variable
```

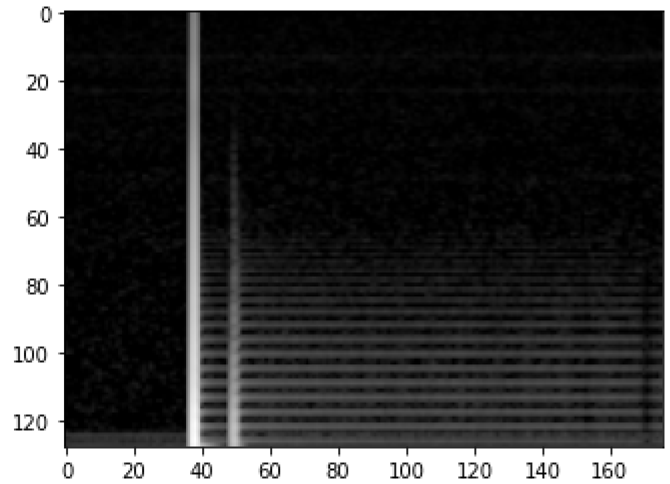
```
Model saved.
```

In [72]:

```python
#Let's load our saved model and take a look at the testing images
#(same procedure we used before)
import os
from random import randint
import numpy as np
from PIL import Image
from keras.models import load_model
from matplotlib import pyplot as plt
%matplotlib inline

# Load the saved model
modelFileName = 'spectogram-classifier.h5'
model = load_model(modelFileName)

#get the list of test image files
image_folder = '/content/drive/My Drive/test/test_files'


fig = plt.figure()
%matplotlib inline
file_names = os.listdir(image_folder)
img_num = 0
for file_name in file_names:
    file_path = os.path.join(image_folder, file_name)
    # Open the file using the matplotlib.image library
    image = mp_image.imread(file_path)
    # Add the image to the figure (which will have 1 row,
    #a column for each filename, and a position based on its index in the file_names list)
    a=fig.add_subplot(1, len(file_names), file_names.index(file_name)+1)
    # Add the image to the plot
    image_plot = plt.imshow(image)
    # Add a caption with the file name
    a.set_title(file_name)
    # Show filenames
    print(file_name)
    # Show image shape
    print(image.shape)
    # Show images
    plt.show()
```




```
1675_c.png
(128, 176, 4)
```



In [40]:
```
#The id's are different from the ones we had for training. Let's take a look at
#the submission csv, we'll create a dataframe and do what we did before to create
#a list of files representative of each id.

submission = pd.read_csv('/content/drive/My Drive/submission_format.csv')
submission
```

Out[40]:

	id	appliance
0	1576	0
1	1577	0
2	1578	0
3	1579	0
4	1580	0
...	...	...
379	1955	0
380	1956	0
381	1957	0
382	1958	0
383	1959	0

384 rows × 2 columns

In [41]:
```
submission['voltage'] = submission['id'].astype(str) + "_v.png"
submission['voltage']
```

Out[41]:
```
0      1576_v.png
1      1577_v.png
2      1578_v.png
3      1579_v.png
4      1580_v.png
          ...
379    1955_v.png
380    1956_v.png
381    1957_v.png
382    1958_v.png
383    1959_v.png
Name: voltage, Length: 384, dtype: object
```

In [42]:
```
submission['current'] = submission['id'].astype(str)+"_c.png"
submission['current']
```

Out[42]:
```
0      1576_c.png
1      1577_c.png
2      1578_c.png
3      1579_c.png
4      1580_c.png
          ...
379    1955_c.png
380    1956_c.png
381    1957_c.png
382    1958_c.png
383    1959_c.png
Name: current, Length: 384, dtype: object
```

In [51]:
```
test_current_df = submission[['id','current']]
test_voltage_df = submission[['id','voltage']]
```

In [52]:
```
test_current_df
```

Out[52]:

	id	current
0	1576	1576_c.png
1	1577	1577_c.png
2	1578	1578_c.png
3	1579	1579_c.png
4	1580	1580_c.png
...	...	...
379	1955	1955_c.png
380	1956	1956_c.png
381	1957	1957_c.png
382	1958	1958_c.png
383	1959	1959_c.png

384 rows × 2 columns

In [53]:
```
# we'll create two test generators. The key differences when testing is to not
# input a class (None), not shuffle, and since the filenames are new, set
#validate_filenames to False

test_datagen=ImageDataGenerator(rescale=1./255.)


test_generator_1 =  test_datagen.flow_from_dataframe(
                    dataframe=test_current_df,
                    directory='/content/drive/My Drive/test/test_files',
                    x_col='current',
                    batch_size=32,
                    shuffle=False,
                    class_mode=None,
                    color_mode='grayscale',
                    validate_filenames=False,
                    target_size=(128, 176))

test_generator_2 =  test_datagen.flow_from_dataframe(
                    dataframe=test_voltage_df,
                    directory='/content/drive/My Drive/test/test_files',
                    x_col='voltage',
                    batch_size=32,
                    shuffle=False,
                    class_mode=None,
                    color_mode='grayscale',
                    validate_filenames=False,
                    target_size=(128, 176))
```

```
Found 384 non-validated image filenames.
Found 384 non-validated image filenames.
```

In [54]:
```
#Let's see the first test batch

combo_val = map(format_gen_outputs, test_generator_1, test_generator_2)
x_test_gen_1 = validation_generator_1[0][0]
x_test_gen_2 = validation_generator_2[0][0]
y_test_gen = validation_generator_1[0][1]

predict = model.predict([x_test_gen_1, x_test_gen_2])
```

In [55]:
```python
predict.shape
```
Out[55]:
```
(32, 11)
```

In [56]:
```python

# Use the model to predict the class
class_probabilities = predict

# The model returns a probability value for each class
# The one with the highest probability is the predicted class
predictions = np.argmax(class_probabilities, axis=1)

print (predictions)
```
```
[ 0  1  8  2  7  8  8  0  8  0  0  7  0  2  5  8  2  0  2  8  0  0  8  8
  7  6  0  2  8  0 10  4]
```

In [57]:
```python
len(predictions)
```
Out[57]:
```
32
```

In [58]:
```python
ApplianceCategories = {0: 'Heater',
1: 'Fridge',
2: 'Hairdryer',
3: 'Microwave',
4: 'Air Conditioner',
5: 'Vacuum',
6: 'Incandescent Light Bulb',
7: 'Laptop',
8: 'Compact Fluorescent Lamp',
9: 'Fan',
10: 'Washing Machine'}
```

In [59]:
```python
#Let's see what the actual predicted classes are
predicted_classes = []
for prediction in predictions:
    ApplianceCategories[prediction]
    predicted_classes.append(ApplianceCategories[prediction])

predicted_classes
```

Out[59]:
```
['Heater',
 'Fridge',
 'Compact Fluorescent Lamp',
 'Hairdryer',
 'Laptop',
 'Compact Fluorescent Lamp',
 'Compact Fluorescent Lamp',
 'Heater',
 'Compact Fluorescent Lamp',
 'Heater',
 'Heater',
 'Laptop',
 'Heater',
 'Hairdryer',
 'Vacuum',
 'Compact Fluorescent Lamp',
 'Hairdryer',
 'Heater',
 'Hairdryer',
 'Compact Fluorescent Lamp',
 'Heater',
 'Heater',
 'Compact Fluorescent Lamp',
 'Compact Fluorescent Lamp',
 'Laptop',
 'Incandescent Light Bulb',
 'Heater',
 'Hairdryer',
 'Compact Fluorescent Lamp',
 'Heater',
 'Washing Machine',
 'Air Conditioner']
```

In [60]:
```python
len(predicted_classes)
```

Out[60]:
```
32
```

In [63]:
```
# Let's use the generators to iterate through all the batches and predict
#the full test set please note that this test set has no labels,
#it was originally intended for a Driven Data competition
#and they had the labels

combo_val = map(format_gen_outputs, test_generator_1, test_generator_2)
x_test_gen_1 = validation_generator_1[0][0]
x_test_gen_2 = validation_generator_2[0][0]
y_test_gen = validation_generator_1[0][1]

test_generator_1.reset()
test_generator_2.reset()

test_1=np.concatenate([test_generator_1.next() for i in range(test_generator_1.__len__())])
test_2=np.concatenate([test_generator_2.next() for i in range(test_generator_2.__len__())])
print(test_1.shape)
print(test_2.shape)

batch_prediction = model.predict([test_1, test_2])
```

```
(384, 128, 176, 1)
(384, 128, 176, 1)
```

In [64]:
```
batch_prediction.shape
```

Out[64]:
```
(384, 11)
```

In [65]:

```python
# Use the model predictions to predict the class
class_probabilities_full = batch_prediction

# The model returns a probability value for each class
# The one with the highest probability is the predicted class
full_predictions = np.argmax(class_probabilities_full, axis=1)

print (full_predictions)
```

```
[ 1  8  8  8  1  7 10  8 10  8  8  8  8  8 10  8  8  8  8  8  8  1  8  8
  8  0  1  8  8  6  6  8  8  6  8  8 10  1  8  0  8  4  8  8  1 10  8  6
  8  0  8  8  8  8  8 10  8  9  0  8  8  8 10  8  8 10  8  8  1  4  8  7
 10  8  4  7  2  3 10  1 10  3  0 10  8  7  8  8  8  8  8  6  8  8  6  8
 10  8  0  7  8  8 10  4  8  4  8  8  8  8 10  8  8  4 10  1  8  8  8  8
  2 10  9  8  8  9  8  6  1  1  8  0  8  0  8  8  1  7  7  0  8  0 10 10
  2  8  4  8  6  8  8  8  0  8 10  8  9 10  7 10  2  8  8  6 10  8  6  7
  9  8  7  1  8 10  8 10  8 10 10  1  8  1  9  6  9  2  8  8  6  8  8  8
  6  8  1  8 10  0  7  9  8 10  8  2  8  8  8  8 10  4  1  8  2  8  8  8
  8  0  8  2  8  8  8  0  8 10  9  1  8  6  6  8 10  8  4  8 10  8  1  8
  8 10  1  8  8  8  4  8  8  7  9  8  1  8  8  8  6  7  8  1  8  6  6  8
 10 10  8  8 10  6  8  1  8  8  0  7  1  1  0  6  1  8  8  9  8  8  8 10
  8  6  1 10  8  8  8  8  0  8 10  4  7  8  8 10  8  1  8  0  8  4 10  8
  1 10  8  0  8  8 10  0  8  8  8  6  6  8  8  7  8  8  4  8  1  8  6  1
  8  8  8  8  8  8  8  8  9  9  4 10  1  8  8  8  8  8  1  6  8 10  8  8
  8  8  8 10  8  1 10  8  1  8  7  8  8  8  8 10  8  6  8  8  6  8 10  8]
```

In [66]:
```
#Let's see what categories these numbers represent
predicted_classes_final = []
for prediction in full_predictions:
    ApplianceCategories[prediction]
    predicted_classes_final.append(ApplianceCategories[prediction])

predicted_classes_final
```

Out[66]:
```
['Fridge',
 'Compact Fluorescent Lamp',
 'Compact Fluorescent Lamp',
 'Compact Fluorescent Lamp',
 'Fridge',
 'Laptop',
 'Washing Machine',
 'Compact Fluorescent Lamp',
 'Washing Machine',
 'Compact Fluorescent Lamp',
 'Compact Fluorescent Lamp',
 'Compact Fluorescent Lamp',
 'Compact Fluorescent Lamp',
 'Compact Fluorescent Lamp',
 'Washing Machine',
 'Compact Fluorescent Lamp',
 'Compact Fluorescent Lamp',
 'Compact Fluorescent Lamp',
 'Compact Fluorescent Lamp',
```

In [67]:
```
submission.loc[:,'appliance'] = full_predictions
```

In [68]:
```python
submission.loc[:,'categories'] = predicted_classes_final
submission
```

Out[68]:

	id	appliance	voltage	current	categories
0	1576	1	1576_v.png	1576_c.png	Fridge
1	1577	8	1577_v.png	1577_c.png	Compact Fluorescent Lamp
2	1578	8	1578_v.png	1578_c.png	Compact Fluorescent Lamp
3	1579	8	1579_v.png	1579_c.png	Compact Fluorescent Lamp
4	1580	1	1580_v.png	1580_c.png	Fridge
...	...	...	...	...	...
379	1955	8	1955_v.png	1955_c.png	Compact Fluorescent Lamp
380	1956	6	1956_v.png	1956_c.png	Incandescent Light Bulb
381	1957	8	1957_v.png	1957_c.png	Compact Fluorescent Lamp
382	1958	10	1958_v.png	1958_c.png	Washing Machine
383	1959	8	1959_v.png	1959_c.png	Compact Fluorescent Lamp

384 rows × 5 columns

In [70]:
```
#The original submission format was requested with only the 'id'and 'appliance'
#columns with integers for 'appliance'
submission_final = submission[['id', 'appliance']]
submission_final
```

Out[70]:

	id	appliance
0	1576	1
1	1577	8
2	1578	8
3	1579	8
4	1580	1
...	...	...
379	1955	8
380	1956	6
381	1957	8
382	1958	10
383	1959	8

384 rows × 2 columns

In [ ]: