Required Packages

In [ ]:
```
!pip3 install h5py
!pip3 install tensorflow
```

In [ ]:
```
from tensorflow.keras import datasets, layers, models
import urllib.request

import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
import h5py
import os
```

Problem and Goals

In [ ]:
```
def download(file_name, url):
    urllib.request.urlretrieve(url, file_name)
```

NNSS GOALS:

1. Implement a more sophisticated ENN/CNN architecture, and see if we can produce improved output results, but via accuracy and statistics.
2. Develop a probability model for the architecture and provide UQ for the model.
3. The goal for the dataset would be to predict the absorption spectra from the sample images.

RESEARCH GOALS:

1. One NN vs esnsembled NN - Is the ensembled network better and if so by how much.
2. How do we structure the cost function for regression. Meaning Mean Squared Error (MSE) or some other type
3. One NN vs esembled NN related to EQ changes.

Dataset

Dataset was pulled from Caltech. Machine learning of optical properties of materials - predicting spectra from images and images from spectra.

Authors Gregoire, John Caltech 0000-0002-2863-5265 ORCID
Stein, Helge Caltech
Soedarmadji, Edwin Caltech
Newhouse, Paul Caltech
Guevarra, Dan Caltech

URL https://data.caltech.edu/records/1103

In [ ]:
```
download("dataset_raw.h5","https://bit.ly/34xI5LW")
download("training_raw.txt","https://bit.ly/3rpWZwP")
download("test_raw.txt","https://bit.ly/34xOneu")
```

In [ ]:
```
hf = h5py.File('dataset_raw.h5', 'r')
list(hf.keys())
```

Out[ ]:
```
['atfrac',
 'atfrac_keys',
 'energy_eV',
 'images',
 'loading_keys',
 'loadings',
 'plate_id',
 'sample_id',
 'spectra']
```

Images and Spectrogram

This section will be more dedicated to understanding what the data is and what it looks like.

In [ ]:

```python
images = hf['images']
labels = hf['spectra']
print(f'Dataset image size:{images.shape}')
print(f'Dataset label size:{labels.shape}')
```

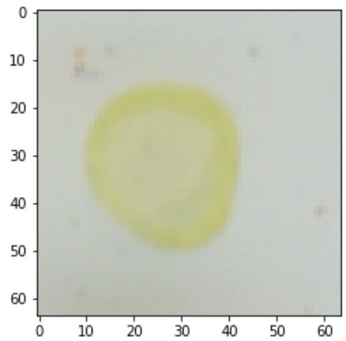
```
Dataset image size:(180902, 64, 64, 3)
Dataset label size:(180902, 220)
```

In [ ]:

```python
#print(images[0])
plt.imshow(images[0])
```

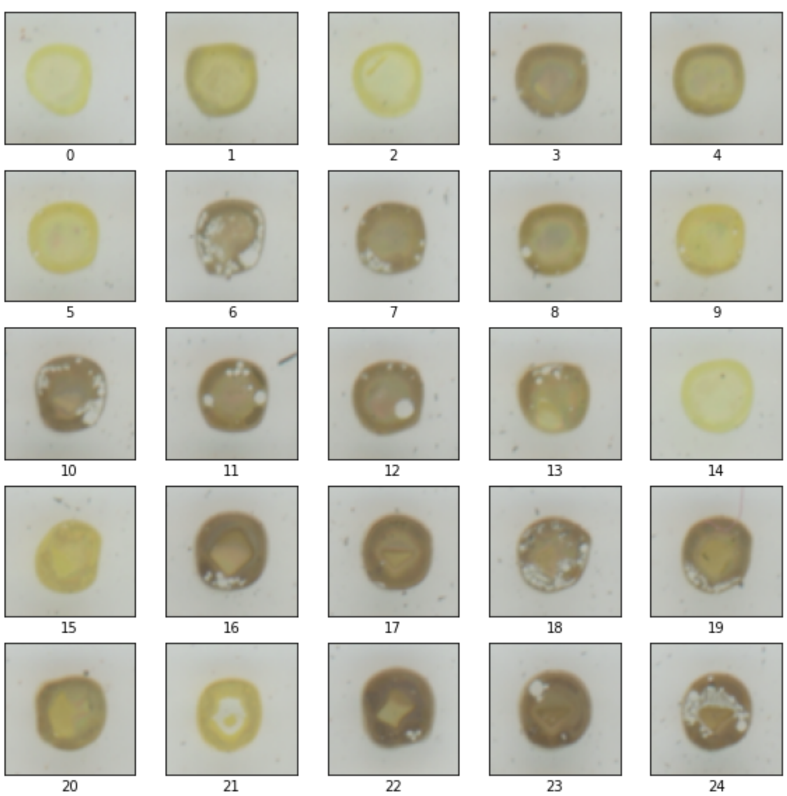
Out[ ]:

```
<matplotlib.image.AxesImage at 0x7fb0e651f9d0>
```



In [ ]:

```python
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(images[i])
    # The CIFAR labels happen to be arrays,
    # which is why you need the extra index
    plt.xlabel(i)
plt.show()
```



Spectra

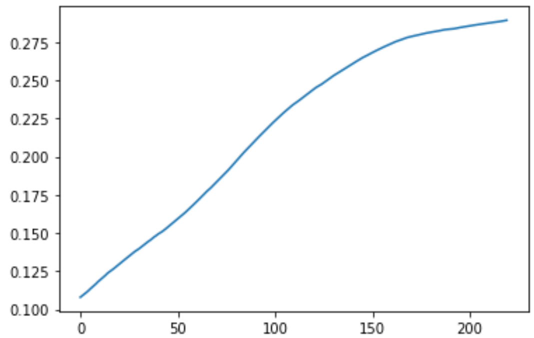
Showing the spectrogram of the example images. This will be what we will be trying to predict.

In [ ]:

```
plt.plot(labels[0])
```

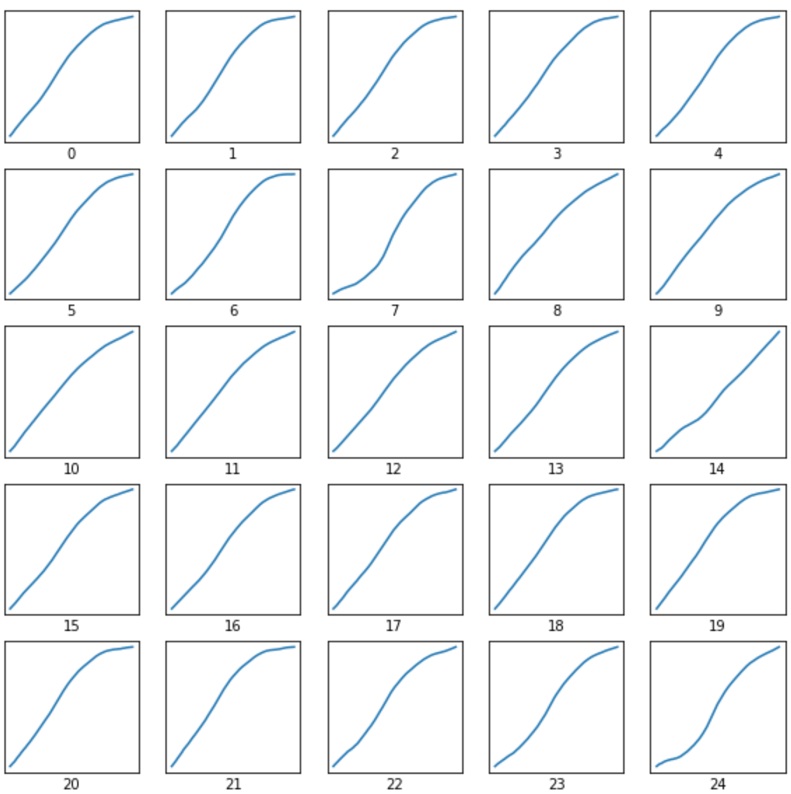
Out[ ]:

```
[<matplotlib.lines.Line2D at 0x7fb166ec2b90>]
```



In [ ]:

```
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.plot(labels[i])
    # The CIFAR labels happen to be arrays,
    # which is why you need the extra index
    plt.xlabel(i)
plt.show()
```



Split the dataset

Splitting the dataset into a 70% training and %30 testing.

In [ ]:

```
train_images = images[:144721]
train_labels = labels[:144721]

test_images = images[144722:]
test_labels = labels[144722:]
```

GPU Information

In [ ]:
```
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
  print('Not connected to a GPU')
else:
  print(gpu_info)
```

```
Num GPUs Available:  1
Tue Feb 15 06:14:41 2022
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.32.03    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla P100-PCIE...  Off  | 00000000:00:04.0 Off |                    0 |
| N/A   37C    P0    33W / 250W |  15981MiB / 16280MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
+-----------------------------------------------------------------------------+
```

Single Model

In [ ]:
```
model = models.Sequential()

# CNN
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(64, 64, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))

# Dense
model.add(layers.Flatten())
model.add(layers.Dense(300, activation='relu'))
model.add(layers.Dense(220))
```

Fitness function using mean squared error.

Might switch the data to polynomial regression. https://medium.com/analytics-vidhya/polynomial-regression-with-keras-ef1797b39b88

In [ ]:
```
#model.compile(optimizer="adam", loss="mse", metrics=["mae"])
model.compile(optimizer='adam', loss=tf.keras.losses.MeanSquaredError(), metrics=["mae"])
```

In [ ]:
```
model.fit(train_images, train_labels, epochs=5)
```

```
Epoch 1/5
4523/4523 [==============================] - 20s 4ms/step - loss: 0.0041 - mae: 0.0447
Epoch 2/5
4523/4523 [==============================] - 20s 4ms/step - loss: 0.0030 - mae: 0.0378
Epoch 3/5
4523/4523 [==============================] - 20s 4ms/step - loss: 0.0026 - mae: 0.0348
Epoch 4/5
4523/4523 [==============================] - 20s 4ms/step - loss: 0.0023 - mae: 0.0329
Epoch 5/5
4523/4523 [==============================] - 20s 4ms/step - loss: 0.0021 - mae: 0.0315
```

Out[ ]:
```
<keras.callbacks.History at 0x7fb0cf70b610>
```

Saving and loading the model.

Save the trained model.

In [ ]:
```
# Code to save the model. This should be the easiest way to do this.
```

Loading the model so it can be used.

In [ ]:
```
# Code to load the model that has been trained.
```

Testing the single Trained Model.

In [ ]:
```
prediction = model.predict(train_images[0].reshape(1,64,64,3))
```

In [ ]:
```
plt.plot(prediction[0], label="Model Prediction")
plt.plot(train_labels[0], label="Actual")
plt.legend()
plt.show()
```