# Software Development for High-Throughput Hydrogen Evolution Reaction

**Dennis Johan Loevlie**

Submitted in partial fulfillment of the requirements for the degree of

Master of Science Department of Chemical Engineering

Carnegie Mellon University

Pittsburgh, PA, USA

December 13, 2020

# Carnegie Mellon University

## CARNEGIE INSTITUTE OF TECHNOLOGY

## REPORT

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF Master of Science

**TITLE** Software Development for High-Throughput Hydrogen Evolution Reaction

**PRESENTED BY** Dennis Loevlie

**ACCEPTED BY THE DEPARTMENT OF**

Chemical Engineering

John Kitchin

JOHN KITCHIN, PROFESSOR AND ADVISOR

12/11/2020

DATE

Ann Robinson

ANNE S. ROBINSON, PROFESSOR AND DEPARTMENT HEAD

12/11/2020

DATE

# Contents

# Abstract

Utilizing clean energy sources in areas such as transportation and power generation is of paramount importance when looking toward the future. One energy source that the world could incorporate is hydrogen as it is a light, energy-dense, and easily storable compound. Curently the majority of the world's hydrogen is being produced in processes that generate carbon emmisions such as, Steam Methane Reforming (SMR). To truly refer to hydrogen as a clean fuel source we must look toward methods of generating it that do not negatively impact the environment in the process. Photocatalytic water splitting is one of those methods but is not being done in industry due to the high cost of the rare metal catalysts required. Dr. Stefan Bernard's group has developed a high-throughput experimental setup that is being used to screen large amounts of bimetallic combinations of catalysts in the hopes of finding cheaper, more plentiful alternatives. The work covered in this paper is the development of software tools to help accelerate the post processing of the data and give deeper insights to what goes on during the reactions. The software developed includes several image analysis tools (including the use of a custom built convolutional neural network classifier), data storage and extraction/location tools for the massive amounts of data being produced, and automated clustering software to group the bimetallic combinations based on the synergetic effects they exhibit.

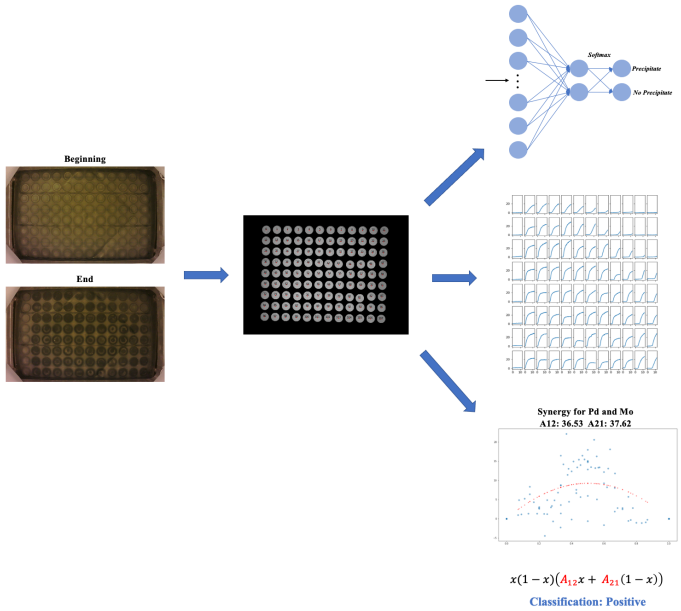


Figure 1: Graphical Abstract

## List of Figures



## List of Tables

# 1 Motivation

Moving toward clean, energy-efficient alternative fuels has become essential. Hydrogen is one ideal candidate for an energy carrier when it is generated in an environmentally friendly manner. Currently, the majority of the world's hydrogen is being produced via steam methane reforming (SMR).[1] On a high-level, SMR has an inlet stream of methane and steam that is fed to a catalytic reformer at high temperatures (700-900 ° C) and pressures (1.5-3 MPa) to form a hydrogen and carbon monoxide mixture. The reforming reaction is shown in equation 1. The outlet is then sent through a series of reactors where the Water Gas Shift (WGS) reaction occurs, as shown in equation 2. There is normally additional steam added before each WGS to push the reaction to produce more hydrogen.

$$CH_4 + H_2O \rightarrow CO + 3\,H_2 \tag{1}$$

$$CO + H_2O \rightarrow CO_2 + H_2 \tag{2}$$

Another hydrogen production process is coal gasification, where coal is partially oxidized with steam and oxygen at high temperatures and pressure to produce hydrogen, carbon monoxide, carbon dioxide, methane, and other compounds. In recent years gasification and coal power plants have become extremely unpopular for many reasons; natural gas provides a more efficient and eco-friendly alternative. Producing hydrogen via gasification or SMR requires an enormous amount of energy and a substantial amount of $CO_2$ is generated as a by-product. For hydrogen to be a clean fuel, we must look to alternative energy sources, such as wind, hydropower, and solar, to carry out a water-splitting reaction. Solar energy seems to be the best approach since

it has fewer region-related limitations than hydropower and wind energy.[1]

Hydrogen production from water via photocatalytic water splitting is an attractive alternative since the inlet material (water) is abundant and inexpensive. This method's efficiency is not yet practical for large scale hydrogen production. The Bernhard group has accomplished efficient water reduction systems using iridium-based photosensitizers with colloidal palladium or platinum catalysts.[2,3] Currently, work is being done to develop a solar fuel system utilizing more abundant and accessible components and to get a deeper understanding of why the catalysts are effective. There are minimal studies into bimetallic colloidal catalysts for visible-light-driven photocatalytic water reduction in solution-based systems. Therefore, a plethora of bimetallic catalysts have been investigated for photocatalytic water reduction in this work. The photocatalytic cycle involved in the photocatalytic water splitting experiments we have conducted is shown in Figure 2. This work revolves around software that has been developed to utilize the data retrieved from the experiments that are conducted to achieve a deeper understanding of the effect of the different bimetallic catalysts on the reaction.

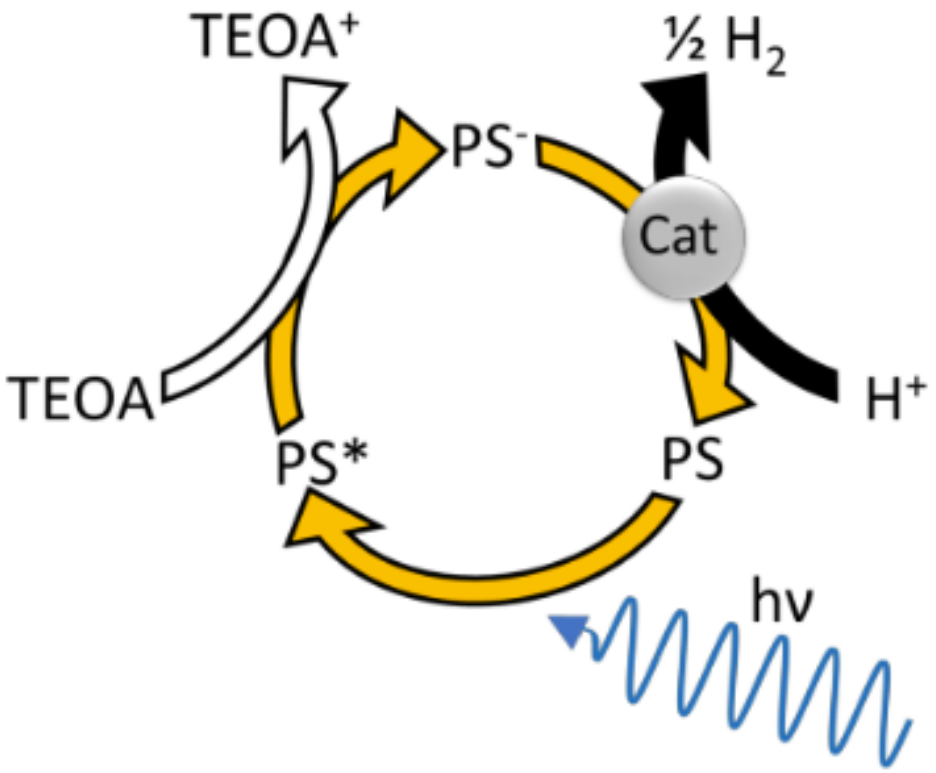


Figure 2: Photocatalytic cycle, involving [Ir(Fmppy)2dtbbpy]PF6 as a photosensitizer (PS), TEOA as the sacrificial electron donor, and the nanoparticulate in-situ formed bimetallic catalyst (Cat), which was varied by composition and component mixture.[4]

A high-throughput 96 well plate experimental setup has been used to drastically increase the screening rate of bimetallics for the hydrogen evolution reaction (HER).[4] This setup has reduced the waste and generated real-time data on the photocatalytic reactions taking place. To rapidly assess hydrogen production in each of the wells, a chemosensory tape is utilized to present a visual, quantifiable measure of the hydrogen content in the headspace above the solutions in the wells. The data is analyzed using a series of software tools that have been developed. To quantify the changes in the chemosensory tape's darkness, which indicates hydrogen presence, images

were taken of the plate every ten minutes during the reaction. A tool was developed to analyze these images and quantify the hydrogen production rate for each well in a plate during the reaction. A Python package, `espyranto`, was built to store and visualize that data and create automated reports for each of the bimetallics tested. The data and automated reports from each experiment are stored in a government supercomputer NERSC that provides easy access to the data for experimental/computational collaborators. As the files in the database grow, the time taken to locate data from a specific bimetallic combination becomes a nuisance. A second Python package, `nb_search`, was developed to efficiently sort through, locate and, open report files. An additional feature allows tags to be added to files that are urgent or have hard deadlines.

Specific groupings of the 31 bimetallics tested became obvious once visualizations of the synergy between the bimetallics were analyzed. Positive synergy was defined as the combination of two metals producing more hydrogen then the summation of the amount of hydrogen each of them produced separately. A quantitative method for clustering those synergy plots was created; this was validated with a qualitative grouping of the bimetallics. In the third generation of experiments being conducted, there were images taken of the bottom of the plate and the top. This allowed for more information to be pulled from each experiment, such as the change in the RGB color scale and whether or not precipitate had formed. To visualize this data more precisely an interactive image analysis tool was developed. This tool allowed the user to select one or more wells of interest, see the bottom of the well(s) with their color change, and the top of the well(s) with the change in the darkness (indicating hydrogen production). A slider was incorporated to change the time during the experiment the user wants to visualize. It was hypothesized that the amount of precipitate formed throughout an experiment could be

a relevant feature when training a model to predict the amount of hydrogen that would be produced with different bimetallic catalysts. Thus, an additional image analysis tool has been developed to accurately predict whether a well has produced precipitate and if so, the area (in pixels) of precipitate produced. This will be used in future computational research to create better models and to achieve a deeper understanding of the data.

# 2 Approach

The following sections will go over the methods used in the different research projects completed.

## 2.1 Image Analysis for Quantifying Hydrogen Production

The Hydrogen Evolution Reaction experiments take several hours, and images are taken of the plate every ~10 minutes throughout the experiment. The experimental setup of a plate consists of 96 wells where one of the metal catalysts varies in concentration along the rows and the other along the columns. The top left well is used as a control variable and no trace of either metal catalyst. This plate configuration is shown in Figure 3.

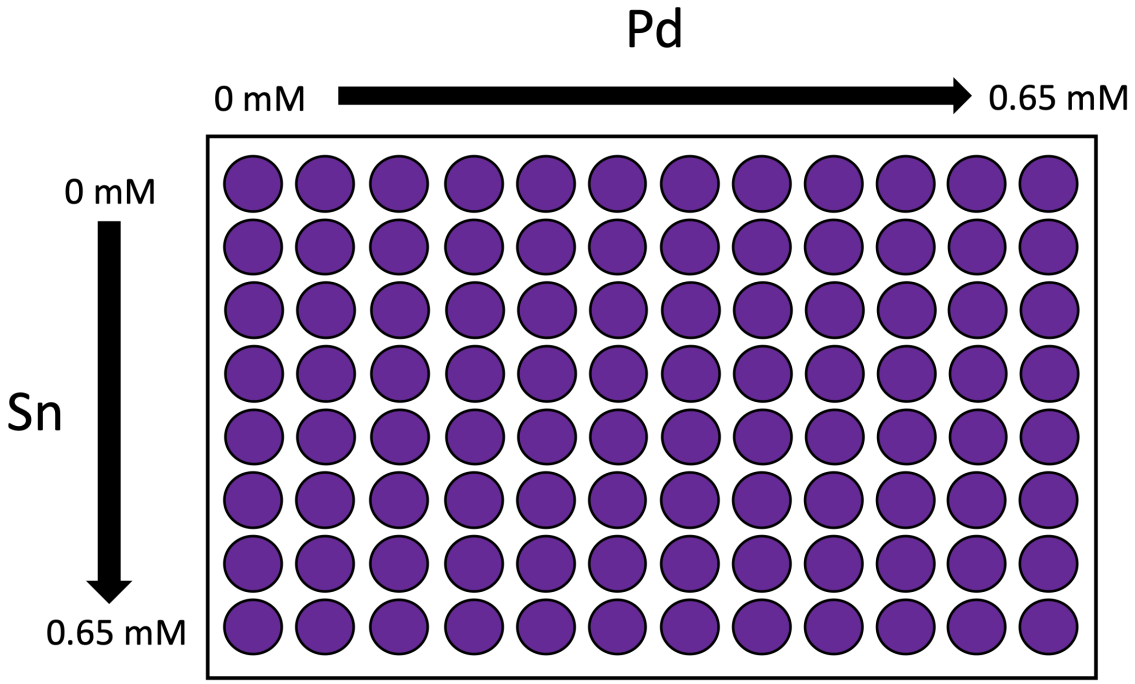


Figure 3: The experimental setup of a 96 well plate for testing bi-metallic catalysts.

The first and most crucial step in data analysis and post-processing is obtaining quantitative measurements of how much hydrogen each bimetallic

combination produced over time. As the reaction takes place, certain wells become darker in color, indicating the presence of hydrogen in the solution. This change in darkness is evident as seen in Figure 4 and Figure 5. Using Figure 4 as a baseline and calculating the change in darkness in each of the subsequent images data can be collected on the amount of hydrogen produced in each of the wells. This is far easier and more accurate when a binary mask is created.

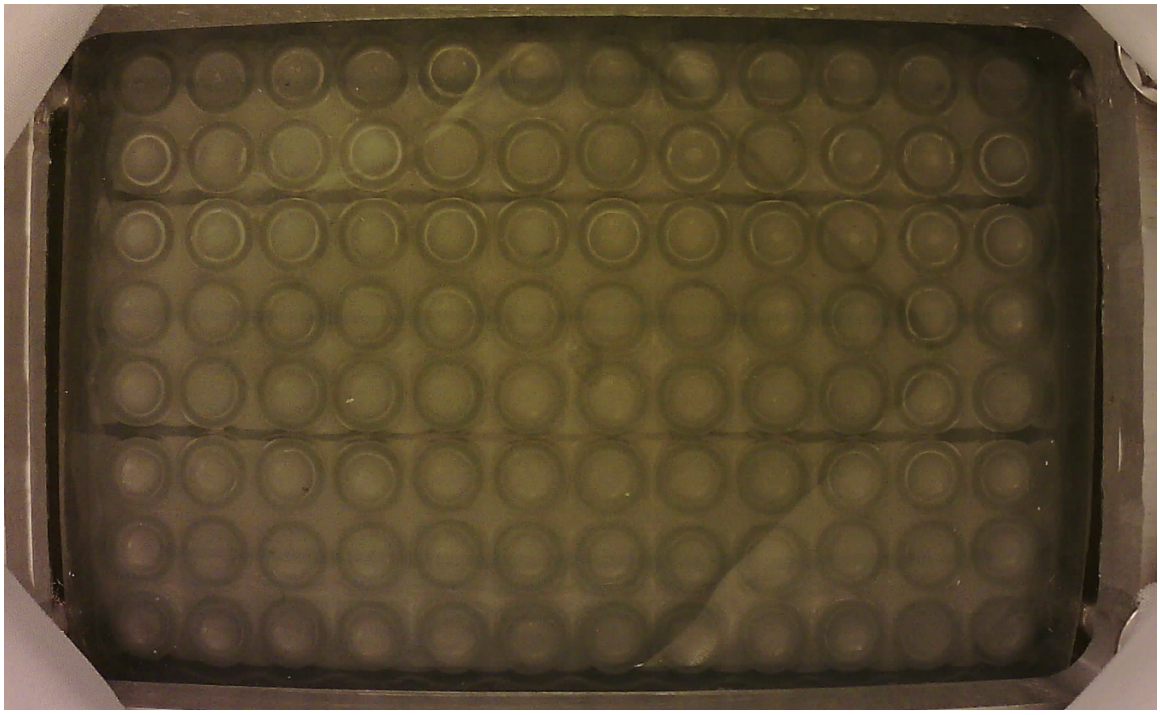

Figure 4: Plate image taken at the begining of the experiment.

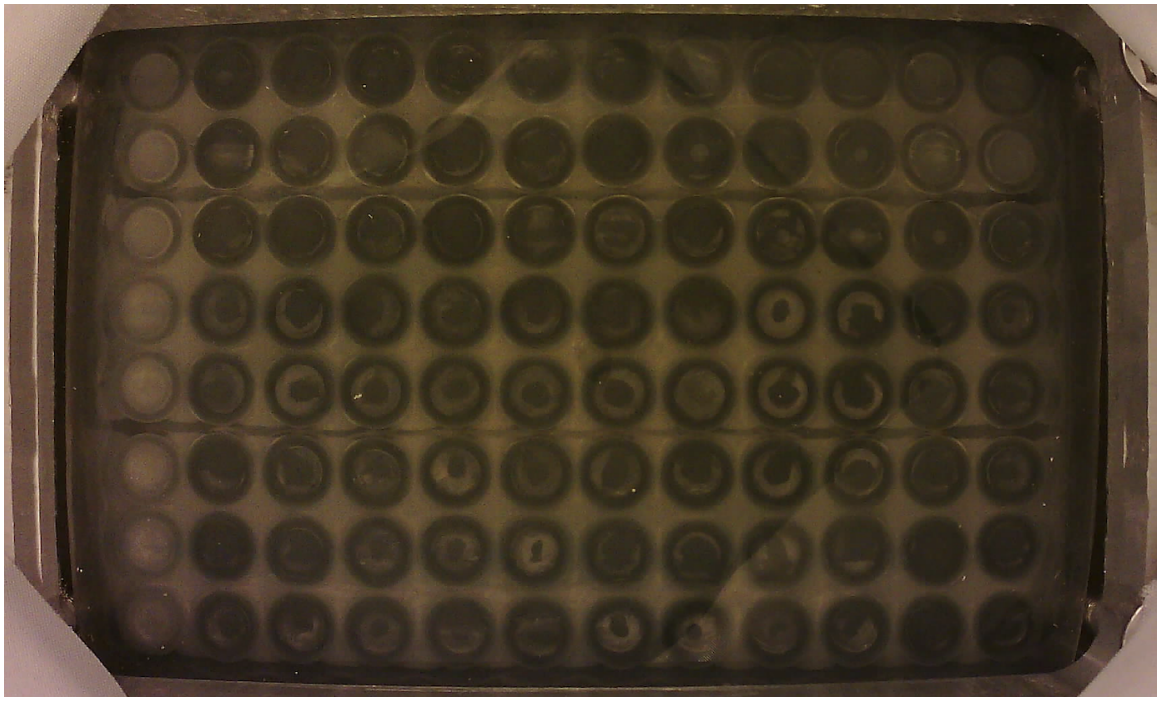

Figure 5: Plate image taken at the end of the experiment (1180 minutes).

To accurately and quickly perform analysis on each of the wells, the first step is to create a binary mask of the plate. The binary mask allows for easy image manipulation on the wells and ignores everything in the image that is not important. It takes the grayscale (pixel values from 0 black to 255 white) plate image and makes the value of every pixel other than the wells equal to zero. The method developed to create this binary mask was interactive, efficient, and accurate. There was a fine line between automation and accuracy, so that the least amount of interaction was required to generate the mask. The first step for generating the mask prompts the user to click on the centers of the four corner wells on the plate. This step was developed using the `matplotlib` function `ginput` but did not work inline in a Jupyter notebook, so the TkAgg backend of `matplotlib` had to be used. This backend created a separate pop-up window for the interactive portions of the code that the user could exit out of when they were finished. A cross-hair interactive cursor was incorporated to help with accuracy using the `matplotlib.widgets` library.

This step can be seen in Figure 6.

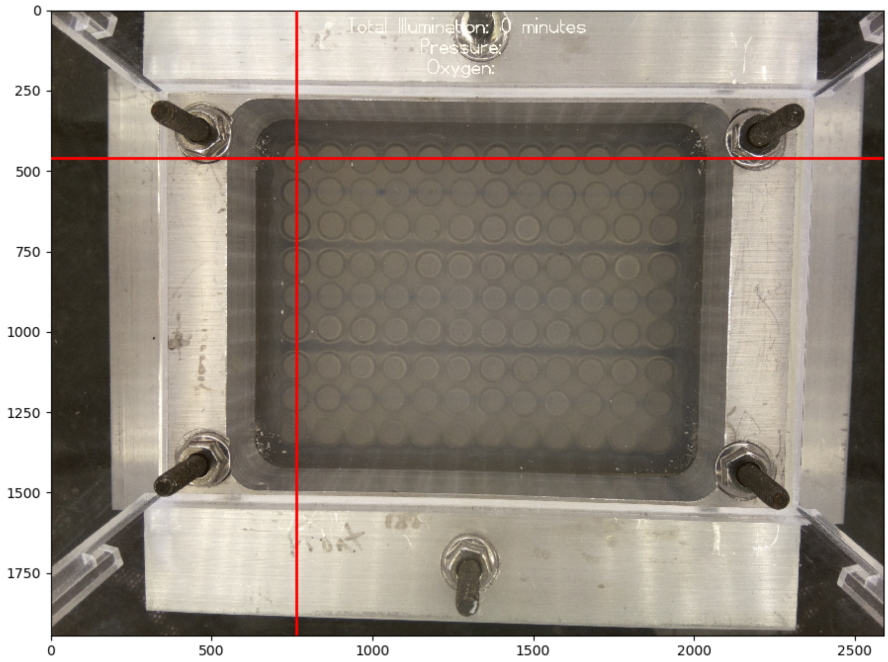


Figure 6: Step one in the interactive binary mask creation.

After step one is complete, the rest of the wells' centers would be generated by assuming there was linear spacing vertically and horizontally between the wells. The user would then be shown the original image with transparent red circles placed where the code guessed a well is located based on the previously generated center values. This step is also interactive; the user can click on any of the red circles and drag them to the correct location, as shown in Figure 7. This was accomplished using the `matplotlib` function `patches.Circle` that generates circle objects that can be manipulated through a custom built class.

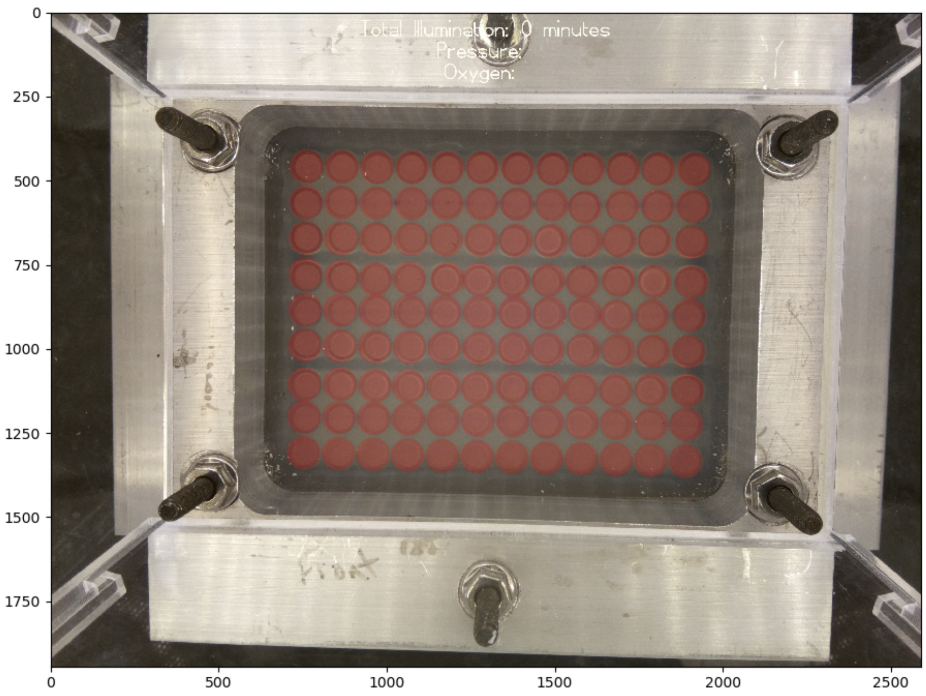


Figure 7: Step two in the interactive binary mask creation.

Next, the new circle centers are collected and saved on a CSV file for later use. Finally, the binary mask can now be easily generated by zeroing out all of the pixels outside the area located in the red circles from step 2. This is done by initializing a boolean array the size of the image with all true values, then using the function `skimage.draw.circle` with the x and y coordinates and the radius of each well to assign those pixels to be False. Finally, the original image can be sliced with the boolean array, and all true values in it are set to zero. The binary mask generated can be seen in Figure 8. At this point, image analysis results can be obtained.

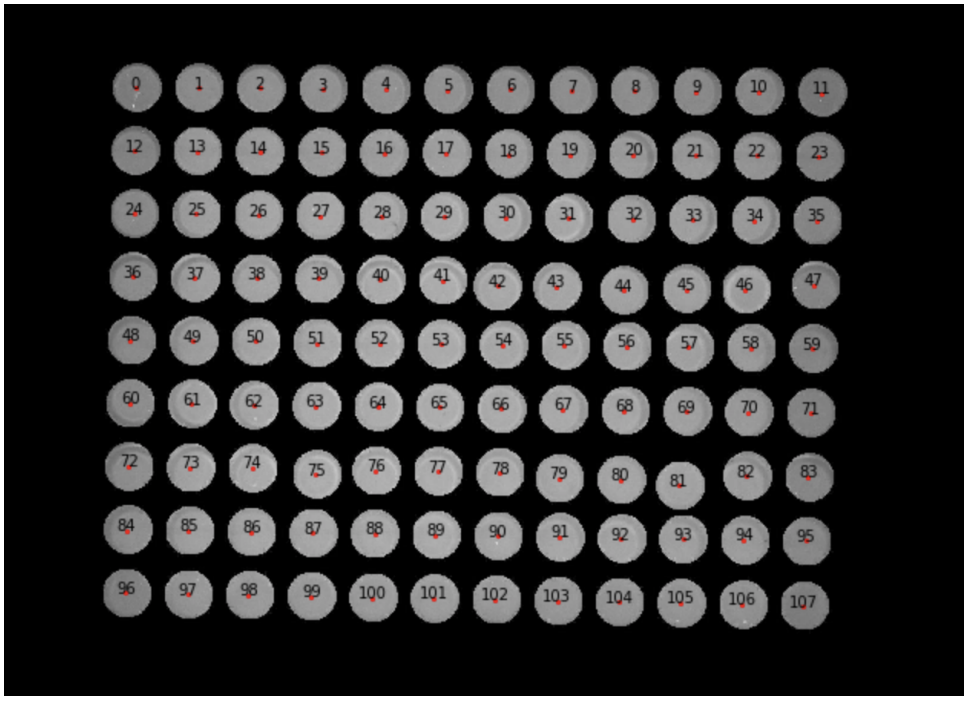


Figure 8: Binary mask generated using the data from the user in step one and two.

## 2.2 Generation Three Data Interactive Visualization Tool

In the third generation of experiments conducted, there were images taken of the top and the bottom of the plate. This provided a few new insights into the data that was not previously possible in generation one and two. One insight was the change in the RGB color scale of a well during the experiment. The other was whether or not the well-produced precipitate and how much was produced. Visualizing specific wells and what happens to them through time as a precipitate is formed is valuable information as it allows for conclusions to be made on what effect the precipitate had on the overall hydrogen production in the well. To do this more efficiently and smoothly, `matplotlib widgets` were used to make an interactive GUI that the user could use to choose any number of wells and visualize them at different time steps using a slider tool.

## 2.3 Generation Three Data Precipitate Quantification Model

Developing a machine learning model to predict the amount of hydrogen that a specific bimetallic combination would produce would be extremely useful to this project. It would not only help choose which metals should be used in future experiments, but also help the researchers get a deeper understanding of why certain bimetallic combinations are superior to others. The amount of precipitate formed in a well was hypothesized to be a effective feature in such a model so work has been done to automate the retrieval of that information.

Using the binary mask of the plate and `skimage` thresholding techniques it is easy to generate an image of each well separately. When choosing a well image at the final time step with precipitate at the bottom of the well, similar `skimage` thresholding can be used to separate the precipitate from the rest of the image. It is then a simple calculation to retrieve the area (in pixels) of precipitate that is in the well. In theory this is a great method of getting the feature data for a plate but when this method is tested on the entire plate to automate the process, the issue arises that wells without precipitate are being misclassified. To fix this issue a convolutional neural network (CNN) binary classifier was used to predict whether a well image had precipitate in it or not.

The CNN was made using the popular open source deep learning package `PyTorch`. The general architecture is shown below in Figure 9. Larger models were tested but due to the low number of images available for training it was decided that a smaller custom model would be more viable. Batch normalization and dropout layers were added to the model architecture to help prevent overfitting. When incorporating dropout regularization, during training some nodes are randomly ignored or "dropped out". More than likely the binary classification task we are implementing this network for does not

need every node to achieve high accuracy, so incorporating dropout layers can help prevent overfitting. A visualization of how dropout regularization works, is shown in Figure 10. In training the samples were fed into the loop with a batch size of 4. The loss function used was the `PyTorch` function `torch.nn.BCEWithLogitsLoss` which is simply a plain Sigmoid activation layer followed by a Binary Cross-Entropy loss function. When combining both of these into one layer, the log-sum-exp trick helps with the numerical stability. The Adam optimizer was used with a learning rate of $1 \times 10^{-4}$ and the PyTorch `ReduceLROnPlateau` learning rate scheduler was used to help prevent overfitting. The model was trained for 1000 epochs on CUDA GPU at Amazon Web Services (AWS) and the parameters were saved for future use.

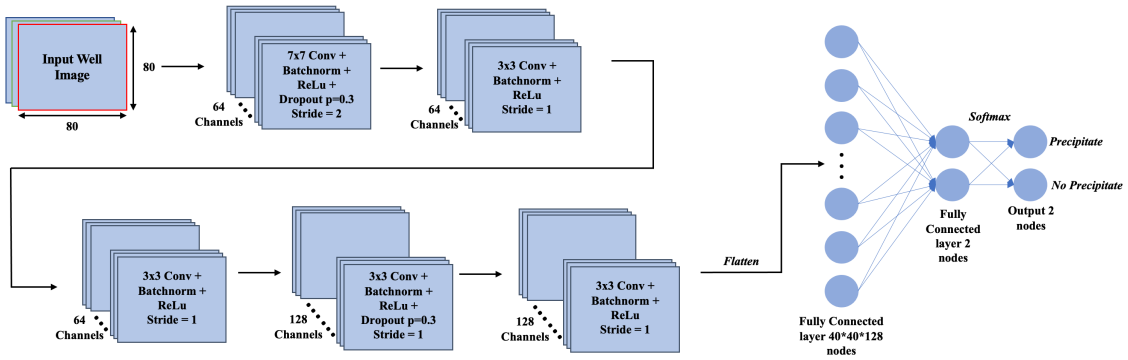


Figure 9: Architecture of the Convolutional Neural Network used to predict the presence of precipitate in a well.

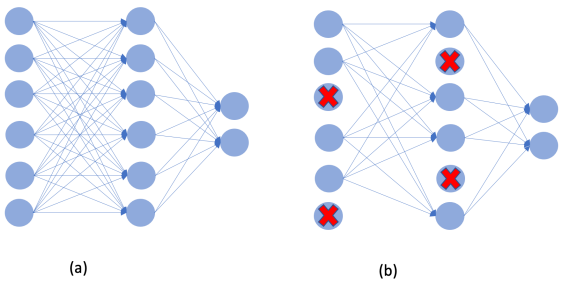


Figure 10: Visual description of dropout regularization. Figure (a) is of a standard Neural Network and Figure (b) is the network after applying dropout.

## 2.4 nb_search Package Development

Being able to efficiently share files and data between the computational and experimental research groups working on this project is extremely important. The group had kept the files on a goverment run supercomputer called NERSC. Each experiment plate had its own folder and a script was created to automatically create report notebooks for each of the bimetallic combinations. As the files in the database grew, the time taken to locate data from a specific bimetallic became a nuisance. A Python package, `nb_search`, was developed to efficiently sort through, locate and open the report files. This package used the package `nbformat` to read through the notebook files in the directory in a JSON format. There were several options for filtering the file search such as keywords in markdown cells, code cells, headings, custom tags, and meta-data that would have the two metal catalysts and the highest amount of hydrogen produced on the plate.

An additional feature allowed tags to be added to files that were urgent or had hard deadlines. This is similar to a much less powerful version of the TODO tags in Emacs orgmode. When a TODO tag is added to a notebook a optional description and due date can also be added. All notebooks with TODO tags can be displayed with the `nb_search` function `search_todo` with the link to the notebooks and the optional description along with the number of days until the due date.

## 2.5 Quantitative Synergy Plot Grouping

The Kitchin group has generated plots of the synergy between the bimetallic combinations for each plate. The definition of synergy used was how much more or less hydrogen the bimetallic combination produces than the summation of the hydrogen produced by each pure metal, at the same ratio. Five groupings of the 31 bimetallics tested became obvious once visualizations

of the synergy between the bimetallics were analyzed. These groupings are listed below:

1. Positive Synergy
2. Negative Synergy
3. Positive and Negative Synergy (with a trend)
4. No Synergy
5. Evenly Distributed Positive and Negative Synergy (no trend)

The plots were grouped qualitatively but a quantitative method for clustering the synergy plots was desired. An example of a synergy plot that was considered to exhibit positive synergy is shown in Figure 11. The plots trends exhibited a Margules type equation and the data was used to regress two parameters for each plot. The Margules equation is shown in Equation 3. Based on the ratio, magnitude, and sign of the fit parameters the plots were categorized into one of the five groupings mentioned above. The equation and an example of a positive synergy plot are shown below.

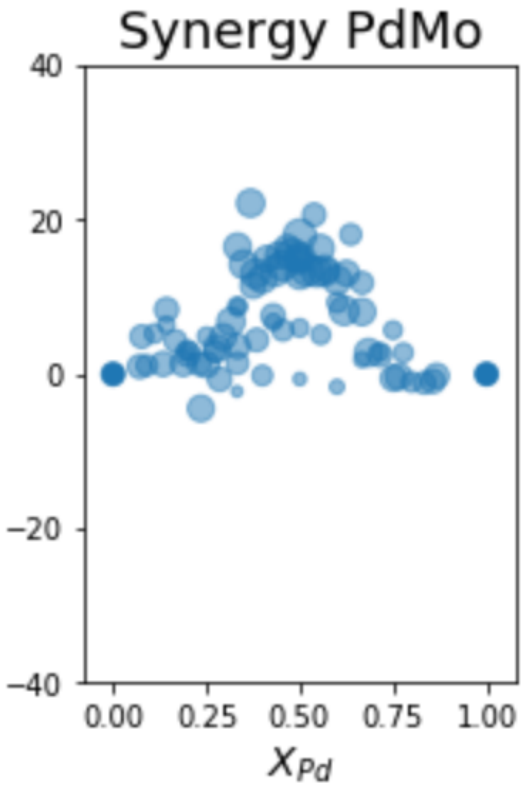


Figure 11: An example of a positive synergy plot with Pd and Mo.

$$f(x) = x(1-x)(A_{12}x + A_{21}(1-x)) \tag{3}$$

# 3 Results

## 3.1 Image Analysis for Quantifying Hydrogen Production

Once the binary mask is generated and the center values for each of the wells are stored, valuable data can be pulled from the plethora of images taken during an experiment. First we use the initial image as a baseline and find the difference in darkness of the wells in the subsequent images with the mask applied. The results for each well can be easily visualized in a grid plot with the package `espyranto` developed by the Kitchin group. An example of this plot is shown in Figure 12. The image analysis on generation three data shown in the following sections also utilize the binary mask that has been generated.

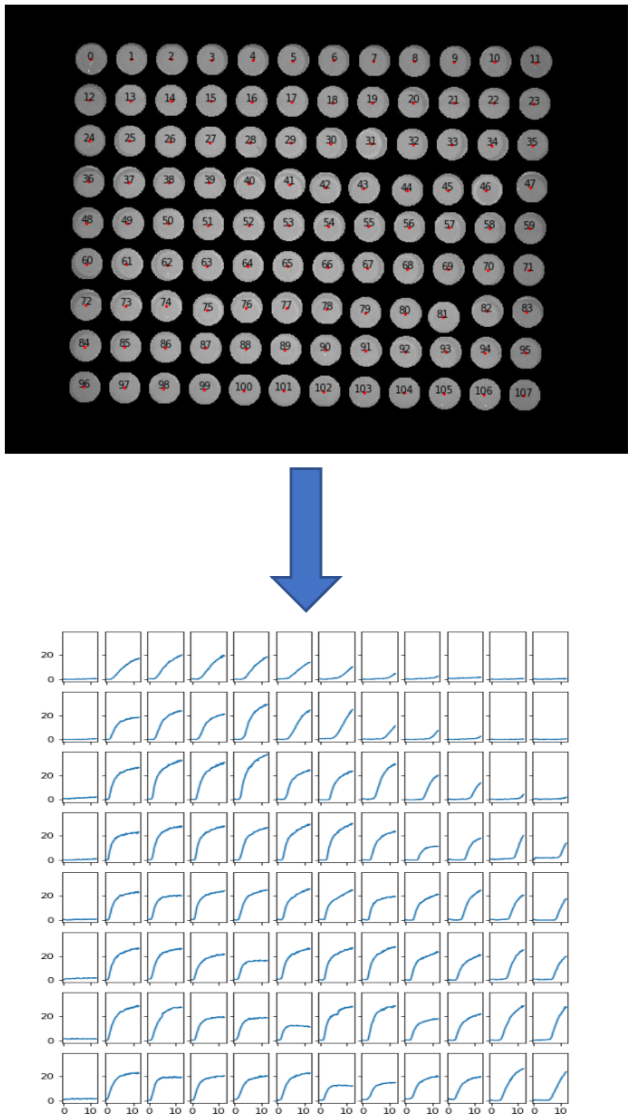

Figure 12: Grid plot of the amount of hydrogen in each well as a function of time.

## 3.2 Generation Three Data Interactive Visualization Tool

The generation three data interactive visualization tool was generated to allow any number of wells to be selected and visualized. An example of selecting two wells (54 and 55) is shown in Figure 13. The first column is the image of the top of the well. The second column is a plot of the change in darkness with time (using the images of the top of the well). The third

column shows the image of the bottom of the well. The fourth column shows a plot of the change in the RGB colors with time (using images of the bottom of the well). The rows are ordered based on the well numbers selected.

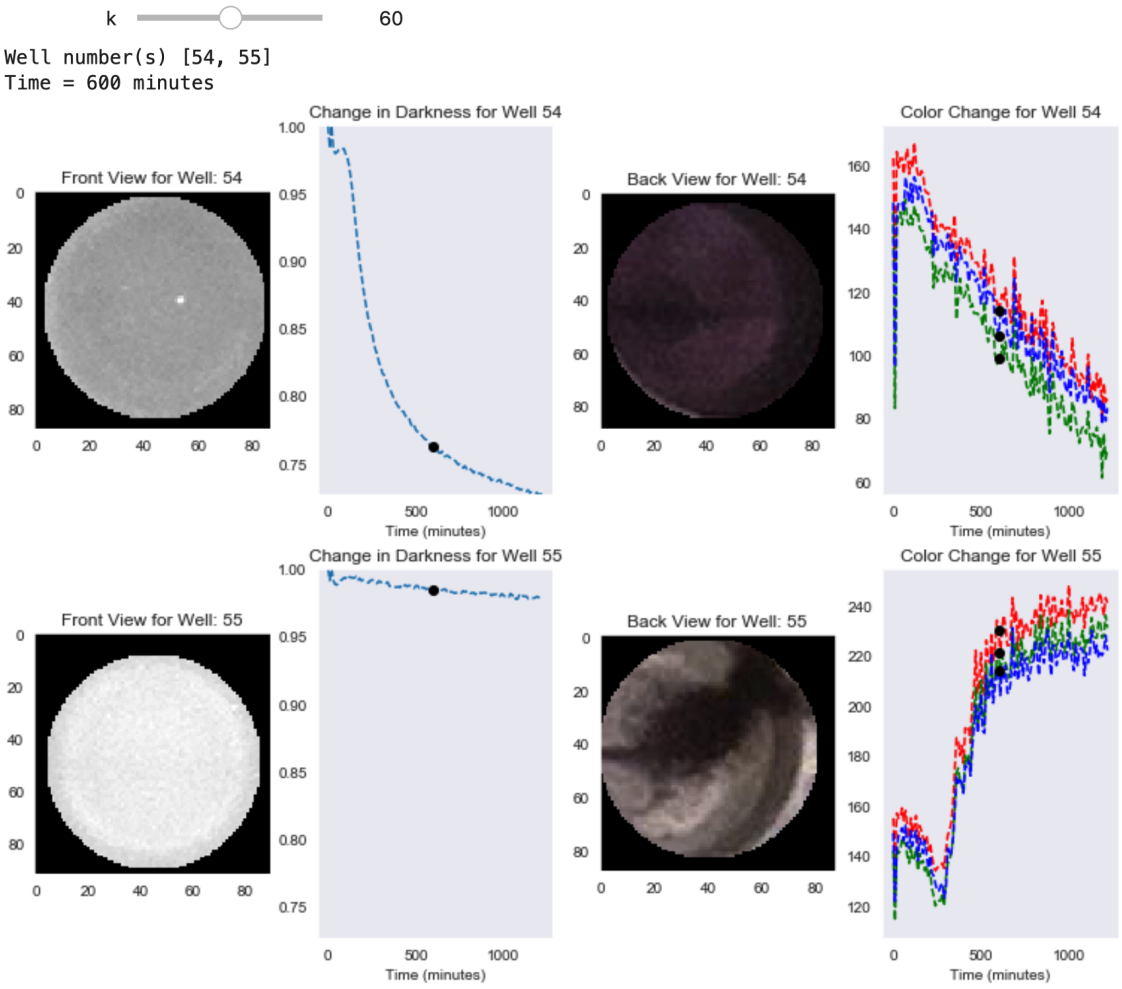


Figure 13: Interactive GUI for visualizing specific wells.

### 3.3 Generation Three Data Precipitate Quantification Model

The precipitate quantification model had two steps. The first was to use the convolutional neural network to classify whether the wells contain precipitate or not. The second step was to use `skimage` thresholding methods to locate the precipitate in the image and output the area of the precipitate in the image in pixels. The results can be seen in Figure 14 and an example of

thresholding to locate the area of precipitate for one of the wells is shown in Figure 15. A snippet of the code used for the thresholding example is located in the appendix in Figure 17.

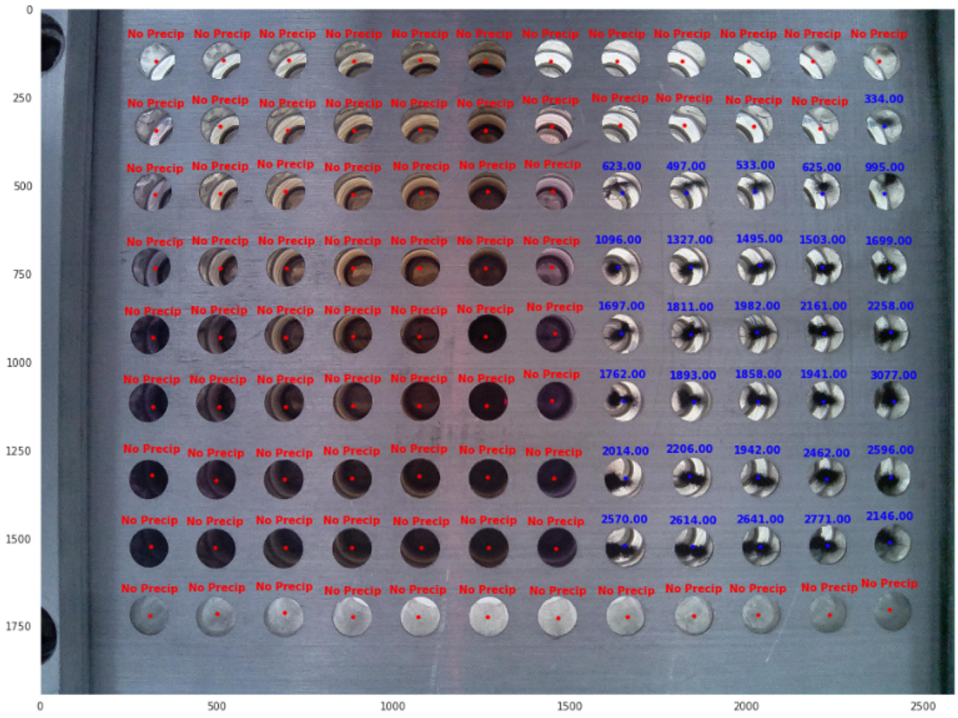


Figure 14: Results of the convolutional neural network classifier and `skimage` thresholding. The numbers in blue are the area of precipitate (in pixels) for the well.

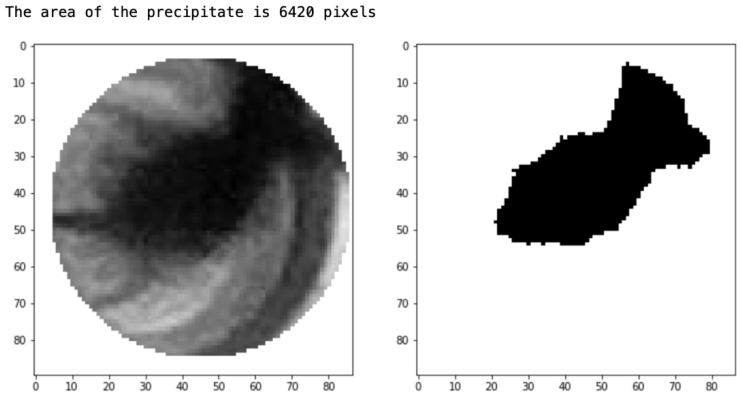


Figure 15: Image showing the original image of the well with the mask whitened out versus the image after thresholding is utilized to locate the precipitate in the well and calculate it's area.

## 3.4 nb_search Package Development

The package `nb_search` had many functions to refine the search through a given directory. The following sections will explain the uses of the package and the continuous integration that was used to ensure the package stayed functional. The uses are explained assuming it is being utilized in an iPython console.

### 3.4.1 Functions for Searching

There are six arguments to help search for, locate, and open Jupyter notebook files. The most basic search argument simply searches through the desired directory for all of the notebook files and displays them as clickable HTML links to the notebooks. The syntax for using that is `%run nb_search.py --all`. The second search argument allows the user to search through the notebooks in a given directory and return the notebooks that contain the specified string in one or more code cells. An example of using this to search for the variable "x" is `%run nb_search.py --code '.' x`. The third argument has the same functionality as the "code" argument but allows you to search in only markdown cells. An example of using this to search for the word "title" is `%run nb_search.py --markdown '.' title`. The fourth argument searches for a given string only in the headings of the notebooks. An example of using this to search for the string "title" in the headings is `%run nb_search.py --heading '.' title`.

The fifth and sixth functionalities of `nb_search` are powerful and specific to our teams data. The fifth argument lets the user search for experiment specific meta data in the notebooks. The three entries that it allows the user to search for are; Metal A, Metal B, and the Maximum Hydrogen produced in the plate. An example of using this to search for a notebook where one of the metals is "Mo" and the maximum hydrogen produced is greater

than 8 micromoles is `%run nb_search.py --property '.' Mo and Max_H < 8.0`. This function is powerful but does not allow for more advanced queries such as asking for a notebook with the Metal "Mo" or maximum hydrogen above 8 micromoles so that is why the final function was developed. This final function takes advantage of Python's built in parser and functions and requires the user to pass a function that returns True or False for the notebooks they want to search for. An example of using this function to find a plate that contains Au and has a max hydrogen production of over 31 micromoles (The output list is printed because this is not a Jupyter Notebook so the HTML link will not work) is shown below:

```
from nb_search.nb_search import fsearch

def f(NB):
    p1 = NB.property['Metal_A'] == 'Au'
    p2 = NB.property['Metal_B'] == 'Au'
    p3 = NB.property['Max_H'] > 31
    return (p1 or p2) and p3

files = fsearch(f,'.')
print(files[0])
```

```
'./nb_search/Test/Notebooks/Test_notebook2.ipynb'
```

### 3.4.2 Task Management Feature

Task management plays a huge part in a successful project, especially when there are hard deadlines. This was the motivation behind the "TODO" feature in the `nb_search` package. The user can add a TODO tag to a notebook with an optional description and due date (in brackets) that would be displayed above the notebook link. This can be added anywhere within a notebook and the sytax is `%TODO [YEAR-MONTH-DAY] Optional Description`. The TODO functionality is simple and only requires the user to input the desired directory to search under for TODO tags like shown below:

```
from nb_search.nb_search import search_todo
files = search_todo('.')
```

### 3.4.3 Travis Continuous Integration

As the package got larger and new features continued to be developed, it was important to ensure that the basic functionalities were still working smoothly. To do this, two programs were primarily used. The first was `pytest` a framework that allows tests to be written for an application. The second was Travis Continuous Integration which is a software that can be used to build and test packages hosted on GitHub. A directory of test Jupyter notebooks was created to sort through and the tests are shown in the table below. Every time a new commit was pushed to GitHub Travis CI would test the new code to make sure it still returned the correct number of notebooks from the test directory. The tests used and the correct number of notebooks for each test are shown below in Table 1.

Table 1: Correct output lengths for each test used.

| Test | Length of Notebook List Found |
|---|---|
| Search code cells for "espyranto" | 2 |
| Search markdown cells for "Loevlie" | 3 |
| Search headings for "Loevlie" | 2 |
| Search for "Au" in metadata | 2 |
| Search for "Au" and "Ga" in metadata | 1 |
| Search for "Au" and "max hydrogen $>$ 30" in metadata | 1 |
| Search TODO's | 3 |

## 3.5 Quantitative Synergy Plot Grouping

The Margules equation fit the trend of the synergy plot graphs well. Using the ratio, magnitude, and sign of the parameters that were fit to each plot, the bimetallic combinations were grouped into the five groups as described in section 2.5. An example of the fit on the synergy plot shown in Figure 11

is shown below in Figure 16. When comparing the qualitative groupings and the quantitative groupings of the synergy plots, the quantitative method was one-hundred percent accurate. The groupings of each bimetallic combination that has been tested by the Bernard group so far are shown in Table 2.

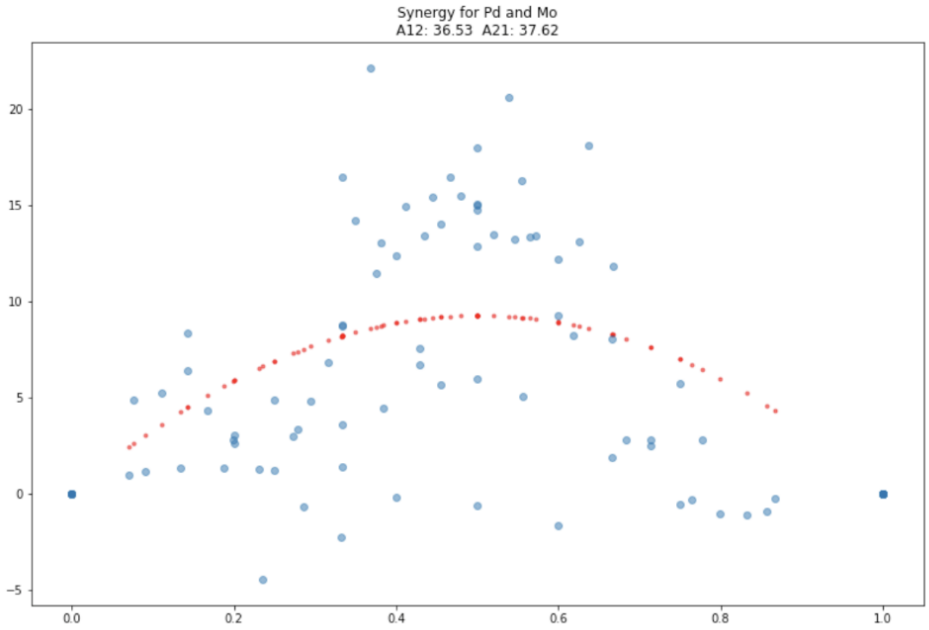


Figure 16: Synergy plot and fit for Pd and Mo. The fit equation is shown in red.

Table 2: Groupings of each bimetallic combination tested.

| Positive | Negative | Positive/Negative (with trend) | No Synergy | Positive/Negative (no trend) |
|---|---|---|---|---|
| Ru Cu | Ni Au | Mo Cu | Fe Mn | Rh Ni |
| Pd Mo | Zn Cu | In Pt | Bi Mo | Ni Cu |
| Os Ni | Pt Bi | Pt Sn | Bi Co | Pd Ru |
| Ni Mo | Ru Au | Pt Co | Ir Co | |
| Pd Sn | Ir Au | V Cu | Mo Sn | |
| | Au Sn | Pd Pb | | |
| | AI Au | Pt Fe | | |
| | Ga Au | | | |
| | Au Ni | | | |
| | Pt Au | | | |
| | Pt Zn | | | |
| | Pd Ni | | | |
| | Ni Sn | | | |
| | Au Cu | | | |
| | Ni Co | | | |
| | Au Pb | | | |
| | Ag Ni | | | |
| | Pt Pd | | | |
| | Au Ai | | | |

# 4 Conclusions

In conclusion, the software tools developed have helped improve a few areas of the post experimental computations and evaluation. Getting quantitative information about the experiment from the images taken has become more efficient. Visualizing the individual wells, how they change with time and how they may differ from one another has become easier. Obtaining valuable features from the images of the bottom of the plates is now an efficient automated process. Sorting through and locating the data has become simpler and organizing tasks for the group has become more organized. Finally, grouping different bimetallic combinations based on their synergy no longer requires any human input. Through my research at CMU my eyes have been opened to how much valuable information can be pulled out of data after an experiment has taken place. The valuable knowledge I have obtained on version control, code management, task automation, Image analysis, and machine learning will be invaluable in future research.

# References


[1] Chi-Hung Liao, Chao-Wei Huang, and Jeffrey C. S. Wu. Hydrogen production from semiconductor-based photocatalysis via water splitting. *Catalysts*, 2(4):490–516, 2012.

[2] Peter N. Curtin, Leonard L. Tinker, Christine M. Burgess, Eric D. Cline, and Stefan Bernhard. Structure-activity correlations among iridium(iii) photosensitizers in a robust water-reducing system. *Inorganic Chemistry*, 48(22):10498–10506, 2009.

[3] Husain N. Kagalwala, Danielle N. Chirdon, Isaac N. Mills, Nikita Budwal, and Stefan Bernhard. Light-driven hydrogen generation from microemulsions using metallosurfactant catalysts and oxalic acid. *Inorganic Chemistry*, 56(17):10162–10171, 2017.

[4] Eric M. Lopato, Emily A. Eikey, Zoe C. Simon, Seoin Back, Kevin Tran, Jacqueline Lewis, Jakub F. Kowalewski, Sadegh Yazdi, John R. Kitchin, Zachary W. Ulissi, Jill E. Millstone, and Stefan Bernhard. Parallelized screening of characterized and dft-modeled bimetallic colloidal cocatalysts for photocatalytic hydrogen evolution. *ACS Catalysis*, 10(7):4244–4252, 2020.

# 5 Appendices

```
from skimage.measure import label, regionprops
from skimage.morphology import opoening, disk

Gray_well[Gray_well==0] == 255
smask = opening(Gray_well,disk(5))
label_image, num = label(40<smask,return_num=True)
fig, ax = plt.subplots(nrows=1,ncols=2,figsize=(12,8))
ax[0].imshow(Gray_well,cmap='gray')
ax[1].imshow(label_image,cmap='gray')
region = regionprops(label_image)[0]
print(f'The area of precipitate is {region.area} pixels')
```

Figure 17: Code used for thresholding the image (after it is confirmed to contain precipitate) and finding the area of precipitate in pixels.

```
from nb_search import search_todo_util, search_notebook_util,
from nb_search import search_heading_util, heading_list, search_data_util, search_todo_util

def test_todo():
    file_list, tag_list = search_todo_util();
    assert len(file_list) == 3 and len(tag_list) == 3

def test_markdown_search():
    file_list = search_notebook_util('Loevlie','markdown')
    assert len(file_list) == 3

def test_code_search():
    file_list = search_notebook_util('espyranto','code')
    assert len(file_list) == 2

def test_heading_search():
    file_list = search_heading_util('Loevlie')
    assert len(file_list) == 2

def test_property1():
    file_list_Au = search_data_util(['Au'])
    file_list_Ga_Au = search_data_util(['Ga','Au'])
    file_list_Au_Hy = search_data_util(['Au','H_max < 30.0'])
    assert len(file_list_Au) == 2
    assert len(file_list_Ga_Au) == 1
    assert len(file_list_Au_Hy) == 1

def test_property2():
    file_list_Ga_Au = search_data_util(['Ga','Au'])
    assert len(file_list_Ga_Au) == 1

def test_property3():
    file_list_Au_Hy = search_data_util(['Au','H_max < 30.0'])
    assert len(file_list_Au_Hy) == 1
```

Figure 18: Code used by pytest and Travis CI to maintain the nb_search package on GitHub

Table 3: Table listing the architecture information of the CNN used to predict the presence of precipitate in wells.

| Unit Type | Output Plane Size |
|---|---|
| Convolution $7 \times 7$ | 64 |
| Batch Normalization | 64 |
| ReLu | 64 |
| Dropout(p=0.3) | 64 |
| Convolution $3 \times 3$ | 64 |
| Batch Normalization | 64 |
| ReLu | 64 |
| Convolution $3 \times 3$ | 64 |
| Batch Normalization | 64 |
| ReLu | 64 |
| Convolution $3 \times 3$ | 128 |
| Batch Normalization | 128 |
| ReLu | 128 |
| Dropout(p=0.3) | 128 |
| Convolution $3 \times 3$ | 128 |
| Batch Normalization | 128 |
| ReLu | 128 |
| Linear | 1 |