Automated Electron Diffraction Analysis

Krystal R. York

ECE 6970

Spring 2021

1 Introduction

1.1 Reflection High-Energy Electron Diffraction

Reflection high-energy electron diffraction (RHEED) is an experimental technique used to observe the growth of material in-situ. This powerful analytical tool is used to examine the structure of the crystal surface at atomic levels. RHEED images are captured during growth to gather information about the material such as the lattice parameters, growth rate, crystal quality, and whether it is rough or smooth. Crystal quality is typically described as polycrystalline or single crystalline. A polycrystalline material has imperfections creating separate crystalline domains within the material with no preferred orientations. Polycrystalline films will display arc shapes in their RHEED images known as Debye-Scherrer rings. On the other hand, single crystalline films will display thin, straight streaks in their RHEED images. The desire is that these streaks will remain during the entire growth, indicating that the material is a single crystal [1–4]. RHEED images of a typical single crystal pattern from a (111) yttria-stabilized zirconia (YSZ) substrate and a typical polycrystalline growth are shown in Fig. 1.1 and Fig. 1.2, respectively.

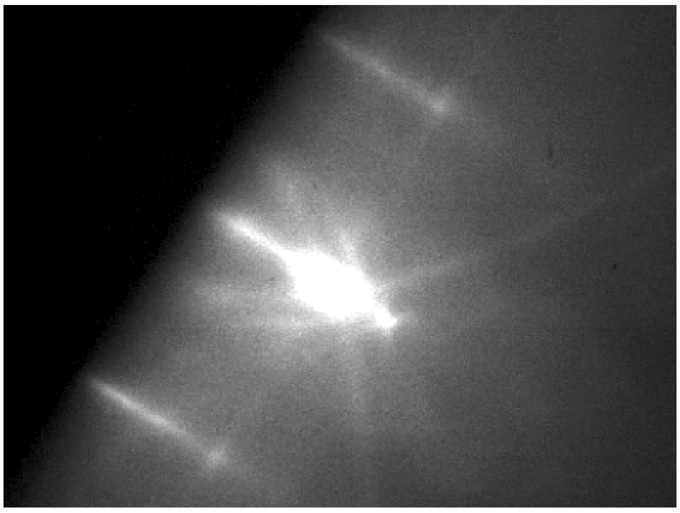


Figure 1.1: *Example RHEED image for (111) yttria-stabilized zirconia (YSZ) substrate.*

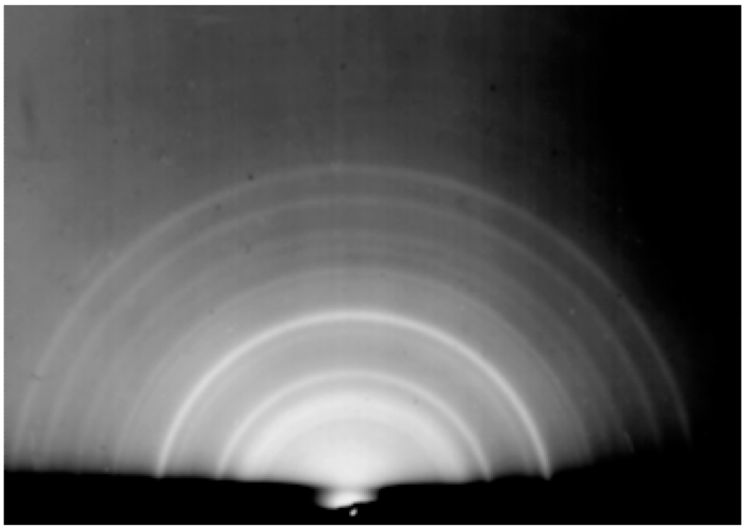


Figure 1.2: *Example RHEED image of a typical polycrystalline material.*

A typical RHEED system, as shown in Fig. 1.3, involves a focused electron source, phosphor screen, and a clean surface to hold the sample. This system is commonly paired with MBE due to ultra-high vacuum environment. The system used in the WMU MBE lab is a 20 kV Staib instruments electron source. The electron source aims an incident beam of electrons at energies that are accelerated to between 10 kV and 20 kV and directed at a low angle θ to the surface of the sample. This glancing angle is varied by both electrostatic deflection and sample motion. Because the incident beam only has a small penetration depth, the RHEED system is able to very precisely observe the first few layers of the sample. The diffracted beams, incident on a phosphor screen, are recorded on a charge-coupled device (CCD) camera. A kSA capture and analysis package (which includes the CCD camera) is employed to analyze the RHEED images [1–3].

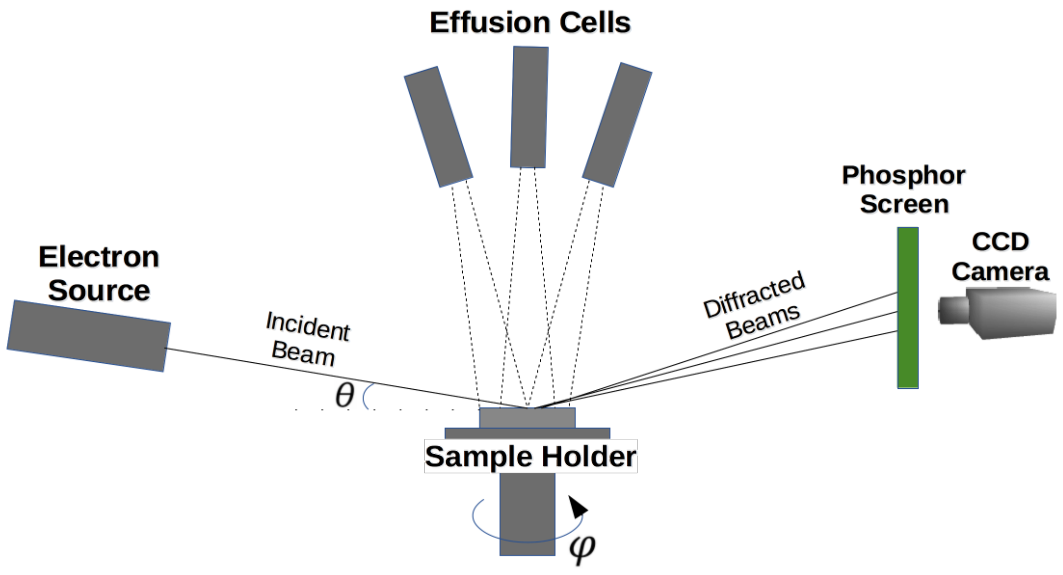


Figure 1.3: *A schematic of the experimental set-up of the RHEED system. The electron gun focuses an electron beam at the sample surface at a small angle of just a few degrees. This beam will scatter in a distinct diffraction pattern at a phosphorescent screen where the pattern is recorded by a CCD camera.*

Without in-situ techniques such as RHEED, thin film growth would have to be analyzed after the growth process with techniques such as x-ray diffraction (XRD), scanning electron microscopy (SEM), Raman spectroscopy, and scanning tunneling microscopy (STM). By comparing RHEED images to XRD, SEM, and Raman spectroscopy results, it has been shown that RHEED can be used directly to determine the degree of cation ordering in a film [5,6]. Consequently, we can use RHEED during crystal growth to study the effect of processing parameters on not only crystal quality and lattice parameters, but cation ordering as well.

1.2 Laboratory Automation and Machine Learning

Implementing automation into a research laboratory can have several benefits such as improved efficiency and productivity that can save the researchers time and money, increased accuracy and repeatability that improves the quality of the data collected, and assistance in data management and traceability especially when dealing with multiple samples [7]. There is also the secondary benefit that comes from the educational experience of applying such automation techniques that is especially important in a university setting. With low-cost development boards or computers such as the Arduino Uno or Raspberry Pi, automation can be easily introduced into an undergraduate laboratory or picked up by a graduate student to improve a process in their research laboratory [8–14]. With a wide range of professions moving towards automation in workplaces, it is increasingly important to begin moving towards automation in research laboratories and other academic settings.

Artificial intelligence is where machines are programmed to display human-like understanding of concepts as well as decisions making and problem solving skills. Machine learning is a subset of artificial intelligence where a large amount of data is provided to a machine to analyze and make a prediction. Both automation and machine learning can simplify and expedite systems, however, automation will repeat a task the same way whereas machine learning is a dynamic, predictive system that can have variable inputs and variable outputs. Nevertheless, machine learning can be used in automated systems [15–18]

2 Objectives

This project aims to automate the acquisition and analysis of reflection high-energy electron diffraction (RHEED) data in a molecular beam epitaxy (MBE) setting:

1. While growing a material, there are times when the substrate holder needs to be rotated especially when observing different RHEED patterns. Doing this requires one to turn a knob on top of the MBE machine. In this project, in conjunction with a suitable stepper motor, I will automate this to be able to turn the sample to different angles or setpoints while taking RHEED images. The angle would then be automatically recorded with the RHEED image that was taken. In Phase I, a manual control will be used so that the rotation can be rocked to achieve an optimal pattern.

2. In Phase II, in conjunction possible with machine learning, a go to next function will be added to enable the system to automatically rotate to fix on the next direction of symmetry, and optimize the angle for balanced image intensity across the pattern.

3. If time permits, a Phase III will be implemented where the data can be pipelined to existing analysis software for streamlined analysis of diffraction intensity that can be used to determine the Bragg-Williams order parameter or the lattice constants.

3 Methodology

3.1 Hardware

A Nema 17 bipolar stepper motor (Fig. 3.1) was bought to be able to rotate the substrate holder. The 100:1 gearbox will be able to finely tune the angle every 0.018°. Sparkfun's Big Easy Driver (Fig. 3.2) was bought to drive the stepper motor. This driver is able to drive the stepper up to a max 2A/phase, which is above the 1.68A needed for this motor. The Sparkfun's Artemis Development Kit (Fig. 3.3) is the microcontroller the code will be uploaded to. This board can be programmed and used like other Arduino boards. Additionally, this board can be used in machine learning projects using its on-board microphone or accelerometer as well as the Himax CMOS imaging camera (Fig. 3.4).

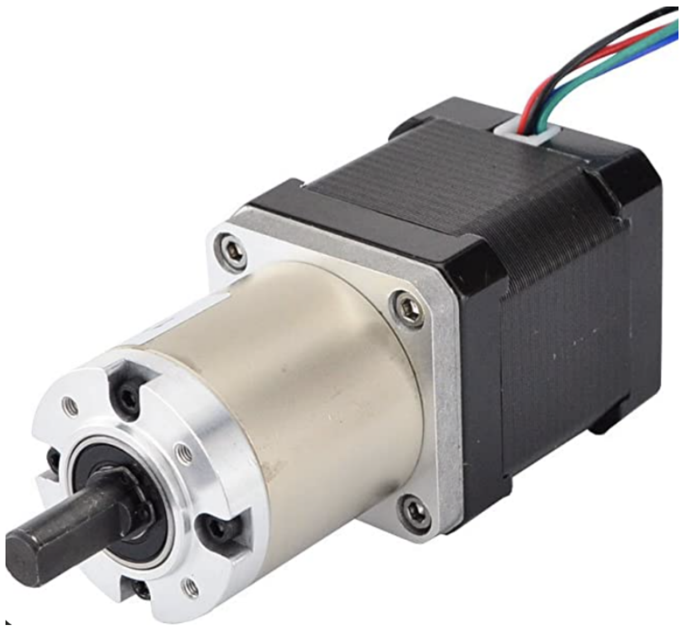


Figure 3.1: *Nema 17 bipolar stepper motor with gearbox 100:1.*

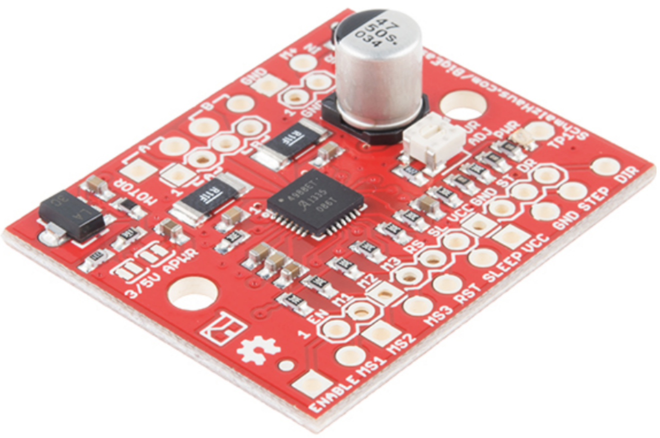


Figure 3.2: *Sparkfun's Big Easy Driver used to drive the stepper motor.*

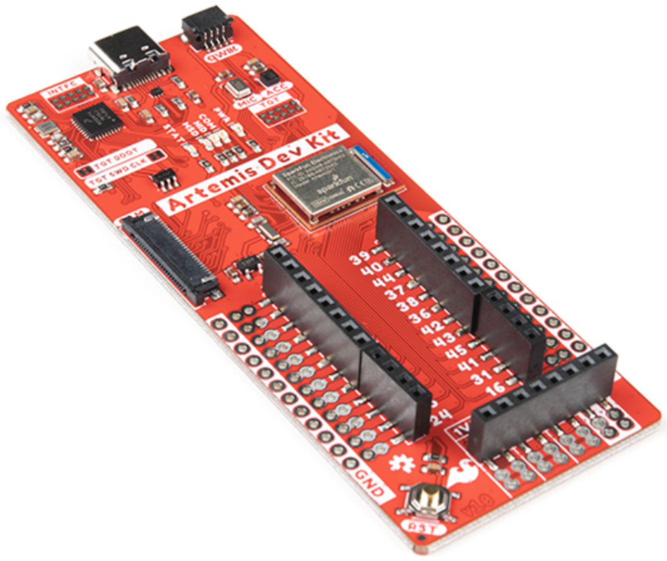


Figure 3.3: *Sparkfun's Artemis Development Kit.*

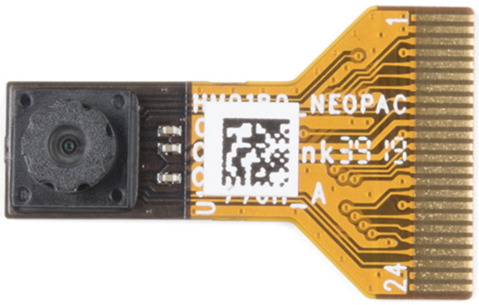


Figure 3.4: *Himax CMOS Imaging Camera.*

3.2 Software

The software that can be used to program Sparkfun's Artemis Development Kit can be either the Arduino IDE, the Arm® Mbed™ OS (Studio and CLI), and AmbiqSDK. For this project, the Arduino IDE was used. The first code (see section 2.1 in the Appendix) was written to fill the first objective. This code will allow you to use the serial monitor to enter the angle. The stepper motor then turns the sample to the specified angle to take the RHEED image.

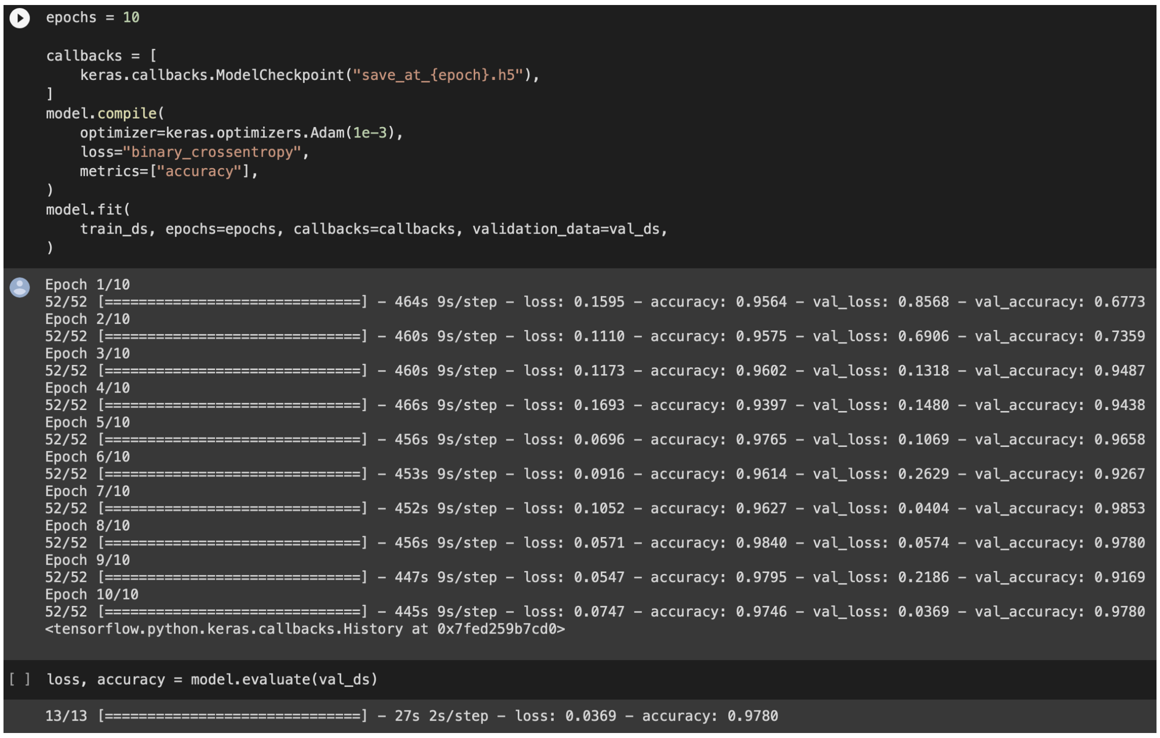
While waiting for the hardware to arrive, Google's Colab was used to train the machine learning model. Google's Colab is free to use and allows people to execute Python code to develop machine learning models using TensorFlow. It is especially useful because it allows free Cloud service with free GPU. Colab makes it relatively easy to learn about machine learning with several examples that are easily executed through a browser. Executing code is done by simply clicking on the play button that can be seen next to the code. Additionally, the code can be saved via GitHub or Google Drive, and the models and other item that are formed when you execute the code (such as images and graphs) can be downloaded to your own computer.

To begin developing the model needed for taking RHEED images, a dataset of about 1000 single crystal and 1000 polycrystalline images was made. While training the model, the software will be able to differentiate between the two type of RHEED images by simply putting these images in two different sub-folders. TensorFlow Lite Model Maker (see section 2.2 in the Appendix) was first used to develop a model since it easily enables machine learning on mobile devices and microcontrollers. Since this model will be used on an Arduino device, it is the most obvious choice.

After exploring more of TensorFlow and machine learning, it was determined that creating a model using Keras (see section 2.3 in the Appendix) will be useful as well. Keras is an API also written in Python and uses the Tensorflow. Keras allows developers to access a wider range of capabilities when creating a model such as image augmentation and customizing the training layers. Additionally, it can be exported to a Lite model to be then easily used on microcontrollers (see section 2.4 in the Appendix). The Lite model can then be used to convert it to a header file that can simply be used in an Arduino IDE code. By using a # include ”model.h” at the top of your code, the already trained model can be accessed throughout the program.

4 Results and Discussion

The both the Lite and Keras models were trained by splitting the polycrystalline and single crystalline images into a training group and a validation group. Usually, about 80-90% of the images will be put in the training group and 10-20% of the images will be put in the validation group. This is so the model can be trained and afterwards use the validation images to double check and improve the model further. Then the model will be trained. After executing each piece of code, print outs of what is happening are sometimes displayed below the code, as seen in Fig. 4.1. This is the code that was training the Keras model. After each epoch, it prints out the accuracy of that step and then moves on to the next epoch, typically becoming more accurate each time.



```
epochs = 10

callbacks = [
    keras.callbacks.ModelCheckpoint("save_at_{epoch}.h5"),
]
model.compile(
    optimizer=keras.optimizers.Adam(1e-3),
    loss="binary_crossentropy",
    metrics=["accuracy"],
)
model.fit(
    train_ds, epochs=epochs, callbacks=callbacks, validation_data=val_ds,
)
```

```
Epoch 1/10
52/52 [==============================] - 464s 9s/step - loss: 0.1595 - accuracy: 0.9564 - val_loss: 0.8568 - val_accuracy: 0.6773
Epoch 2/10
52/52 [==============================] - 460s 9s/step - loss: 0.1110 - accuracy: 0.9575 - val_loss: 0.6906 - val_accuracy: 0.7359
Epoch 3/10
52/52 [==============================] - 460s 9s/step - loss: 0.1173 - accuracy: 0.9602 - val_loss: 0.1318 - val_accuracy: 0.9487
Epoch 4/10
52/52 [==============================] - 466s 9s/step - loss: 0.1693 - accuracy: 0.9397 - val_loss: 0.1480 - val_accuracy: 0.9438
Epoch 5/10
52/52 [==============================] - 456s 9s/step - loss: 0.0696 - accuracy: 0.9765 - val_loss: 0.1069 - val_accuracy: 0.9658
Epoch 6/10
52/52 [==============================] - 453s 9s/step - loss: 0.0916 - accuracy: 0.9614 - val_loss: 0.2629 - val_accuracy: 0.9267
Epoch 7/10
52/52 [==============================] - 452s 9s/step - loss: 0.1052 - accuracy: 0.9627 - val_loss: 0.0404 - val_accuracy: 0.9853
Epoch 8/10
52/52 [==============================] - 456s 9s/step - loss: 0.0571 - accuracy: 0.9840 - val_loss: 0.0574 - val_accuracy: 0.9780
Epoch 9/10
52/52 [==============================] - 447s 9s/step - loss: 0.0547 - accuracy: 0.9795 - val_loss: 0.2186 - val_accuracy: 0.9169
Epoch 10/10
52/52 [==============================] - 445s 9s/step - loss: 0.0747 - accuracy: 0.9746 - val_loss: 0.0369 - val_accuracy: 0.9780
<tensorflow.python.keras.callbacks.History at 0x7fed259b7cd0>
```

```
[ ] loss, accuracy = model.evaluate(val_ds)
```

```
13/13 [==============================] - 27s 2s/step - loss: 0.0369 - accuracy: 0.9780
```

Figure 4.1: *The model being trained in Google's Colab. The model will repeatably run the training steps depending on the number of epochs you assign. The accuracy and validation accuracy typically improves after each round of training.*

Afterwards, a figure was created in order to better visualize how accurate the training was with 100 of the validation images, as seen in Fig. 4.2. Since the accuracy of the model was about 99%, it makes sense that one image was labeled incorrectly.

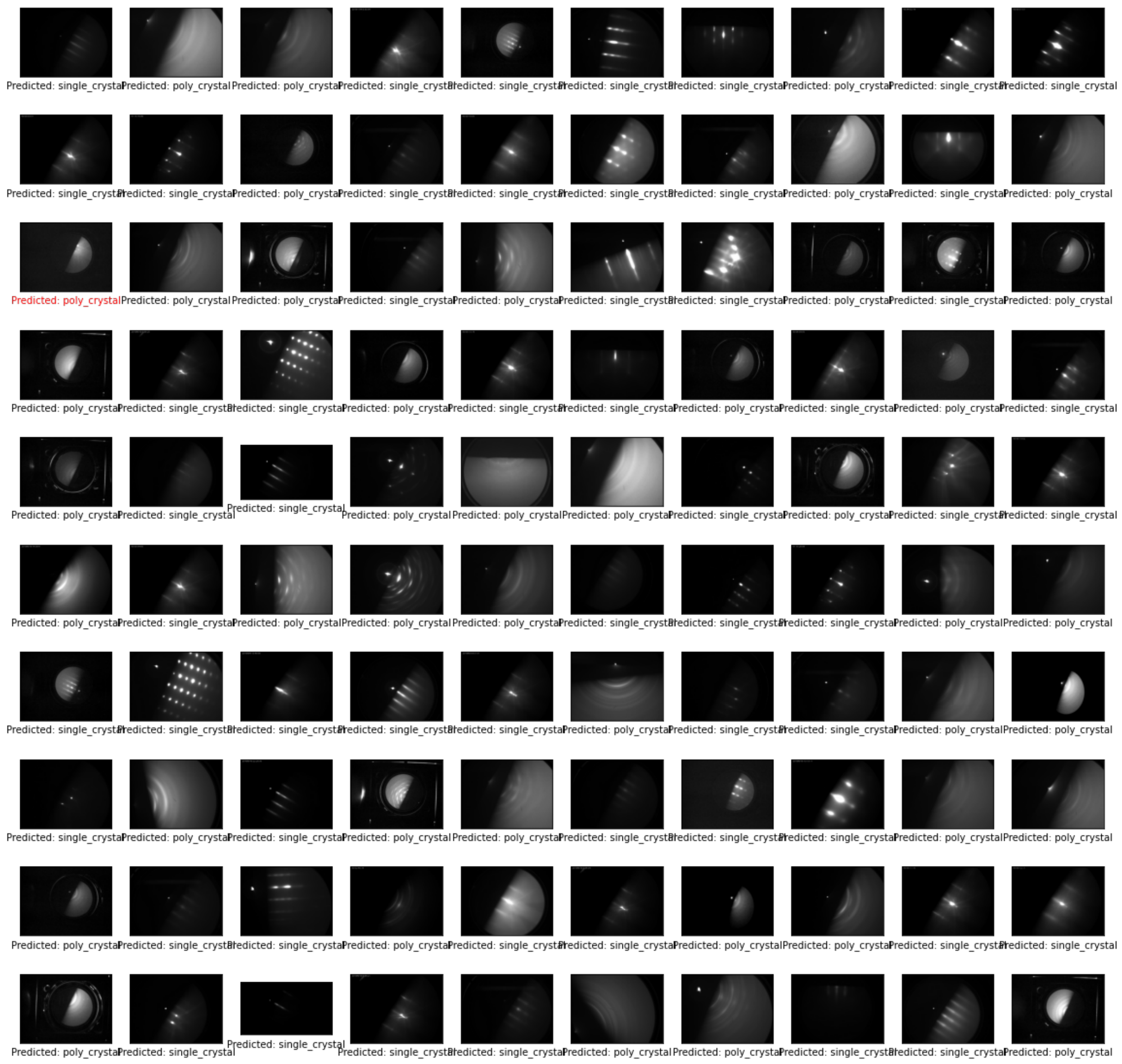
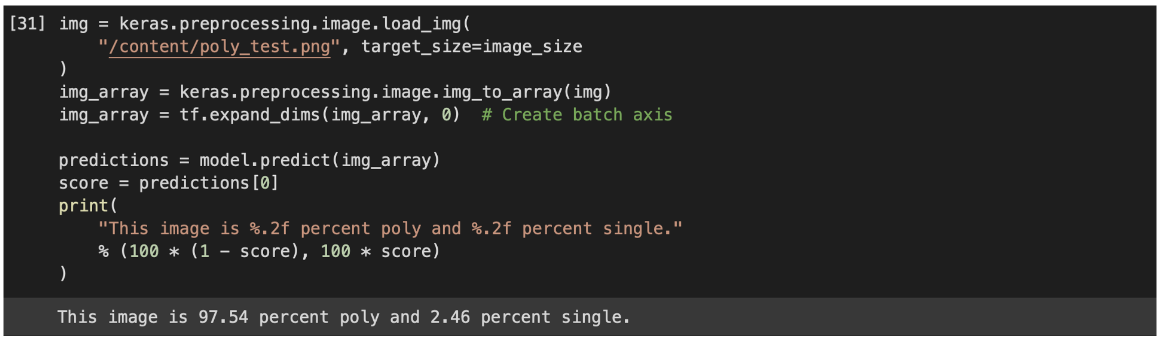

Predicted: single_crystal
Predicted: poly_crystal
Predicted: poly_crystal
Predicted: single_crystal
Predicted: single_crystal
Predicted: single_crystal
Predicted: single_crystal
Predicted: poly_crystal
Predicted: single_crystal
Predicted: single_crystal
Predicted: single_crystal
Predicted: single_crystal
Predicted: poly_crystal
Predicted: single_crystal
Predicted: single_crystal
Predicted: single_crystal
Predicted: single_crystal
Predicted: poly_crystal
Predicted: single_crystal
Predicted: poly_crystal
Predicted: poly_crystal
Predicted: poly_crystal
Predicted: poly_crystal
Predicted: single_crystal
Predicted: poly_crystal
Predicted: single_crystal
Predicted: single_crystal
Predicted: poly_crystal
Predicted: single_crystal
Predicted: poly_crystal
Predicted: poly_crystal
Predicted: single_crystal
Predicted: single_crystal
Predicted: poly_crystal
Predicted: single_crystal
Predicted: single_crystal
Predicted: poly_crystal
Predicted: single_crystal
Predicted: poly_crystal
Predicted: single_crystal
Predicted: poly_crystal
Predicted: single_crystal
Predicted: single_crystal
Predicted: poly_crystal
Predicted: poly_crystal
Predicted: poly_crystal
Predicted: single_crystal
Predicted: poly_crystal
Predicted: single_crystal
Predicted: single_crystal
Predicted: poly_crystal
Predicted: single_crystal
Predicted: poly_crystal
Predicted: poly_crystal
Predicted: poly_crystal
Predicted: single_crystal
Predicted: single_crystal
Predicted: single_crystal
Predicted: poly_crystal
Predicted: poly_crystal
Predicted: single_crystal
Predicted: single_crystal
Predicted: single_crystal
Predicted: single_crystal
Predicted: single_crystal
Predicted: poly_crystal
Predicted: single_crystal
Predicted: single_crystal
Predicted: poly_crystal
Predicted: poly_crystal
Predicted: single_crystal
Predicted: poly_crystal
Predicted: single_crystal
Predicted: poly_crystal
Predicted: poly_crystal
Predicted: single_crystal
Predicted: single_crystal
Predicted: single_crystal
Predicted: poly_crystal
Predicted: poly_crystal
Predicted: poly_crystal
Predicted: single_crystal
Predicted: single_crystal
Predicted: poly_crystal
Predicted: single_crystal
Predicted: single_crystal
Predicted: poly_crystal
Predicted: poly_crystal
Predicted: single_crystal
Predicted: single_crystal
Predicted: poly_crystal
Predicted: single_crystal
Predicted: single_crystal
Predicted: single_crystal
Predicted: single_crystal
Predicted: poly_crystal
Predicted: poly_crystal
Predicted: single_crystal
Predicted: single_crystal
Predicted: poly_crystal

Figure 4.2: *A plot of 100 test images and their predicted labels. Notice that there is one red label (first column, third row) because that prediction result was different from the label provided in the test dataset.*

To further test the accuracy of this model, a polycrystalline image (from Fig. 1.2) and a single crystalline image (from Fig. 1.1) was fed into the model afterwards. Both of these images were not a part of the training or validation dataset. The certainties were also printed to determine how confident the model was in its prediction (see Fig. 4.3 and Fig. 4.4).

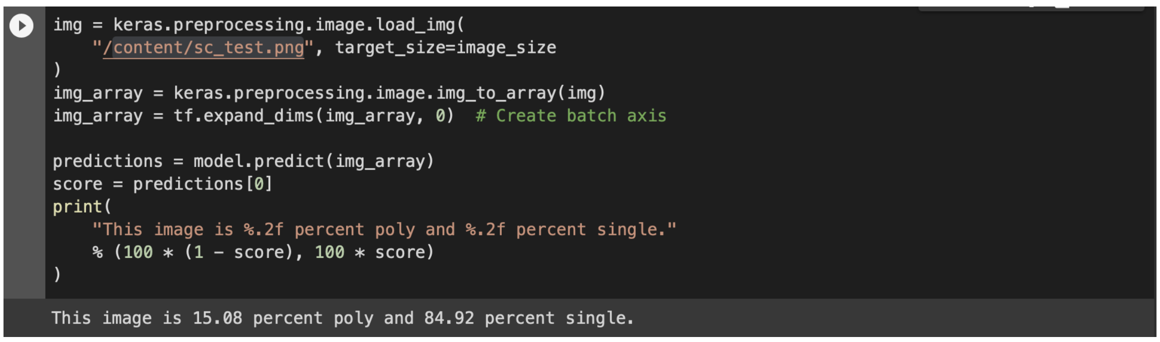


```
[31] img = keras.preprocessing.image.load_img(
         "/content/poly_test.png", target_size=image_size
     )
     img_array = keras.preprocessing.image.img_to_array(img)
     img_array = tf.expand_dims(img_array, 0)  # Create batch axis

     predictions = model.predict(img_array)
     score = predictions[0]
     print(
         "This image is %.2f percent poly and %.2f percent single."
         % (100 * (1 - score), 100 * score)
     )

     This image is 97.54 percent poly and 2.46 percent single.
```

Figure 4.3: *The model predicting whether the image is polycrystalline or single crystalline. The model was more certain that this image was polycrystalline, which is correct.*



```
img = keras.preprocessing.image.load_img(
    "/content/sc_test.png", target_size=image_size
)
img_array = keras.preprocessing.image.img_to_array(img)
img_array = tf.expand_dims(img_array, 0)  # Create batch axis

predictions = model.predict(img_array)
score = predictions[0]
print(
    "This image is %.2f percent poly and %.2f percent single."
    % (100 * (1 - score), 100 * score)
)

This image is 15.08 percent poly and 84.92 percent single.
```

Figure 4.4: *The model predicting whether the image is polycrystalline or single crystalline. The model was more certain that this image was single crystalline, which is correct.*

5 Conclusions and Further Steps

After going through all of the training steps for the Tensorflow Lite model and the Tensorflow Keras model, the accuracy for these models were 99% and 98%, respectively. However, these accuracy values refer to the accuracy of the model correctly labeling the images in the validation dataset. When introducing images from outside of the datasets, the confidence levels of the models seemed to decrease. For example, in Fig. 4.4, the print out in this section of code indicates that the model was only 84.92% confident that the image was single crystal. This could most likely be fixed by adding more single and polycrystalline images to their respective datasets.

A machine learning model for determining whether a RHEED image is polycrystalline or single crystalline would be useful for pipelining to existing analysis software in order to streamline the analysis of diffraction intensity to determine the Bragg-Williams order parameter or the lattice constants. Since the process and code for determining the order parameter of a polycrystalline image is different than for a single crystalline image, this model could classify the image and then feed the image to the next, appropriate step. This is also the case for when determining the lattice parameters.

Now that there is a model that can predict whether a RHEED image is polycrystalline or single crystalline, we could expand this model to be able to automatically rotate to fix on the next direction. To do so, we would have to add another dataset of images that are slightly off of the direction of symmetry. This would involve taking several pictures of single crystal RHEED images are not at optimized angles. Since these images are not typically taken on purpose, this dataset has not been made yet.

The immediate next step will be including the converted model.h file into an Arduino IDE code. This code would be able to use a RHEED image taken by the Himax CMOS Imaging camera and be able to determine whether this image is single or polycrystalline. Depending on the result, the code would then be able to instruct the stepper motor to do something and/or pipeline the image into already existing software for further analysis.

6 Appendix: Code

6.1 Motor Controller Code

```
#include <AccelStepper.h>
#define stp 2
#define dir 3
#define MS1 4
#define MS2 5
#define MS3 6
#define EN  7
AccelStepper stepper(AccelStepper::DRIVER, 9, 8);
char user_input;
int angleGo = 0;

void setup() {
  stepper.setMaxSpeed(2000);
  stepper.setAcceleration(1000);
  pinMode(stp, OUTPUT);
  pinMode(dir, OUTPUT);
  pinMode(MS1, OUTPUT);
  pinMode(MS2, OUTPUT);
  pinMode(MS3, OUTPUT);
  pinMode(EN, OUTPUT);
  //resetBEDPins(); //Set step, direction, microstep and enable pins to default states
  Serial.begin(115200); //Open Serial connection for debugging
  Serial.println("Begin motor control");
  Serial.println();
  Serial.println("Enter angle to take RHEED picture at:");
  Serial.println();
}

void loop() {
  while(Serial.available()){
      angleGo = Serial.parseInt(); //Read user input and trigger appropriate function
      digitalWrite(EN, LOW); //Pull enable pin low to set FETs active and allow motor control
      UserSpecify();
  }
}

void UserSpecify(){
  digitalWrite(dir, LOW); //Pull direction pin low to move "forward"
  if (stepper.distanceToGo() > 0){
    stepper.moveTo(angleGo);
    Serial.print("Moving to ");
    Serial.println(angleGo);
  }
  stepper.run();
  Serial.println("Enter new option");
  Serial.println();
}
```

6.2 Tensorflow Lite

```
import matplotlib.pyplot as plt
import numpy as np
import os
import tensorflow as tf
assert tf.__version__.startswith('2')
import pathlib
import tempfile
from tflite_model_maker import configs
from tflite_model_maker import ExportFormat
from tflite_model_maker import image_classifier
from tflite_model_maker import ImageClassifierDataLoader
from tflite_model_maker import model_spec

image_path = os.path.join(os.path.dirname('/content/rheed/rheed_images'), 'rheed_images')
data = ImageClassifierDataLoader.from_folder(image_path)
train_data, test_data = data.split(0.9)

model = image_classifier.create(train_data)

loss, accuracy = model.evaluate(test_data)
model.export(export_dir='.')
model.summary()

# helper function that returns 'red'/'black' depending on if its input parameter matches or not.
def get_label_color(val1, val2):
  if val1 == val2:
    return 'black'
  else:
    return 'red'

# Then plot 100 test images and their predicted labels.
# If a prediction result is different from the label provided label in "test"
# dataset, we will highlight it in red color.
plt.figure(figsize=(20, 20))
predicts = model.predict_top_k(test_data)
for i, (image, label) in enumerate(test_data.gen_dataset().unbatch().take(100)):
  ax = plt.subplot(10, 10, i+1)
  plt.xticks([])
  plt.yticks([])
  plt.grid(False)
  plt.imshow(image.numpy(), cmap=plt.cm.gray)
  predict_label = predicts[i][0][0]
  color = get_label_color(predict_label,
                          test_data.index_to_label[label.numpy()])
  ax.xaxis.label.set_color(color)
  plt.xlabel('Predicted: %s' % predict_label)
plt.show()
```

6.3 Tensorflow Keras

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import pathlib
import matplotlib.pyplot as plt

data_dir = pathlib.Path('/content/rheed/rheed_images')
image_size = (180, 180)
batch_size = 32

train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    data_dir,
    validation_split=0.2,
    subset="training",
    seed=1337,
    image_size=image_size,
    batch_size=batch_size,
)
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    data_dir,
    validation_split=0.2,
    subset="validation",
    seed=1337,
    image_size=image_size,
    batch_size=batch_size,
)

class_names = train_ds.class_names
print(class_names)
plt.figure(figsize=(10, 10))
for images, labels in train_ds.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(int(labels[i]))
        plt.axis("off")

data_augmentation = keras.Sequential(
    [
        layers.experimental.preprocessing.RandomFlip("horizontal"),
        layers.experimental.preprocessing.RandomRotation(0.1),
    ]
)

plt.figure(figsize=(10, 10))
for images, _ in train_ds.take(1):
    for i in range(9):
        augmented_images = data_augmentation(images)
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(augmented_images[0].numpy().astype("uint8"))
        plt.axis("off")

train_ds = train_ds.prefetch(buffer_size=32)
val_ds = val_ds.prefetch(buffer_size=32)

def make_model(input_shape, num_classes):
    inputs = keras.Input(shape=input_shape)
    # Image augmentation block
    x = data_augmentation(inputs)

    # Entry block
    x = layers.experimental.preprocessing.Rescaling(1.0 / 255)(x)
    x = layers.Conv2D(32, 3, strides=2, padding="same")(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation("relu")(x)

    x = layers.Conv2D(64, 3, padding="same")(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation("relu")(x)
    previous_block_activation = x  # Set aside residual

    for size in [128, 256, 512, 728]:
        x = layers.Activation("relu")(x)
        x = layers.SeparableConv2D(size, 3, padding="same")(x)
        x = layers.BatchNormalization()(x)

        x = layers.Activation("relu")(x)
        x = layers.SeparableConv2D(size, 3, padding="same")(x)
        x = layers.BatchNormalization()(x)

        x = layers.MaxPooling2D(3, strides=2, padding="same")(x)

        # Project residual
        residual = layers.Conv2D(size, 1, strides=2, padding="same")(
            previous_block_activation
        )
        x = layers.add([x, residual])  # Add back residual
        previous_block_activation = x  # Set aside next residual

    x = layers.SeparableConv2D(1024, 3, padding="same")(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation("relu")(x)

    x = layers.GlobalAveragePooling2D()(x)
    if num_classes == 2:
        activation = "sigmoid"
        units = 1
    else:
        activation = "softmax"
        units = num_classes

    x = layers.Dropout(0.5)(x)
    outputs = layers.Dense(units, activation=activation)(x)
    return keras.Model(inputs, outputs)

model = make_model(input_shape=image_size + (3,), num_classes=2)
keras.utils.plot_model(model, show_shapes=True)

epochs = 10
callbacks = [
    keras.callbacks.ModelCheckpoint("save_at_{epoch}.h5"),
]
```

```
model.compile(
    optimizer=keras.optimizers.Adam(1e-3),
    loss="binary_crossentropy",
    metrics=["accuracy"],
)
model.fit(
    train_ds, epochs=epochs, callbacks=callbacks, validation_data=val_ds,
)

model.summary()
loss, accuracy = model.evaluate(val_ds)

img = keras.preprocessing.image.load_img(
    "/content/rheed/crystals-10-00523-g010.png", target_size=image_size
)
img_array = keras.preprocessing.image.img_to_array(img)
img_array = tf.expand_dims(img_array, 0)  # Create batch axis

predictions = model.predict(img_array)
score = predictions[0]
print(
    "This image is %.2f percent poly and %.2f percent single."
    % (100 * (1 - score), 100 * score)
)

img = keras.preprocessing.image.load_img(
    "/content/rheed/rheed3.jpg", target_size=image_size
)
img_array = keras.preprocessing.image.img_to_array(img)
img_array = tf.expand_dims(img_array, 0)  # Create batch axis

predictions = model.predict(img_array)
score = predictions[0]
print(
    "This image is %.2f percent poly and %.2f percent single."
    % (100 * (1 - score), 100 * score)
)
```

6.4 Converting Keras to Lite to Header File

```
!pip install tflite-model-maker
!pip3 install --extra-index-url https://google-coral.github.io/py-repo/ tflite_runtime
# Setup environment
!apt-get -qq install xxd
!pip install pandas numpy matplotlib
%tensorflow_version 2.x
!pip install tensorflow

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import tensorflow as tf
import os
import fileinput
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import matplotlib.pyplot as plt

model = keras.models.load_model("save_at_12.h5")
model.summary()
image_size = (180, 180)
batch_size = 32

# Convert the model to the TensorFlow Lite format without quantization
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()
# Save the model to disk
open("rheed.tflite", "wb").write(tflite_model)
basic_model_size = os.path.getsize("rheed.tflite")
print("Model is %d bytes" % basic_model_size)

!echo "const unsigned char model[] = {" > /content/model.h
!cat rheed.tflite | xxd -i              >> /content/model.h
!echo "};"                              >> /content/model.h

model_h_size = os.path.getsize("model.h")
print(f"Header file, model.h, is {model_h_size:,} bytes.")
print("\nOpen the side panel (refresh if needed). Double click model.h to download the file.")

img = keras.preprocessing.image.load_img(
    "/content/sc_test.png", target_size=image_size
)
img_array = keras.preprocessing.image.img_to_array(img)
img_array = tf.expand_dims(img_array, 0)  # Create batch axis

predictions = model.predict(img_array)
score = predictions[0]
print(
    "This image is %.2f percent poly and %.2f percent single."
    % (100 * (1 - score), 100 * score)
)

img = keras.preprocessing.image.load_img(
    "/content/poly_test.png", target_size=image_size
)
img_array = keras.preprocessing.image.img_to_array(img)
img_array = tf.expand_dims(img_array, 0)  # Create batch axis

predictions = model.predict(img_array)
score = predictions[0]
print(
    "This image is %.2f percent poly and %.2f percent single."
    % (100 * (1 - score), 100 * score)
)
```

Bibliography

[1] S. Hasegawa, “Reflection HighEnergy Electron Diffraction,” in *Characterization of Materials*, pp. 1925–1938, John Wiley & Sons, Inc, 2 ed., Oct. 2012.

[2] M. Dabrowska-Szata, “Analysis of RHEED pattern from semiconductor surfaces,” *Materials Chemistry and Physics*, vol. 81, pp. 257–259, 2003.

[3] A. Ichimiya and P. Cohen, *Reflection High-Energy Electron Diffraction*. Cambridge University Press, 2004.

[4] N. Feldberg, *$ZnSnN_2$: Growth and Characterization of an Earth Abundant Element Material with Order Dependent Properties*. PhD thesis, University at Buffalo, New York, May 2015.

[5] R. A. Makin, K. York, S. M. Durbin, N. Senabulya, J. Mathis, R. Clarke, N. Feldberg, P. Miska, C. M. Jones, Z. Deng, L. Williams, E. Kioupakis, and R. J. Reeves, “Alloy-Free Band Gap Tuning across the Visible Spectrum,” *Physical Review Letters*, vol. 122, p. 256403, June 2019.

[6] R. A. Makin, K. York, S. M. Durbin, and R. J. Reeves, “Revisiting semiconductor band gaps through structural motifs: An Ising model perspective,” *Physical Review B*, vol. 102, p. 115202, Sept. 2020.

[7] K. K. Rout, S. Mishra, and A. Routray, “Development of an Internet of Things (IoT) Based Introductory Laboratory for Undergraduate Engineering Students,” in *2017 International Conference on Information Technology (ICIT)*, (Bhubaneswar), pp. 113–118, IEEE, Dec. 2017.

[8] M. Su, G. Lu, H. Xu, Q. Peng, and L. Huang, “Design and implementation of semiconductor Laser reliability testing platform,” *th International Conference on Electronic Packaging Technology*, p. 3, 2018.

[9] S. Baltayan, C. Kreiter, and A. Pester, “An online DC-motor test bench for engineering education,” in *2018 IEEE Global Engineering Education Conference (EDUCON)*, (Tenerife), pp. 1484–1488, IEEE, Apr. 2018.

[10] M. Poongothai, P. M. Subramanian, and A. Rajeswari, “Design and implementation of IoT based smart laboratory,” in *2018 5th International Conference on Industrial Engineering and Applications (ICIEA)*, (Singapore), pp. 169–173, IEEE, Apr. 2018.

[11] W. Neil, G. Zipp, G. Nemeth, M. F. Russo, and D. S. Nirschl, “End-to-End Sample Tracking in the Laboratory Using a Custom Internet of Things Device,” *SLAS TECHNOLOGY: Translating Life Sciences Innovation*, vol. 23, pp. 412–422, Oct. 2018.

[12] G. Lippi and G. Da Rin, “Advantages and limitations of total laboratory automation: a personal overview,” *Clinical Chemistry and Laboratory Medicine (CCLM)*, vol. 57, pp. 802–811, May 2019.

[13] R. Zaitsev, M. Kirichenko, L. Zaitseva, and S. Radoguz, “Automation Measurement System of Semiconductor Devices Parameters,” p. 4, 2020.

[14] A. Wolf, P. Galambos, and K. Szell, “Device Integration Concepts in Laboratory Automation,” in *2020 IEEE 24th International Conference on Intelligent Engineering Systems (INES)*, (Reykjavk, Iceland), pp. 171–178, IEEE, July 2020.

[15] D. Rafique and L. Velasco, “Machine Learning for Network Automation: Overview, Architecture, and Applications [Invited Tutorial],” *Journal of Optical Communications and Networking*, vol. 10, p. D126, Oct. 2018.

[16] J.-P. Correa-Baena, K. Hippalgaonkar, J. van Duren, S. Jaffer, V. R. Chandrasekhar, V. Stevanovic, C. Wadia, S. Guha, and T. Buonassisi, “Accelerating Materials Development via Automation, Machine Learning, and High-Performance Computing,” *Joule*, vol. 2, pp. 1410–1420, Aug. 2018.

[17] M. S. Mahdavinejad, M. Rezvan, M. Barekatain, P. Adibi, P. Barnaghi, and A. P. Sheth, “Machine learning for internet of things data analysis: a survey,” *Digital Communications and Networks*, vol. 4, pp. 161–175, Aug. 2018.

[18] I. J. Marshall and B. C. Wallace, “Toward systematic review automation: a practical guide to using machine learning tools in research synthesis,” *Systematic Reviews*, vol. 8, pp. 163, s13643–019–1074–9, Dec. 2019.