Break Through Tech AI
December 11, 2024

AI STUDIO PROJECT: FINAL PRESENTATION

Introductions

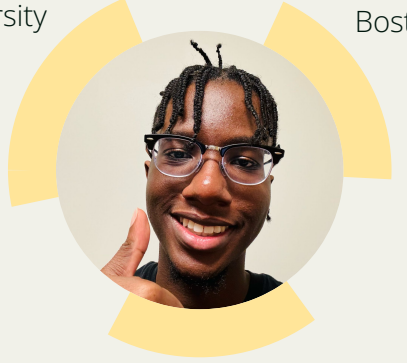
01

The Team

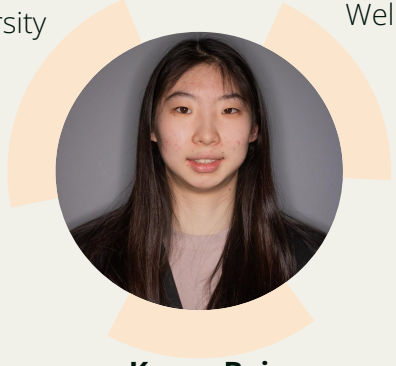
Giselle Yang
Simmons University

Valentina Haddad
Boston University

Marelyn Gonzalez
Wellesley College



Andersen Prince
Tufts University



Karen Bei
Tufts University

Our Leaders

Andrii Zahorodnii
AI Studio TA

Taylor Hogan
Challenge Advisor

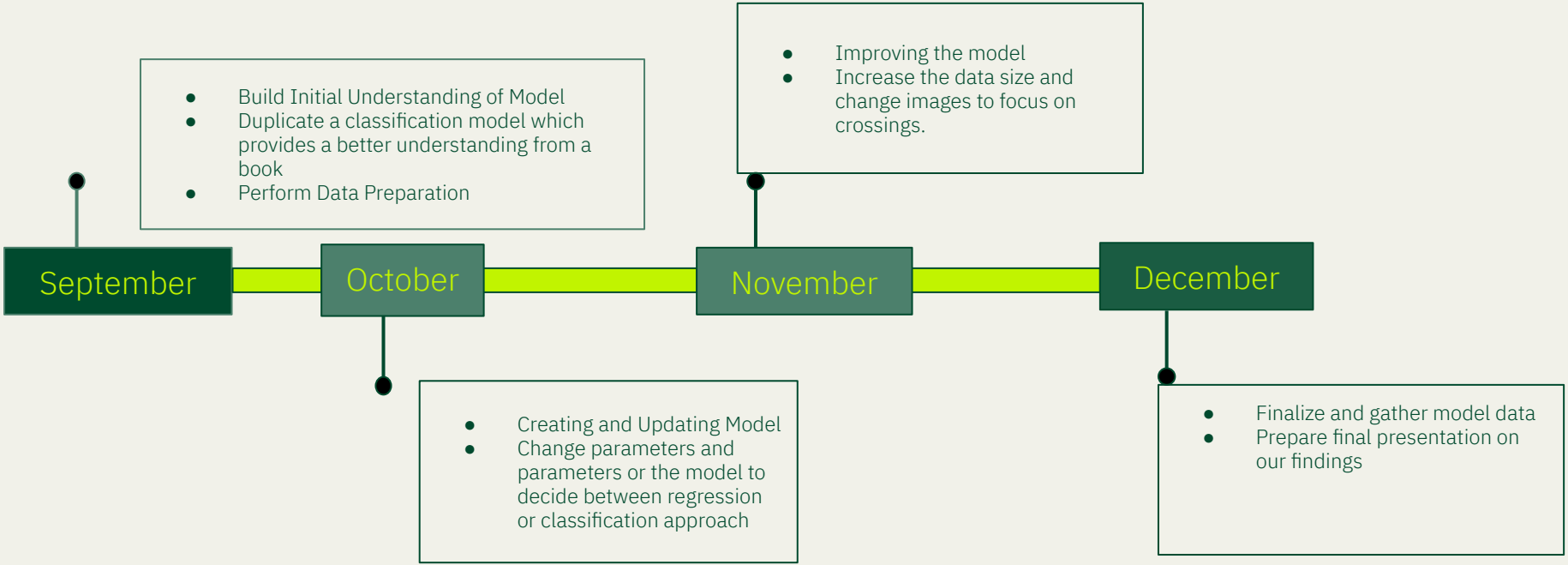
Ksenia Roze
Challenge Advisor

Contents

Project Overview

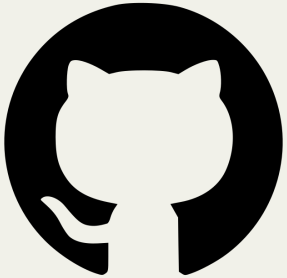
02

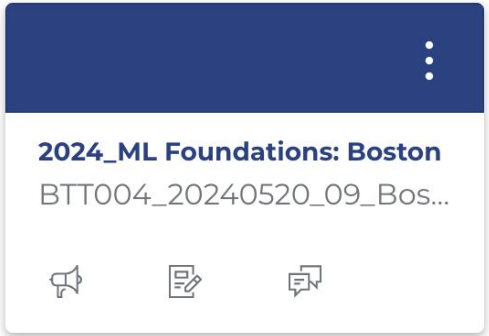
TIMELINE

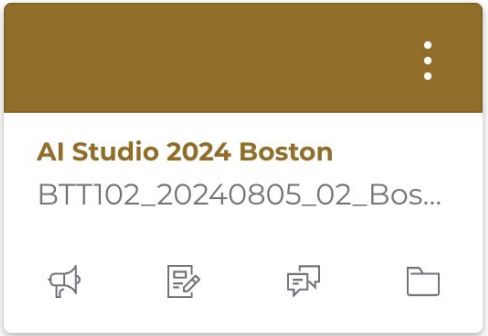

Build Initial Understanding of Model
Duplicate a classification model which provides a better understanding from a book
Perform Data Preparation
Improving the model
Increase the data size and change images to focus on crossings.
September
October
November
December
Creating and Updating Model
Change parameters and parameters or the model to decide between regression or classification approach
Finalize and gather model data
Prepare final presentation on our findings

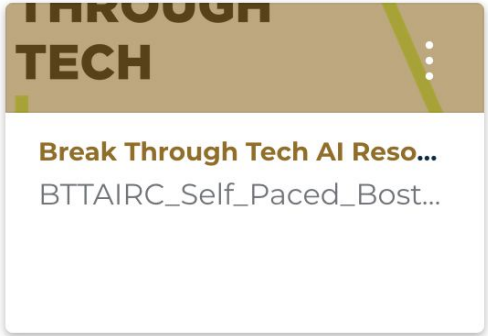
RESOURCES WE LEVERAGED











PROJECT OVERVIEW

TASK

To build a Convolutional Neural Network (CNN) that predicts the quality of PCB/electronic designs. The CNN will analyze key metrics, generating a numerical score correlated with the "goodness" of a design.

BUSINESS IMPACT

A CNN for PCB design checks for problems automatically, saving time and money by catching issues early. It makes designs better and more reliable while helping companies stand out with advanced tech.

WHY?

Using a CNN is important because it saves time and money by finding problems early, which means less waste. It also ensures designs are high-quality and reliable, so the final product works better.

IN DEPTH – OUR GOAL

Objective: Determine if a given chip design is "good" or "bad" based on the number of signal crossings.

1. **Key Insight:** Fewer crossings lead to less signal interference, which indicates a better chip design.
2. **Purpose:** Demonstrate how machine learning (ML) can accelerate design automation by evaluating chip designs efficiently.
3. **Approach:** Assess and explore various ML models to improve the accuracy of design quality scoring
    - Example model: "Book" model
    - Classification model
    - And others...

ML IN HARDWARE ENGINEERING

TRADITIONALLY

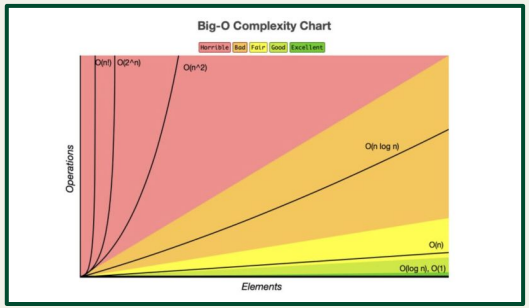
- Single thread execution
- Non-parallel programming → slower processing

ADVANTAGE

- Neural Network inference = constant time
- on GPUs, can process hundreds of evaluations per second, regardless of input size
- Allows parallel processing of multiple tasks

EXAMPLE

- Time complexity of determining # of crossings w/ different methods...



In traditional computational methods:

Best: O(n log n)

Worst: O(N^2)

With neural networks on a GPU:

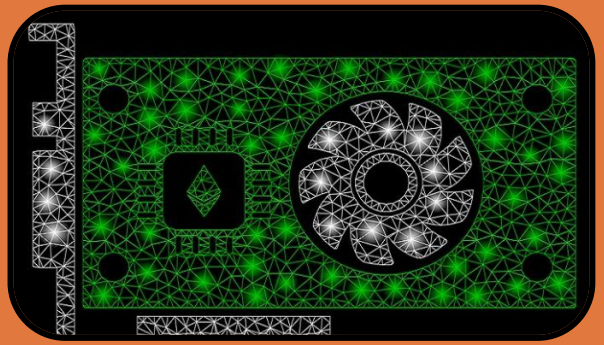
O(# of layers * avg # neurons/layer * avg # of neurons/layer)

With GPU: O(1) b/c of parallel execution!

ML IN HARDWARE ENGINEERING

Importance in Automation and Optimization

- Most optimization algorithms rely on a goal/fitness function to evaluate potential solutions
- Testing solutions → speed = limiting factor in finding optimal results
- Neural networks act with human intuition but faster and more accurate



Benefits of GPU Utilization

- Costly, but enables parallel evaluation of multiple fitness functions (ex: crossings, congestion, alignment)
- Entire optimization algorithms can be run on the GPU = enhanced speed/efficiency

Data Understanding

03

CODE FLOW

SET UP THE DATABASE

We start with a simple EDA database that looks like real electronics. It's built to act like real designs with different parts and layers.

MAKE SAMPLE DATA

The database creates samples with different shapes and setups to have lots of variety.

TURN INTO PICTURES

Each sample is turned into a picture that shows the design.

SCORE & SORT

The pictures get a "good" or "bad" score, saved in folders, and tracked in a CSV file.

TRAIN

The pictures and scores are used to teach the CNN how to tell good designs from bad ones.

TEST DESIGNS

We can use eval_model.py to test new pictures and see if they're good or bad.

LABELS

1	good/t0.svg	0.0
2	bad/t1.svg	0.9611111111111111
3	bad/t2.svg	0.9277777777777778
4	good/t3.svg	0.0
5	good/t4.svg	0.20555555555555555

45	good/t44.svg	0.35
46	bad/t45.svg	0.9166666666666666
47	bad/t46.svg	0.9444444444444444
48	good/t47.svg	0.0
49	bad/t48.svg	0.8666666666666667

0.5 > BAD
0.5 < GOOD

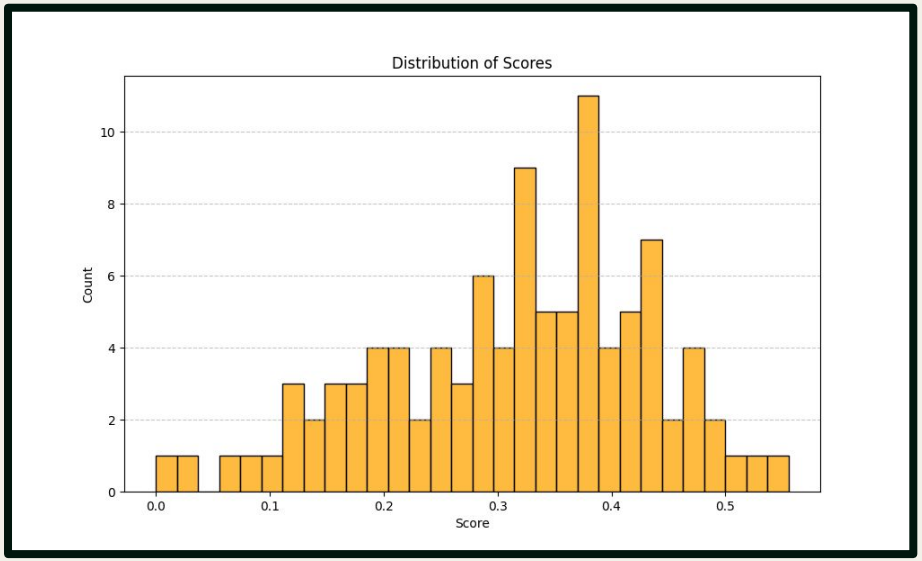
HOW TO TRAIN & EVALUATE?

```
Using 4367 files for training.
Using 1091 files for validation.
Epoch 1/25
137/137 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 156s 1s/step - acc: 0.9590 - loss: 0.0721 - val_acc: 0.5555 -
val_loss: 0.8566
Epoch 2/25
137/137 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 153s 1s/step - acc: 0.9873 - loss: 0.0298 - val_acc: 0.5555 -
val_loss: 1.6911
```

• • •

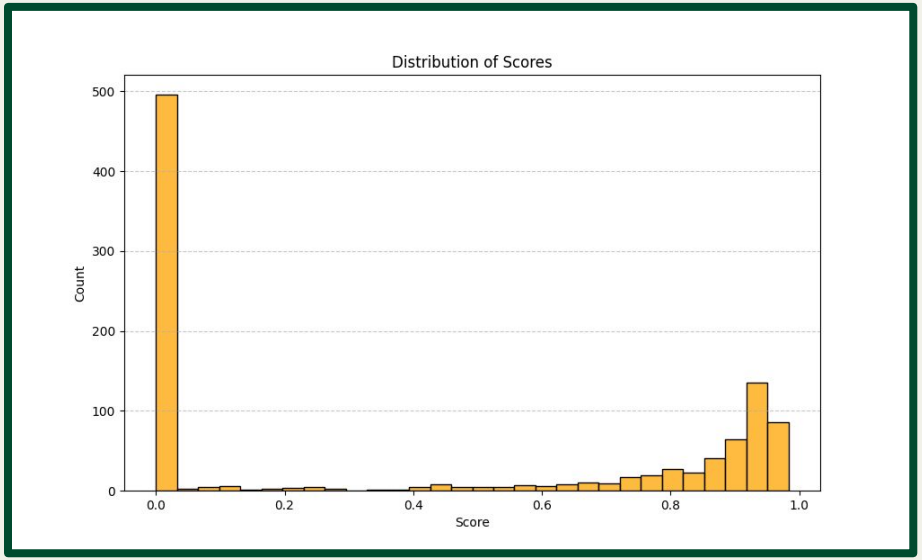
```
Epoch 25/25
137/137 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 156s 1s/step - acc: 0.9962 - loss: 0.0117 - val_acc: 0.9945 -
val_loss: 0.0130
137/137 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 27s 199ms/step - acc: 0.9163 - loss: 0.4414
Test accuracy: 0.916
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 97ms/step
This image is 100.00% Good and 0.00% Bad.
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
This image is 0.06% Good and 99.94% Bad.
```

- Data Preparation
- Training the Model
- Model Testing

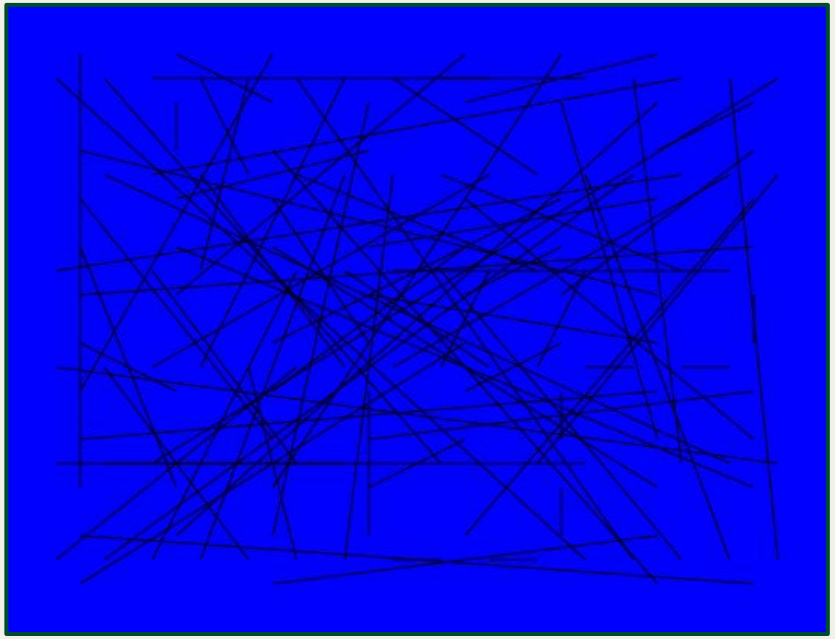


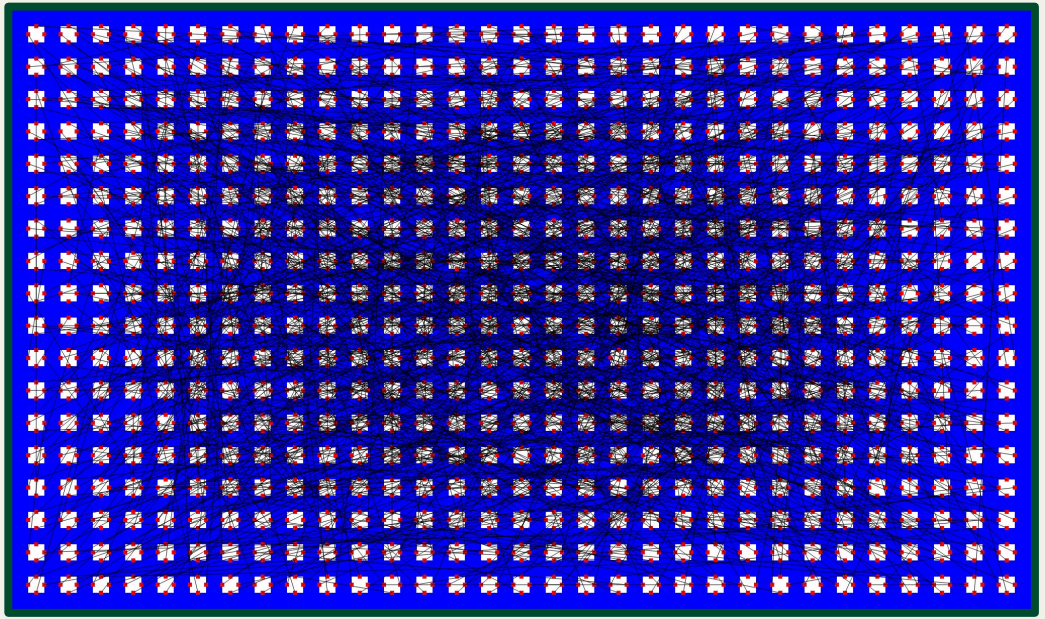
PEAK AT THE DATA

1/2 good, 1/2 bad with the bad being clumped on the bad side



TRAINING IMAGE





Lots of Crossings:

- These designs are messy and not as efficient.
- They can cause problems like signal issues or make the board harder to build.

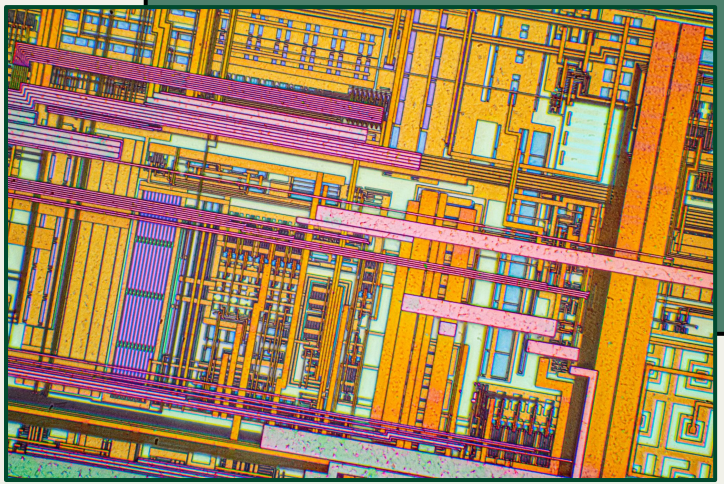
Fewer Crossings:

- These are better designs. The components are placed neatly, and the wiring is cleaner.

Model
Approach

04

KEY METRICS



Crossings & Congestion

Crossing and congestion are ways to measure how good the layout of a chip, package, or PCB is. They show if the design is organized or messy, which can affect how well it works and how easy it is to make.

Why Crossings are BAD

Crossings happen when wires or connections overlap, which can cause problems like bad signals or making it harder to build. Keeping crossings low helps the design work better and be more reliable.

WHY USE NEURAL NETWORKS?

Advantages of Neural Networks for Determining Crossings

- Encodes more than crossing metrics: considers congestion, overlaps
- Constant time metric determination, regardless of design complexity.
- Faster “fitness function” speeds up solution searching.

Efficiency & Scalability

- Parallel processing: One image can score multiple metrics via federated systems.
- Less code compared to traditional C++ implementations.
- Adding new metrics requires only additional training—not re-coding.

Time-Saving Benefits

- Training on an off-the-shelf Mac laptop takes
- Takes approximately 3 minutes.

STRUCTURE OF NEURAL NETWORKS

Convolutional Layers:

- These are like little detectives that scan the image to find patterns.
- For PCBs, they look for things like wires, crossings, and where components are placed.

Pooling Layers:

- These layers shrink the image down but keep the most important parts.
- It’s like zooming out so the model can focus on the big picture instead of tiny details.

Fully Connected Layers:

- These layers are the decision-makers. They take all the information the earlier layers found and figure out if the PCB is "good" or "bad."

Model Evaluation

05

Dog vs Cat Example

Model

- CNN binary classification model
- Supervised learning
  - Label: dog/cat
  - Feature: Cat and Dog Images
- Base Model

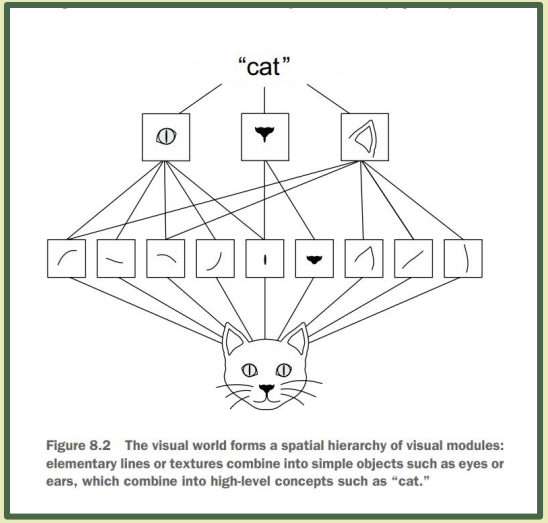


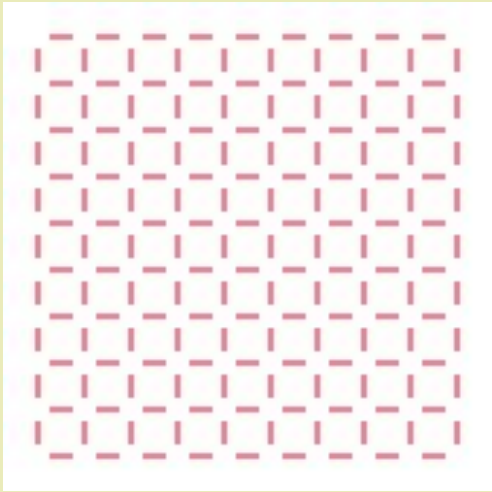
Figure 8.2 The visual world forms a spatial hierarchy of visual modules: elementary lines or textures combine into simple objects such as eyes or ears, which combine into high-level concepts such as "cat."

Source: Deep Learning with Python (p. 205)

Good vs Bad Chip

Model

- CNN binary classification model
- Supervised learning
  - Label: good/bad
  - Feature: Chip Images
- Final Model



MODEL COMPARISON

Book Classification	CNN Classification	CNN Regression
Model that is able to classify an image as a dog or cat	Model that determine if a chip is good or bad, where good chips have less crossings	Model that predicts the labels of generated chip images
High Accuracy Low Loss Is able to accurately classify dogs vs cats	High Accuracy Good for binary problems (good/bad)	Low Accuracy Able to predict actual label scores for the chips

MODEL ANALYSIS

Reasonably high prediction, showing overall good performance

Suspect overfitting?

– Accuracy is high due to simplicity and binary problem

OUR MODEL IS

92%

ACCURATE

PRELIMINARY INSIGHTS

Using 800 files for training.
Using 200 files for validation.
Epoch 1/5
25/25 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8060s 325/step - acc: 0.6993 - loss: 0.4778
Epoch 2/5
25/25 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7013s 276s/step - acc: 0.8672 - loss: 0.2929
Epoch 3/5
25/25 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5076s 203s/step - acc: 0.9007 - loss: 0.2.644
Epoch 4/5
25/25 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5011s 201s/step - acc: 0.9067 - loss: 0.2604
Epoch 5/5
25/25 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4690s 186s/step - acc: 0.9126 - loss: 0.2488

25/25 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 33s 5s/step - acc: 0.2599 - loss: 1.3807

Loss: 1.382626533... Test accuracy: 0.259999990...

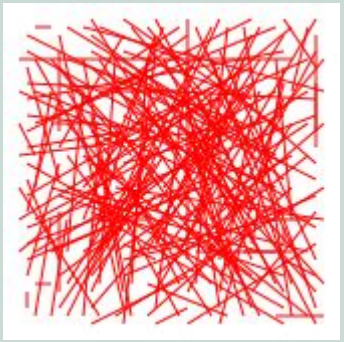
INSIGHT & KEY FINDINGS

Found 1000 files belonging to 2 classes.
Using 800 files for training.
Using 200 files for validation.
Epoch 1/5
25/25 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 29s 1s/step - acc: 0.9277 - loss: 0.1326
Epoch 2/5
25/25 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 28s 1s/step - acc: 0.9901 - loss: 0.0227
Epoch 3/5
25/25 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 28s 1s/step - acc: 0.9934 - loss: 0.0149
Epoch 4/5
25/25 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 28s 1s/step - acc: 0.9964 - loss: 0.0128
Epoch 5/5
25/25 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 28s 1s/step - acc: 0.9980 - loss: 0.0097

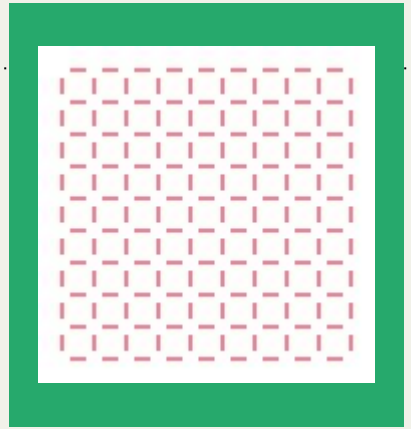
25/25 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5s 192ms/step - acc: 0.9216 - loss: 0.3646

Test accuracy: 0.926

INSIGHT &
KEY FINDINGS...



1/1 ——————————————————— 0s
90ms/step
This image is 17.61% Good and 82.39% Bad.

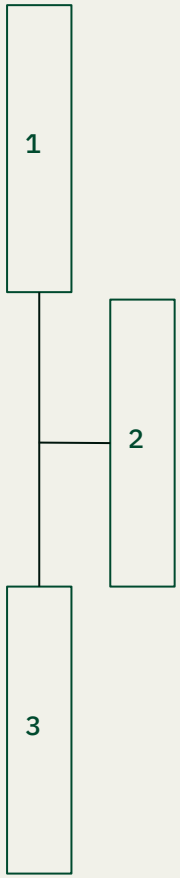


1/1 ——————————————————— 0s
23ms/step
This image is 83.74% Good and 16.26 % Bad.

Final Thoughts

06

WHAT COULD HAVE BEEN DONE BETTER ?



- Collecting more examples of PCB designs could have made the model work better.
- Checking random samples to make sure the labels were right would have improved the data.
- Adding more datasets or features might have helped the model learn more about what makes a good design.

- Adding features like congestion or alignment could have improved our results.
- Using powerful computers could have sped up working with big datasets.
- Real-world data is tricky, and we could have spent more time understanding what affects the model.

- We should have checked our data more to make sure the labels were right.
- Trying other models, like ones that are already trained, might have worked better.
- Testing different settings, like hyperparameters, could have made the model more accurate.

POTENTIAL NEXT STEPS

- Add more ways to measure how "good" a design is to give more helpful scores.
- Test how changes in training settings, like epochs, batch size, or learning rate, affect accuracy.
- Look into different data morphologies and examine how they affect the model's ability to generalize

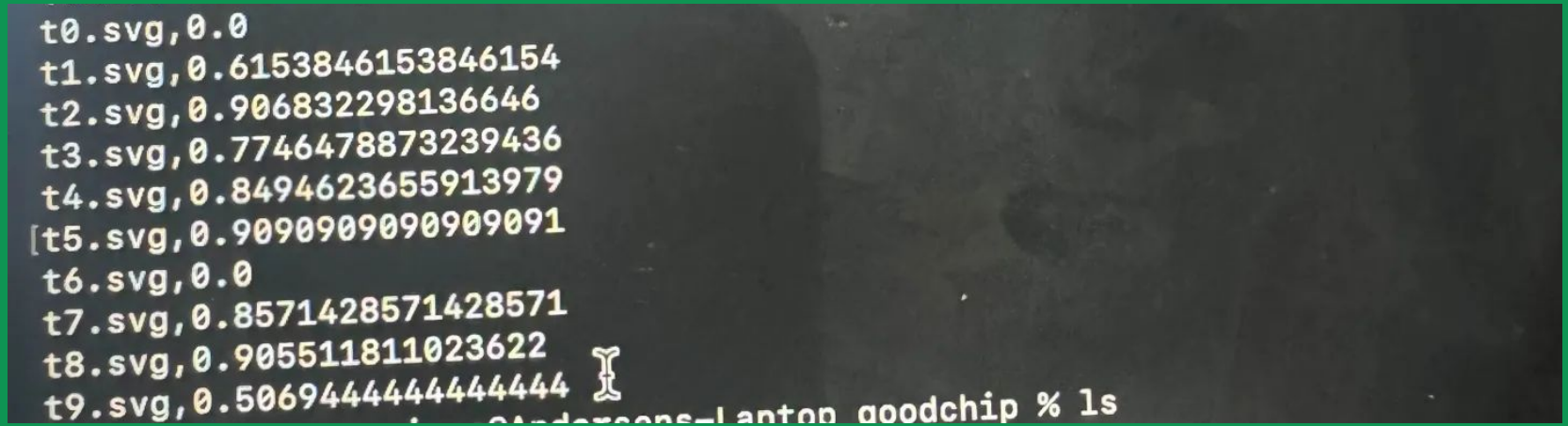
THANK YOU

Any Questions?

Appendix

Data Understanding

- Built the training data and generated our own test images
- Noticed that more crossings led to higher scores
- Formed foundation for what it means to be “good” and “bad”



Emitting Crossings

- Initially had red and white squares in the images
- There was a function in generatory.py that was drawing the components → commented out the function and any lines that called it
- images have a higher signal to noise ratio and are smaller

```
65  def draw_component(display, component) -> None:
66      r = component.macro.rect.move_by(component.loc.x, component.loc.y)
67      draw_geometry(display, r, "white")
68      for p in component.macro.pins:
69          r = p.rect.move_by(component.loc.x, component.loc.y)
70          draw_geometry(display, r, "red")
```

Loading Images

- Resized from 800x800 to 400x400 (improve efficiency)
- Converted from SVG to PNG image format
- Generated array of images and their labels
- Extracted image file names and their corresponding label
    - Stored all images in a list
- Stored all the processed images and their labels as NumPy arrays to allow for training the CNN

```
(venv) andersenprince@Andersens-Laptop goodchip % python3 loadimages.py

Reading in file: t1.svg
Label: 0.6153846153846154
Reading in file: t2.svg
Label: 0.906832298136646
Reading in file: t3.svg
Label: 0.7746478873239436
Reading in file: t4.svg
Label: 0.8494623655913979
Reading in file: t5.svg
Label: 0.9090909090909092
Reading in file: t6.svg
Label: 0.0
Reading in file: t7.svg
Label: 0.8571428571428571
Reading in file: t8.svg
Label: 0.905511811023622
Reading in file: t9.svg
Label: 0.5069444444444444
Read in 9 images and 9 labels.
(venv) andersenprince@Andersens-Laptop goodchip %
```

```
2       def __init__(self) -> None:
3           self.params = {
4               "max_components": 100,
5               "num_designs": 10,
6               "component_ds": 32,
7               "pin_ds": 8,
8               "same_design_size": False,
9               "random_changes": True
10
11          }
```

```
65  def draw_component(display, component) -> None:
66      r = component.macro.rect.move_by(component.loc.x, component.loc.y)
67      draw_geometry(display, r, "white")
68      for p in component.macro.pins:
69          r = p.rect.move_by(component.loc.x, component.loc.y)
70          draw_geometry(display, r, "red")
```

Business Impact


TECH

Nvidia passes Alphabet in market cap and is now the third most valuable U.S. company

PUBLISHED WED, FEB 14 2024•5:33 PM EST


Nvidia passes Microsoft in market cap to become most valuable public company

PUBLISHED TUE, JUN 18 2024•1:03 PM EDT | UPDATED TUE, JUN 18 2024•4:09 PM EDT

- Efficiency: With billions of transistors, chip design is challenging. AI models help by labeling designs as good or bad, speeding up the process.
- Optimization: AI ensures PCB designs are optimized for top performance.
- Money Maker: Chips bring in huge revenue, $527 billion globally, with U.S. companies contributing $264 billion.
- Going Green: Companies like NVIDIA are using more renewable energy in chip production to help the planet.
- Big Competition: NVIDIA leads in AI chips, but rivals are trying to catch up.

Algorithm Selection (CNN classification model)

- Supervised learning
  - Label: good/bad

```
65    chip_list = [] #convert into binary
66    for label in label_list:
67        if label > 0.5:
68            chip = 0 #bad
69        else:
70            chip = 1 #good
71        chip_list.append(chip)
72    y = np.array(chip_list)  # good/bad
```

  - Feature: PCB images (generated virtually, EDA)
- Binary classification model

Model Training CNN classification model

- Training and testing sets + normalizing pixel values

```
11  X_data = np.load('X.npy')
12  y_data = np.load('y.npy')
13
14
15  # Create training and test sets
16  from sklearn.model_selection import train_test_split
17  X_train, X_test, y_train, y_test = train_test_split(X_data, y_data, test_size=0.2, random_state=1234)
18
19  #normalize pixel values
20  X_train = X_train/ 255
21  X_test = X_test/ 255
```

- Shapes of X_train, X_test, Y_train, and Y_test

```
(venv) yzy@Zs-MacBook-Air goodchip % python3 cnn_model.py


X_train shape: (800, 400, 400, 3)
y_train.shape: (800,)
X_test.shape: (200, 400, 400, 3)
y_test.shape: (200,)
```

Model Training CNN classification model

- Train CNN model
  - Filters = 16, 32, 64, 128
  - GlobalAveragePooling2D
  - Dense layer: unit = 2

```
def train_model(size,strides):
    # 1. Create CNN model object
    cnn_model = keras.Sequential()


    # 2. Create the input layer and add it to the model object:
    input_shape=(X_train.shape[1:])
    input_layer = keras.layers.InputLayer(input_shape = input_shape)
    cnn_model.add(input_layer)


    # 3. Create the first convolutional layer and add it to the model object:
    conv_1 = keras.layers.Conv2D(filters = 16, kernel_size = size, strides=strides)
    batchNorm_1 = keras.layers.BatchNormalization()
    ReLU_1 = keras.layers.ReLU()
    cnn_model.add(conv_1)
    cnn_model.add(batchNorm_1)
    cnn_model.add(ReLU_1)


    # 4. Create the second convolutional layer and add it to the model object:
    conv_2 = keras.layers.Conv2D(filters = 32, kernel_size = size, strides=strides)
    batchNorm_2 = keras.layers.BatchNormalization()
    ReLU_2 = keras.layers.ReLU()
    cnn_model.add(conv_2)
    cnn_model.add(batchNorm_2)
    cnn_model.add(ReLU_2)
```

```
    # 5. Create the third convolutional layer and add it to the model object:
    conv_3 = keras.layers.Conv2D(filters = 64, kernel_size = size, strides=strides)
    batchNorm_3 = keras.layers.BatchNormalization()
    ReLU_3 = keras.layers.ReLU()
    cnn_model.add(conv_3)
    cnn_model.add(batchNorm_3)
    cnn_model.add(ReLU_3)


    # 6. Create the fourth convolutional layer and add it to the model object:
    conv_4 = keras.layers.Conv2D(filters = 128, kernel_size = size, strides=strides)
    batchNorm_4 = keras.layers.BatchNormalization()
    ReLU_4 = keras.layers.ReLU()
    cnn_model.add(conv_4)
    cnn_model.add(batchNorm_4)
    cnn_model.add(ReLU_4)


    # 7. Create the pooling layer and add it to the model object:
    pooling_layer = keras.layers.GlobalAveragePooling2D()
    cnn_model.add(pooling_layer)


    # 8. Create the output layer and add it to the model object:
    output_layer = keras.layers.Dense(units = 2, activation='softmax')
    cnn_model.add(output_layer)


    cnn_model.summary()
```

Model Training CNN classification model

- Train CNN model
  - Total params: 98,658 (385.38 KB)
  - Trainable params: 98,178 (383.51 KB)
  - Non-trainable params: 480 (1.88 KB)

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 398, 398, 16)	448
batch_normalization (BatchNormalization)	(None, 398, 398, 16)	64
re_lu (ReLU)	(None, 398, 398, 16)	0
conv2d_1 (Conv2D)	(None, 396, 396, 32)	4,640
batch_normalization_1 (BatchNormalization)	(None, 396, 396, 32)	128
re_lu_1 (ReLU)	(None, 396, 396, 32)	0
conv2d_2 (Conv2D)	(None, 394, 394, 64)	18,496
batch_normalization_2 (BatchNormalization)	(None, 394, 394, 64)	256
re_lu_2 (ReLU)	(None, 394, 394, 64)	0
conv2d_3 (Conv2D)	(None, 392, 392, 128)	73,856
batch_normalization_3 (BatchNormalization)	(None, 392, 392, 128)	512
re_lu_3 (ReLU)	(None, 392, 392, 128)	0
global_average_pooling2d (GlobalAveragePooling2D)	(None, 128)	0
dense (Dense)	(None, 2)	258

Total params: 98,658 (385.38 KB)
Trainable params: 98,178 (383.51 KB)
Non-trainable params: 480 (1.88 KB)

Hyperparameter Tuning CNN classification model

- Changed kernel sizes and strides
  - Kernel sizes: 3 and 5
  - Strides: (1,1) (default), (1,2), and (2,2)

```
 99  # change kernel sizes
100  kernel_sizes = [3, 5]
101  for size in kernel_sizes:
102     print(f'kernel size = {size}, strides = (1,1) (default)')
103     cnn_model = train_model(size,strides=(1,1)) #default strides
104     opti(cnn_model)
105     display()
106
107
108  #change strides
109  strides_values = [(1,2), (2,2)]
110  for strides in strides_values:
111     print(f'kernel size = 3, strides = {strides}')
112     cnn_model = train_model(3,strides=strides) #kernel size fixed to 3
113     opti(cnn_model)
114     display()
```

Model Evaluation CNN classification model

- Loss & accuracy
  - Loss raised
  - Accuracy dropped

```
Epoch 1/5
25/25 ━━━━━━━━━━━━━━━━━━━━ 8060s 325s/step - accuracy: 0.6993 - loss: 0.4778
Epoch 2/5
25/25 ━━━━━━━━━━━━━━━━━━━━ 7013s 276s/step - accuracy: 0.8672 - loss: 0.2929
Epoch 3/5
25/25 ━━━━━━━━━━━━━━━━━━━━ 5076s 203s/step - accuracy: 0.9007 - loss: 0.2644
Epoch 4/5
25/25 ━━━━━━━━━━━━━━━━━━━━ 5011s 201s/step - accuracy: 0.9067 - loss: 0.2604
Epoch 5/5
25/25 ━━━━━━━━━━━━━━━━━━━━ 4690s 186s/step - accuracy: 0.9126 - loss: 0.2485
Elapsed time: 29852.17s
7/7 ━━━━━━━━━━━━━━━━━━━━ 33s 5s/step - accuracy: 0.2599 - loss: 1.3807
Loss:  1.3826265335083008 Accuracy:  0.25999999046325684
7/7 ━━━━━━━━━━━━━━━━━━━━ 34s 5s/step
```

Model Evaluation CNN classification model

- Loss & accuracy
  - **Kernel size = 3, strides = (1,1)**: loss: 1.3087 accuracy: 0.2599
  - **Kernel size = 5, strides = (1,1)**: loss: 1.8253 accuracy: 0.2599
  - **Kernel size = 3, strides = (1,2)**: loss: 1.7610 accuracy: 0.2599
  - **Kernel size = 3, strides = (2,2)**: loss: 1.9242 accuracy: 0.2599
- Elapsed time
  - **Kernel size = 3, strides = (1,1)**: 29852.17s
  - **Kernel size = 5, strides = (1,1)**: 33894.48s
  - **Kernel size = 3, strides = (1,2)**: 252.36s
  - **Kernel size = 3, strides = (2,2)**: 41.23s

Model Improvement

CNN classification model

- Data format
  - Converted svg files to png files
  - Keep it as svg files
- Metrics to determine a good chip
  - Crossings
  - Congestions, component alignment, and distributed connection density
- Data size
  - 100 -> 1000
  - More than 1000 images (10,000)
- Dense layer in CNN model
  - Unit = 2
  - Unit = 1
- Epochs
  - Epoch = 5
  - Other number of epochs