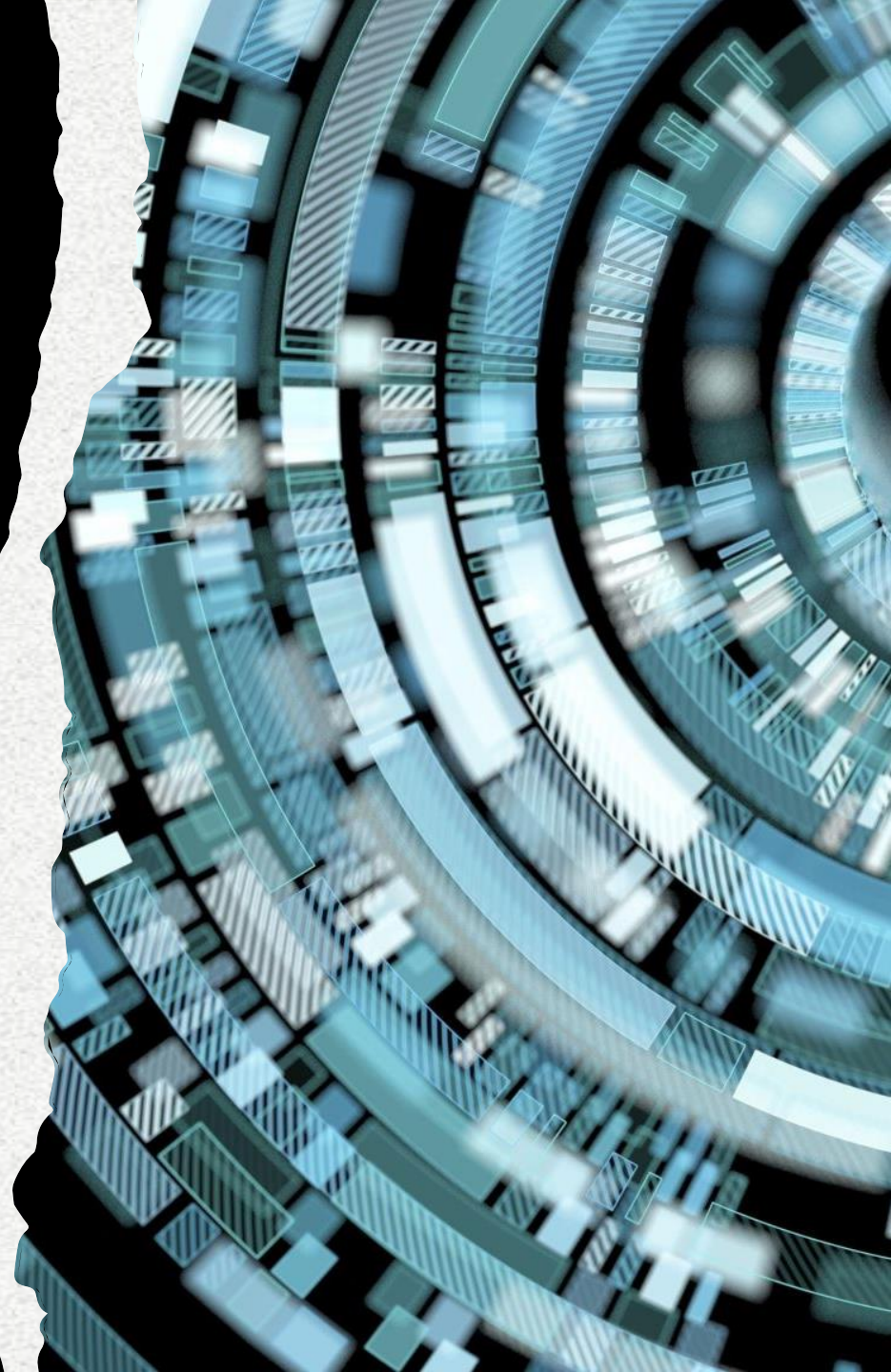
Defect Detection in Additive Manufacturing using a Convolutional Neural Network

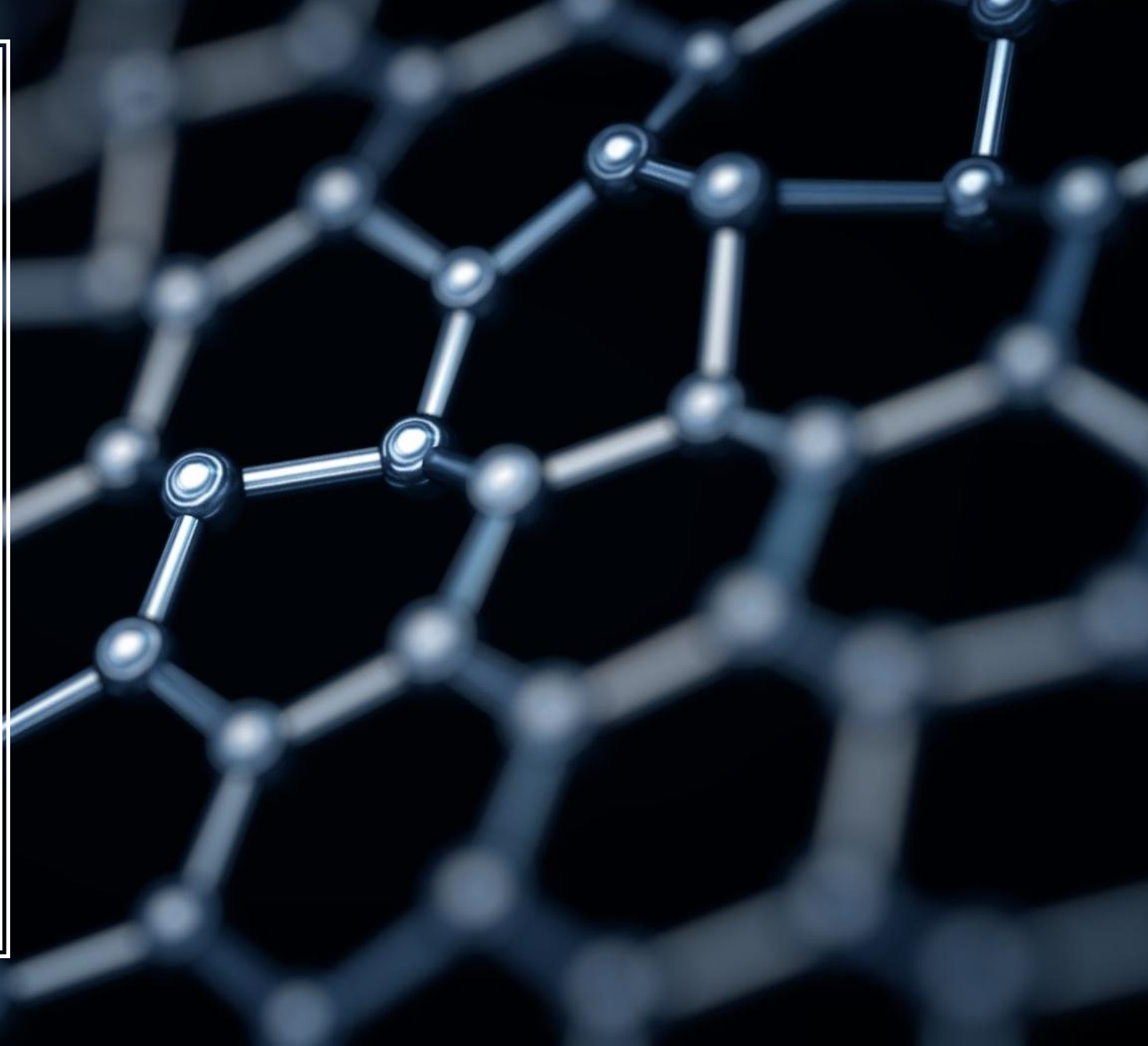
Noah Hamilton

AI in CPS Spring 2021



Outline

- Introduction/Motivation
- Problem Definition
- Background/Importance to CPS
- Approach/Struggles
- Data Acquisition
- Creating the model
- Evaluation
- Conclusions



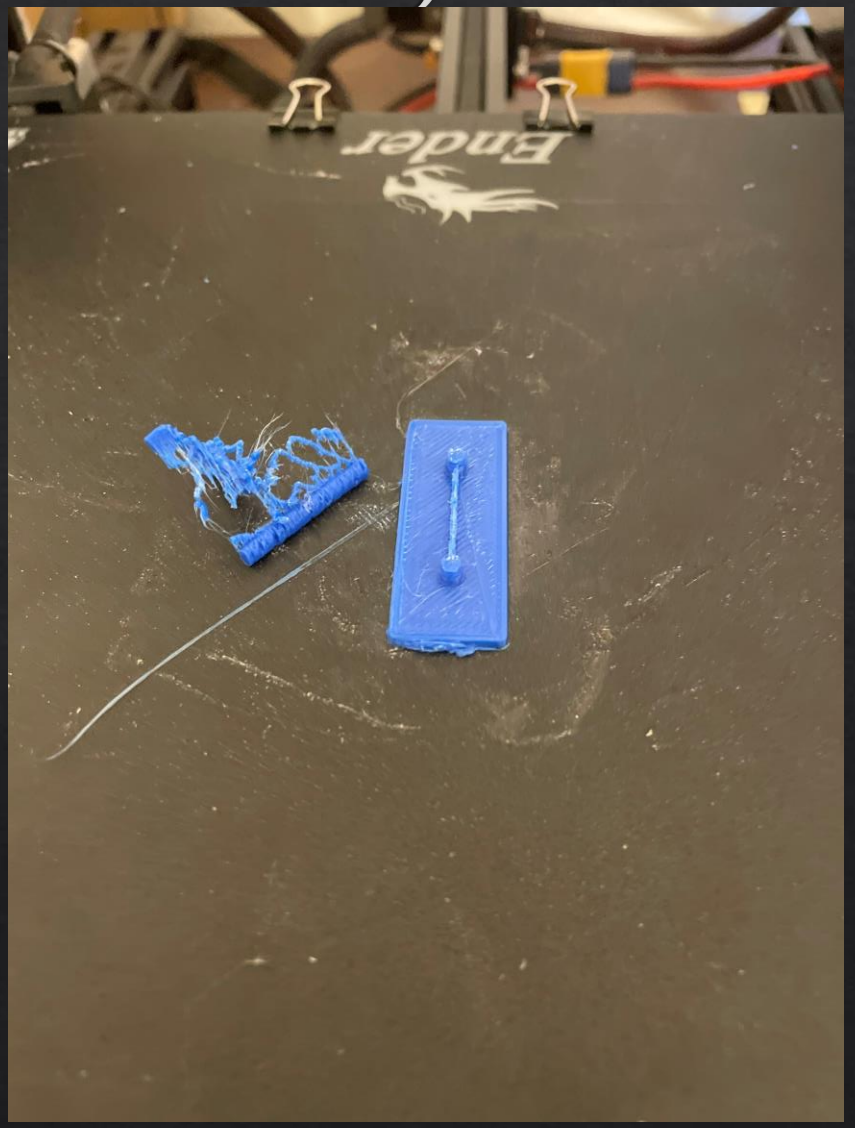
Introduction and Motivation

- Why is the right dog much better quality than the left dog?
- Process Parameters - Print Speed (mm/s), Layer Height (mm), Shell Thickness (mm), Infill density (%)
- What can we do?
  - Genetic Algorithms (GA)
    - Uses minimal ranges to test combination and compare surface roughness. (Very Inefficient)
  - Machine Learning
    - Reduce cost, gives wider range, and more parameters

Figure 1: 2 3D printed dogs with substantial differences between print quality. Photo Credits- Noah Hamilton

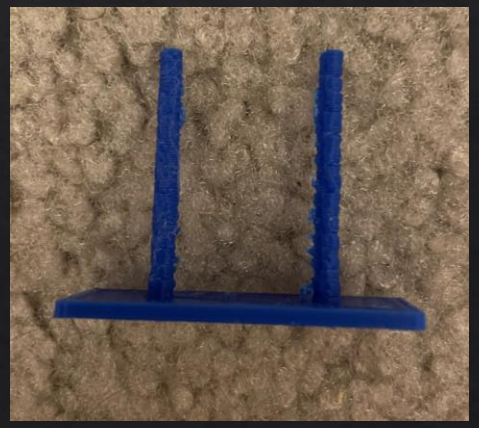
Initial Troubles (change direction)

- Get data… need to print as many models as possible.
- At first I tried to come up with 4 categories of parameters with binary values so 2^4 combinations or 16 and that became too cumbersome, then I went down to 2^3 or 8 combinations.
- After printing I realized this would fit better as a project to identify a clean print from a failed or stringing print.



Problem Definition

- Create a Convolutional Neural Network to classify images of prints into 3 categories: Clean/Processed, Stringing, and Failed Prints
- Classes: Stringing is usually identified as filament between towers, failed is not completing both towers, and clean is two solid towers without stringing.
- Sample size: 14 – stringing, 14 fail, 12, success (used data augmentation to increase)
- Goal: Predict the correct class with highest accuracy and minimal loss without overfitting

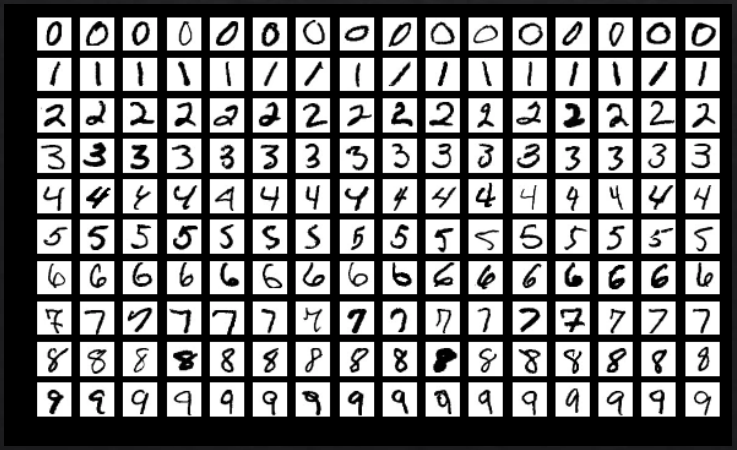


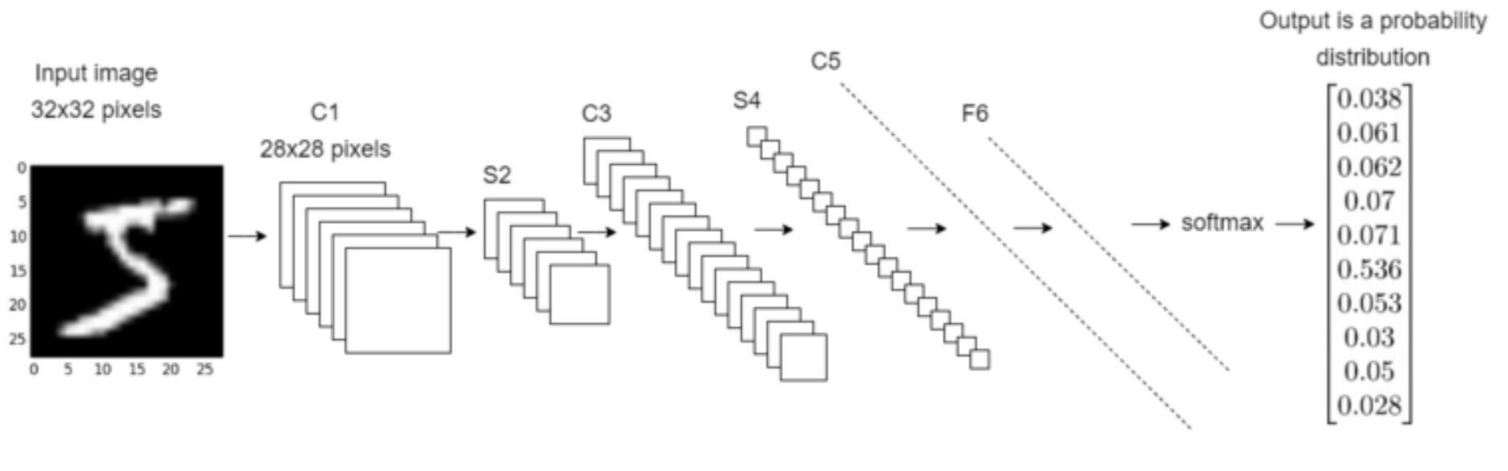
Application to Cyber Physical Systems

- Additive Manufacturing is a growing field that is very physical in nature.
- *Cyber* – Camera and Image Processing Software to detect defects in 3d printing as a proof of concept to larger applications.
- *Physical* - Manufacturing is a physical process focused on creating objects from other materials.
- *Cyber-Physical* Application is the analysis of the physical object using a computer based Machine Learning Model.

What is a Convolution Neural Network?

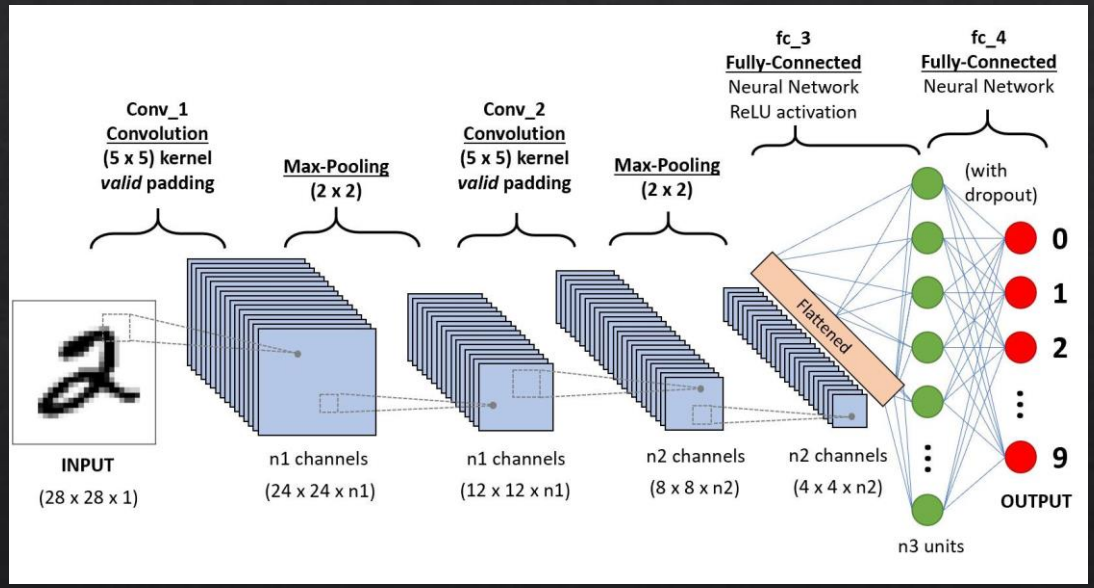
- Deep Neural Network combined with Convolution.
- Uses filters to extract features from images (maintaining positional data)
- Convolutional neural networks are very good at picking up on patterns in the input image, such as lines, gradients, circles, or even eyes and faces.
- Feed forward networks with many layers, specifically with a convolutional layer which mimics the humans eyes ability to see patterns in images





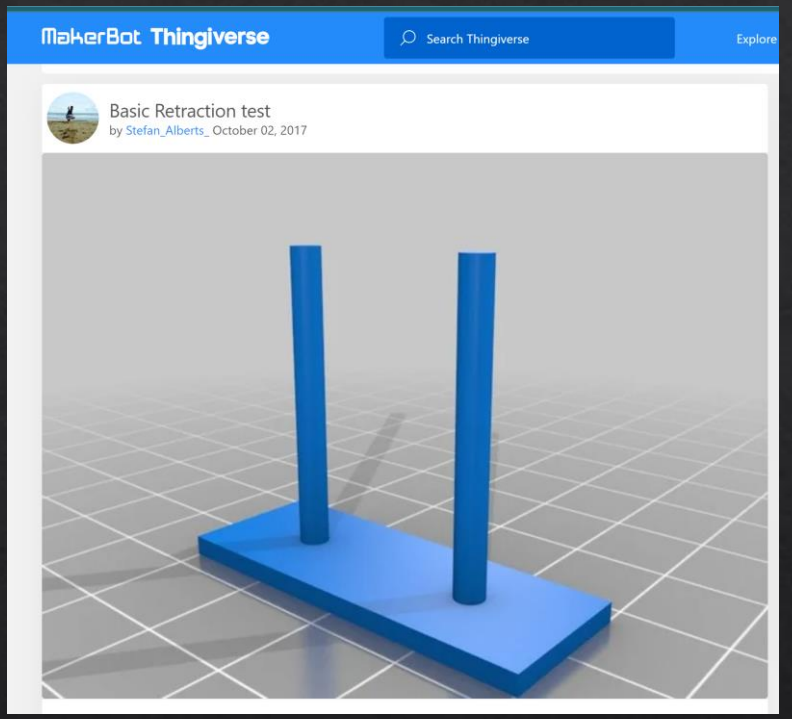
Layers

- Convolution- Multiplies each number by each other number and adds them together to get a summarized version of a larger image.
- Pooling – Takes the highest value of each region to form a new matrix with only those values. (Reduces the spatial dimension to increase speed)
- Dropout – Forces a neural network to disable some neurons in the learning phase to increase accuracy
- ReLU Activation – Normalize the values obtained
- SoftMax – turns the real values for weights into probabilities



Technical Specifications

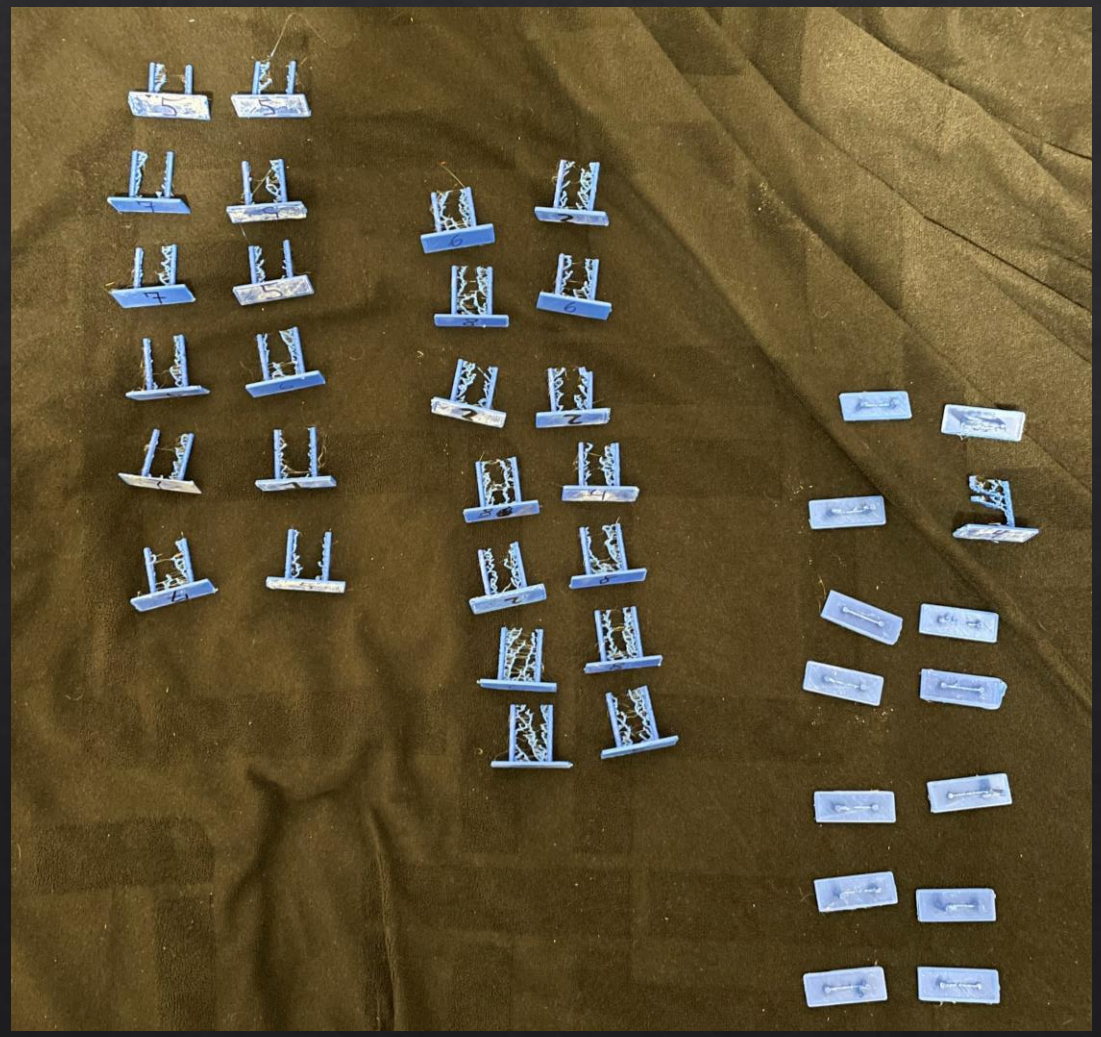
- Models Created using Crealty Ender3 3D Printer.
- Retraction Test Design: https://www.thingiverse.com/thing:2563909
- Cura Slicer Software
  - Print Time: 30-45 minutes (depends on print speed parameter)
  - Print Filament: CCTREE 1.75mm PLA (2g per print)
- Python3 using TensorFlow with Keras for CNN




```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten, Conv2D, MaxPooling2D
import pickle
from keras.models import model_from_json
from keras.models import load_model
import matplotlib.pyplot as plt
```

Acquiring the data

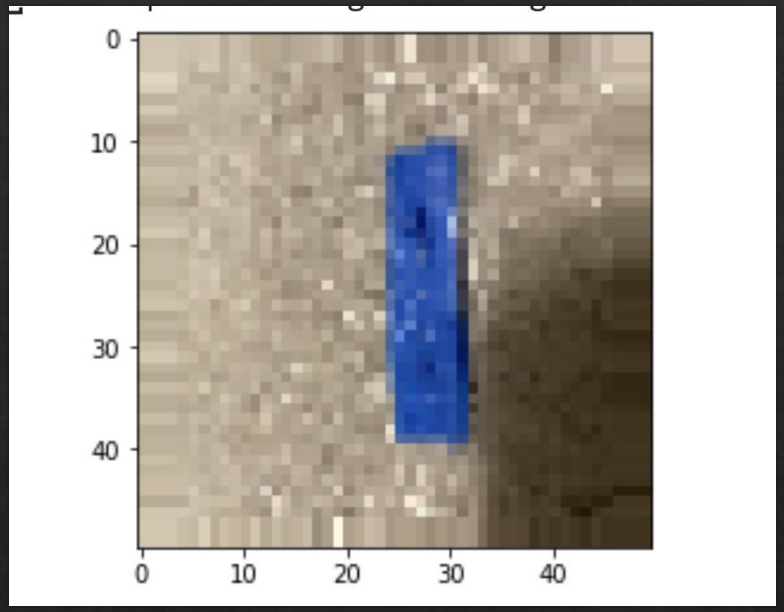
- I used a combination of changes to the printing speed, layer height, and retraction settings to print roughly 40 models, which I have photographed both sides as an input into the Neural Network



Model Number	Max Resolution (mm)	Layer height (mm)	Print Speed (mm/s)	Retraction (mm at mm/s)
1	0.5	0.12	50	5 at 50
2	0.5	0.12	50	None
3	0.5	0.12	30	5 at 50
4	0.5	0.12	30	None
5	0.5	0.2	50	5 at 50
6	0.5	0.2	50	None
7	0.5	0.2	30	5 at 50
8	0.5	0.2	30	None

Data Manipulation/Augmentation

Data augmentation help creates more and more models out of a single image by rotating, scaling, zooming, cropping, etc. to create more data points for better training.



```
# specify augmentation - zoom, flip, shift, etc. - here.
image_generator = tf.keras.preprocessing.image.ImageDataGenerator(
    rescale=1/255,
    zoom_range = 0.2,
    horizontal_flip=True,
    validation_split=0.2 # validation split is set here.
)
```

Creating the model

1. Convolutional layer to identify features
2. Pooling layer to reduce the size
3. Repeat 3 times to identify a good amount of features
4. Flatten the model and create a Dense Layer
5. Then compress the layer into 3 neurons
6. Use a SoftMax function for the output to transform into a probability distribution

```
# Building the model
model = Sequential()
# 3 convolutional layers
model.add(Conv2D(32, (3, 3), input_shape=(50, 50, 3)))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

# 2 hidden layers
model.add(Flatten())
model.add(Dense(128))
model.add(Activation("relu"))

model.add(Dense(128))
model.add(Activation("relu"))

# The output layer with 3 neurons, for 3 classes
model.add(Dense(3))
model.add(Activation("softmax"))
```

Initial Training

- Purpose of training: Assign weights to each of the branches and adjust the weight based on the accuracy of each input to output.
- Settings used initially:
  - 40 Epochs,
  - 8 batch size
  - 80/20 Validation Split to train on 80% of the data and evaluate of on 20%

```
# Training the model, with 40 iterations
# validation_split corresponds to the percentage of images
# used for the validation phase compared to all the images
# history = model.fit(X, y, batch_size=32, epochs=40, validation_split=0.1)
batch_size, epochs = batch_size, 40
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // batch_size,
    validation_data = validation_generator,
    validation_steps = validation_generator.samples // batch_size,
    epochs=epochs,
    # validation split and shuffling is included.
)
```

```
Found 66 images belonging to 3 classes.
Found 14 images belonging to 3 classes.
```

What is Accuracy and What is Loss?

- Accuracy- Number of errors the model made on the data (percent of data classified correctly)
- Loss – Distance between the true value of the problem and the values predicted by the model.

What does this mean?

Low accuracy, high loss – large errors on a lot of data

Low accuracy, low loss, - little errors on a lot of data

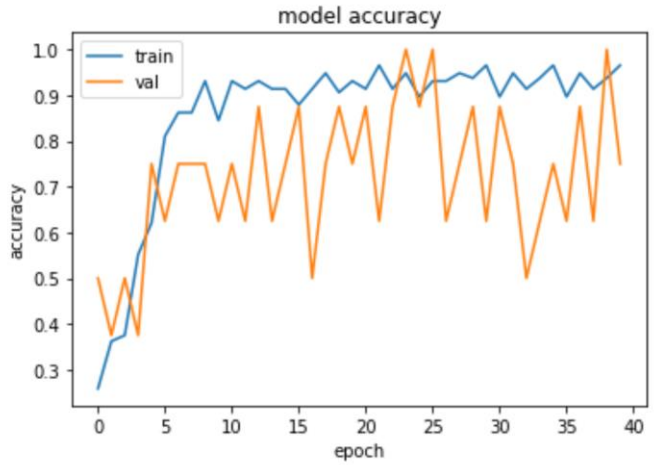
High accuracy, low loss – low errors on few data (BEST CASE)

High accuracy, high loss – large errors on a few data (MY CASE)

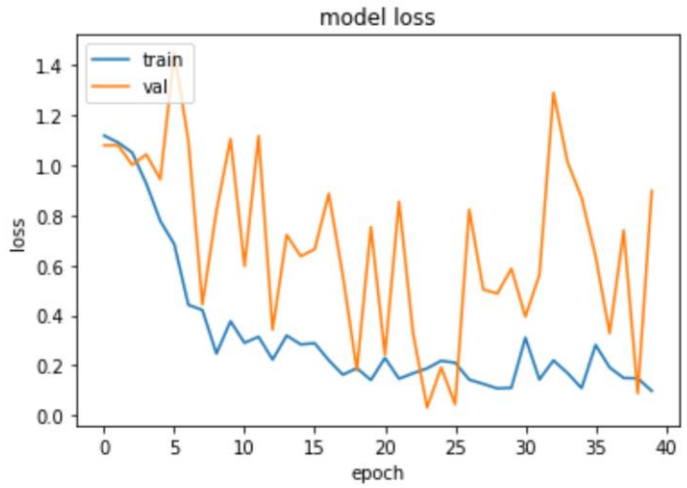
Loss is subjective as it depends on the problem. This project is classifying images between 0,1,2, so loss should be very low, but some project classify data between 0 and 255 so 0.5 loss would be acceptable

Accuracy and Loss (planning on trying to improve this)

```
# Plotting accuracy for yourself.
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```



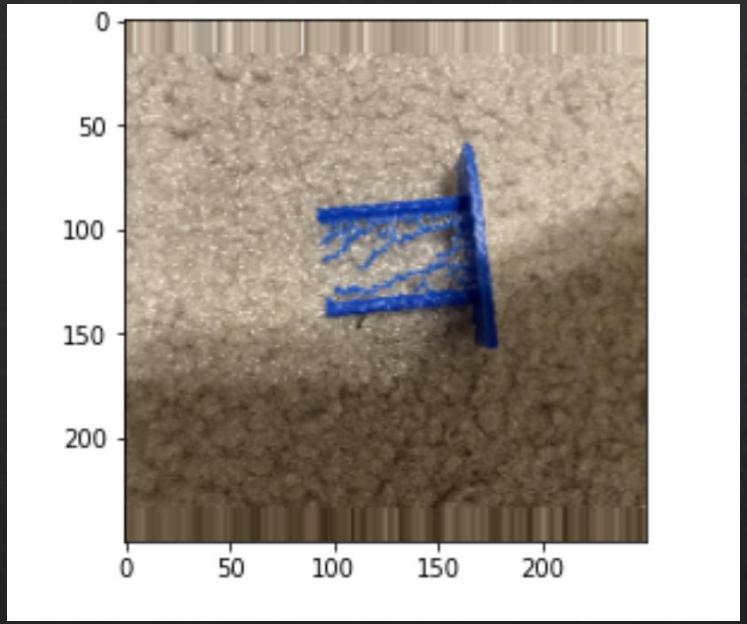

```
# Plotting loss is more important.
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```

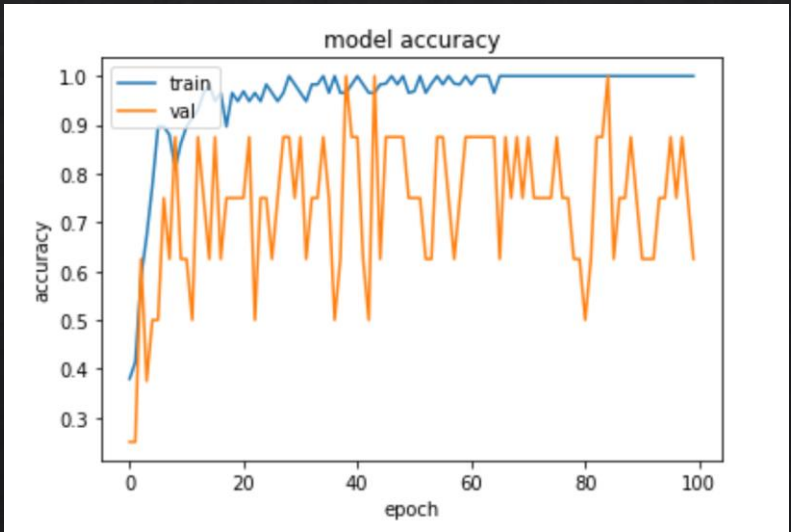


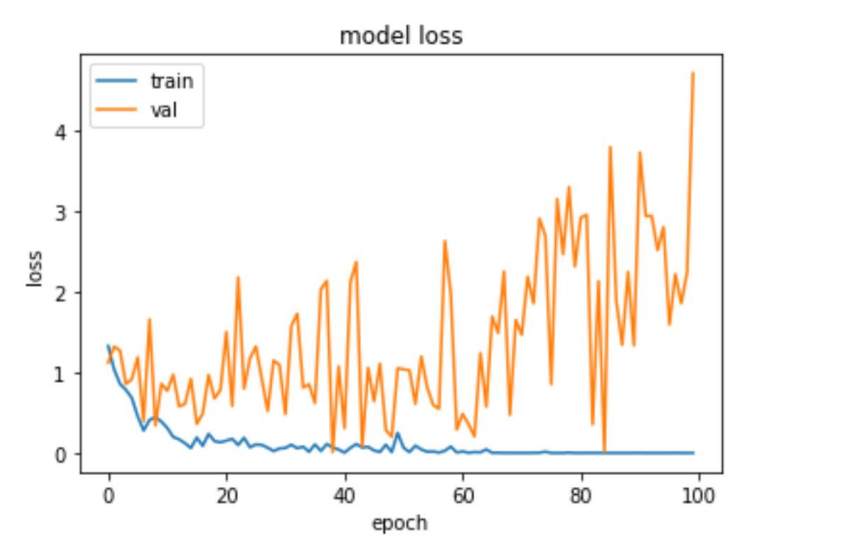

```
Epoch 1/40
8/8 [==============================] - 98s 10s/step - loss: 1.0788 - accuracy: 0.4331 - val_loss: 1.1268 - val_accuracy: 0.2500
Epoch 2/40
8/8 [==============================] - 14s 2s/step - loss: 1.0871 - accuracy: 0.5024 - val_loss: 1.0849 - val_accuracy: 0.5000
Epoch 3/40
8/8 [==============================] - 14s 2s/step - loss: 1.0766 - accuracy: 0.4039 - val_loss: 1.0897 - val_accuracy: 0.3750
```

Attempted Correction

- Increased Resolution to 250x250
- Increased epochs to 100 instead of 40.
- Led to overfitting







Predictions using the initial model

- I plan on running the model through a test and seeing how it performs on new data but I need to print some more models for testing or remove some from the current training set.
- Correctly identifies the failed print. (93.8%)
- Wrongly identifies clean and stringing prints.
  - Possible causes- small dataset, resolution of image (details are too fine)


```
test_generator = image_generator.flow_from_directory(
        '/content/drive/MyDrive/testData/',
        target_size=(50, 50),
        color_mode="rgb",
        shuffle = False,
        class_mode='sparse',
        batch_size=1)

filenames = test_generator.filenames
nb_samples = len(filenames)

predict = model.predict(test_generator,steps = nb_samples)
print(predict)
predict = model.predict_classes(test_generator)
print(predict)
```

```
Found 3 images belonging to 1 classes.
[[6.8435529e-10 3.1829923e-01 6.8170077e-01]
 [1.0460904e-01 9.2819132e-02 8.0257189e-01]
 [1.1480008e-02 4.9868431e-02 9.3865156e-01]]
```

Ways to improve model

Here are a few more approaches you can try to get to above 0.95:

- more aggressive data augmentation
- more aggressive dropout
- fine-tuning one more convolutional block (alongside greater regularization)
- Greater Pixel Resolution than 50x50

- ***Need more data***

Conclusion/Results

- I created a convolutional neural network to analyze defects within 3d printed models that was approximately 70% accurate.
- This would be a great tool to implement into a live machine where it could provide feedback to the printer to stop it after a failed print has been detected assuming the accuracy was increased. This would help reduce cost in the future.
- I also learned a bunch about how neural networks work and how they can apply to Cyber Physical Systems

Further Applications to CPS

- Possibility to include in a live camera monitoring system
- Could be extended to other devices besides 3d printers especially on a large scale
- It would help cost and efficiency
- A process similar to this could be implemented in any aspect where defects are needed to be identified for example in food processing or car manufacturing.
- This could also be used to optimize prints if the settings could be manipulated during build.
- The possibilities are endless
- CNNs are also used for facial recognition, Optical Character Recognition, and IOT Devices

Bibliography

[1] Jin, Z., Zhang, Z., Demir, K. and Gu, G., 2020. Machine Learning for Advanced Additive Manufacturing. Matter, 3(5), pp.1541-1556.

https://towardsdatascience.com/all-the-steps-to-build-your-first-image-classifier-with-code-cf244b015799

https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

https://colab.research.google.com/notebooks/io.ipynb#scrollTo=p2E4EKhCWEC5

https://courses.analyticsvidhya.com/courses/convolutional-neural-networks-cnn-from-scratch?utm_source=blog&utm_medium=build_image_classification_model_using_python_and_keras

https://datascience.stackexchange.com/questions/42599/what-is-the-relationship-between-the-accuracy-and-the-loss-in-deep-learning#:~:text=Accuracy%20can%20be%20seen%20as,on%20a%20lot%20of%20data

Appendix

- https://colab.research.google.com/drive/15lnxYv9uKvHlH52uKPbkO8n2y4V6rD20?usp=sharing
- https://drive.google.com/drive/folders/1BAE5OIu7aLTQ2nxsrvR7k7NerTCNiYBh?usp=sharing
- Link to Google Colab code and data folder I created