Weights & Biases Demo

Recyclable Material Classifier

January 8th, 2020

Casey Duncan

About Me

Education

Colorado School of Mines

MS - Mechanical Engineering, Robotics & Automation

Minor - Computer Science

- Main Focus:
    - Robot path planning
    - Robot perception
    - Robot human interaction
- Favorite Topics
    - Computer Vision
    - Reinforcement Learning for path planning

University of California, Santa Barbara

BS - Mechanical Engineering (2010 - 2014)

Professional Background

Raytheon Vision Systems

Manufacturing Engineer (2013 - Current)

- Main Responsibilities:
    - Write documentation
    - Train & Assist lab personnel
    - Product-to-Customer delivery process

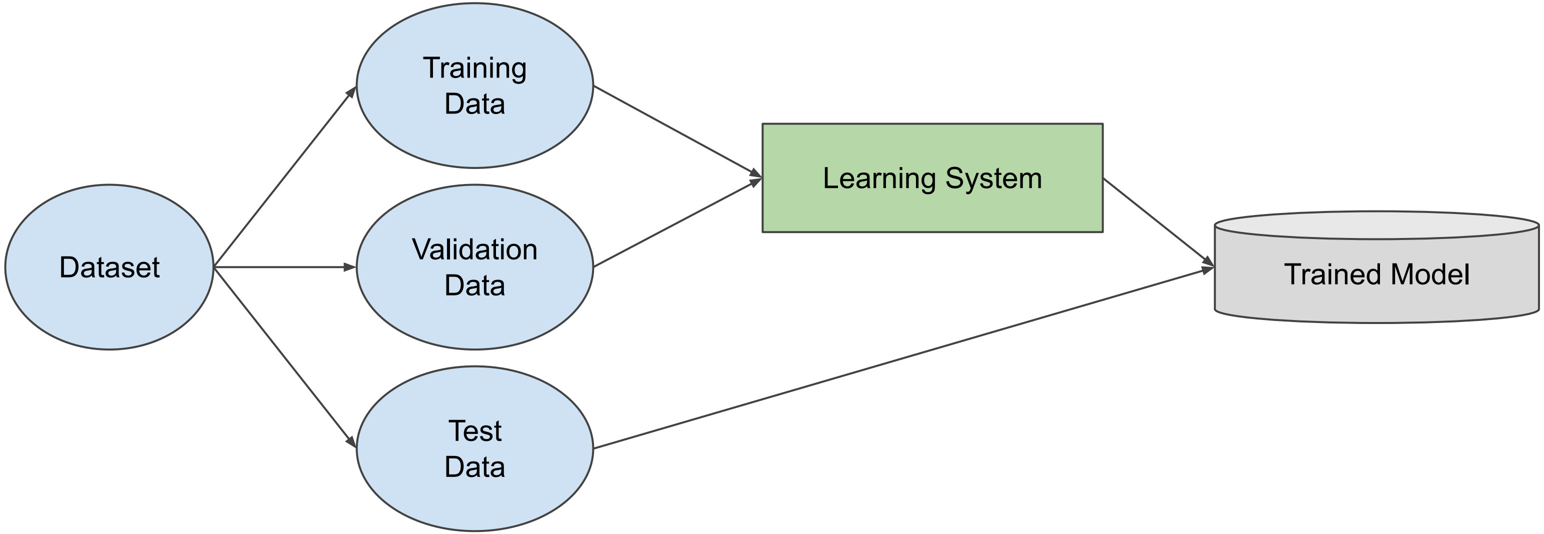
Overview

1. Machine Learning / Computer Vision
2. Project Description
3. ML Model Management Approach
   - Previous Approach
   - Weights & Biases
4. Benefits of Weights & Biases
5. Questions

Machine Learning

What is Machine Learning?

- Application of artificial intelligence (AI)
- Allows computer to learn & improve from experience.



Computer Vision

What is Computer Vision?

- Allows computers to gain a high-level understanding from digital photos & videos
- Modern Computer Vision utilizes Machine Learning to increase intelligence

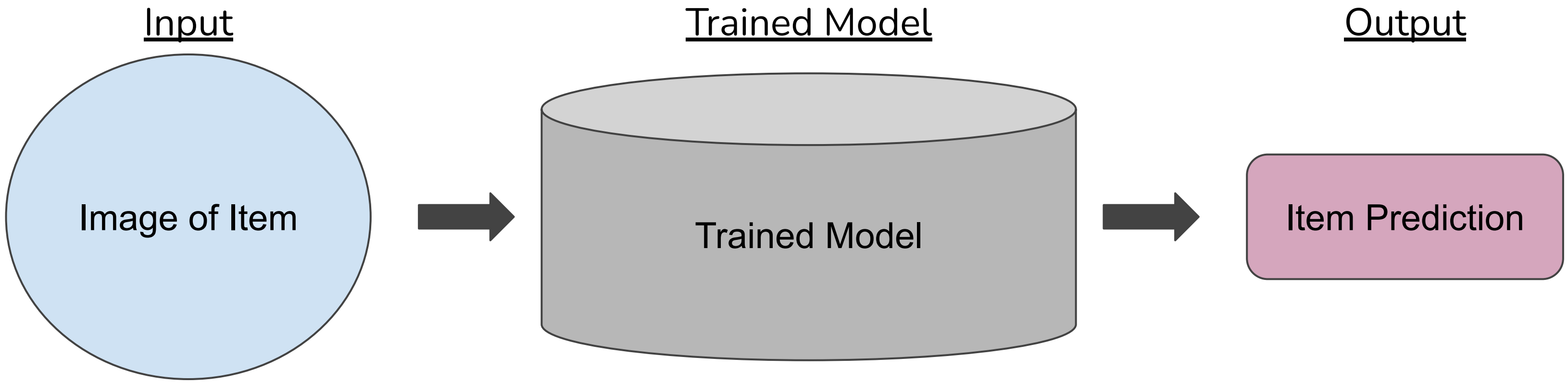
How is it used?

- Facial Recognition
- Autonomous Driving
- Agriculture Crop Selection

Image Classifier

What is an Image Classifier?

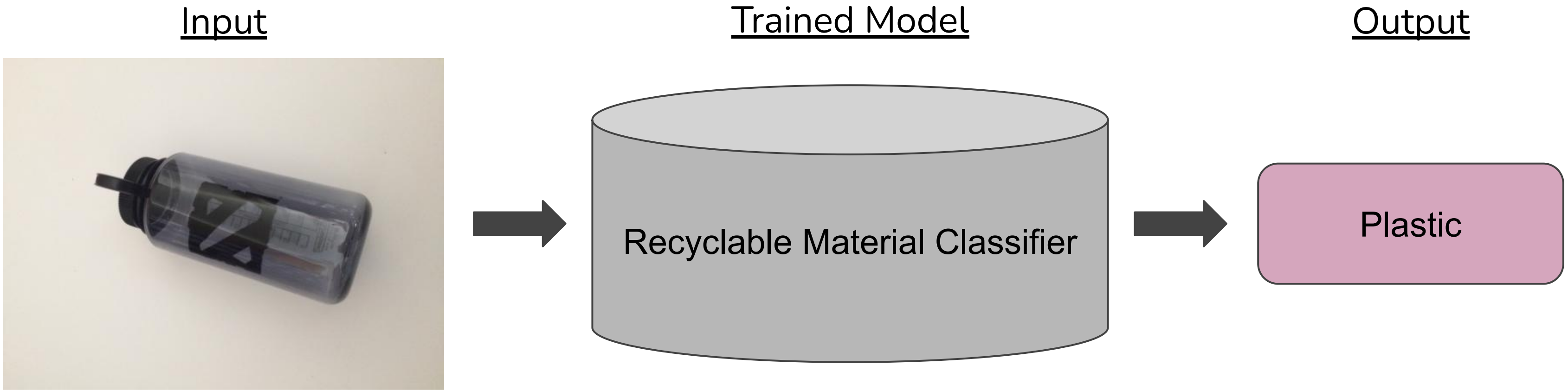
- Built using Machine Learning & Computer Vision
- **Input:** Image of an item
- **Output:** Prediction of what item is



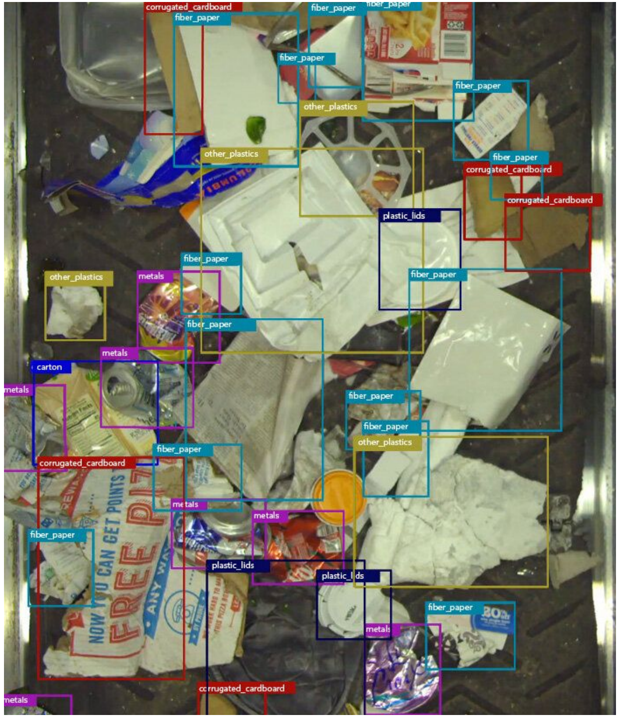
Recyclable Material Classifier

What is a Recyclable Material Classifier?

- **Input:** Image of Material
- **Output:** Material Type



Robotic Recyclable Sorter



Motivation

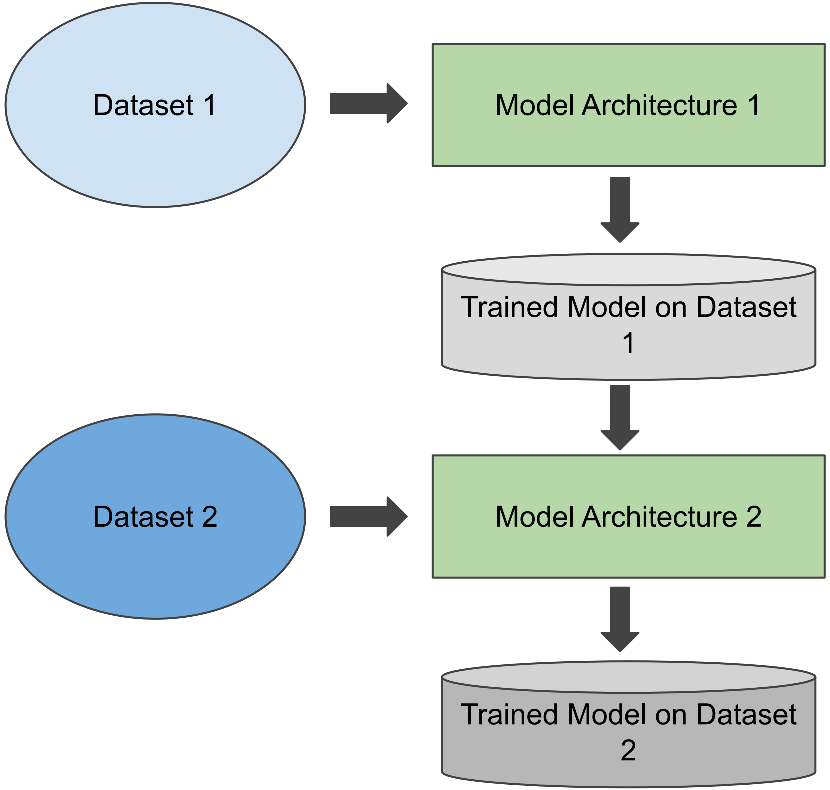
Why Recycling?

- Recycling is hard, even for people
  - Dynamic market, desirable materials constantly shifting
  - Best practices are confusing and locality dependent
- Contribute to the Circular Economy
  - Reduces waste in landfills
  - Reduces overall carbon footprint
- Critical Industry with exceptional returns[1]
  - Largest US operator: Waste Management, $42.2bn Market Cap
  - Up ~300% from 2012

[1] Macrotrends, Waste Management Net Worth 2006 -2019. https://www.macrotrends.net/stocks/charts/WM/waste-management/net-worth

Goal

- Train model with 90% prediction accuracy
  - Leverage transfer learning of proven successful classification models
  - Compare different model performance

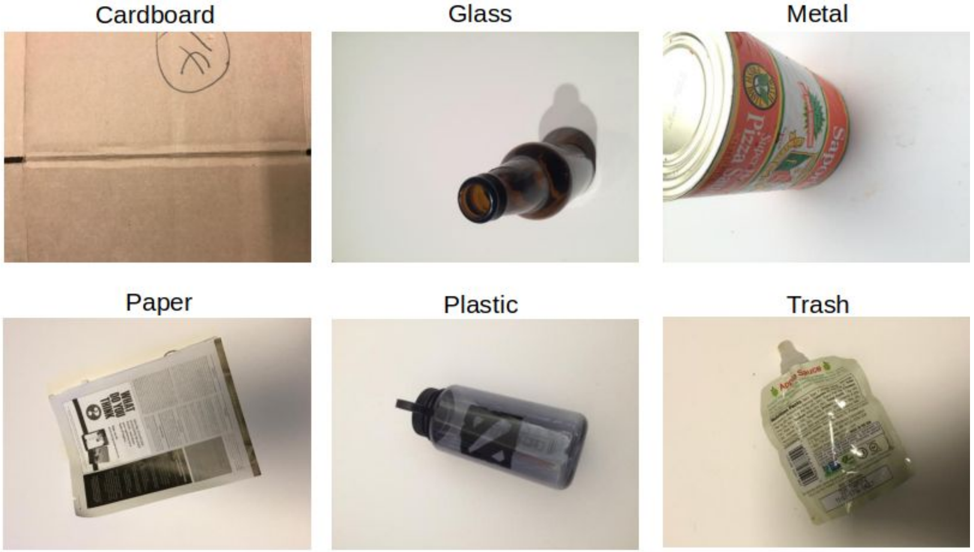


Dataset

Collect a dataset of images[2]

- 6 Classes
  - Paper (594 images)
  - Plastic (482 images)
  - Glass (501 images)
  - Metal (410 images)
  - Cardboard (403 images)
  - Trash (137 images)

- Total Number of Images: 2527 images



[2] Kaggle, Garbage Classification. https://www.kaggle.com/asdasdasasdas/garbage-classification

Model Optimization

What are Hyperparameters?

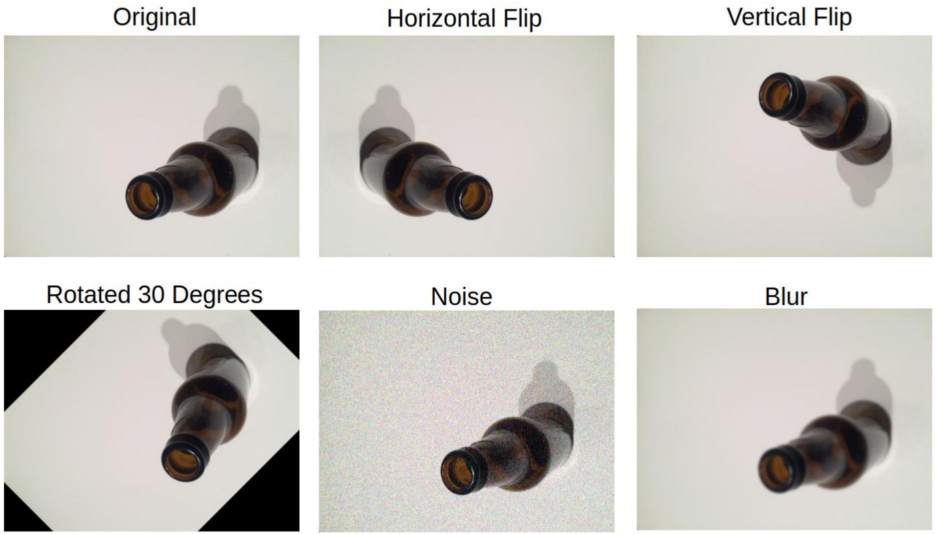
- Values used to control the learning process
- Defined prior to training

My Hyperparameters include:

- Data augmentation
- Train / Validation / Test Data Split Ratios
- Training Epoch Quantity
- Learning Rate
- Model Architecture

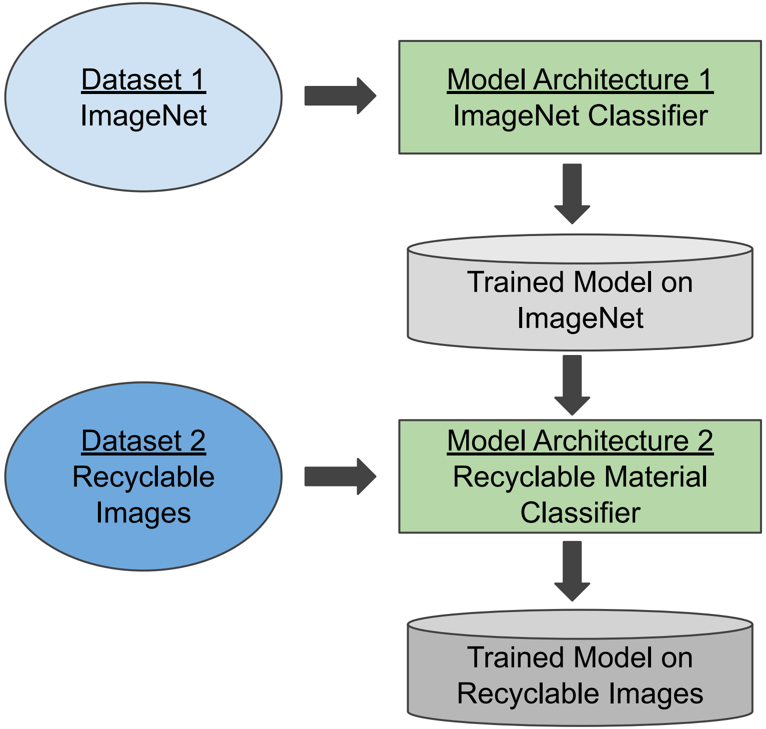
Hyperparameters
Data Augmentations

- Great for when not enough data to produce robust results
- Augmentation Types:
  - Random Horizontal Flip
  - Random Vertical Flip
  - Random Rotation (Up to ± 30°)
  - Random Artificial Noise
  - Random Gaussian Blur
- Any combination of augmentations can be chosen



Hyperparameters
Model Architectures

- Pretrained models
  - ResNet 18
  - ResNet 34
  - ResNet 50
- Customized Model
  - ResNet50-ModNet



Model Performance Reporting

Model Performance Metrics

- Training
  - Loss - penalty for a wrong prediction
- Validation
  - Material Accuracy per Epoch
  - Overall Accuracy per Epoch
- Test
  - Material Accuracy
  - Overall Accuracy
  - Confusion Matrix
    - Table displaying predictions vs. actuals
      - **Rows:** Predicted Material
      - **Columns:** Actual Material

Model Performance Reporting

Previous Approach

	A	B	C	D	E	F
1	Model file name:	resnet18_v1				
2						
3	Selected Transforms:					
4	1) Random Horizontal Flip					
5						
6	Training Ratio:	0.68				
7	Validation Ratio:	0.16				
8	Testing Ratio:	0.16				
9						
10	Epoch Quantity:	15				
11						
12	Confusion Matrix					
13	Cardboard	Glass	Metal	Paper	Plastic	Trash
14	57	0	3	1	2	1
15	0	18	29	0	33	2
16	1	1	62	0	2	1
17	3	0	10	54	4	14
18	0	0	11	0	73	1
19	0	0	2	0	0	19
20						
21	Class Accuracy					
22	Cardboard	Glass	Metal	Paper	Plastic	Trash
23	0.890625	0.219512195121951	0.925373134328358	0.635294117647059	0.858823529411765	0.904761904761905
24						
25	Overall Accuracy:	0.70049504950495				

Model Performance Reporting

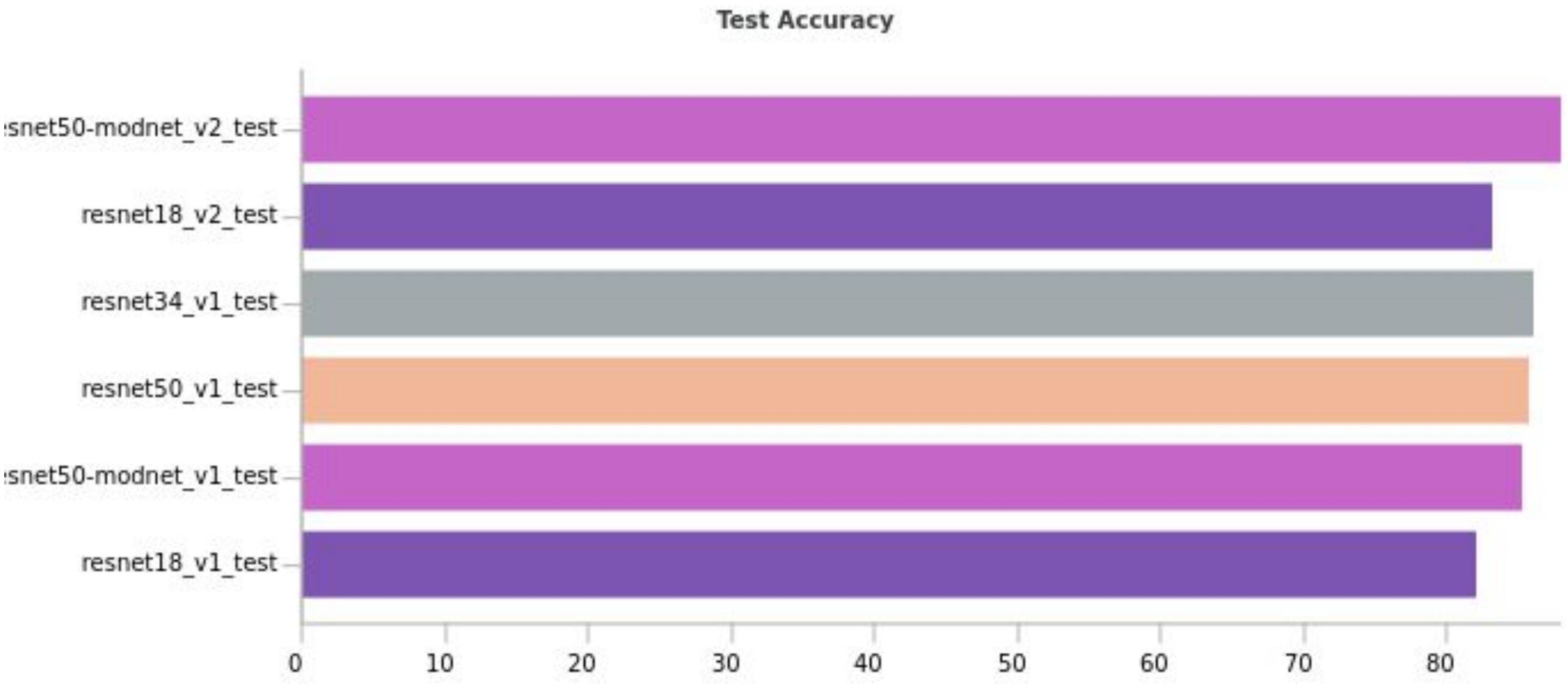
Weights & Biases - Logging

Logging performance using Weights & Biases is easy!

Example: Logging Test Accuracy

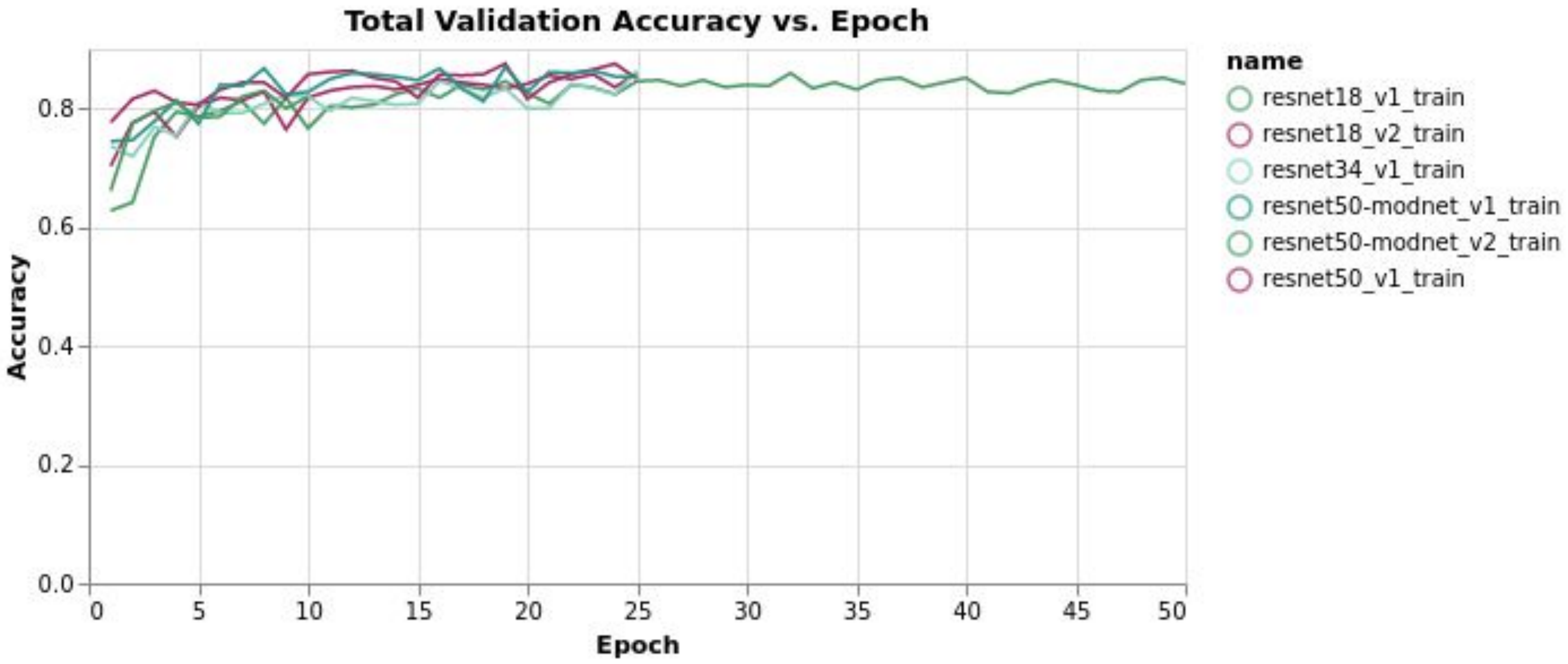
1. Initialize new job before Testing Model
    - ```
      run = wandb.init(project="Computer-Vision-Recyclable-Classifier",
                       name="resnet18_v1_test",
                       config=hyperparameters)
      ```
2. Test Model
3. Calculate Test Accuracy
    - Save to variable: `test_accuracy`
4. Log Test Accuracy in Weights & Biases Project
    - `wandb.log({"Test Accuracy": test_accuracy})`
5. Finish Run
    - `run.finish()`
      ```

Model Performance Reporting
Weights & Biases - Logging



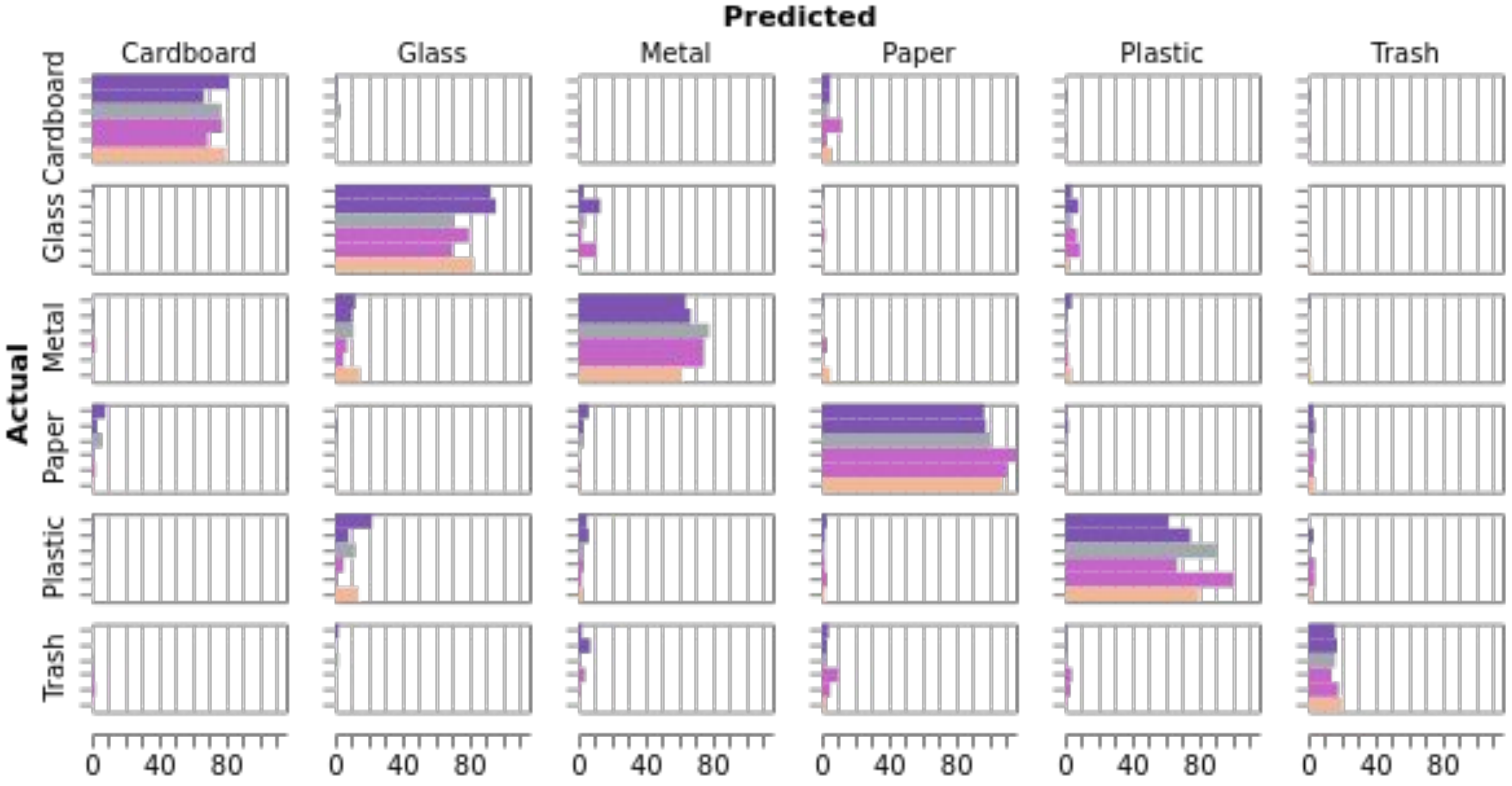
Model Performance Reporting

Weights & Biases - Logging



Model Performance Reporting

Weights & Biases - Logging



Saving Files

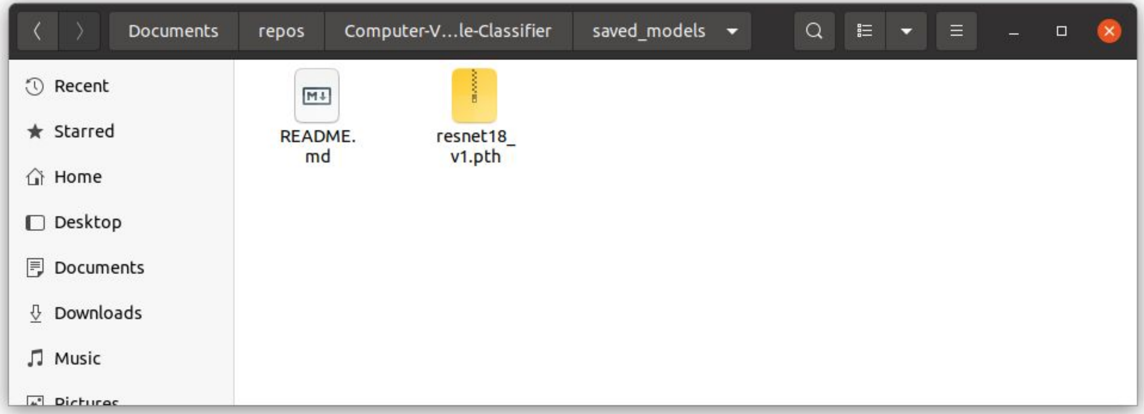
When training & testing new models, it is important to save:

- Model Files
- Data
  - Raw image data files
  - Training / Validation / Test data files

Saving Files
Previous Approach

Popular method for this is saving to local computer:

- Model Files
  - ```
    PATH = "/Computer-Vision-Recyclable-Classifier/saved_models/resnet18_v1.pth"
    torch.save(model.state_dict(), PATH)
    ```



- Saving Training / Validation / Test data
  - Did not save
    ```

Saving Files
Weights & Biases - Artifacts

Weights & Biases ‘Artifacts’ feature:

- Saves important files when training & testing new models. For example:
  - Models
  - Raw Data
  - Data after splitting into Training / Validation / Test
- Creates diagram for how files all tie together

Saving Files
Weights & Biases - Artifacts

Saving File
Example: Saving Model

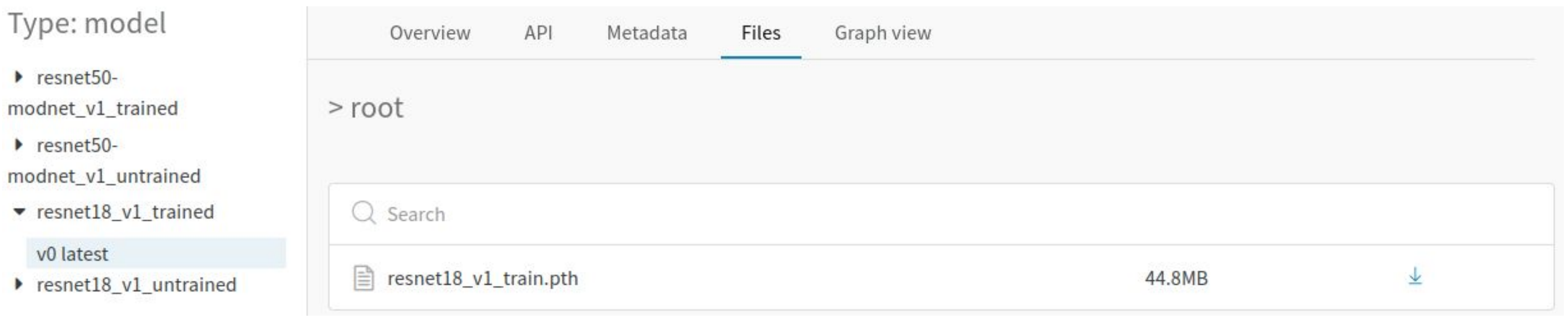
1. Initialize new job before Training Model
    - ```
      run = wandb.init(project="Computer-Vision-Recyclable-Classifier",
                       name="resnet18_v1_train",
                       config=hyperparameters)
      ```

2. Train Model
3. Saved to Trained Model file on local computer
4. Log Model Artifact in Weights & Biases Project
    - ```
      trained_model_artifact = wandb.Artifact("resnet18_v1_trained",
                                              type="model",
                                              metadata=dict(wandb.config))
      trained_model_artifact.add_file(PATH)
      run.log_artifact(trained_model_artifact)
      ```

5. Finish Run
    - ```
      run.finish()
      ```
      ```

Saving Files

Weights & Biases - Artifacts



Saving Files

Weights & Biases - Artifacts

Type: model

- resnet18_v1_trained
  - v0 latest
- resnet18_v1_untrained

Overview | API | Metadata | Files | Graph view

Metadata

Search | Page 1 of 1

Name	Value
architecture	resnet18
batch_size	8
epoch_qty	10
learn_rate	0.001
test_ratio	0.2
train_ratio	0.6
transform_blur	false
transform_horz	false
transform_noise	false
transform_rot	false

Saving Files

Weights & Biases - Artifacts

Type: split_data

- Test_Data_505
  - v0 latest
- Val_Data_505
- Train_Data_1517

Overview | API | Metadata | **Files** | Graph view

> root

Search

Cardboard /	1.6MB	88 files
Glass /	1.3MB	102 files
Metal /	1.4MB	82 files
Paper /	2.6MB	115 files
Plastic /	1.3MB	93 files
Trash /	312.3KB	25 files

Saving Files

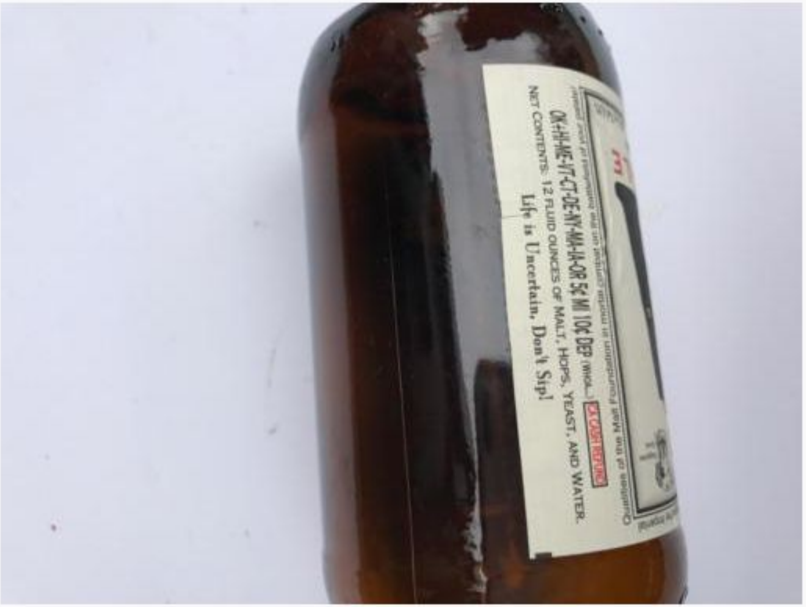
Weights & Biases - Artifacts

Type: split_data

- ▾ Test_Data_505
  - v0 latest
- ▸ Val_Data_505
- ▸ Train_Data_1517

Overview | API | Metadata | **Files** | Graph view

> root / Glass / glass107.jpg



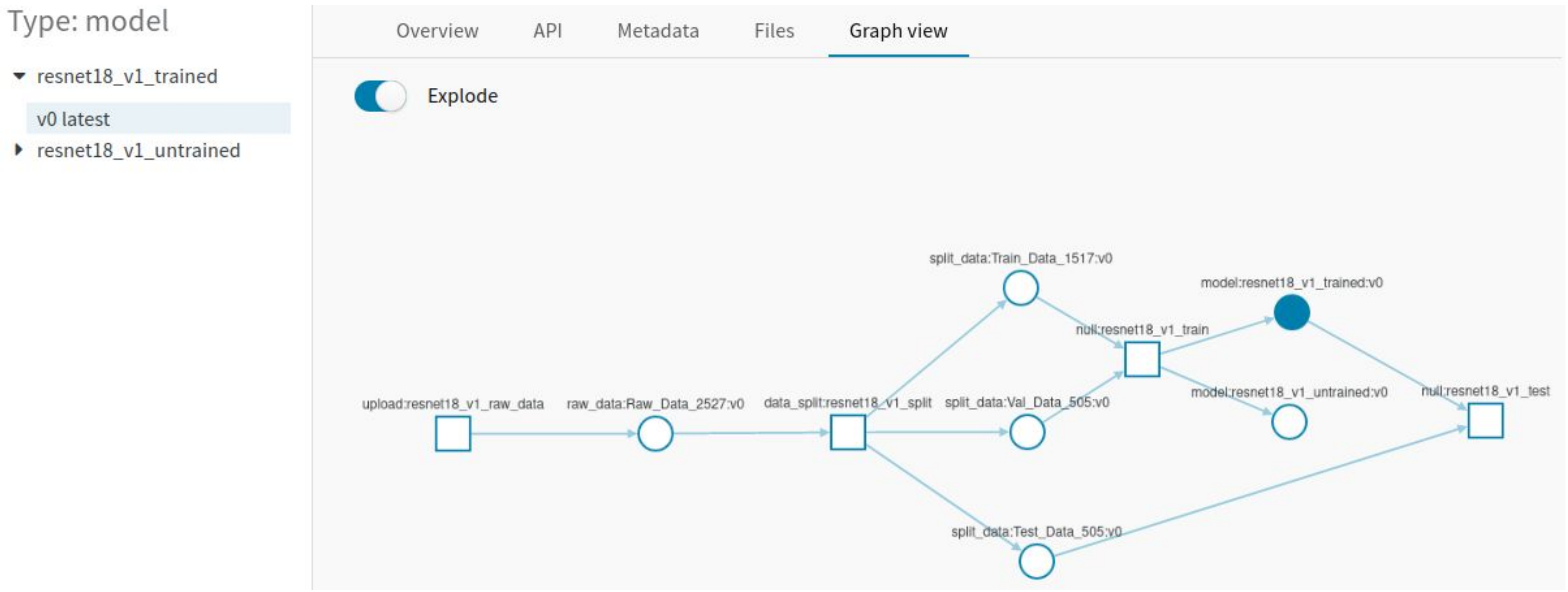
Saving Files
Weights & Biases - Artifacts

Create Diagram through ‘using Artifacts’

Example: Linking Train / Val Data to Model

1. Initialize new job before Training Model
   - ```
     run = wandb.init(project="Computer-Vision-Recyclable-Classifier",
                      name="resnet18_v1_train",
                      config=hyperparameters)
     ```
2. Use Training & Validation data Artifacts
   - ```
     run.use_artifact("Train_Data_180:latest")
     run.use_artifact("Val_Data_60:latest")
     ```
3. Train Model
4. Saved to Trained Model file on local computer
5. Log Model Artifact in Weights & Biases Project
6. Finish Run
   - ```
     run.finish()
     ```
     ```

Saving Files
Weights & Biases - Artifacts

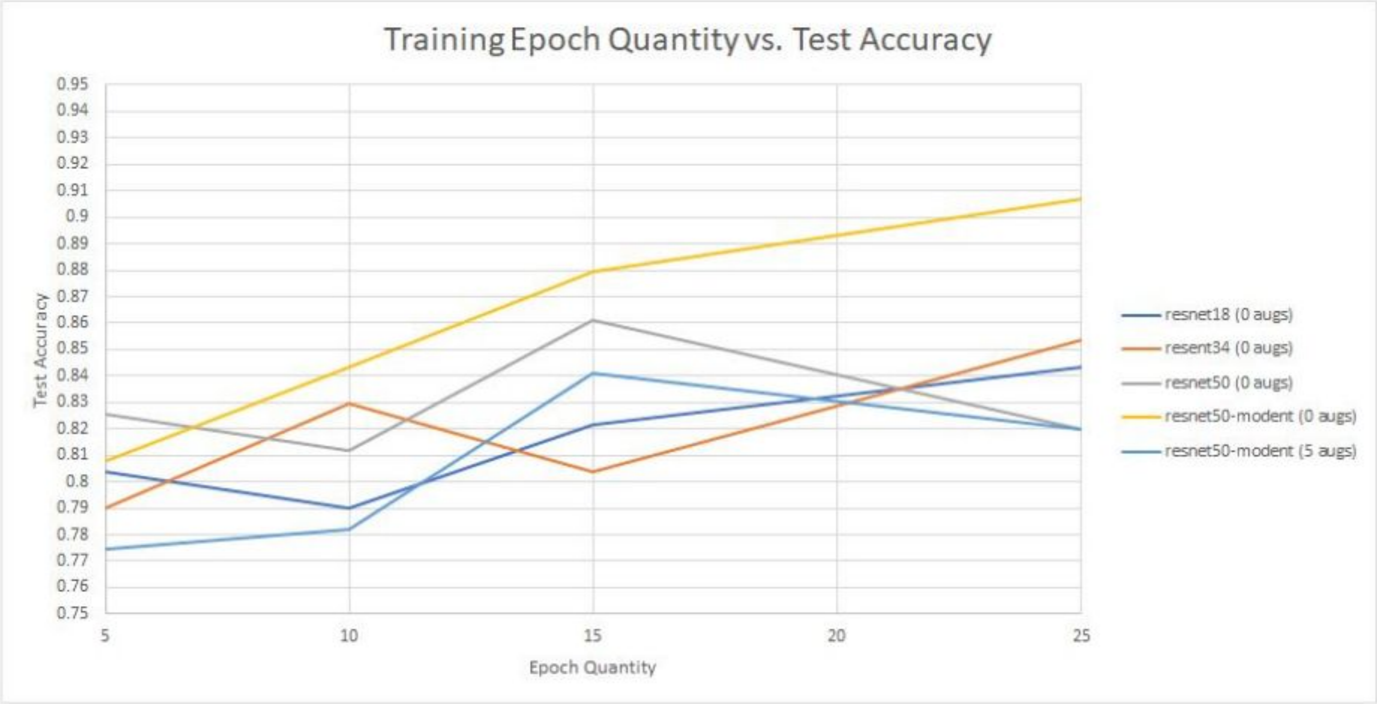


Model Optimization

- How is model performance optimized?
  - Choosing best hyperparameter values
  - There is an art to this due to:
    - Model Architecture
    - Raw Data
    - Project Type

Model Optimization
Previous Approach

- Used excel files to compare different training results



Model Optimization

Weights & Biases - Sweeps

Weights & Biases assists with Model Optimization through the ‘Sweeps’ feature.

1. Define dictionary of Hyperparameters Defaults at beginning of main file.
    - ```
      hyperparameters = dict(train_ratio = 0.6,
                             val_ratio = 0.2,
                             test_ratio = 0.2,
                             epoch_qty = 5,
                             learn_rate = 0.001,
                             transform_horz = False,
                             transform_vert = False,
                             transform_rot30 = False,
                             transform_blur = False,
                             transform_noise = False,
                             architecture = "resnet18")
      ```
    - Tailor code to:
        - Use Hyperparameter dictionary
        - Accept arguments from command line using hyperparameter variable names.
            - For example, the following should run:

              ```
              $ python main_wandb.py --epoch_qty=8 --transform_horz=True
              ```
              ```

Model Optimization

Weights & Biases - Sweeps

2. Create a YAML file (sweep.yaml) in base directory containing sweep parameters
   - Add metric Goal, Name, & Target
   - Change hyperparameter ranges

3. Initialize sweep
   - Run the following in the command line

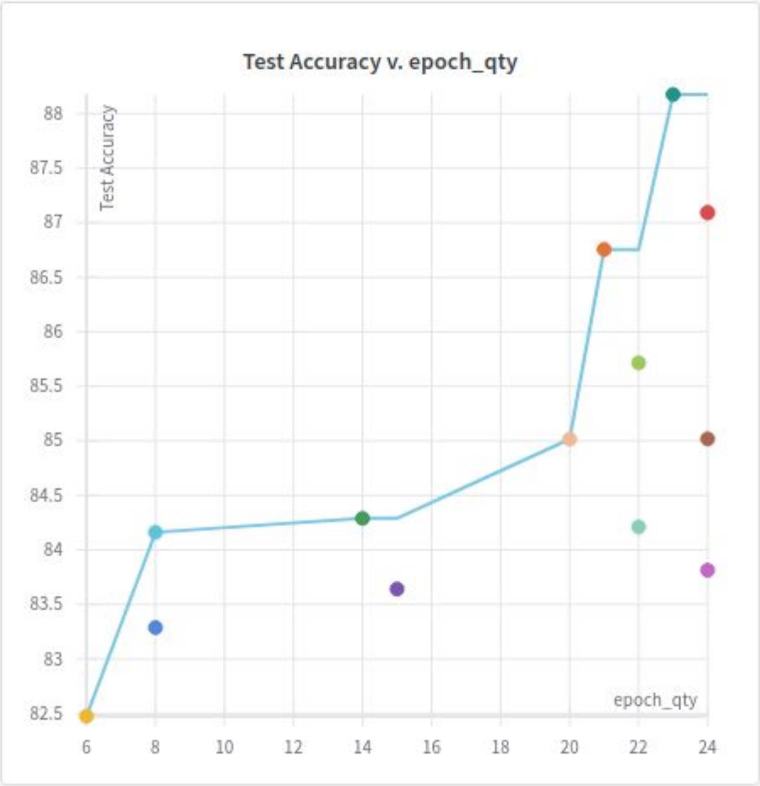
     ```
     $ wandb sweep sweep.yaml
     ```

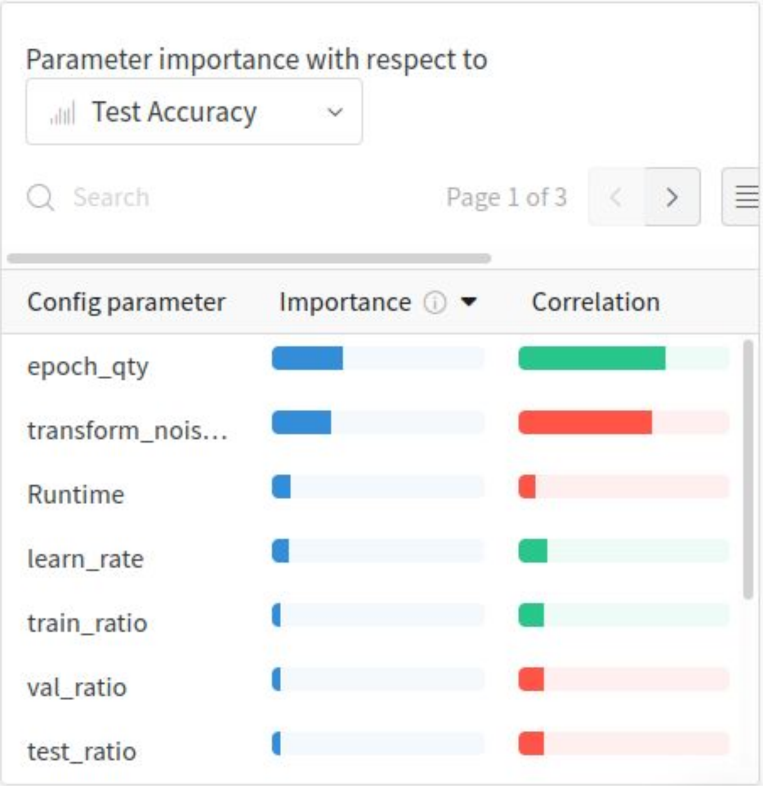
4. Run sweep
   - Run sweep command in the command line. Sweep command obtained from step 4.
     For example:

     ```
     $ wandb agent caseyduncan/recycling-classifier-demo/7x90t06x
     ```

Model Optimization

Weights & Biases - Sweeps





Using Plots

Weights & Biases - Reports

Weights & Biases ‘Reports’ feature lets users easily share model results.

- Simple to build report
- Easily shareable
- Similar to project dashboard

Example Report

Benefits to Weights & Biases

- Logging Metrics
  - Easily save all performance metrics to W&B project
    - No plotting required
    - Easy to compare different models
- Saving Files
  - Model files
  - Raw & Split Data
  - Creates diagram for how ‘Artifacts’ connect
- Optimizing Model
  - Intelligently tunes hyperparameters
  - Ranks hyperparameter importance
- Other:
  - No need to build own management tool
  - Easy to share with others
  - Easy to implement on existing projects

QUESTIONS?

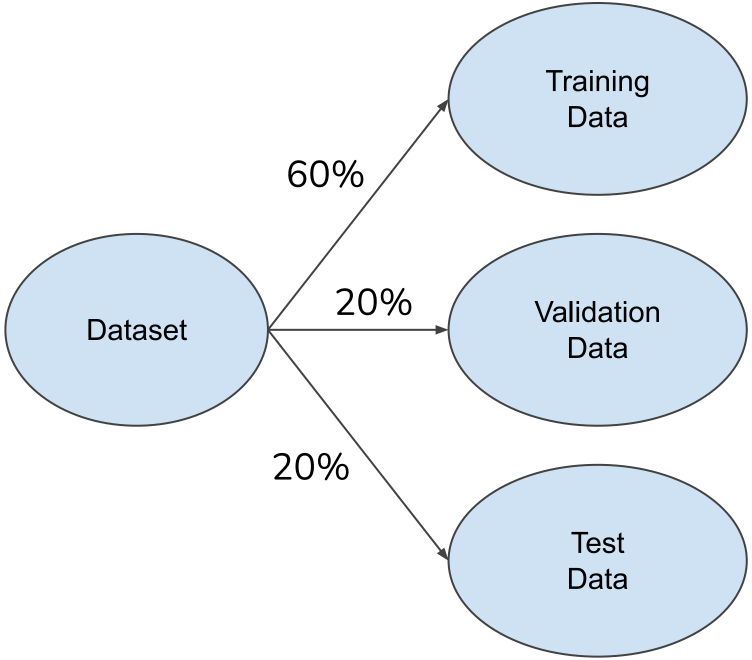
Background

Selected Toolset

- Python
  - Weights & Biases wandb
  - Pytorch
  - OpenCV
  - Various python modules for providing interfaces

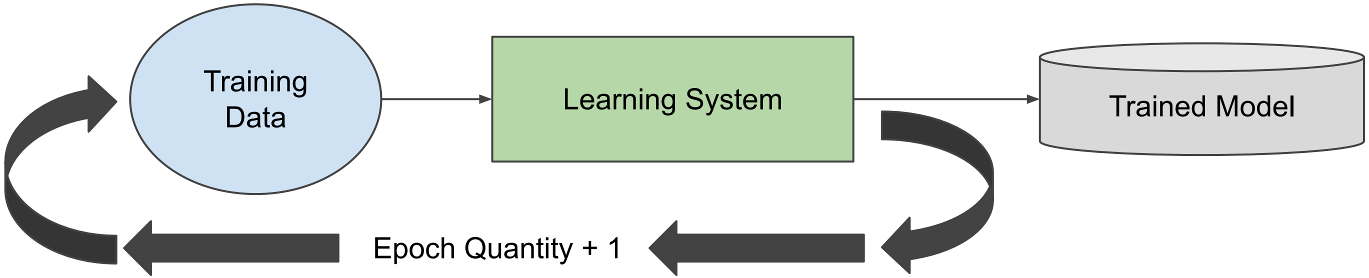
Hyperparameters
Data Split

- Data is split into training, validation, and test data
    - **Training** - used to train model
    - **Validation** - used to check model accuracy during training
    - **Test** - used to test model accuracy
- Augmentations are applied after split
    - Only applied to training data
    - Not applied to validation & testing
- Split Ratios for my project
    - **Training:** 60% - 80%
    - **Validation:** 10% - 20%
    - **Test:** 10% - 20%



Hyperparameters
Epoch Quantity

- The number of epochs defined governs how many times the model is trained using the training data.
  - If epoch quantity is 2, then the model is trained with the training data two times.
  - Training model with the training data only one time is typically not enough to achieve a high test accuracy

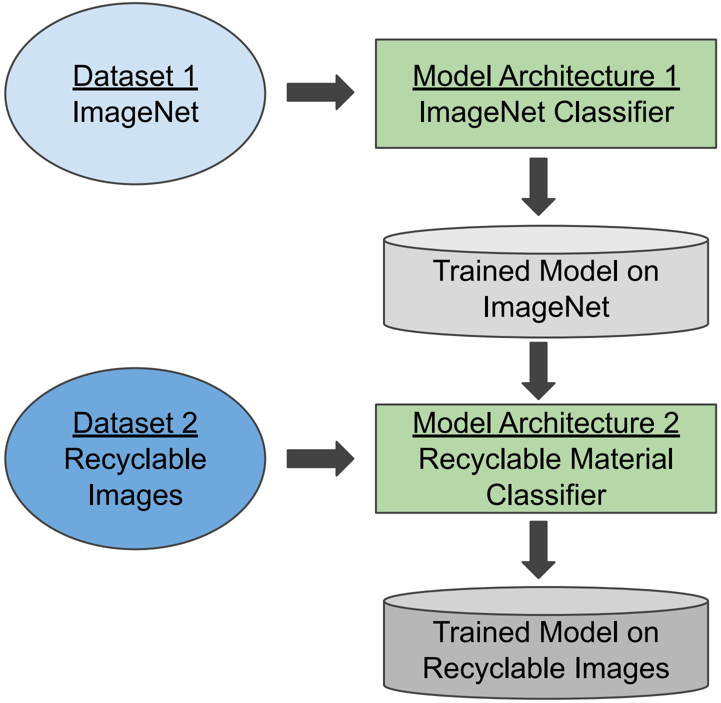


Hyperparameters
Learning Rate

- Learning Rate controls how much to change the model in response to the prediction error each time the model trained.
  - Too small of learning rate results in long training process that may get stuck learning.
  - Too large of learning rate results in unstable training process where optimal learned features (weights & biases) may be skipped.
- Example
  - You're blind & walking down mountain to find bottom of valley.

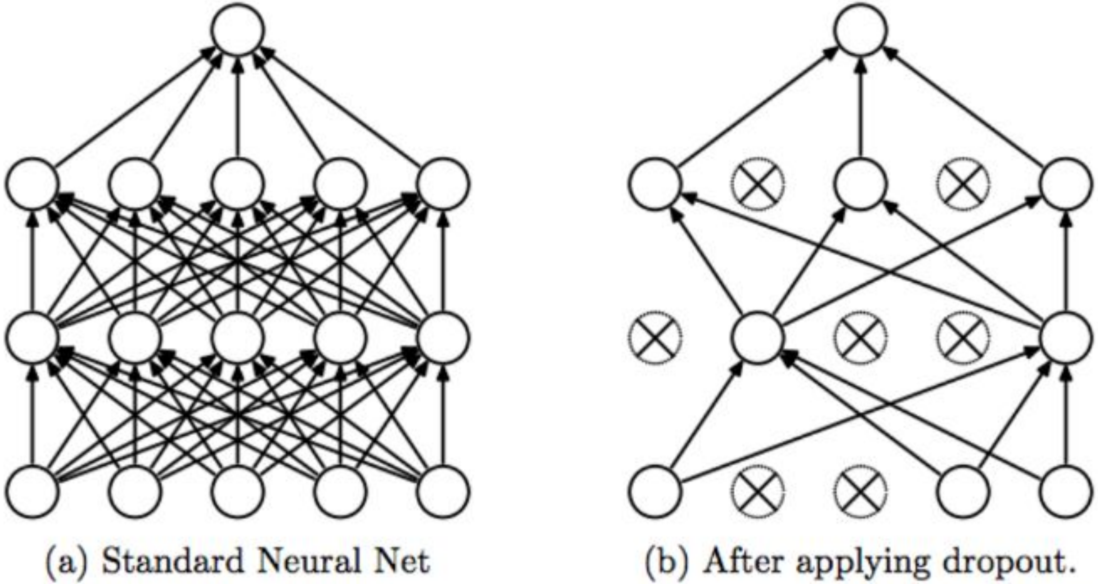
Hyperparameters
Model Architectures

- Leverage transfer learning
    - Pretrained models
        - ResNet 18, ResNet 34, ResNet 50
        - Freeze pre-trained layers to prevent back propagation
        - Define Fully Connected (FC) layer at end to learn the 6 classes
    - Customized Model
        - ResNet50-ModNet
            - Added ReLu, Dropout, FC, and LogSoftmax to last layer



Hyperparameters
Dropout

- Rate at which neurons in a layer are ignored
  - Great for preventing overfitting
  - Forces network to learn more robust features
  - Reduces training time



ImageNet Dataset

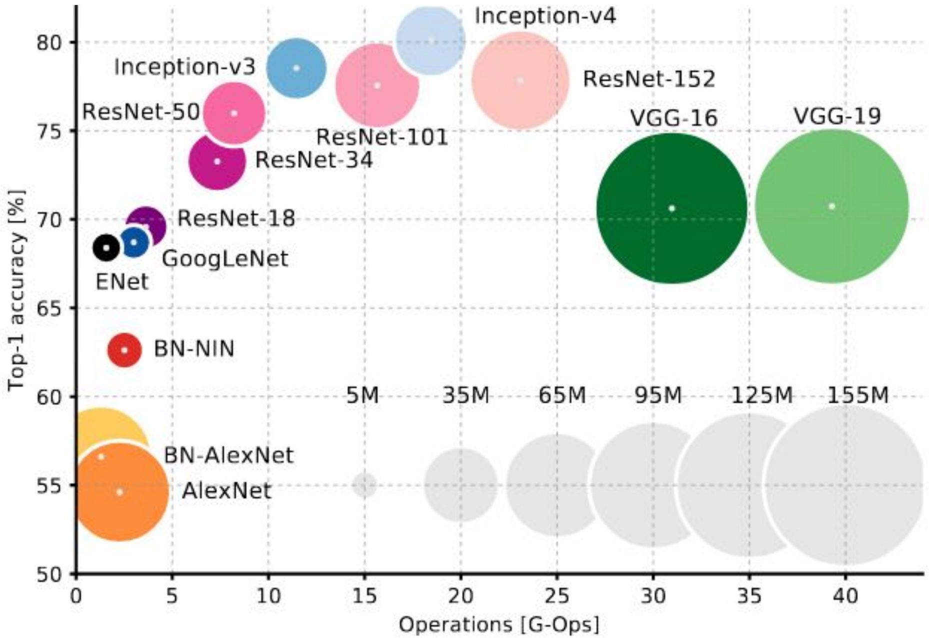
ImageNet[3] 2011 Fall Release (32326)

- plant, flora, plant life (4486)
- geological formation, formation (175)
- natural object (1112)
- sport, athletics (176)
- artifact, artefact (10504)
- fungus (308)
- person, individual, someone, somebody, mortal, soul (6978)
- animal, animate being, beast, brute, creature, fauna (3998)
- Misc (20400)

[3] ImageNet Dataset. http://www.image-net.org/

Previous Work

- Top-1 vs. Top-5 Accuracy Scoring
  - **Top-1:** ground truth matching the greatest prediction
  - **Top-5:** ground truth matching one of the top 5 predictions
- ResNet[4]
  - Implements Skip Connections
    - adds outputs from previous layers to outputs of stacked layers
  - Avoids vanishing / exploding gradient
- ResNeXt[5]
  - Same as ResNet but increases number of channels from one layer to the next)
- EfficientNet[6]
  - Top-1 Accuracy: 88.61%
- Effectiveness of Data Augmentation[7]



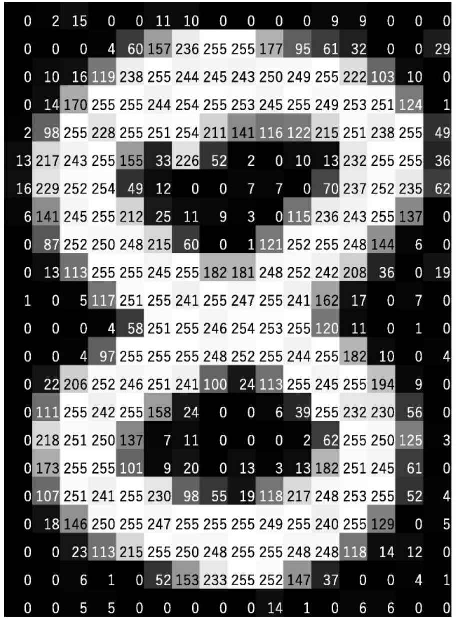
Deep Neural Network Comparison[8]
(The size of the blobs is proportional to the number of network parameters)

[4] Deep Residual Learning for Image Recognition. https://arxiv.org/abs/1512.03385
[5] Aggregated Residual Transformations for Deep Neural Networks. https://arxiv.org/abs/1611.05431
[6] EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. https://arxiv.org/pdf/1905.11946.pdf
[7] The Effectiveness of Data Augmentation in Image Classification using Deep Learning. https://arxiv.org/pdf/1712.04621.pdf?source=post_page
[8] An Analysis of Deep Neural Network Models for Practical Applications. https://arxiv.org/abs/1605.07678

Convolutional Neural Network (CNN)

Computer Interpretation of Image

- Image Size: 22 x 16
- Number of Pixels = 352
- Activation Number = Pixel Value

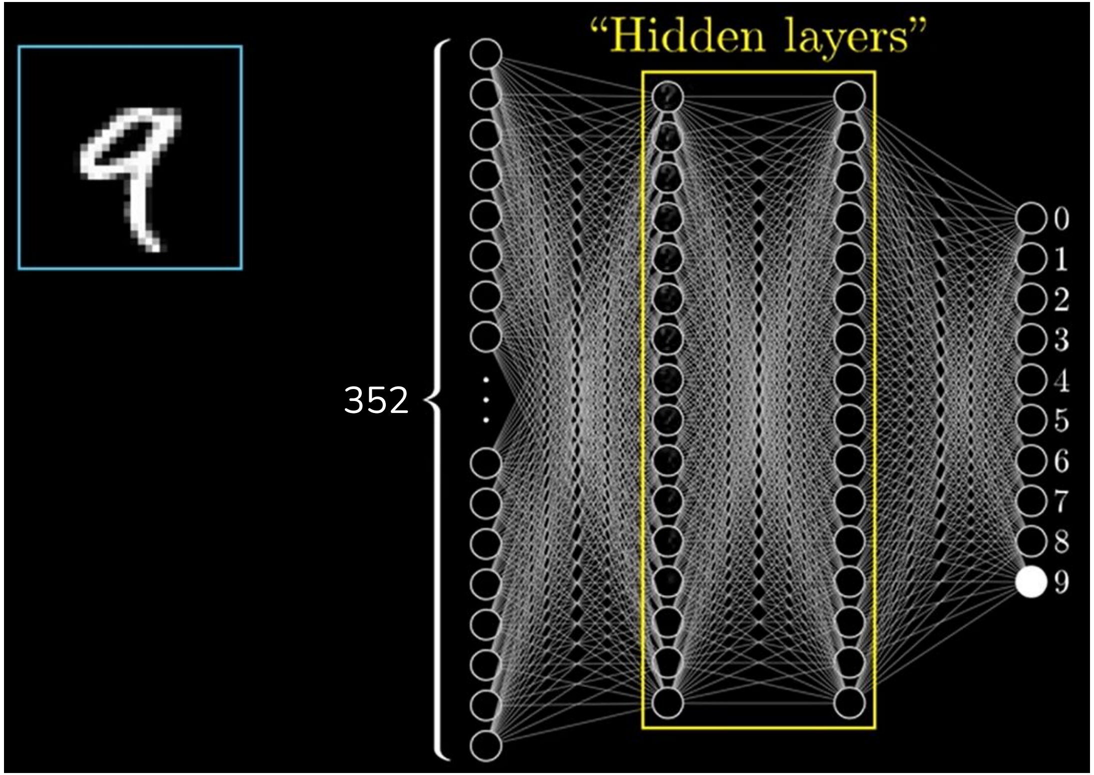


0	2	15	0	0	11	10	0	0	0	0	9	9	0	0	0
0	0	0	4	60	157	236	255	255	177	95	61	32	0	0	29
0	10	16	119	238	255	244	245	243	250	249	255	222	103	10	0
0	14	170	255	255	244	254	255	253	245	255	249	253	251	124	1
2	98	255	228	255	251	254	211	141	116	122	215	251	238	255	49
13	217	243	255	155	33	226	52	2	0	10	13	232	255	255	36
16	229	252	254	49	12	0	0	7	7	0	70	237	252	235	62
6	141	245	255	212	25	11	9	3	0	115	236	243	255	137	0
0	87	252	250	248	215	60	0	1	121	252	255	248	144	6	0
0	13	113	255	255	245	255	182	181	248	252	242	208	36	0	19
1	0	5	117	251	255	241	255	247	255	241	162	17	0	7	0
0	0	0	4	58	251	255	246	254	253	255	120	11	0	1	0
0	0	4	97	255	255	255	248	252	255	244	255	182	10	0	4
0	22	206	252	246	251	241	100	24	113	255	245	255	194	9	0
0	111	255	242	255	158	24	0	0	6	39	255	232	230	56	0
0	218	251	250	137	7	11	0	0	0	2	62	255	250	125	3
0	173	255	255	101	9	20	0	13	3	13	182	251	245	61	0
0	107	251	241	255	230	98	55	19	118	217	248	253	255	52	4
0	18	146	250	255	247	255	255	255	249	255	240	255	129	0	5
0	0	23	113	215	255	250	248	255	255	248	248	118	14	12	0
0	0	6	1	0	52	153	233	255	252	147	37	0	0	4	1
0	0	5	5	0	0	0	0	0	14	1	0	6	6	0	0

Convolutional Neural Network (CNN)

Neural Networks

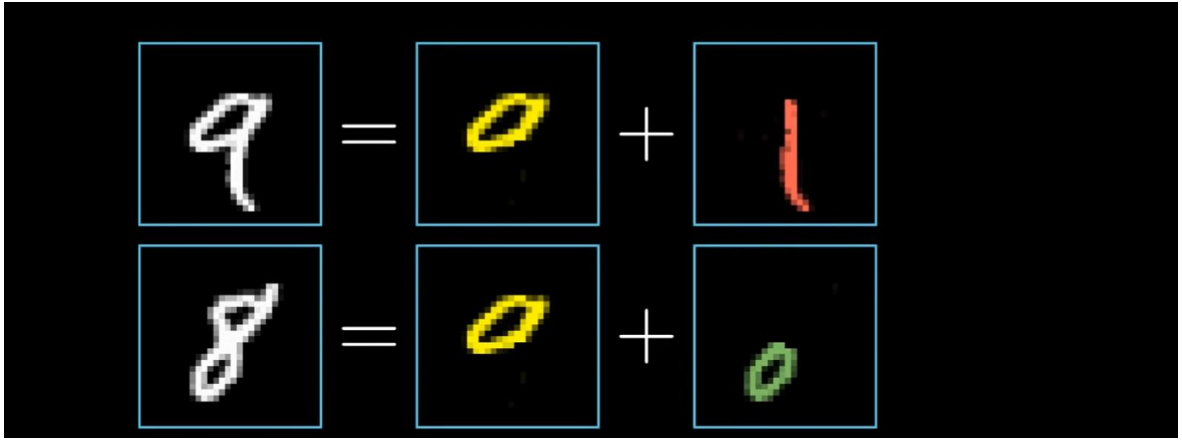
- First Layer
  - **Input:** 352 pixel values
- Last Layer
  - **Output:** 10 values corresponding to number in image
- Hidden Layers
  - 16 neurons per hidden layer
  - Each input neuron has a weight
  - Each neuron recognizes feature in image



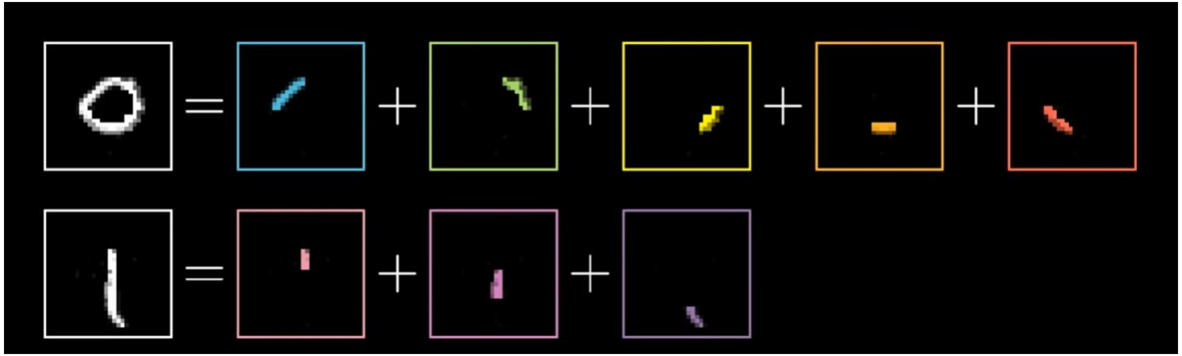
Convolutional Neural Network (CNN)

Neural Network - Hidden Layers

2nd Hidden Layer



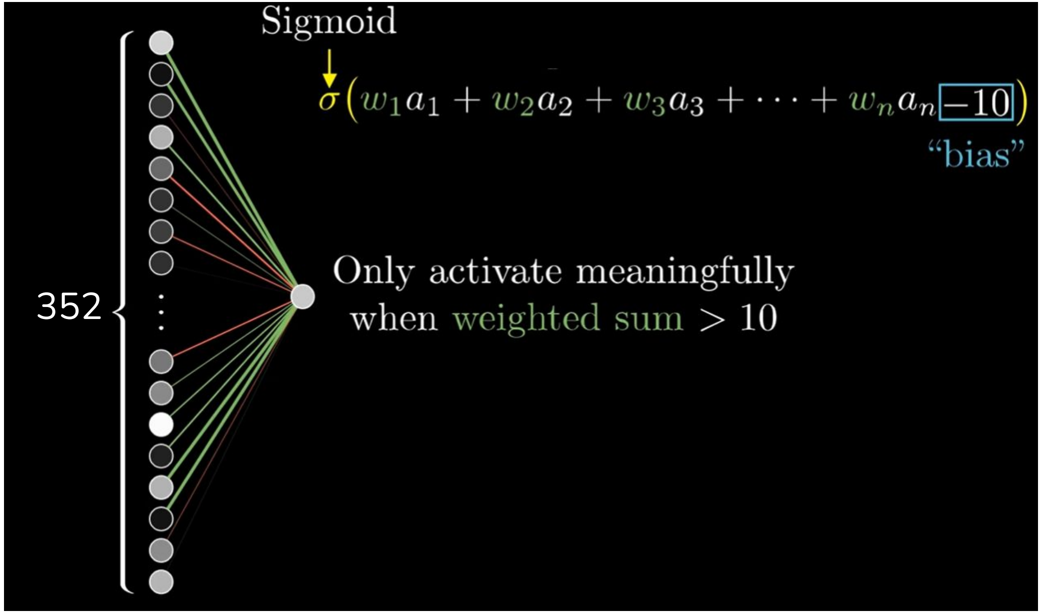
1st Hidden Layer



Convolutional Neural Network (CNN)

Neural Network - Weights & Biases

- Connection between each neuron has a **weight** (w_n)
- **Activations** (a_n) are multiplied by weights and summed
- Activation function is applied to squish weighted sum to between 0 and 1
- Each weighted sum is assigned a **bias** (b)
  - Neuron activates if weighted sum is larger than bias
- Learnable Values = 6090
  - Weights = 6048
    (352x16)+(16x16)+(16x10) = 6048
  - Biases = 42
    16 + 16 + 10 = 42



Convolutional Neural Network (CNN)

Neural Network - Back Propagation

1. Assign random value for weights (w_n) & biases (b)
2. Train on new image
3. Apply Cost Function to model output
    - Quantifies model predictions vs true value performance
4. Minimize Cost
    - This is called **gradient descent**
    - Step towards minimum is called **learning rate**
5. Change weight (w_n) & bias (b) values by negative gradient of cost function for each layer
6. Repeat steps 2 - 5

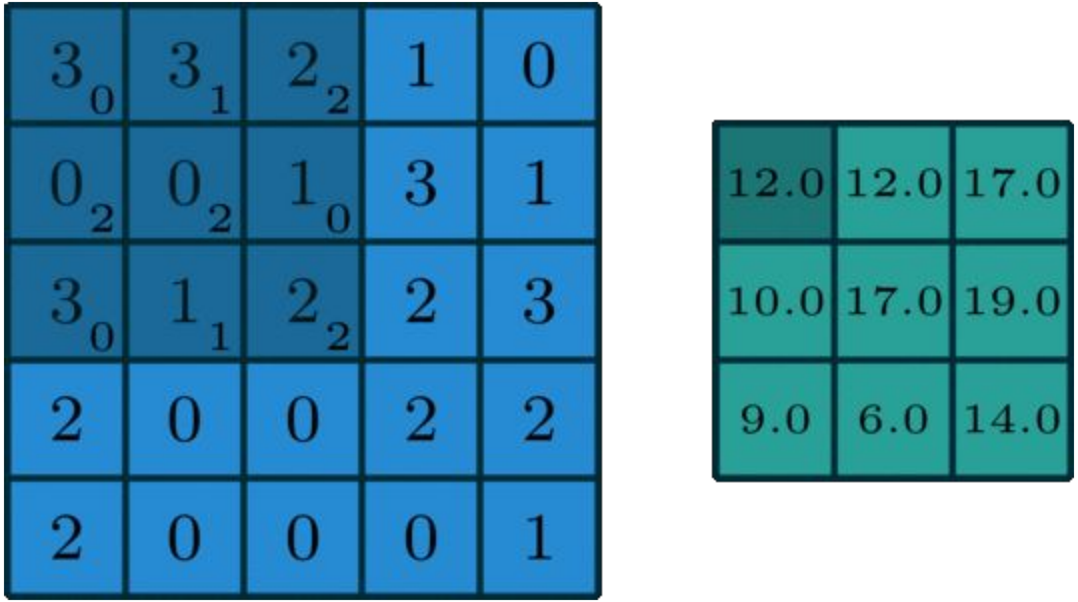
For more information, reference these videos.

Convolutional Neural Network (CNN)
Convolutions

1. Start with Kernel:

0	1	2
2	2	0
0	1	2

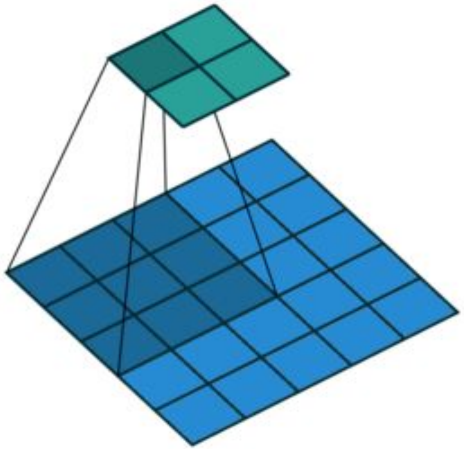
2. Slide along image pixel values
    a. Perform an element-wise multiplication
    b. Sum results
3. Place results into image of new pixel values



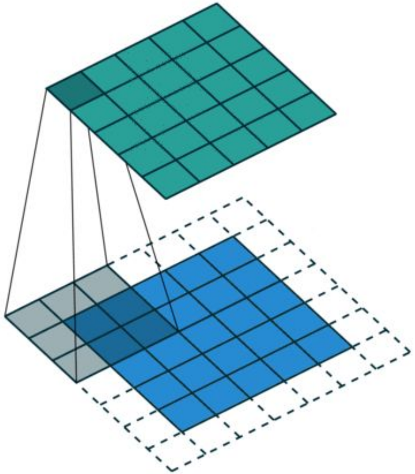
Convolutional Neural Network (CNN)

Convolutions

- Convolution Parameters
    - Kernel Values
    - Kernel Size
    - Padding
    - Stride
    - Max / Min Pooling
- More Information



Stride



Padding