Machine Learning Image Based Pest Species Identification For Embedded Systems

Joshua Bruce Mehlman
SFSU ENGR 859
SFSU ID: 903035606
JMehlman@sfsu.edu

***Abstract*—Pest species are problematic in a wide variety of environments from agriculture to vulnerable ecosystems, to personal gardens. Invasive species are often intermixed with native and beneficial species, often consuming resources and causing significant damage [1] [2] and are by definition difficult to dissuade [3].**

The environments that we most care about these species are difficult or prohibitively expensive to deploy significant computational hardware, from power considerations to weatherproofing, larger computational hardware presents challenges for field deploying. There are also often poor cellar service making a cloud based approach difficult, expensive, or impossible. Field deployed equipment must be protected from the elements and often relies on battery power. This equipment is also prone to being damaged, destroyed, and stolen. These constraints require the use of low powered low cost hardware.

The system was built by developing a Convolutional Neural Network based on LeNetV5 using PyTorch, then converting that model to Tensor Flow Light and installing it on a low cost, low power, Sony Spresense Microcontroller [4]

The model proved to be adept at predicting if target was empty. And if properly trained was fairly good at detecting the target squirrels, and distinguishing them from larger birds. However, the system was poor at predicting small birds at the feeder.
It is believed by improving the training the system can be made to be more than adequate for detecting a squirrel at a bird feeder. It is also not necessary to predict small birds, as they are not confused with the target species, the Eastern Fox Squirrel [8].

I. Introduction

There have been a few projects using machine learning to detect an invasive species, even with a squirrel as the target. However most of the work has been focused on cloud based computing [6] or using expensive and energy intensive hardware [5].

In our garden we have several bird feeders. These feeders can help offset the resource scarcity that is triggered by our increasing development for the native native bird populations. However, invasive squirrels (in my garden, the Eastern Fox Squirrel) will over consume the feed set out for the birds. Often a single animal will devour a weeks worth of bird feed in a single sitting. These pests are notoriously difficult to dissuade. Any garden store will be filled with failed gizmos to prevent squirrels from getting at the feed, and there are 1000s of books on the topic using a verity of techniques, even capsaicin [7].

We have made a low cost, field deployable system to target a specific invasive species, the Eastern Fox Squirrel [8], with a computer vision convolutional neural network (CNN) running on a low cost, low power microcontroller ‘Fig. 1’.

II. Materials And Methods

A. Outline

There are three separate phases in the pipeline ‘Fig. 2’. Model training, realtime, and post-processing. During the initial training phase there was a preprocessing image gathering. During subsequent training, the images were gathered during training runs. After a model is trained on a pre-acquired image set. The camera was placed in a fixed position relative to the bird feeder for each run. The images from the run are then manualy tagged and re-run through the model as validation to analyze and improve the models success rate.
The system has two streams of camera data available. Both were used:

- “Still” stream at QVGA (320x240), JPEG: File Save. Triggered on demand.
- ”Video” stream QVGA (320x240), YUV442: Model Processing. Callback at 5Hz
  - The library will only crop a YUV442 format image, Attempting to crop from RGB565 will fail.

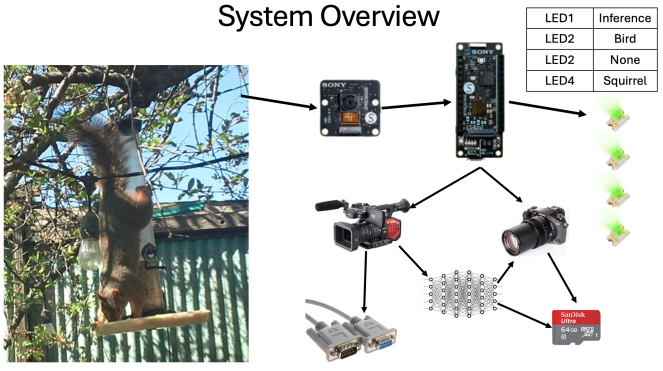

Fig. 1. System Diagram

The system registers a callback to execute at the set frequency with the video capture. This call back is set to the slowest speed available (5Hz). The realtime data-flow is as follows:

- Capture the still immediately on video callback.

- Center crop the video image to 96x96
- The cropped image is converted to RGB565
- The RGB image is converted to Float32 and normalized to 1.0
- The RGB656 is loaded to TFLite
- Inference is run
  - An LED is lit just prior to inference, and extinguished at completion
- Interprete the results as:
  - "Bird"
  - "Empty"
  - "Squirrel"
- Light up the LED corresponding to the inference category
- If "Bird" or "Squirrel" (Or while gathering training data)
  - Save the JPEG "still" to the SD card
- A log file is written to the SD card
  - inference results
  - image number (-1 for no save)
- Send to serial (config at build) either:
  - Results, and tallies
  - Cropped RGB565 image stream

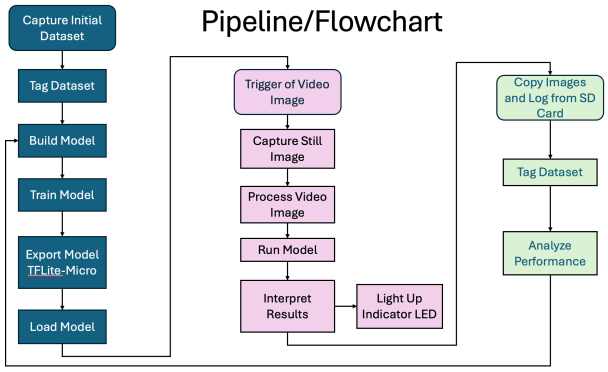

Fig. 2. Process Flow

B. Materials

The system is built around a Sony Spresense CXD5602 [4] Microcontroller. This microcontroller sits in a sweet spot of power, cost, and features. With a 6 core ARM Cortex M4F processor running at 156 MHz, and 1.5 MB of RAM, it has just enough capabilities to run the required image processing and Machine Learning model. Yet it is low cost, and low power. The board only consumes 100mW (A Raspberry Pi 3B uses over 2W) [4]. The total hardware cost was less than $200 'Table. I".

TABLE I
SYSTEM COST

Component	***Model***	***Price (USD)***
Development Board	Sony Spresense CXD5602	$65
Expansion Board	Sony Spresense Extension Board	$45
Memory Card	SanDisk 32GB Micro SD	$20
Camera	Sony Spresense Camera	$35
Tripod	Torjim Phone Tripod	$10
USB Cable	Micro USB A to Micro-B	$3

Microcontroller:

- ARM Cortex-M4F @ 156 MHz
- SRAM: 1.5MB
- Power: 100mW

Expansion Board:

- SD Memory Card Slot
- 32GB High Speed SD Card

Camera:

- Image Transfer: CMOS 8-Bit Parallel
- Control interface: I2C
- Sensor: 2,608x 1,960 (≈5M pixel)

```
==========================================================================================
Layer (type (var_name))  Input Shape          Output Shape         Param #          Trainable
==========================================================================================
leNetV5 (leNetV5)        [1, 2, 96, 96]       [1, 3]               --               True
├─Sequential (features)  [1, 2, 96, 96]       [1, 30, 22, 22]      --               True
│    └─Conv2d (0)        [1, 2, 96, 96]       [1, 12, 98, 98]      228              True
│    └─BatchNorm2d (1)   [1, 12, 98, 98]      [1, 12, 98, 98]      24               True
│    └─ReLU (2)          [1, 12, 98, 98]      [1, 12, 98, 98]      --               --
│    └─MaxPool2d (3)     [1, 12, 98, 98]      [1, 12, 49, 49]      --               --
│    └─Conv2d (4)        [1, 12, 49, 49]      [1, 24, 47, 47]      2,616            True
│    └─BatchNorm2d (5)   [1, 24, 47, 47]      [1, 24, 47, 47]      48               True
│    └─ReLU (6)          [1, 24, 47, 47]      [1, 24, 47, 47]      --               --
│    └─Conv2d (7)        [1, 24, 47, 47]      [1, 30, 45, 45]      6,510            True
│    └─BatchNorm2d (8)   [1, 30, 45, 45]      [1, 30, 45, 45]      60               True
│    └─ReLU (9)          [1, 30, 45, 45]      [1, 30, 45, 45]      --               --
│    └─MaxPool2d (10)    [1, 30, 45, 45]      [1, 30, 22, 22]      --               --
├─Sequential (linear)    [1, 30, 22, 22]      [1, 32]              --               True
│    └─Flatten (0)       [1, 30, 22, 22]      [1, 14520]           --               --
│    └─Linear (1)        [1, 14520]           [1, 32]              464,672          True
│    └─ReLU (2)          [1, 32]              [1, 32]              --               --
│    └─Linear (3)        [1, 32]              [1, 32]              1,056            True
│    └─ReLU (4)          [1, 32]              [1, 32]              --               --
├─Sequential (classifier)[1, 32]              [1, 3]               --               True
│    └─Linear (0)        [1, 32]              [1, 3]               99               True
==========================================================================================
Total params    : 475,313
MACs            : 22,304,008
Input size (MB) : 0.07
==========================================================================================
```

Fig. 3. LeNetV5 Summary and Size

C. Image Format

The image format used as input to the model is RGB565. This is a three color bitmap format. Each color is stored is a 5-bit byte into 2 8 bit bytes "Fig. 4". RGB565 was selected for its 2 Byte bit depth and ease of use. The image options available from the Sony Spresense Camera Library are:

- RGB565
- YUV422
- JPEG
- Gray-scale

RGB565	8-Bit High Byte								8-Bit Low Byte							
8-Bit Data	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
5-Bit Data	x	R4	R3	R2	R1	R0	G4	G3	G2	G1	G0	B4	B3	B2	B1	B0

Fig. 4. RGB565 Byte Format

D. Training Procedure

To train the model we first resize the training images to the largest image size we were able to use with the limited memory available. This was QVGA (320x240). The images are then center croped to the largest image size we were able to use while keeping the model small enough to fit. This is 96x96. Which is also the smallest size the Spresense camera library will crop to. The images are then converted to RGB565 and then converted to a PyTorch tensor. It is important to note that torchvision.transorms.ToTensor also converts the image data from UINT8 (0 to 255) to Float32 (0.0 to 1.0). This must be taken into account when running the model in realtime.

Since we did not have a lot of images to work with, and the images were captured over a relatively small number of days and conditions, we augmented the image set using rotation and perspective distortion. The number of "Empty" images also far outstretched the number of "Bird" and "Squirrel" images. The "Empty" images were randomly culled to bring the number of images in all classes somewhat the same count for training and a smaller number (about 10% of the training count) for validation. Once we had achieved a good set of images, we successively trained and re-trained the model using a variety of hyperperamiters until we got a good training loss "Fig. 5'".

Fig. 5. Training Loss

The model was then run on the testing images and a confusion matrix generated "Fig. 6'". This matrix is and images from poor performing categories are studied, the system is tweaked and re-run until we are happy(ish) with the performance. The model is then loaded onto the microcontroller, and the camera is setup to capture images of the feeder for a day. These images are manually tagged, and run the performance analyzed. The model is then re-tweeked, and retrained on the growing image set. Wash, rinse, repeat. An example training validation confusion matrix 'Fig. 6".

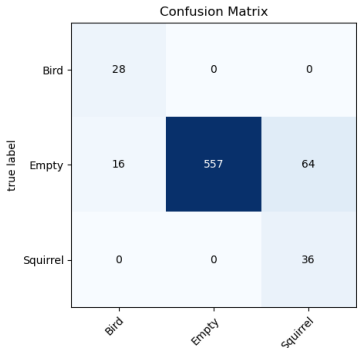

Fig. 6. Training Validation Confusion Matrix

Summary of training procedure:

- Image Conversion
  - Resize to QVGA 320x240
  - Center Crop to 96x96
  - Convert to RGB565
    - Byte swap R and B
  - Convert images ToTensor
- Image augmentation
  - For each bird and squirrel make a copy that is
    - Randomly rotated -10 to 10 degrees
    - Randomly perspective distorted up to 25%
  - For Empty
    - Cull image set to same numbers and augmented birds and squirrels
- Send the training images in batches to the trainer
- Run the trained model with the validation set.
- The final hyperparameters of the LeNetV5 [11] Model are summarized in "Table. II".

TABLE II
MODEL TRAINING HYPERPARAMETERS

Hyperparameter	***Value***
Epochs	8
Learning Rate	0.01
Batch Size	16
Criterion	Cross Entropy Loss
Optimizer	SGD

E. Installing Model on MicroController

One of the more complex procedures is the converting and installing of the machine learning model into the resource limited microcontroller. The first step in this process is to determine how large a model and image size can fit onto the selected microcontroller. Our board has 1.5MB to work with, and this space must be shared between the model, the images, the software (Including the Tensor Flow Lite library set), as well as the base operating system, and the memory space required to run the model (The Arena Size). These sizes are all interdependent on each other and some must be iteratively

determined “Table. III”. With our image size, the model must be significantly less than 1MB.

TABLE III
QUANTIZATION SUMARY

Item	*Size (kB)*
Camera Image	154
Save Image	50
Model Image	18
Model	482
Arena	166
Libraries, Program and OS	67
Memory Installed	1,573
Total Used	938

Once the model is built and trained it is converted to TensorFlow Lite for Microcontrollers “Fig. 11” then installed on the microcontroller.

- Convert the model to ONNX using torch.onnx
  - Required input: Image Shape (2x96x96)
- Convert the ONNX file to TensorFlow using onnx2tf
  - Required input: mean and standard deviation of training image set by pixel
- Export the TensorFlow file to TensorFlow Lite
  - Required input: Representative Dataset
  - Quantization of the model from 32 Bit Float to 8 Bit Int is done at this step.
- Hex dump the TensorFlow Lite file using xdd to save as a c header
  - shell: xxd -i leNetV5.tflite >leNetV5

TABLE IV
QUANTIZATION SUMARY

Quantization	*Size (kB)*
Float 32	1,961
Float 16	956
INT 8	482

During the conversion process the model is also quantized. As “Table. IV” shows, quantizing to INT8 is critical to get the model small enough to fit on the limited (1.5MB) memory space of the microcontroller. The model size + the input image size is 18 kB + 482 kB = 500 kB. It may be tempting to look at the table and think that we can use Float 16, rather than INT 8, however this change would also require an increase in Arena size. Similarly one might be tempted to increase the models image size to QQVGA (160x120) as that would only bring 20kB more. However this would increase the model size, also requiring a larger arena. The domino effect would be an addition of more than the 600kB available space. Further complicating the process, if we went much over 90% memory usage the board would not boot.

III. RESULTS

Once everything was functional, and a training set with fairly good validation numbers was achieved “Fig. 6’” the system was set up to perform a realtime run. The results of the initial Real Time Run Confusion Matrix ‘Fig. 7” were pretty good for squirrel and Empty, but disappointing for detection of birds. The model was subsequently re-trained including the images gathered on the first successful run. After retraining on larger image set Real Time Run Confusion Matrix ‘Fig. 8”. This proved very successful in determining positive hits on the ”Empty” case. However, this progress came at the cost of poor performance in the ”Squirrel” case. The detection of ”Bird” also remained poor.

TABLE V
TRAINING IMAGE COUNTS

Classification	Training Images	Validation Images
Bird	195	25
Empty	467	89
Squirrel	549	61

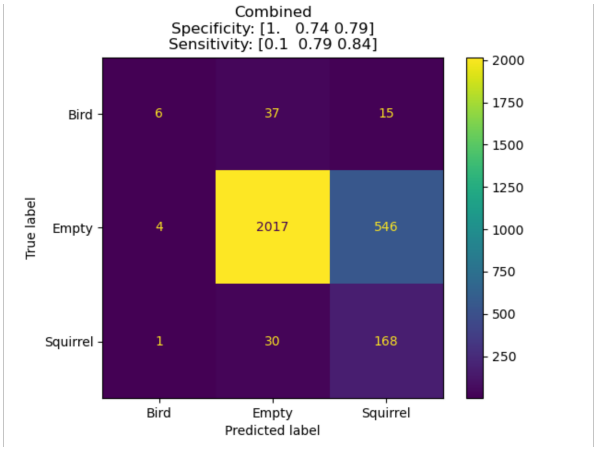

Fig. 7. Initial Real Time Run Confusion Matrix

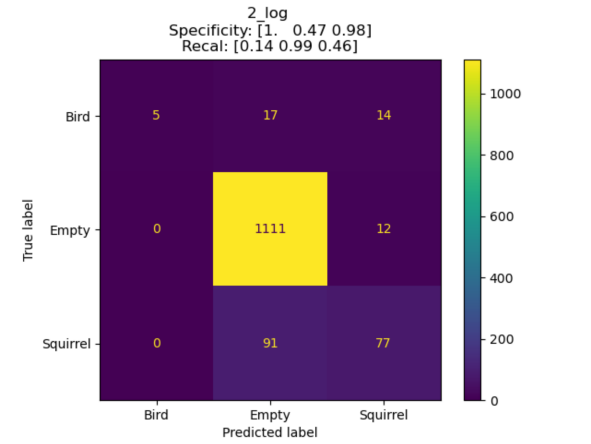

Fig. 8. After retraining on larger image set Real Time Run Confusion Matrix

After analyzing the images that were successful and unsuccessful in determining ”Bird”, the reason was discovered “Fig. 9”. It turns out that once the images are cropped to

96x96, the small birds are indistinguishable from no bird. Another major factor in the overall system performance is the lack of training images “Table. V”. These counts are very small, even when image augmentation was used. We made a decision to only use a few days worth of images for the final training due to a storm that had come in and radically rearranged the tree branches that the feeders were hanging from.

Fig. 9. Incorrect (small) Bird. Correct (Large) Bird.

The performance of the system can be improved by using a larger image size and by training on a much larger image set. If a larger image size proves impossible due to memory space limitations, then the ”Bird” focus should be limited to larger birds. (e.x. Blue Jay, Mourning Dove, and Crows: All of which are daily visitors). The singular goal of the system is the detection of the Squirrels, the bird detection is secondary, and mostly there to be useful in avoiding false positive hits for squirrels. Small birds are never confused with squirrels
One final note, is that the inference time is long. With the model as build it takes 20 seconds to run. This, while extraordinarily long, is not detrimental to the detection of squirrels, as when they do come, they will stay for many minutes at a time. However, we believe that the inference time can be dramatically improved by switching the LeNetV5 model to a MobileNetV1. However, the MobileNetV1 model trained on the 2x96x96 image set is 71.2MB in PyTorch, 18MB after quantization and converting to Tensor Flow Lite. The initial MobileNetV1, however, does show promising initial results “Fig. 10”. The size can be remedied by vastly trimming the layer count and size of each layer. The performance can be further improved by increasing the number of training images, augmentation, and tweaking the hyperparameters.
Once the system is performing better the final step would be to waterproof, and use a water jet to deter the squirrel once detected.

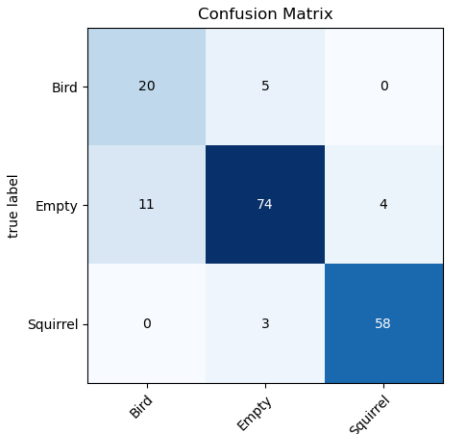

Fig. 10. MobileNetV1 Confusion Matrix

REFERENCES

[1] Paini, D. R., Sheppard, A. W., Cook, D. C., De Barro, P. J., Worner, S. P., & Thomas, M. B. (2016). Global threat to agriculture from invasive species. Proceedings of the National Academy of Sciences, 113(27), 7575-7579.

[2] Anderies, J. M., Katti, M., & Shochat, E. (2007). Living in the city: resource availability, predation, and bird population dynamics in urban areas. Journal of theoretical biology, 247(1), 36-49.

[3] Koehler, A. E., Marsh, R. E., & Salmon, T. P. (1990). Frightening methods and devices/stimuli to prevent mammal damage–a review.

[4] Sony. (2004). Sony Spresense Specifications. https://developer.sony.com/spresense/product-specifications

[5] King Nicola, Sadowski, (2001) ML-based Bird and Squirrel Detector, MagPi, https://magpi.raspberrypi.com/articles/ml-based-bird-and-squirrel-detector

[6] T. Mary. (2021). Solving the Problem of Squirrels Stealing from the Bird feeder: Prototyping Image Classification with the Deep Learning Workbench in Intel® DevCloud for the Edge. Intel Community, https://community.intel.com/t5/Blogs/Tech-Innovation/Artificial-Intelligence-AI/Solving-the-Problem-of-Squirrels-Stealing-from-the-Bird-feeder/post/1335676

[7] Chapman, J. B. (1996). Effectiveness of capsaicin as a squirrel repellent at bird feeders. Mississippi State University.

[8] Krause, S. K., Kelt, D. A., & Van Vuren, D. H. (2010). Invasion, damage, and control options for eastern fox squirrels. In Proceedings of the Vertebrate Pest Conference (Vol. 24, No. 24).

[9] MIC Lab. (2024). http://sfsu-miclab.org/

[10] Singla, N. (2014). Motion detection based on frame difference method. International Journal of Information & Computation Technology, 4(15), 1559-1565.

[11] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. Neural computation, 1(4), 541-551.

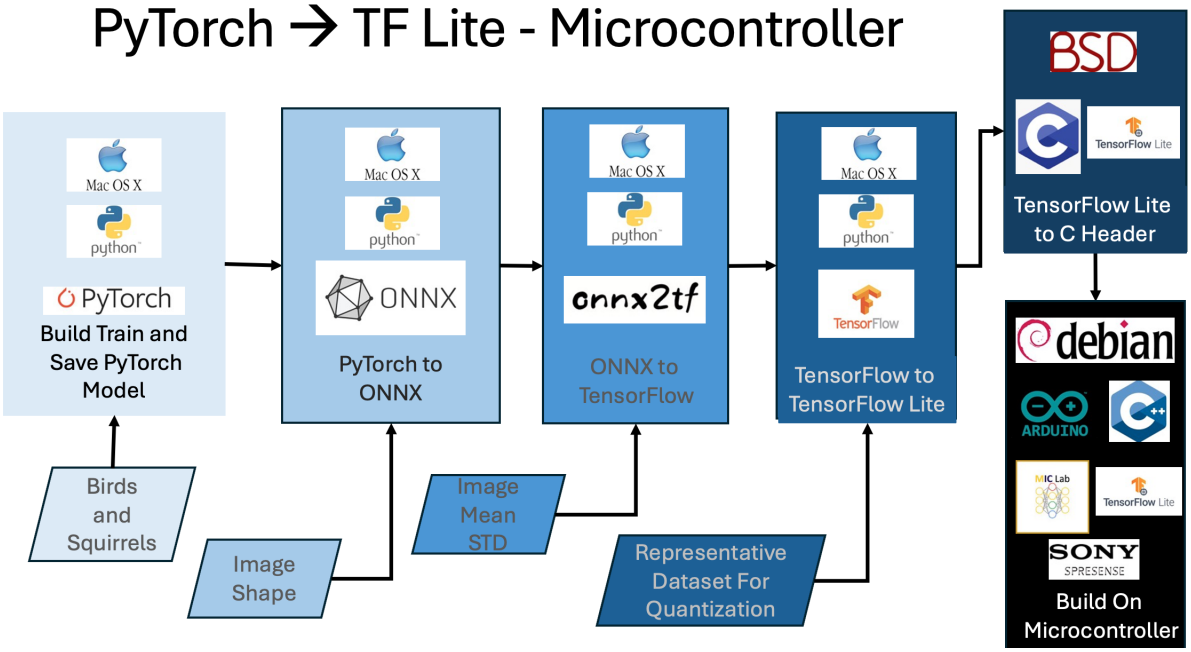

Fig. 11. PyTorch –>Tensor Flow Lite Workflow

TABLE VI
SUMMARY OF CODE BASE

Module	Hardware/OS/Platform	File	Language
RealTime	Sony Spresense/Arduino	sqVB1rd_ard.ino	C++
RealTime Model	TensorFlow Lite	RGB2BGRleNetV5_trained.h	C Header
RealTime Serial Image Diaplay	Processing: https://processing.org	processingScript.pde	Java
Image Capture/First Attempt at RealTime	Sony SpreSense/SDK	sqb_main.c, camera.c, camera.h, cardUtils.c, cartdUtils.h, DSC.config	C
Data Augmentation	torchvision	DataAugment.py	Python
Image Loading	TorchVision	DataPreparation.py	Python
Convert Image	CV2	BGR2RGB565.py [9]	Python
Rename Images	OS	fileOpps.py	Python
Model Training	PyTorch	main.py	Python
Model Training	PyTorch	Model.py	Python
Model Definition	PyTorch	Trainer.py	Python
Model Converting	'Fig. 11"	saveModel.py	Python
Model Optimization	MIC Pruning Engine	optimize.py	Python
Model Validation	Tensor Flow	RunTfLite.py	Python
Model Validation	PyTorch	validate.py	Python
Data Analysis	Torch Metrics/SSklearn	analysis.py	Python
Model Analysis	thop	OpCounter.py	Python