Wildfire Detection enabled Camera
Jetson TX2 GPU accelerated

Nikos Georgis

17-Feb-2018

Motivation

- The 2017 California wildfire season was the most destructive wildfire season on record.
- Early detection of ignition can result in faster response by fire agencies therefore minimizing destruction.
- San Diego, CA residents were invited to watch strategically installed cameras and report fires:
  - *Public can use webcams to watch for wildfires across San Diego County*
- This is a task that could potentially be automated.
- NVIDIA Jetson TX2 is an ideal platform for this kind of applications.

Goals

- Design an Intelligent Camera capable of
  - Accurate and fast wildfire detection.
  - Accurate even when minimal amount of training data is available.
  - Operate in real time, processing multiple video streams.
  - Use the TensorRT model optimizer.

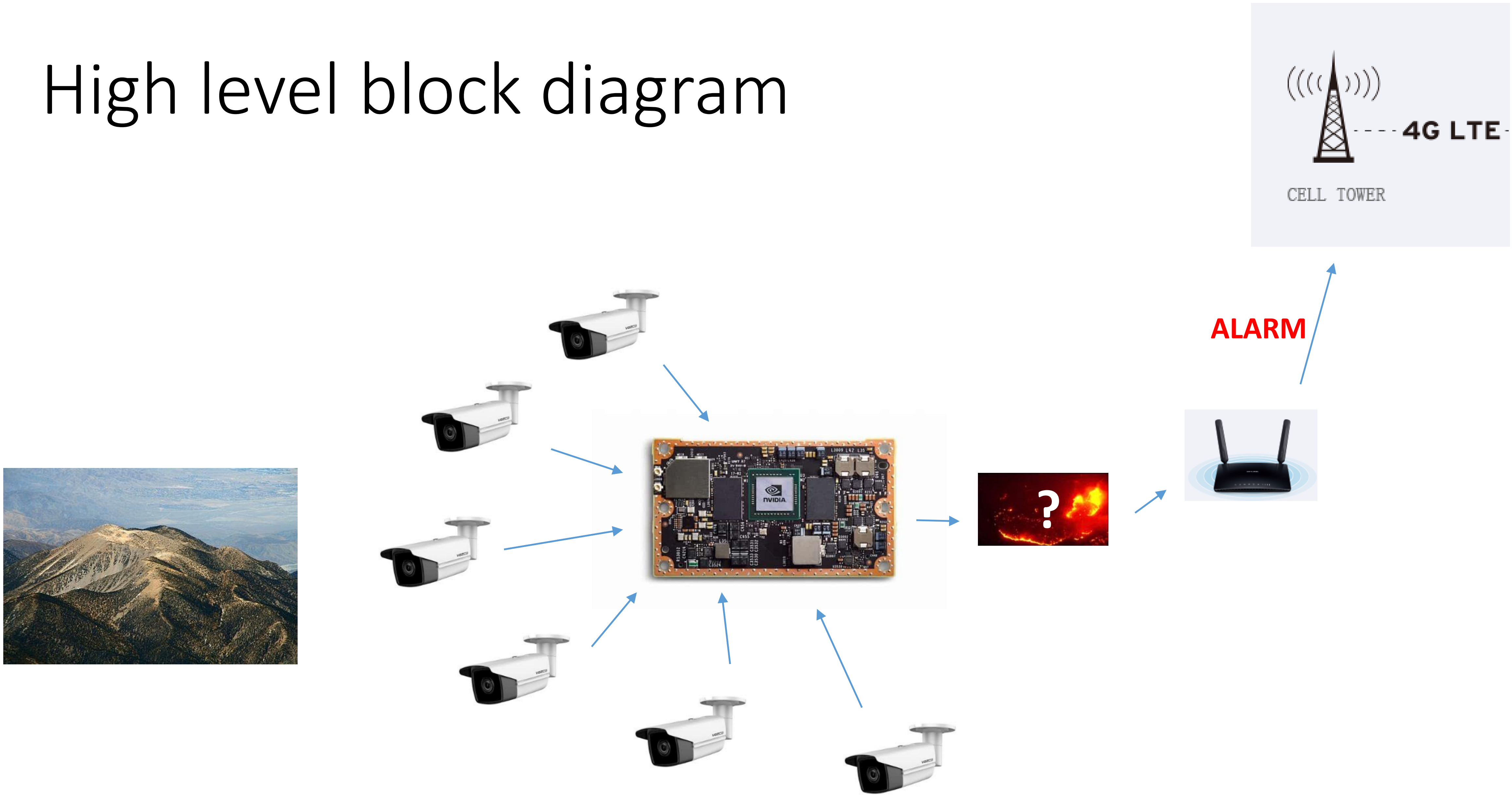
- Educational value of this project
  - Demonstrate end-to-end pipeline: from idea to implementation.
  - Show how transfer learning can be used in real projects.
  - Show how Tensorflow can be used to generate UFF models.
  - Use NVIDIA profiling tools to measure performance and identify bottlenecks.

Proposed solution

- Able to operate standalone.
- Battery rechargeable by solar panels.
- Real-time processing
  - Process video streams from multiple local cameras.
  - Wildfire and smoke detection.
- Always connected
  - 4G/LTE connectivity.
  - Send alarms in real-time.
- Rigid enclosure
  - Water proof.
  - Heat proof.

Photoshop by Anthony G

High level block diagram



Software Implementation

- Development on a NVIDIA GTX1080 desktop
  - Used Tensorflow 1.3 for training.
  - Jupyter notebook capturing the training and test phases:
  - https://github.com/ngeorgis/ca-fire-detector/blob/master/fire-detection-jetson-save-ca.ipynb
- Deployment phase
  - Jetson TX2
  - JetPack 3.2
  - TensorRT 3
  - `Linux tegra-ubuntu 4.4.38-tegra #1 SMP PREEMPT Fri Dec 1 06:08:28 PST 2017 aarch64 aarch64 aarch64 GNU/Linux`

CNN Fire detection

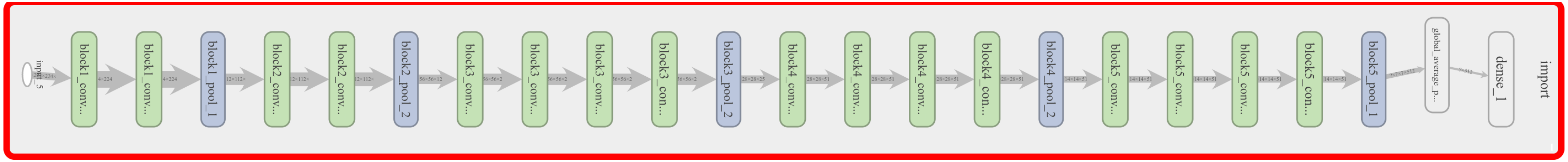
Although an advanced CNN or even RNN (LSTM or GRU) could be used for sophisticated fire detection, in this project focus was on robustness and speed.

A relatively simple but deep enough CNN was found to be suitable for this purpose.

- A number of classes were evaluated
  - Fire, Smoke, Safe
  - Fire, Safe

- Several models were evaluated
  - VGG16_model
  - VGG19_model
  - InceptionV3_model
  - Xception_model
  - ResNet50_model

Finally, a two-class Fire/Safe VGG-19 was used.

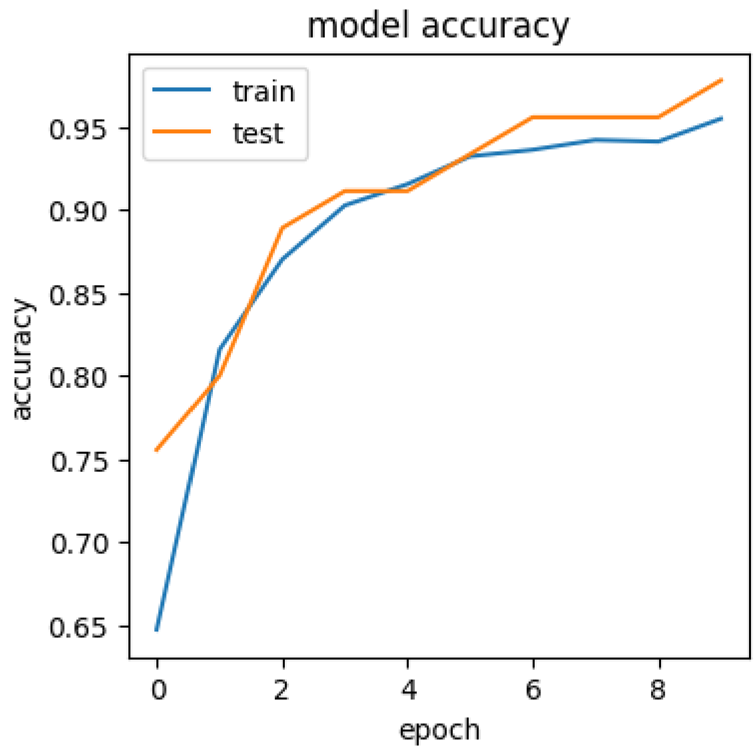


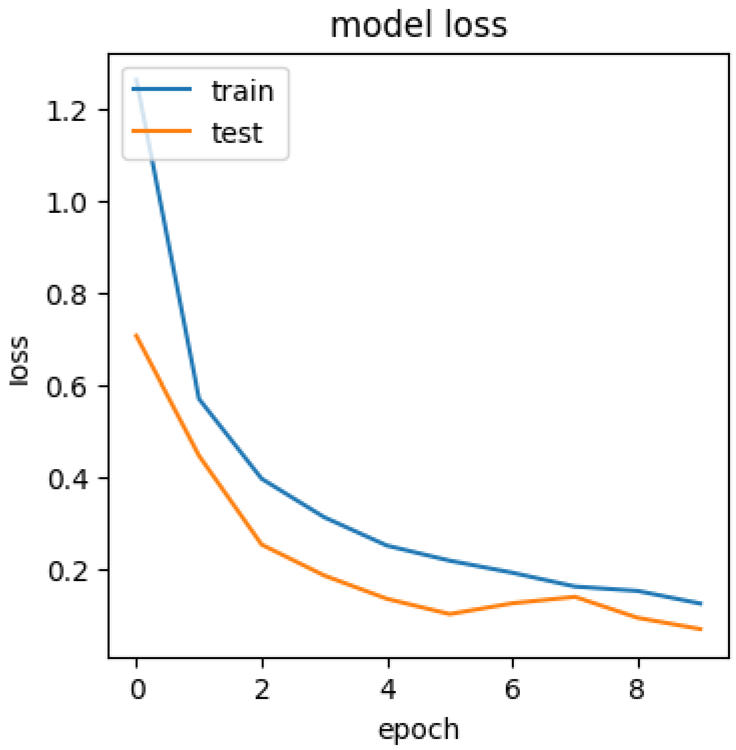
Transfer learning

- The amount of training data required to train VGG-19 is difficult to be captured and manually annotated.
- Pretrained models are available with significant accuracy for many classes similar to the two classes used here: Fire/Safe
- The decision was made to freeze the base model and only trained the last layers.
  - `Total params: 20,025,410`
  - **`Trainable params: 1,026`**
  - `Non-trainable params: 20,024,384`
- Resulted in an accurate fire detector that can generalize well.
- Reference: [Udacity AIND Dog Project](), [Transfer Learning]()

```
In [12]: # Freeze the layers which you don't want to train.
         for layer in base_model.layers:
             layer.trainable = False
```

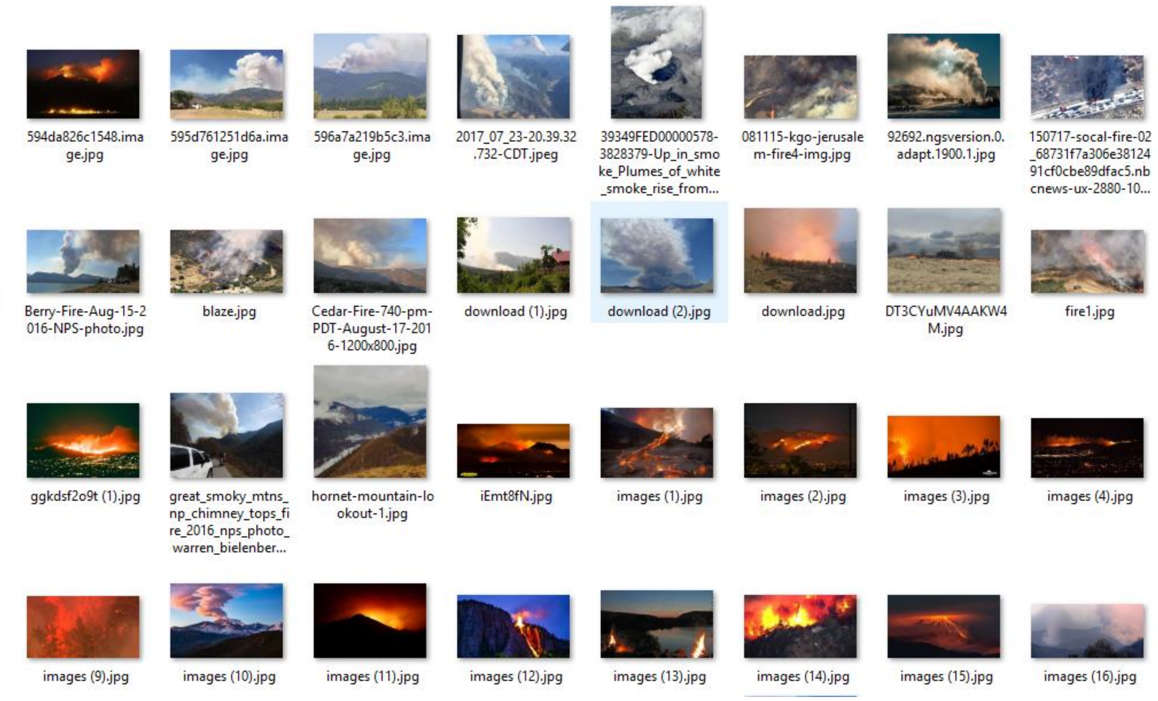
Accuracy and loss graphs





Training data

- Less than 1000 training images were used.
  - Training data
  - Validation data
  - Test data



- Reference : [How to create a deep learning dataset using Google Images]

From Tensorflow to TensorRT and Jetson TX2

- Details in https://github.com/ngeorgis/ca-fire-detector/blob/master/fire-detection-jetson-save-ca.ipynb
  - Section: *Convert the Keras / TF model to something that Jetson TX2 understands*
- Freeze the TF graph
- Convert to UFF
  - `convert-to-uff tensorflow --input-file frozen_fire_detector.pb -l`
  - `convert-to-uff tensorflow -o fire_detector.uff --input-file frozen_fire_detector.pb -O "dense_1/Softmax"`

From Tensorflow to TensorRT and Jetson TX2

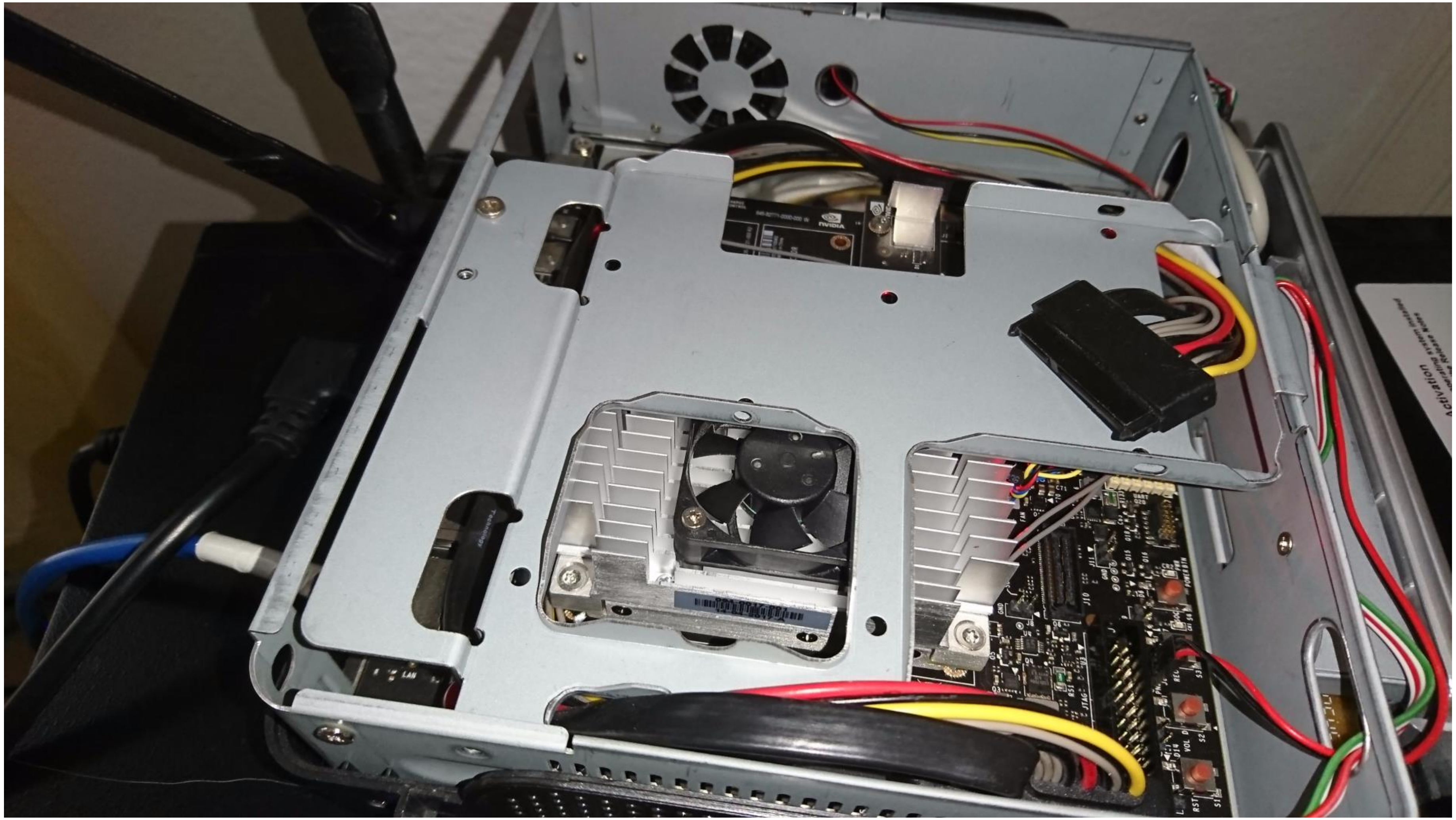
- Deploy using minor modifications to
  - `sample TensorRT-3.0.0\samples\sampleUffMNIST\sampleUffMNIST.cpp`

```
auto fileName = locateFile("fire_detector.uff");

std::cout << fileName << std::endl;
int maxBatchSize = 1;
auto parser = createUffParser();

/* Register tensorflow input */
parser->registerInput("input_5", DimsCHW(3, 224, 224));
parser->registerOutput("dense_1/Softmax");

ICudaEngine* engine = loadModelAndCreateEngine(fileName.c_str(), maxBatchSize, parser);
if (!engine)
    RETURN_AND_LOG(EXIT_FAILURE, ERROR, "Model load failed");

/* we need to keep the memory created by the parser */
parser->destroy();

execute(*engine);

engine->destroy();
shutdownProtobufLibrary();
return EXIT_SUCCESS;
```

Status of hardware development

- Jetson TX2 acquired.
- No carrier board yet for Fire Detector miniaturization.
- Mini-ITX enclosure.
- Raspberry Pi cameras
  - Can be accessed over IP using NVIDIA Gstreamer.
  - AVC packets over IP.
  - NVDec decoding pipeline.
  - 3D printer M12 mount -> M12 -> CS adapter -> Sony lenses
- No waterproofing or recharging battery effort yet.

Current hardware status



Experimental results

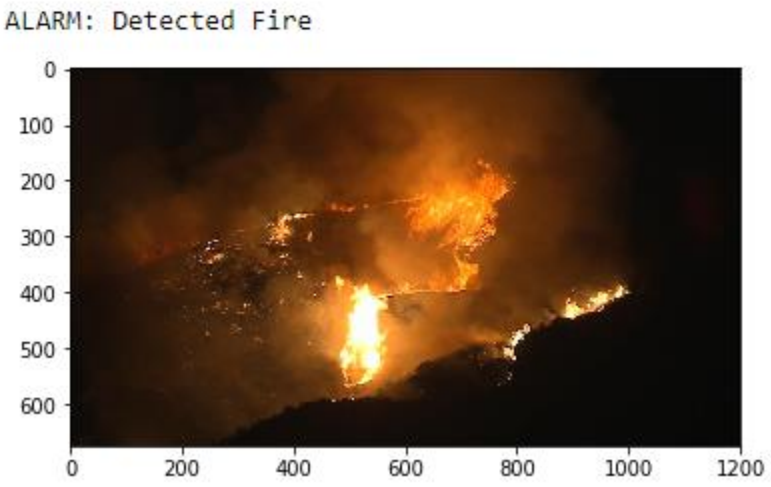
- Class: Safe

Looking good: Safe

Looking good: Safe

Experimental results

- Class: Fire



Profiling results

- Profiled using nvprof on Jetson TX2
  - Clocks were set to high using nvpmodel and jetson-clock.sh
  - Can process over 15 fps of HD video stream from IP cameras.
  - Average over 10 runs is 65.1052 ms
  - Mainly cycles spent on `fusedConvolutionReluKernel` and `cudnn_winograd_128x128`
- Used the NVIDIA System profiler
  - Remote attach and profile fire detection pipeline
  - Opportunity to overlap transfers and compute - TBD

```
==3743== Profiling application: ../../bin/sample_uff_fire_detector
==3743== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   23.16%  2.19217s      2315  946.94us     320ns  6.4950ms  [CUDA memcpy HtoD]
                    7.12%  674.20ms       166  4.0615ms  1.3785ms  8.0820ms  trtwell_scudnn_winograd_128x128_ldg1_ldg4_mobile_relu_tile148t_nt
                    2.57%  243.28ms        32  7.6024ms  2.6775ms  11.893ms  void fused::fusedConvolutionReluKernel<fused::SrcChwcPtr_FltTex_Re
                    fused::KpqkPtrWriter<float, int=1, int=2>, float, float, int=7, int=5, int=1, int=3, int=3, int=1, int=1>(fused::Convolutic
                    float)
                    2.16%  204.12ms        32  6.3788ms  2.6318ms  9.5032ms  void fused::fusedConvolutionReluKernel<fused::SrcChwcPtr_FltTex_Re
                    fused::KpqkPtrWriter<float, int=1, int=2>, float, float, int=4, int=8, int=1, int=3, int=3, int=1, int=1>(fused::Convolutic
                    float)
                    2.15%  203.34ms        30  6.7781ms  2.4396ms  10.121ms  void fused::fusedConvolutionReluKernel<fused::SrcChwcPtr_FltTex_Re
                    fused::KpqkPtrWriter<float, int=1, int=1>, float, float, int=7, int=8, int=4, int=3, int=3, int=1, int=1>(fused::Convolutic
                    float)
                    2.08%  196.95ms        15  13.130ms  4.8616ms  19.804ms  void fused::fusedConvolutionReluKernel<fused::SrcChwcPtr_FltTex_Re
                    fused::KpqkPtrWriter<float, int=1, int=1>, float, float, int=7, int=6, int=8, int=3, int=3, int=1, int=1>(fused::Convolutic
                    float)
                    1.78%  168.75ms        15  11.250ms  3.5959ms  27.921ms  void fused::fusedConvolutionReluKernel<fused::SrcChwcPtr_FltTex_Re
                    fused::KpqkPtrWriter<float, int=1, int=4>, float, float, int=5, int=7, int=4, int=3, int=3, int=1, int=1>(fused::Convolutic
                    float)
                    1.77%  167.09ms        15  11.139ms  3.2848ms  19.074ms  void fused::fusedConvolutionReluKernel<fused::SrcChwcPtr_FltTex_Re
                    fused::KpqkPtrWriter<float, int=1, int=4>, float, float, int=2, int=5, int=2, int=3, int=3, int=1, int=1>(fused::Convolutic
                    float)
```

Under development

- Add dropout layers and re-train VGG-19
- Drop the dropout for inference
  - TensorRT PB -> UFF issue
  - Need to understand how to manipulate TF graph and remove dropouts
  - Open issue in GitHub project: https://github.com/ngeorgis/ca-fire-detector/issues/1
- Better training data.
- Waterproof enclosure.
- Add solar panel / battery / more cameras.
- Optimize pipeline and deploy

Conclusions

- Successful design of a wildfire early detection system using deep learning.
- Better to have intelligent locally so that wildfire cameras can process in real-time and respond faster.
- NVIDIA Jetson TX2 ideal for this task.
- Amazing application of transfer learning to make this fire detector work with minimal training data.
- Successful deployment using the NVIDIA Jetson TX2 tools.
- Implementation of the end-to-end pipeline
  - Idea -> Keras / Tensorflow -> Freeze to pb -> pf to uff -> TensorRT -> TX2 inference
- Profiling of the complete pipeline using NVIDIA tools.

Acknowledgements

- Udacity Artificial Intelligent Nano Degree for teaching me deep learning and introducing me to the powerful transfer learning concept and how to freeze layers properly.
- Anthony and Elias, junior data scientists, for assistance including training data collection.

https://www.udacity.com/ai UDACITY