SYNTIANT

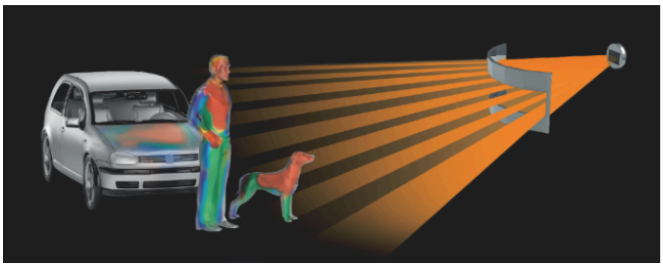
Power-Efficient Object Detection in PIR Data Using Syntiant NDP

Overview

Unlocking Hidden Information in Passive Infrared (PIR)

A passive infrared sensor (PIR sensor) is an electronic sensor that measures infrared (IR) light radiating from objects in its field of view. Most often used in motion detectors, they are small, inexpensive, low-power, easy-to-use, and durable. These characteristics make them suitable for use in appliances and gadgets used in homes or businesses, such as security alarms and automatic lighting applications.

PIR sensors are typically used to detect general movement, but not to give information on who or what moved. This is because traditionally they are deployed in low-power compute environments paired with very simple threshold-based triggering mechanisms. This paper describes how advanced AI algorithms can be used to unlock more information inherently present in PIR sensor data.



Standard PIR sensitivity

PIR sensitivity using Syntiant's ultra-low-power Neural Decision Processor (NDP)

PIR sensors are almost always used to detect whether a human has moved within the sensor's range, but the simple triggering algorithms used are sensitive to other types of movement. Therefore, they tend to produce high rates of *false alarms,* for example arising from pets and automobiles. Attempts to reduce false alarms by lowering the sensitivity instead causes high rates of *missed detections*. If instead advanced AI algorithms are employed, both false alarms and missed detections can be drastically reduced. The Syntiant Neural Decision Processor (NDP) enables such algorithms to be deployed in an ultra-low-power computing environment.

How PIR Sensors Work

PIRs are made of a pyroelectric sensor that can detect levels of infrared radiation. Everything emits some low-level radiation, and the hotter something is, the more radiation is emitted. The sensor in a motion detector is actually split in two halves, both of which are configured to detect changes in IR levels.

When the sensor is idle, both slots detect the same amount of IR, the ambient amount radiated from the room or walls or outdoors. When a warm body like a human or animal passes by, it first intercepts one half of the PIR sensor, causing a *positive differential* change between the two halves. When the warm body leaves the sensing area, the reverse happens -- the sensor generates a negative differential change. These change pulses are what is detected.

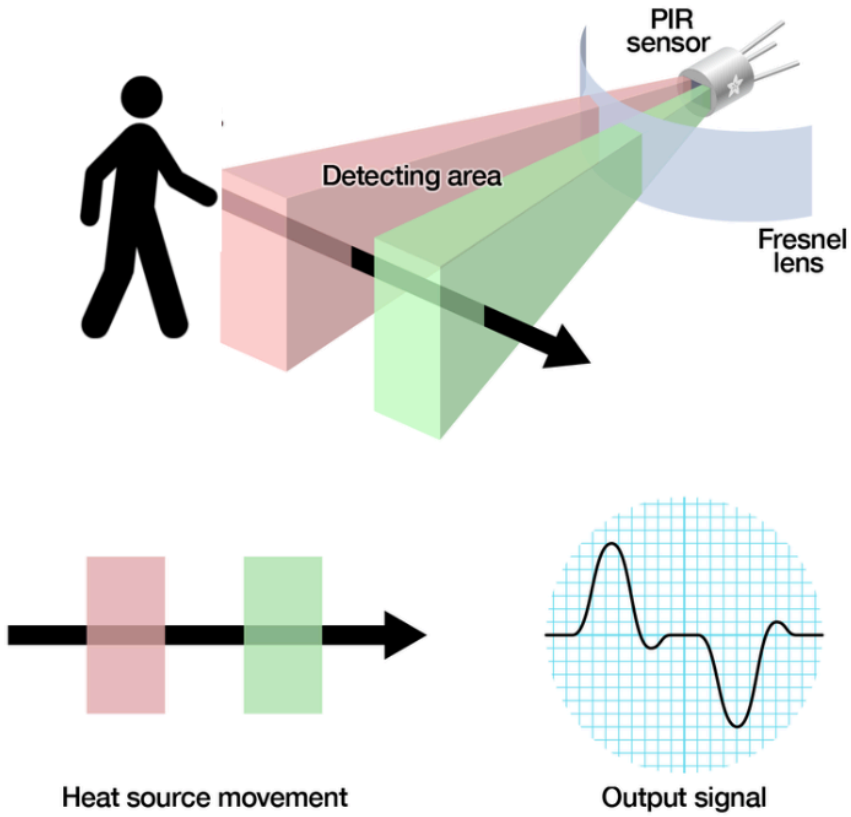


Illustration 1: PIR sensors measure temperature changes in multiple sub-regions of a monitoring area caused by the motion of heat-emitting objects.

Most of the variation in PIR sensor specifications are not due to the sensors themselves, but instead due to the lenses used to focus light onto the sensors. The lenses -- commonly Fresnel lenses -- collect light from large areas of a room or landscape. They have multiple facets that focus light from different sectors of an area to different sensors. The different faceting and sub-lenses create a range of detection areas, interleaved with each other.

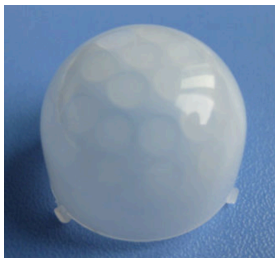


Illustration 2: Lenses

The Advantages of AI Over Traditional PIR Triggering Algorithms

In a typical deployment, a device may contain multiple PIR sensors, which, when combined with a suitable lens, monitor different areas of a room or landscape. The different sensor "channels" are constantly active and contain different information on each channel.

Illustration 3 (below) shows a time-series trace of a 3-channel PIR deployment in a home surveillance application. In this example, a person walks in front of the device in the middle of the time period. Positive and negative sensor values correspond to increases and decreases in temperature, respectively.

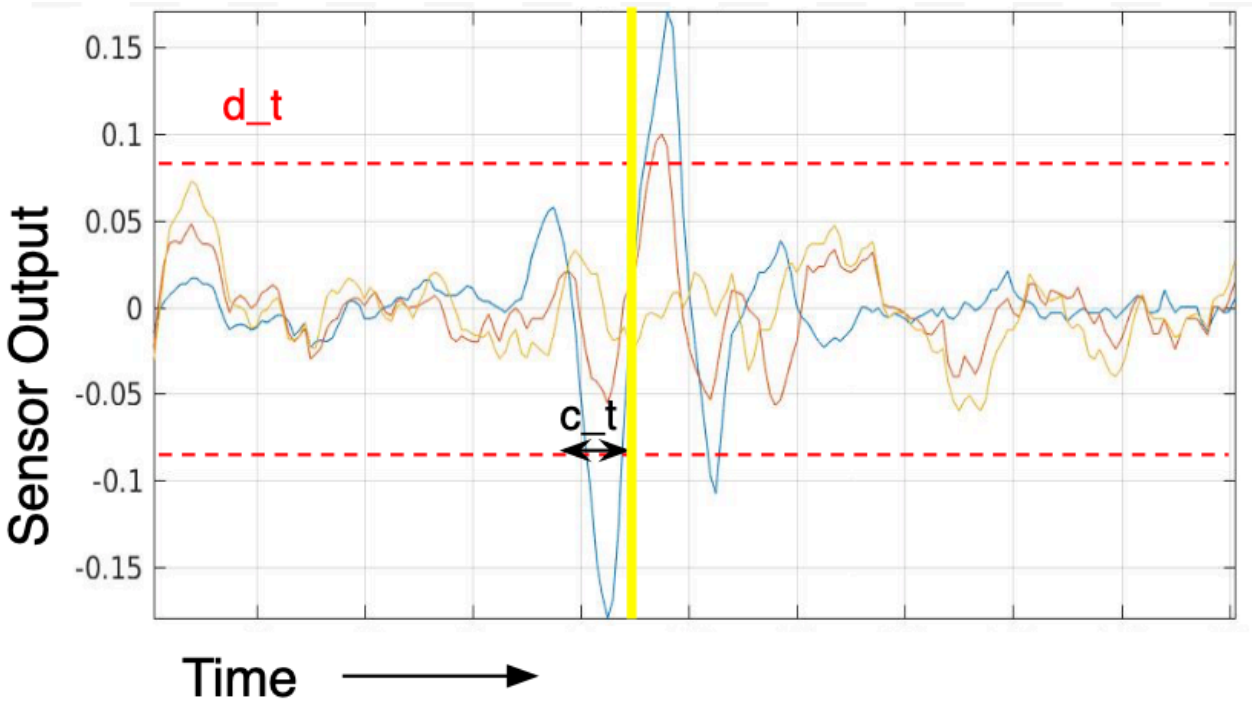


Illustration 3: A simple threshold-based triggering mechanism typically used with PIR sensors

Because PIR sensors are usually deployed in ultra-low-power conditions, coupled with lightweight microcontroller units (MCUs) running off a battery, very simple detection algorithms must be used. These algorithms are usually based on thresholds and counters. As shown in **Illustration 3**, if any of the sensor channels deviate from zero (positive or negative) by an amount exceeding a detection threshold **d_t**, the detector counter increments by one. If any of the channels remain above **d_t** in subsequent time steps, the counter continues to increment; otherwise a time-out occurs, and the counter is reset to zero. If the counter exceeds a predefined count threshold **c_th**, the final detection is triggered (the yellow bar in **Illustration 3**) and the counter is reset.

Simple algorithms like this can be effective at detecting motion in the environment. However, because they are simply based on the overall sensor levels exceeding a threshold in short time windows, they ignore much of the information in the complex patterns of sensor change over time, as well as the patterns of how the different sensor channels relate to each other.

AI algorithms can learn to take advantage of such information using Machine Learning (ML) models such as Deep Neural Networks (DNNs). DNNs can use much longer time spans of sensor data from multiple channels simultaneously and learn how this complex sensor data differentiates the presence of different kinds of objects (such as humans, animals, or cars), the speed of an object's motion, and its size and position. To learn these patterns, ML algorithms require many examples of the sensor data traces in response to the different kinds of events important in the application.

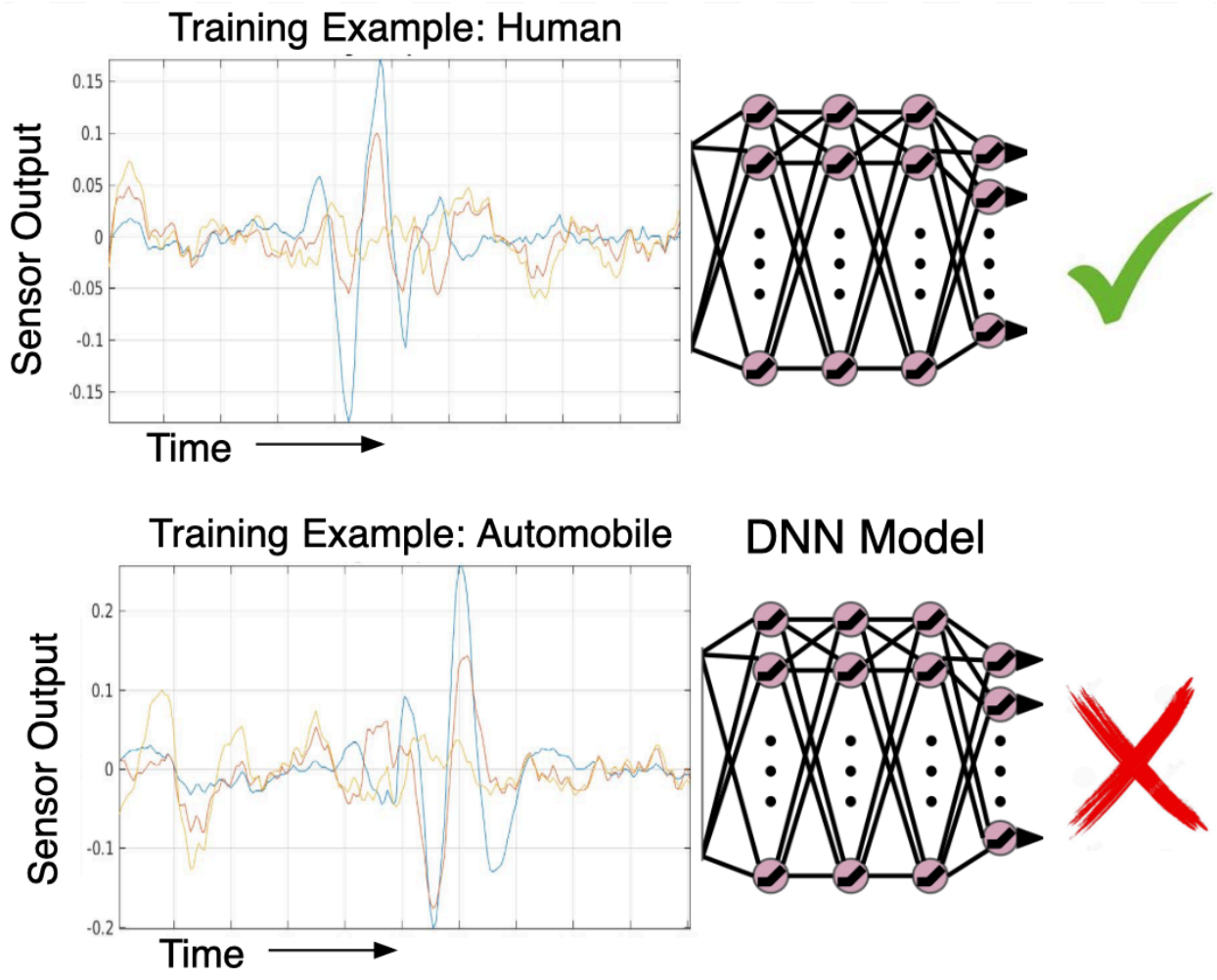


Illustration 4: Deep Neural Networks (DNNs) can learn to distinguish types of moving objects based on the complex time-varying sensor output patterns across multiple sensor channels.

Example Application: Home Entry Door Surveillance

A common application for PIR sensors is in a home surveillance application of an entry door (see **Illustration 5**). In this application the sensors enable a homeowner to receive alerts, sometimes with accompanying video data, if a person approaches the door. The PIR sensors are included with a small battery-powered camera that the homeowner can easily mount outside the door.

To attempt to save power, the camera is only enabled if the PIR sensors detect motion. However, because of the simplicity of the detection algorithms and the complexity of the outdoor environment, such systems are heavily susceptible to false alarms from pets, cars, and even people walking past on the sidewalk.

Illustration 5: Video snapshot and accompanying PIR sensor data trace for a home entry door surveillance deployment.

Illustration 6 shows how a properly trained DNN can be used to dramatically reduce both false alarms and missed detections for this application. Compared to the simple threshold-based algorithm, the DNN could be used to reduce false alarms by 80%, missed detections by 60%, or both types of errors simultaneously by 40%.

The model used in this example was trained on the outputs of PIR sensors exposed to approaching humans and passing cars, as well as instances with no moving objects. The training involved two hundred 1-minute long PIR time series from a wide variety of locations. Models were then tested on a held out set of PIR sensor locations and compared to a simple threshold-and-count algorithm supplied by an experienced customer known for commercial PIR deployments.

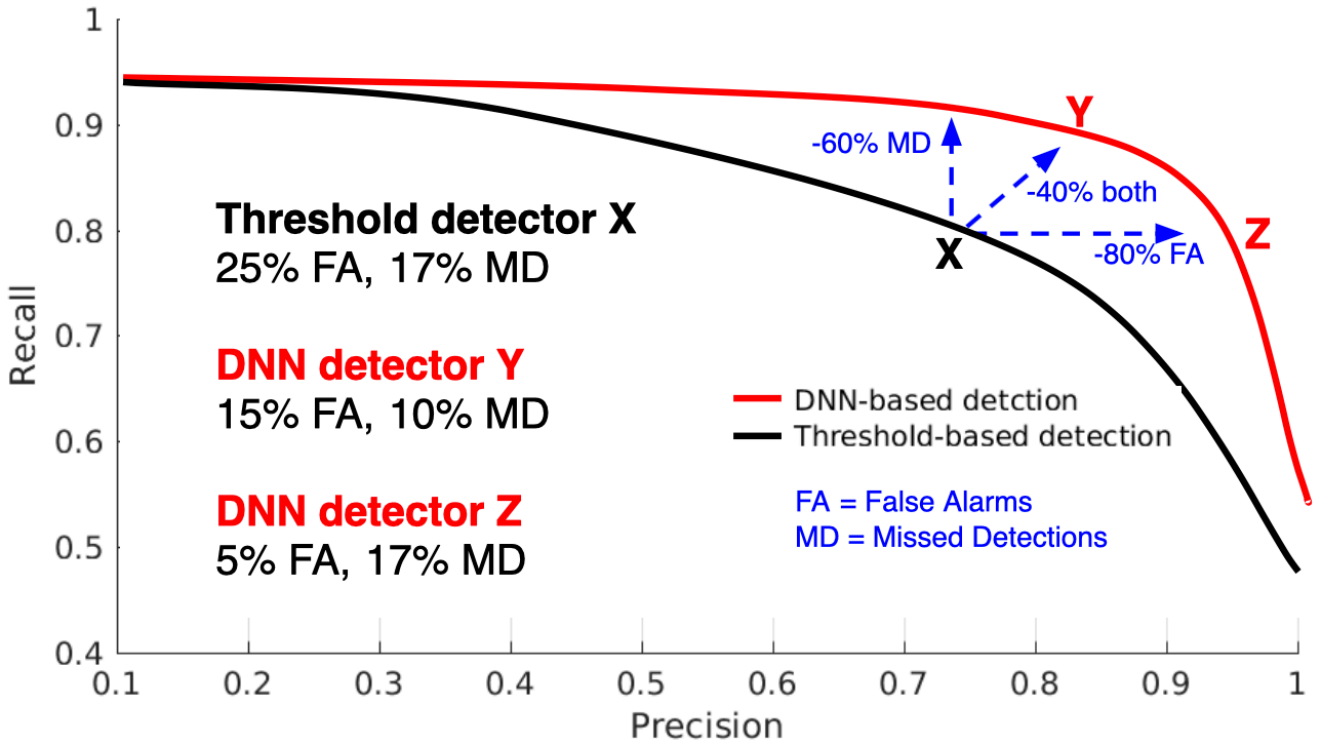


Illustration 6: DNN-based detectors greatly reduce both false alarms and missed detections in a human vs. vehicle detection task.

While DNNs can use longer time spans of sensor outputs to their advantage, this introduces *signal latency*, if sensor outputs are being used to make a decision about motion activity in the recent past. Latency is similarly introduced by the detector counters in threshold-based detectors. Latency is potentially an issue in applications where video data is being buffered and streamed off device upon a detection event. In order to catch the video corresponding to the motion event, longer video clips have to be buffered and streamed for detectors with longer latency. This consumes more system resources and power.

As shown in **Illustration 7(a)**, latency is easily configurable in DNN-based detectors by excluding future sensor data from models making decisions about the current frame. The amount of future context used to train a model will set its corresponding signal latency. **Illustration 7(b)** shows the performance impact of reducing the signal latency from 3 seconds to 0.25 seconds, compared to the threshold-based detector with a latency of 0.25 seconds. In this case, the DNN-based detector is still able to maintain a large advantage over the threshold-based detector, possibly because of its remaining ability to use long time spans in the past to make a decision about the current frame.

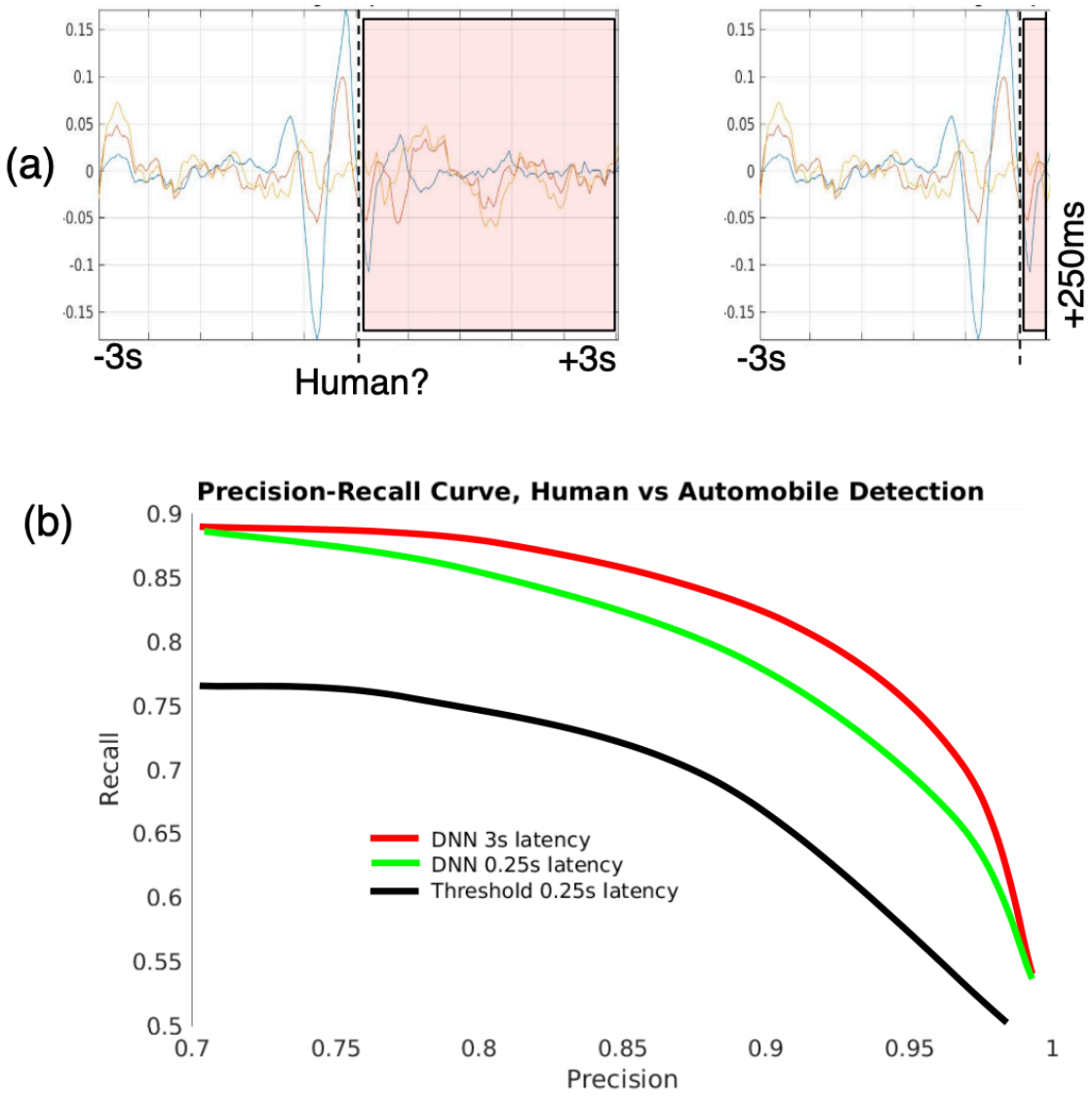


Illustration 7: Configuring DNN latency by excluding future sensor data from models making decisions about the current frame. (a) Shows the difference in the model data inputs for a detector having 3s (left) and 250ms (right) of latency with respect to the current frame (dashed line). (b) Shows that there is some impact to the precision-recall curve to reducing latency, but the DNN detector still holds a large advantage over the threshold-based detector with the same latency.

Using the Syntiant NDP to Run AI Algorithms at Ultra-Low Power

Previously, DNN-based PIR detection algorithms had been too compute intensive to be deployed in low-power systems. However, the Syntiant NDP was designed to run DNNs with up to four hidden layers while consuming under 200 microwatts (uW) of power during active sensor processing. One such example is the DNN used to achieve the results shown in **Illustration 6**.

Once appropriate training data is obtained and prepared for training DNNs, the Syntiant Training Development Kit (TDK) can easily be used to train and test models that are ready for deployment on the NDP. Model training and testing is performed through a familiar Keras Python interface to TensorFlow, the most widely used library for DNN-based system development. Models can be trained on premises or on standard AWS EC2 instances running Deep Learning AMIs, potentially employing GPU co-processors to speed up training. Syntiant also offers services to accelerate data collection, data preparation, model training, and developer training.

```
60000 train samples
10000 test samples
Train on 60000 samples, validate on 10000 samples
Epoch 1/4
2018-08-29 21:46:13.605822: I tensorflow/core/platform/cpu_feature_guard.cc:137] Your CPU supports instructions that this TensorF
2018-08-29 21:46:15.458819: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:895] successful NUMA node read from SysFS had
2018-08-29 21:46:15.459151: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1105] Found device 0 with properties:
name: Tesla K80 major: 3 minor: 7 memoryClockRate(GHz): 0.8235
pciBusID: 0000:00:1e.0
totalMemory: 11.17GiB freeMemory: 11.10GiB
2018-08-29 21:46:15.459183: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1195] Creating TensorFlow device (/device:GPU:0) -
60000/60000 [==============================] - 15s 251us/step - loss: 0.9844 - acc: 0.6935 - val_loss: 0.2746 - val_acc: 0.9194
Epoch 2/4
60000/60000 [==============================] - 1s 10us/step - loss: 0.2881 - acc: 0.9163 - val_loss: 0.1715 - val_acc: 0.9509
Epoch 3/4
60000/60000 [==============================] - 1s 10us/step - loss: 0.1998 - acc: 0.9434 - val_loss: 0.1334 - val_acc: 0.9614
Epoch 4/4
60000/60000 [==============================] - 1s 10us/step - loss: 0.1567 - acc: 0.9555 - val_loss: 0.1110 - val_acc: 0.9666
Test loss: 0.11102039980888366
Test accuracy: 0.9666
Your classification accuracy is 'good enough'.
################################################################################
Now we are going to quantize the weights we just learned and re-run the test set.
The test accuracy should not get substantially worse.
################################################################################
Quantized Test loss: 0.11325708038210869
Quantized Test accuracy: 0.9672
The loss changed by:  0.0022366805732250278
The accuracy changed by:  0.0005999999999999339
```

Illustration 8: Syntiant's TDK developer interface is the familiar Keras API to Tensorflow

Models trained with Syntiant's TDK are ready for system integration using the Syntiant Software Development Kit (SDK), either for live testing on the Syntiant NDP9101 Rasberry Pi evaluation system, or for production deployment. The SDK provides a collection of software for execution on companion processor(s) responsible for controlling NDP devices, such as an associated application processor (AP) or an NDP-embedded processor (for example, the ARM M0 microcontroller of an NDP device). The SDK source code may also be used as an example in deployments for which direct integration is not appropriate.

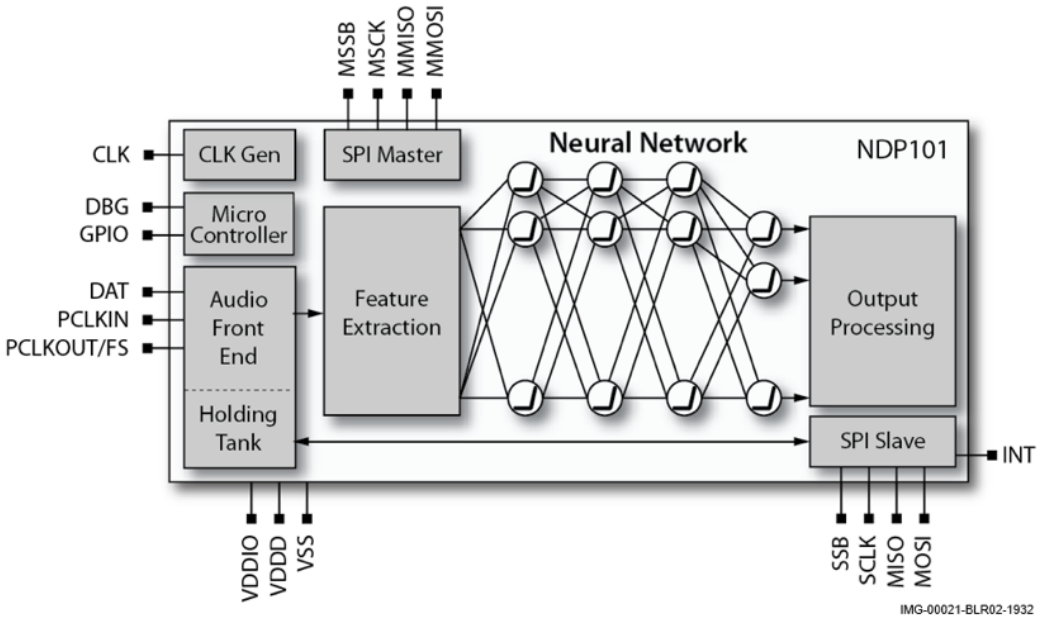


Illustration 9: Syntiant NDP101 Block Diagram

Summary

Increasing PIR's Powers of Observation Without Increasing its Power Consumption

AI algorithms employing DNNs enable dramatic improvements in distinguishing humans from other moving objects using PIR sensors. This exemplifies what is achievable in many other PIR sensor use cases, including in manufacturing, robot navigation, and agriculture -- as well as with many other types of inexpensive sensors. These sensors have had limited observational powers because they've been deployed in ultra-low-power compute environments that are severely limited in on-device intelligence and the ability to continuously stream data off the devices.

The Syntiant NDP removes the computation bottleneck for AI algorithms at ultra-low-power consumption. The NDP architecture is built from the ground up to run DNNs. It achieves breakthrough performance by highly coupled computation and memory, exploiting the vast inherent parallelism of DNN computation and computing only at required numerical precision. The devices combine these benefits to achieve approximately 100x efficiency improvement over stored program architectures such as CPUs and DSPs. Furthermore, they are easily programmed and integrated using Syntiant's accompanying TDK and SDK.