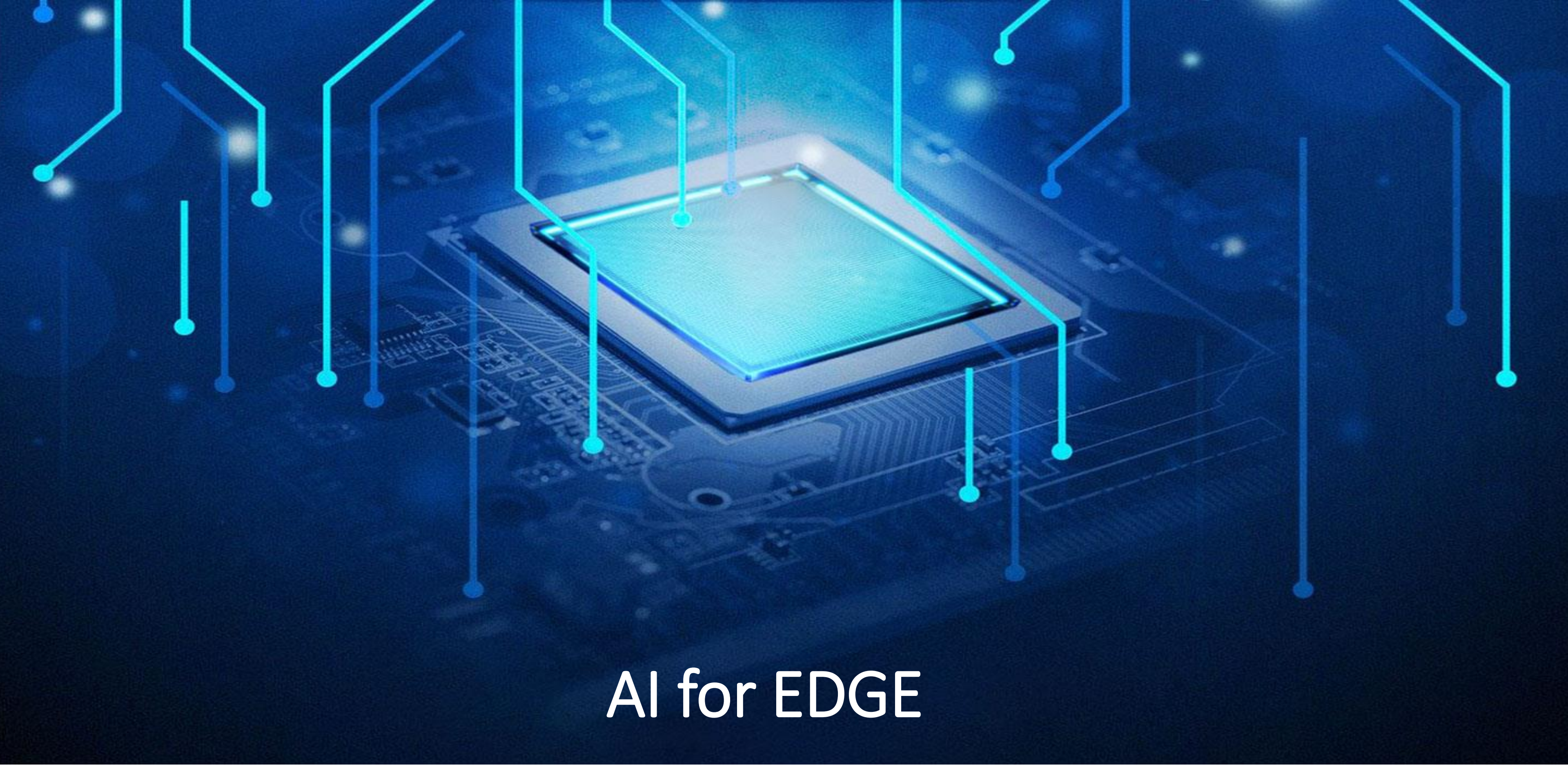


AI for EDGE

Machine learning process

Outline

- Machine learning process
- Model development
- Optimization
- Model deployment
- Transfer learning

Machine learning process

The machine learning process is all the activities performed, from defining the problem that our model will solve, to its implementation.

- Define the problem
- Data acquisition
- Data cleaning
- Exploratory Data Analysis (EDA)
- Feature engineering
- Model development
- Deployment

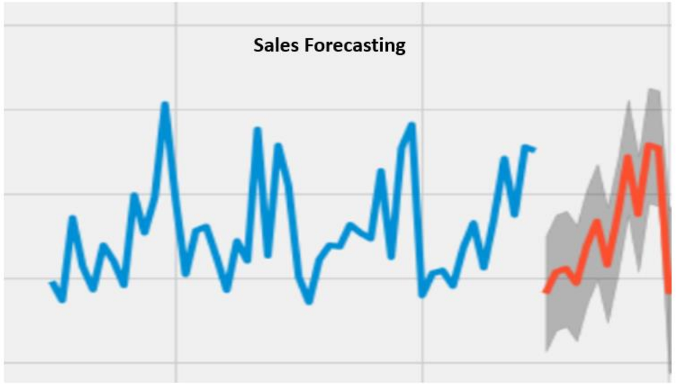
The first five points usually take up around 70% of the total machine learning time

Define the problem

At this stage, we identify the problem that we are supposed to solve and select the task that our model is to fulfill.

Example 1

- **Problem**: Recognition of plants in bed segments.
- Machine learning problem: multi-label classification, because many types of plants can be planted in one part of the bed



Example 2

- **Problem**: Prediction of the monthly profit from sales in a home electronics store.
- Machine learning problem: regression, because we are trying to predict the profit for the next month

Data acquisition

At this stage, we assess data availability and quality. If the data is insufficient, we obtain it.

Example 1

- **Problem**: Recognition of plants in bed segments.
- Data acquisition:
  - taking pictures of flower bed segments from above using a drone
  - manual labeling of photos with the classes of individual plants

Example 2

- **Problem**: Prediction of the monthly profit from sales in a home electronics store.
- Data acquisition:
  - using historical data on profit from sales
  - labels (monthly profits from sales) in this case are already ready

Sales Forecasting

Data cleaning

At this stage, we check whether the data does not have values that strongly deviate from the norm, i.e. the so-called outliers. These values are often the result of errors in the data acquisition process. Getting rid of outliers is very important, because their presence can fool ML model

Example 1

- **Problem**: Recognition of plants in bed segments.
- Data cleaning:
  - removal of images that do not include flower beds (drone program error)
  - resetting values in images that are outside the allowed range (depending on how images are saved)

Example 2

- **Problem**: Prediction of the monthly profit from sales in a home electronics store.
- Data cleaning:
  - replacing negative sales values with the average/median of adjacent months
  - replacing extremely high sales values (above a realistic threshold) with the average/median of the adjacent months

Sales Forecasting

Exploratory Data Analysis (EDA)

The process of visualizing and calculating various statistical indicators to better understand the available data.

Example 1

- **Problem**: Recognition of plants in bed segments.
- EDA:
  - comparison of the number of examples for each class (balanced/unbalanced problem)
  - visualization of images
  - calculation of the average pixel value in the image, median and standard deviation
  - determination of class covariance images

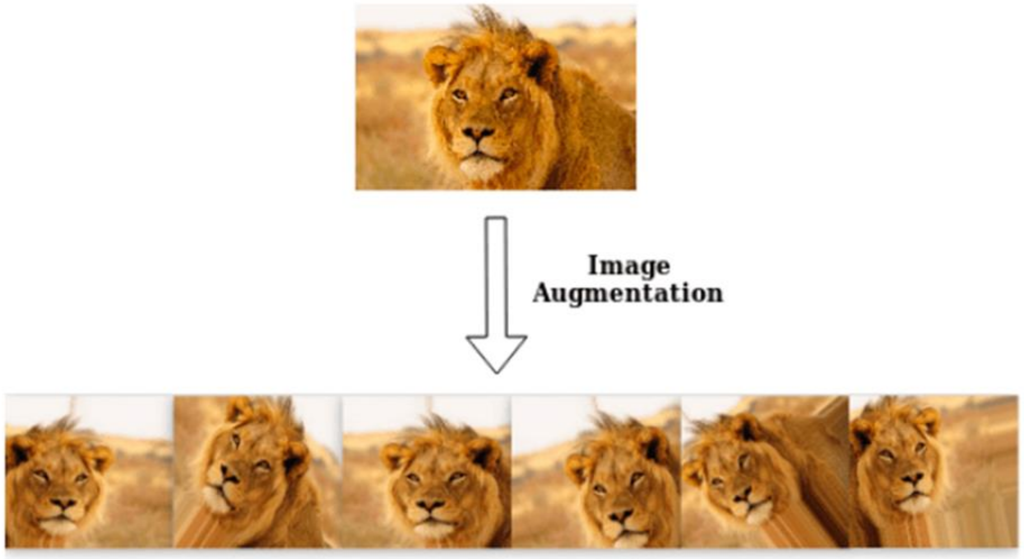
Example 2

- **Problem**: Prediction of the monthly profit from sales in a home electronics store.
- EDA:
  - visualization of profit curves with other feature values (e.g. number of customers in a month) to find correlations
  - calculation of the average value of profit, median and standard deviation
  - compare the amount of available data for each month

Feature Engineering

Feature Engineering is the stage in which we select specific numerical values (features) on the basis of which our model will make predictions. In addition, we decide on the transformations and augmentations that they will undergo before the training process.

Augmentation is the process of tweaking data with various manipulations to produce more training examples. In the case of time series, an example of augmentation may be the addition of random noise, and in the case of images, their rotation.



Feature Engineering

Feature Engineering is the stage in which we select specific numerical values (features) on the basis of which our model will make predictions. In addition, we decide on the transformations and augmentations that they will undergo before the training process.

Example 1

- **Problem**: Recognition of plants in bed segments.
- Feature engineering:
  - input values will be three-channel images (e.g. RGB)
  - global normalization of images to <-0.5, 0.5> range
  - augmentation by rotation and noise

Example 2

- **Problem**: Prediction of the monthly profit from sales in a home electronics store.
- Feature engineering:
  - the input characteristics will be the values of profit and number of clients from the last 6 months (rolling window technique)
  - global normalization of features to the range <-0.5, 0.5>
  - augmentation by noise

Model development

At this stage, we perform 4 steps:

1. Algorithm selection (e.g. decision tree, neural network)
2. Selection of metrics
3. Training and evaluation
4. Repeat step three as you change hyperparameters and add additional techniques (until you are satisfied with the results)

Model development

Example 1

- **Problem**: Recognition of plants in bed segments.
- Model:
  - Architecture: Convolutional neural network with ReLU and sigmoid activation functions
  - Optimizer: Adam
  - Cost function: Binary cross entropy (multi-label classification)
  - Metrics: PR AUC (assume the problem is imbalanced)

Example 2

- **Problem**: Prediction of the monthly profit from sales in a home electronics store.
- Model:
  - Architecture: feedforward neural network with ReLU activation and identity functions
  - Optimizer: Adam
  - Cost function: Mean squared error
  - Metrics: Mean squared error

Imbalanced Classification

The class balance problem must be addressed before training, otherwise the model will always favor the majority class

The method to solve the problem is to weight the loss function for the classes.

$$loss = \frac{1}{n}\sum_{i=1}^{n} -y_i log\hat{y}$$

will change form to

$$loss = \frac{1}{n}\sum_{i=1}^{n} -y_i log\hat{y} * w_c$$

where w_c is a vector with weights for each class

Multiplying the loss function for individual classes by their weights will affect the size of parameter changes resulting from the presence of individual classes.

Imbalanced Classification

The weights for each class can be calculated from the formula

$$w_c = \frac{num\_examples}{num\_classes * num\_samples\_per\_class}$$

After applying the above formula, the majority class will have a lower weight than the minority class, which will result in larger parameter changes for minority class examples.

Overfitting

- Overfitting is a common problem in neural network training. A common feature of overfitting is the presence of large values of weights in the network.
- A few techniques that help combat this phenomenon are:
  - data augmentation (increasing the amount and variety of data)
  - regularization of weights
  - limiting the range of weights
  - dropout

Weight regularization

- Weight regularization is a method that promotes low absolute values of weights by directly affecting the cost function.
- The three most common types of regularization are:
  - **L1**
    loss = loss + α*($|w1|$ + $|w2|$ + $|w3|$...)
  - **L2**
    loss = loss + α*($w1^2 + w2^2 + w3^2$ ...)
  - **L1L2**
    loss = loss + $α_1$*($|w1|$ + $|w2|$ + $|w3|$...) + $α_2$*($w1^2 + w2^2 + w3^2$ ...)
- In Keras, regularization can be applied via the "kernel_regularizer" layer argument.
- **Regularization promotes small weights but does not guarantee them.**

Limiting the range of weights

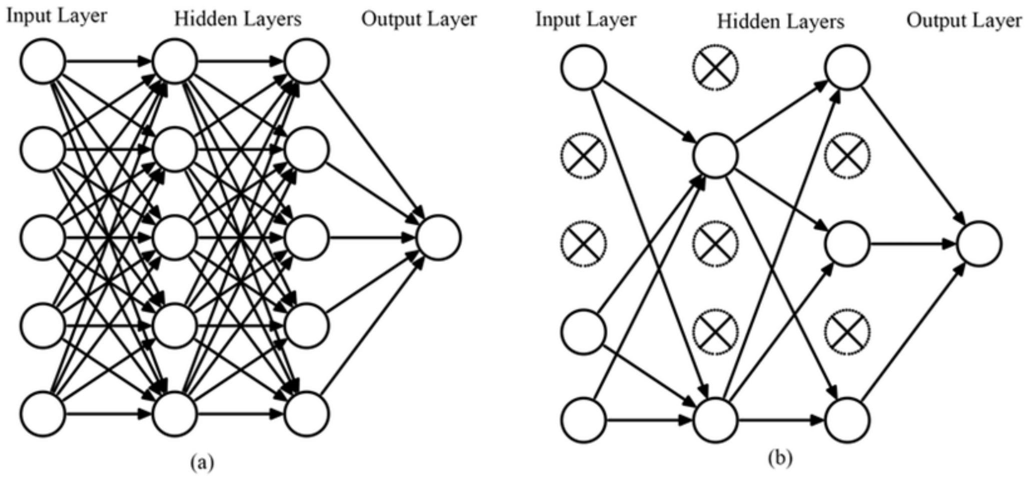
- The weight constraints available in Keras are usually based on the length of the weight vector in the weight space (vector norm):

$$\boldsymbol{Norm} = \sqrt{\boldsymbol{w_1}^2 + \boldsymbol{w_2}^2 + \boldsymbol{w_3}^2 \ldots}$$

- Constraints guarantee keeping the weights small.
- Typical limitations are:
  - **MaxNorm** - scales the weights after exceeding the maximum value by their norm
  - **MinMaxNorm** - rescales weights after exceeding the maximum or minimum value by their norm
  - **UnitNorm** - scales the weights to have a norm equal to 1
  - **NonNeg** - trims negative weights to zero

Dropout

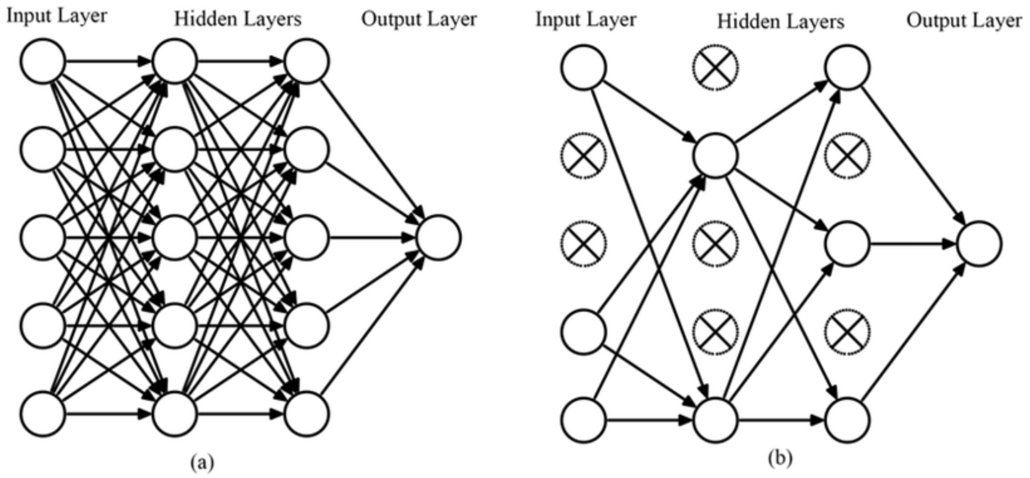
- Dropout consists in randomly switching off neurons during learning by resetting their output values to zero.
- It is used by placing the Dropout layer after the layer we want to influence.
- Resetting the values takes place at the stage of generating the network response to batch examples and results in resetting the gradients for specific parameters (prevents their frequent changes).



Dropout

- Dropout accepts a number from 0 to 1 as input, specifying how many neurons from the previous layer should be turned off.
- Zeroing the responses of neurons causes weaker activation of the next neurons, so Dropout in Keras, in addition to resetting, also scales non-zeroed responses using the formula:

$$y = \frac{1}{1 - rate}$$



Batch normalization

- Batch normalization is a technique of standardizing inputs to a given layer. Standardization scales a set of values so that they have a mean of 0 and a standard deviation of 1.

- Standardization equation:

$$z = \frac{X - u}{\sigma}$$

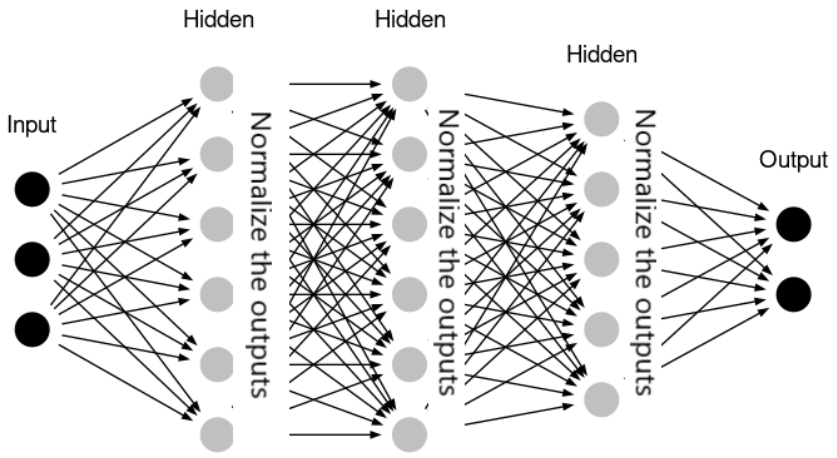
where u is the mean and σ is the standard deviation.

- In Keras, batch normalization is available as a layer that we place between layers in the network.
- During training, standardization is done for each training iteration with mini-batches of different training examples.
- After training, the values are standardized using the average u and σ values of all training iterations.

Batch normalization

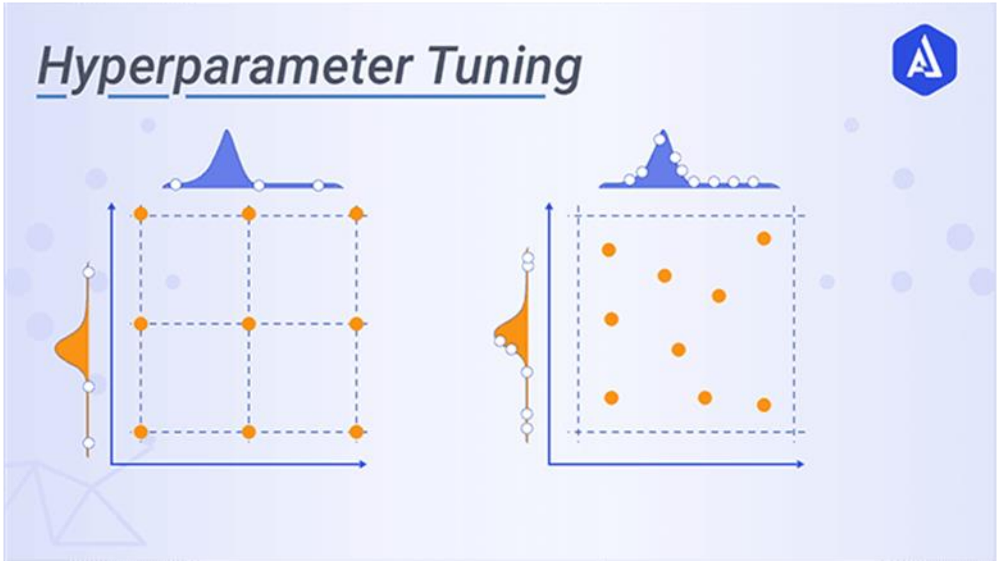
- Batch normalization stabilizes and accelerates training, which allows the use of higher learning rates in learning algorithms!

- Do not apply dropout and batch normalization layers after the last layer in the network, because it does not make any sense!
  Both will then distort the predictions of the network.



Hyperparameter tuning

- In large projects, the search for optimal hyperparameter values is often automated.
- For this purpose, special tools are used to search the hyperparameter space.
- Example tools include:
  - SMAC
  - BOHB
  - HyperBand



Deployment

- The last stage of the machine learning process concerns the implementation of the trained model on the target hardware platform.
- It can be a cloud server, a desktop computer or even an edge device.
- From a TensorFlow perspective, a graph can be loaded directly into an application that uses Python, C++, C, Java, C#, etc.
- It can also be used in a TensorFlow Serving application that devices or programs can communicate with.

Transfer learning

- Resnet, vgg, and other ILSVRC competition-winning models are available off-the-shelf in Tensorflow Keras with the appropriate parameters.
- Assuming that some visual features that these models have learned to extract are common to many images, we can try to use them as a starting point for other tasks.
- Using the finished model from one task as a starting point for another task is known as transfer learning.

Deployment

This stage concerns the implementation of the trained model on the target hardware platform

Example 1

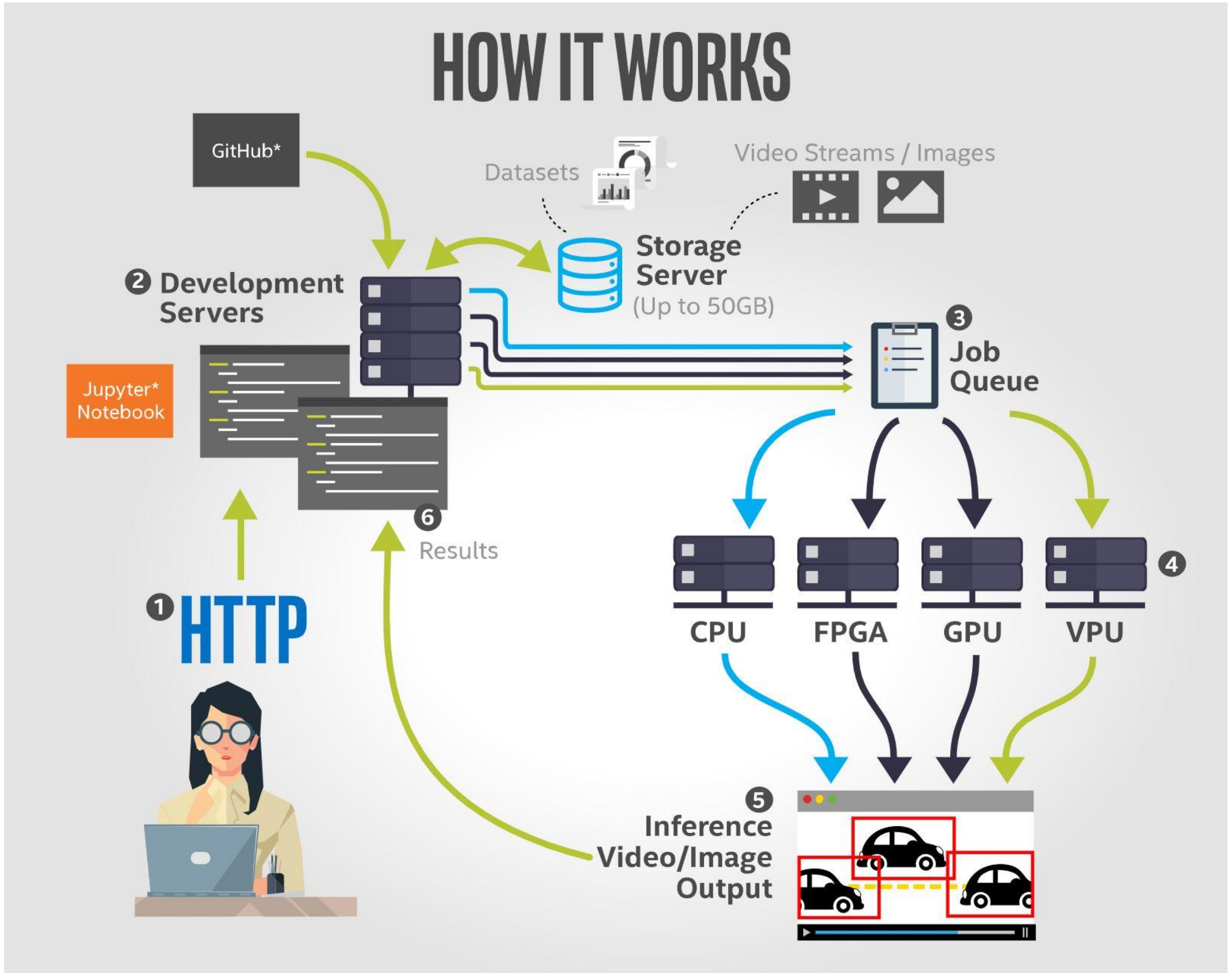
- **Problem**: Recognition of plants in bed segments.
- Deployment: Edge device
  - TensorFlow Lite
  - Raspberry Pi

Example 2

- **Problem**: Prediction of the monthly profit from sales in a home electronics store.
- Deployment: Edge device
  - TensorFlow
  - OpenVINO
  - Intel DevCloud
  - UP Squared AI Vision X

Sales Forecasting

Intel DevCloud For The Edge



Intel DevCloud for the Edge lets developers prototype computer-vision solutions in Intel's cloud environment and watch their code run on any combination of its available hardware resources. (Source: Intel)

Related topics

https://www.tensorflow.org/guide

Post-Training Quantization Best Practices
https://docs.openvino.ai/latest/pot_docs_BestPractices.html#doxid-pot-docs-best-practices

Movidius Myriad X VPU
https://www.aaeon.ai/eu/product/detail/ai-core-x