The edge can be used where low latency is necessary, or where the network itself may not always be available. The use of it can come from a desire for real-time decision-making in certain applications.

Many applications with the cloud get data locally, send the data to the cloud, process it, and send it back. The edge means there's no need to send to the cloud; it can often be more secure (depending on edge device security) and have less impact on a network. Edge AI algorithms can still be trained in the cloud, but get run at the edge.

For Example,
Voice assistants send your query to the **cloud for processing**, while most **self-driving cars** need to be able to **perform their computations at the edge.** Additionally, while gathering insights from millions of sales transactions probably is fine to use the higher compute available in the cloud, a remote nature camera may not be able to always send its data over a connection.

Why is AI at the edge is Important:
Network Impacts:
Network communication can be expensive (bandwidth, power consumption, etc.) and sometimes impossible (think remote locations or during natural disasters).

Latency Consideration:
Real-time processing is necessary for applications, like self-driving cars, that can't handle latency in making important decisions.

Security Concern:
Edge applications could be using personal data (like health data) that could be sensitive if sent to cloud.

Optimization for local inference:
Optimization software, especially made for specific hardware, can help achieve great efficiency with edge AI models.

The OpenVINO Tool Kit:
The OpenVINO™ Toolkit's name comes from "**Open V**isual **I**nferencing and Neural **N**etwork **O**ptimization". It is largely focused around optimizing neural network inference, and is open source.

It is developed by Intel®, and helps support fast inference across Intel® CPUs, GPUs, FPGAs and Neural Compute Stick with a common API. OpenVINO™ can take models built with multiple different frameworks, like TensorFlow or Caffe, and use its Model Optimizer to optimize for inference. This optimized model can then be used with the Inference Engine, which helps speed inference on the related hardware. It also has a wide variety of Pre-Trained Models already put through Model Optimizer.

By optimizing for model speed and size, OpenVINO™ enables running at the edge. **This does not mean an increase in inference accuracy** - this needs to be done in training beforehand. The smaller, quicker models OpenVINO™ generates, along with the hardware optimizations it provides, are great for lower resource applications. For example, an IoT device does not have the benefit of multiple GPUs and unlimited memory space to run its apps.

In OpenVINO™, **Pre-Trained Models refer specifically to the Model Zoo,** in which the Free Model Set contains pre-trained models already **converted using the Model Optimizer.** These models can be used directly with the Inference Engine.

Pre-trained Model:
https://software.intel.com/en-us/openvino-toolkit/documentation/pretrained-models

Types of Computer Vision Models:
Classification determines a given "class" that an image, or an object in an image, belongs to, from a simple yes/no to thousands of classes. These usually have some sort of "probability" by class, so that the highest probability is the determined class, but you can also see the top 5 predictions as well.

Detection gets into determining that objects appear at different places in an image, and oftentimes draws bounding boxes around the detected objects. It also usually has some form of classification that determines the class of an object in a given bounding box. The bounding boxes have a confidence threshold so you can throw out low-confidence detection.

Segmentation classifies sections of an image by classifying each and every pixel. These networks are often post-processed in some way to avoid phantom classes here and there. Within segmentation are the subsets of **semantic segmentation** and **instance segmentation**:
Semantic Segmentation - The first where in all instances of a class are considered as one.
**instance segmentation - ** Consider separates instances of a class as separate objects.

Read:
https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484

Case Studies in Computer Vision:
To Do: Arch Understanding
SSD
ResNet
MobileNet

Loading Pre-Trained Models:
Human Pose Estimation - human-pose-estimation-0001
Text Detection - text-detection-0004
Determining Car Type & Color - vehicle-attributes-recognition-barrier-0039

Optimizations on the Pre-Trained Models:
Different Precisions (Quantization)are related to floating point values - less precision means less memory used by the model, and less compute resources. However, there are some trade-offs with accuracy when using lower precision.

Fusing Layer into multiple Operation, Where multiple layers can be fused into a single operation. These are achieved through the Model Optimizer in OpenVINO™, although the Pre-Trained Models have already been run through that process.

Pre-Processing Input:
There are a number of pre-processing steps we might wish to carry out during inference. In this lesson resize, transpose and reshape are explained.

Why Resize:
In order to get the prediction of the CNN network, we need to read & pre-process the input image in the same way(as training). We can get this information from the Pre-trained model zoo.

Why transpose:
i.e human-pose-estimation-0001 - OpenVINO accept the input [Batch * Channel * Height * Width]. But What we have from the cv2.resize() method is [Height-0 * Width-1 * Channel-2] to covert to OpenVINO format we do image.transpose(C-2, h-0, W-1).

Why Reshape:
Still, We need to add Batch Size equal to 1 but the batch size could be any positive number. So, We do image.reshape(1, Channel, Height, Width).

```
def preprocessing(input_image, height, width):
    '''
    Given an input image, height and width:
    - Resize to height and width
    - Transpose the final "channel" dimension to be first
    - Reshape the image to add a "batch" of 1 at the start
    '''
```

```
    image = cv2.resize(input_image, (width, height))
    image = image.transpose((2,0,1))
    image = image.reshape(1, 3, height, width)

    return image
```

Handling Network Outputs:

Classification networks typically output an array with the softmax probabilities by class; the argmax of those probabilities can be matched up to an array by class for the prediction.

Bounding boxes typically come out with multiple bounding box detections per image, which each box first having a class and