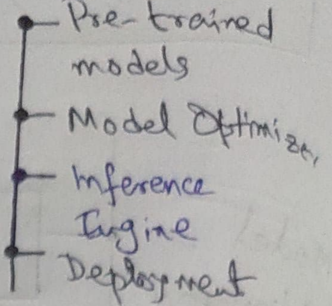
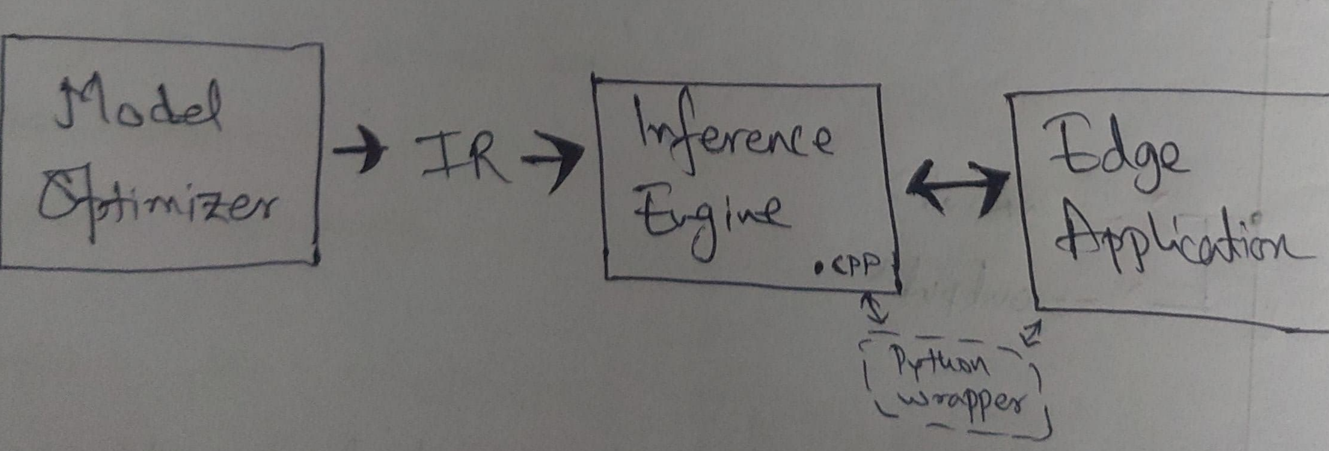
4 THE INFERENCE ENGINE



- The Inference Engine runs the actual inference on the model.

- It only work w/ models that are:
  - already in the IR format as a pre-trained model, ~~models~~ OR
  - models converted to the IR format from the model optimizer.

The Inference Engine provides a library of CV functions needed for ~~the~~ inference.



Devices Supported by the Inference Engine

1. CPUs ~~&~~
2. GPUs
3. FPGAs or Field Programmable Gate Arrays
4. VPUs or Visual Processing Units

   Eg: Intel Neural Compute Stick (NCS)

* All these need to be Intel hardware only.

Using the Inference Engine with an IR

- Library
  - openvino.inference_engine
  - Class — IECore
  - Class — IENetwork

- ~~IE~~ IECore is the Python wrapper to work w/ the Inference Engine.
- IENetwork will initially hold the network & get loaded into IECore.

Steps to load an IR into an IE

1. Get the IR model (.xml) and weights (.bin), ~~as~~ and pass them to the program.

2. In the program, import the openvino IE library, and from it, the classes IECore and IENetwork.

3. Create an IECore instance (say iecore)
4. Create an IENetwork instance (say net)
5. Add a CPU extension if needed
6. Get the supported layers of the network
7. From this list, check and see if there is any unsupported layer by comparing with all layers from net.layers.keys(). If any gets found, perform proper handling of this error

8. Finally, load the iecore network by providing ~~to~~ the net and device name.

Methods used in each step (refer docs)

3. IECore()
4. IENetwork(...)
5. iecore.add_extension(...)
6. iecore.query_network(...)
7. net.layers.keys(), exit(...)
8. iecore.load_network(...)

Sending Inference Requests to the IE

- The load_network method of IE Core class returns an ExecutableNetwork object.
- The inference requests are made to this object.
- Types of requests:

	Synchronous	Asynchronous
(i)	the app sits and waits for the inference.	~~The~~ The app can perform other tasks while making the inference.

(ii)	Method used:
	infer(...)
		Methods used:
		start_async(...)
		wait()

(iii)	The main thread is "blocked".	Does not block any thread as the response may be slow.
(iv)	The next frame is not gathered until the current frame's request is complete.	Sends a frame for inference while processing goes on in the next frame waiting f inference result.

Handling Results & App Integration

- Attributes of InferRequest instances:

  (i) inputs → the image frame

  (ii) outputs → the results

  (iii) latency → the inference time of current request

- All inference requests are stored in a requests attribute in ExecutableNetwork.

- To fetch an o/p:

  exec_net.requests[request_id]. outputs[output_blob]

- For app integration, we would need all the concepts learned so far, and more.

Summary

- Inference Engine
  - performs inference on models in the IR format
- Supported Devices
  - Intel proprietary hardware
  - Eg. CPUs, GPUs, FPGAs, VPUs (like NCS)
- Using the IE
  - library: `openvino.inference_engine`
  - classes
    - `IE Core`
    - `IE Network`
  - methods: `IECore()`, `IENetwork()`, `iecore.add_extension`, `iecore.query_network(...)`, `net.layers.keys()`, `ie core.load_network`
- IE requests
  - Synchronous → `infer(..)`
  - Asynchronous → `start_asyn(...)`, `wait(..)`
- Handle Results
  - `inputs` → the image frame
  - `outputs` → the results
  - `latency` → the inference time of current requests