Procedure to Run an AI (CNN dogs vs cats) on KV260 with Burn

1 Model Creation and Training

1. Choose a model:
   a. Example: a simple CNN (MobileNet, lightweight ResNet, or a custom model).

2. Prepare the data:
   a. Organize images into two folders: `cat/` and `dog/`.
   b. Create training, validation, and test datasets.

3. Train the model (on PC):
   a. With TensorFlow/Keras (`.h5`) or PyTorch.
   b. Adjust hyperparameters (batch size, epochs, learning rate).

4. Evaluate performance:
   a. Check accuracy and loss on the test set.
   b. At this stage, you have a working model... but only usable on your PC.

---

2 Model Conversion

1. **Export to ONNX**
   - a. If you have a `.h5`, convert it to `.onnx`
   - b. Install ONNX:
     1. pip install onnx
2. Check ONNX compatibility:

```
python3 - <<'PY'
import onnx
m = onnx.load("models/dog_vs_cat.onnx")
print("IR version:", m.ir_version)
print("Opset imports:", [(d.domain, d.version) for d in m.opset_import])
print("First nodes:", [n.op_type for n in m.graph.node][:30])
PY
```

- If you see exotic operations, check whether they are supported by Burn.
- Otherwise, re-export your model with another opset or adjust its structure.

---

3 Preparing the KV260

1. Clone Burn:
   - `git clone https://github.com/tracel-ai/burn.git`
   - `cd burn`

2. Install dependencies:
   - Rust + Cargo (build tool).
   - Update the KV260 (internet required).

3. Organize project files:
   - Create a folder for your model (`model/dog_vs_cat.onnx`).
   - Create a folder for test images (`images/`).

---

4 Code Generation with onnx2burn

1. Install the Burn ONNX import tool:
   - `cargo install burn-import`

2. Convert your model:
   - `burn-import onnx models/dog_vs_cat.onnx --out generated_model`

3. This generates Rust code (`generated_model.rs`) that can be used directly.

---

5 Rust Project Creation

1. Create a new project:

```
cargo new dog_vs_cat_burn --bin
cd dog_vs_cat_burn
```

2. Copy the generated code (`generated_model/*`) into `src/`.

3. Edit your `Cargo.toml`:

```
[dependencies]
burn = "0.18"
burn-ndarray = "0.18"
image = "0.25"   # for image loading and preprocessing
```

---

6 Inference Implementation (`main.rs`)

Minimal example structure:

```
mod generated_model; // code produced by onnx2burn
use generated_model::MyModel; // name depends on the generator

fn main() -> Result<(), Box<dyn std::error::Error>> {
    // 1) load and preprocess the image (use image crate)
    // 2) convert into Burn tensor
    // 3) load weights (if necessary: generated_model::load_state("..."))
    // 4) run inference
    // let pred = model.forward(input_tensor);
    // 5) display label
    Ok(())
}
```

7️⃣ Compilation and Execution

On the KV260:

```
cargo build --release
./target/release/dog_vs_cat_burn images/test1.jpg
```

- ⚠️ Compilation on ARM can be slow.
- If needed, cross-compile from your PC to ARM (more complex to set up).

8️⃣ Debugging and Common Errors

- ONNX import error: unsupported operation → simplify your model or use onnxruntime instead.
- Preprocessing error: wrong shape or channel order → ensure Rust preprocessing matches your Python pipeline.
- Very slow build on KV260: possible → use `cargo build --release` or cross-compilation.