**Machine Learning project**

Goal in this project is to build and end-to-end AI/ML deployment on mobile device, in this assignment I use CSC puhti supercomputer to run my code and train my model. After I am done with this project, I have knowledge of how to use Neural Networks in machine learning and also how to apply those in android applications.

Steps for this project

1. **Define & Pla**

The application I selected is to help people to see how healthy they are eating; in my application you can take pictures using your phone’s camera and then model scan image and see what foods there is and then it tells you how healthy that food is. There is many of these kinds of applications already and many health apps are using this already, but I try eating healthy again, so with this app I can look my eating habits and also this is something that is interesting for me and also practical.

**Input:** Image

Image is taken by phone camera in application

**Output:** Food nutrition – Healthy food and not healthy food

**Version 1** I have simple binary output so I just show if my model can classify if food in image is healthy when nutrition’s are certain and unhealthy if they are not

**Success metric:** Accuracy

Success metrics are how accurately model can classify if food in image is healthy or unhealthy food, later when I do more versions, I can add that it’s also looking nutrition and metrics

\*\*Model\*\*

My pretrained model will be **MobileNetV4** because I use vision/camera in my application.

**MobileNet** is a type of **convolutional neural network (CNN**), and it is designed for image classification, object detection and other computer vision tasks. MobileNet’s are designed for small sizes, and they have low power consumption, so they are good for android app. Version 4 was published in September 2024 and I am using this version in my project. It includes the 'universal inverted bottleneck,' which integrates both inverted residuals and inverted bottlenecks as special cases. This allows the model to efficiently use attention modules, including multi-query attention.

\*\*Data\*\*

Because time frame is short for this project and I have only 9 days I decide to use readymade datasets, I have images from [256-kind food datase](http://foodcam.mobi/dataset256.html) and I have around 31 395 images from different foods. This way I collect my image dataset fast and I get good quality images. I also have txt file that tells category and foods Japanese name and English name. This dataset can be used freely if it’s non-commercial research purpose and this project is like that for me.

My other dataset is CSV file that contains 555 rows of nutrition data from different foods like pesto, white beans, kimchi etc. License for this dataset is under [creative commons V4.0](https://creativecommons.org/licenses/by/4.0/)  and it’s free to use.

For this project I combine these datasets to classify if food is healthy or not healthy, I didn’t want to use the same images that are with nutrition data because I want little more challenge when I didn’t collect my images by myself, so I need now do more data cleaning when nutrition data and images are not from same place.

1. **Collect & Prepare Data**
   1. Record or capture raw data on your phone or other devices.
   2. Label the data (CSV, JSON, etc.).
   3. Augment / clean (e.g. image flips, audio gain, text normalization).
   4. Split into train / val / test
   5. Document the collection process. If you have used any tool, please document it.
2. **Train & Evaluate on CSC Puhti Supercomputer**
   1. **Fine‑tune the pretrained model** using your Slurm script. It means you will start training your selected pretrained model (in Step 1.3) with your collected dataset in Step 2 on CSC Puhti.
      1. Your pre-trained model is not specifically trained for the dataset collected for your application setting, so accuracy of the pre-trained model will not be high for your application. So, it is important to specialise this model by training this model on your selected dataset.
      2. *Please note that this (i.e. fine tuning the pre-trained model) is a very common step in the AI/ML domain, i.e. starting your training  on a pre-trained model. It can substantially reduce the overall amount of training you need to do.*
   2. **Log metrics** with TensorBoard or Weights & Biases.
   3. **Checkpoint the best epoch** based on validation accuracy or loss.
   4. **Profile GPU utilisation & time‑per‑epoch**; revise batch size/learning‑rate as needed.
   5. **Save the trained .pt model**
3. Optimize for Mobile Device
   1. **Apply optimization techniques on trained model (.pt) from Step 3 for memory optimization.**
      1. We have covered INT8 quantization only.
      2. Advanced courses have other techniques as well: pruning, distillation. Motivated learners may also implement these, though you are not expected to do so.
      3. You can see full list of model optimization techniques available in Pytorch here: <https://github.com/pytorch/ao>
   2. **Run experiments** and keep an ablation table of model size / latency / accuracy (including this in your final report).
   3. **Export the best model to ExecuTorch format**:
      1. import executorch
      2. executorch.export(model, sample\_input, "model.et")
   4. Verify the .et file loads and runs on x86 (ExecuTorch CPU) before moving to Android.

**Suggestion for this step (**Target after applying optimization**):**

1. ≤ 30 MB total trained model size (as much as lower is better). If the original model is already less than 30 MB, try to reduce  the size by at least 30% of the original model size. You are welcome to reduce even more than 30% if your time permits.
   1. E.g. If the original model size is let’s say 20 MB, 30% of 20 MB, means 6 MB, and after optimization, size of the optimized model should be 14 MB without losing too much accuracy. See the next point for accuracy.
2. ≥ 85 % of baseline accuracy (as much as higher is better). Baseline accuracy is the accuracy of the model before applying quantization.

After applying quantization, your model size will be reduced, and accuracy will also drop to some extent. You may train this reduced-size model again using your training data set.

1. Build the Android App
   1. **Create or clone** an Android Studio project (Kotlin or Java).
   2. **Add ExecuTorch dependency** in build.gradle:

implementation("org.pytorch:executorch-android:0.6.+")

Copy model.et to app/src/main/assets/.

* 1. **Implement sensor input pipeline** for your modality (CameraX, AudioRecord, SensorManager, etc.).
  2. **Wrap the model call**:

val etModule = ExecuTorchModule(assetManager, "model.et")

val output = etModule.run(inputTensor)

* 1. **Measure on‑device latency** (avg 30 runs) with Android Profiler.

1. Test & Demo
   1. **Test offline functionality** (Turn on flight mode on your phone while testing this, just like when you do when board an airplane).
   2. **Ensure APK size ≤ 30 MB**, try to get this value as low as possible without losing too much accuracy or incurring high latency during inference. If the original model is already less than 30 MB, try to reduce  the size by at least 30% of the original model size.
      1. Note that Google play Store limit is 100 MB.
   3. **Record a 1‑minute demo video** showing real‑time inference from mobile sensors.
      1. Data is collected and your ML model is doing inference in real time.

**Project deadline is 16 of May 2025**

References

MobileNet:

<https://en.wikipedia.org/wiki/MobileNet>

<https://docsaid.org/en/papers/lightweight/mobilenet-v4/>

<https://blog.roboflow.com/how-to-use-mobilenetv4-for-classification/>

Data

Food nutrition data: <https://github.com/google-research-datasets/Nutrition5k>

Food images: <http://foodcam.mobi/dataset256.html>