NAIC (AI Technical) team: CantByteUs

Team members:

Chin Zhi Xian
Ng Tze Yang
Ong Chong Yao
Terrence Ong Jun Han

```
What was your data collection process?:

# Clearly describe how you collected and prepared your dataset. Include the
following:

# How many images did you collect per kuih muih category?

# Where did the images come from?

# How did you ensure your dataset is diverse?
```

(1) Scraped 2500~ images per class from the internet using engines Bing and Google.
(2) Convert the images to tensors.
(3) Compute the tensor differences between images to find out the image similarities, thus removing duplicates effectively.
(4) Manually filtered unrelated images out (e.g. images with a huge YouTube logo on it), then combined them with some images taken by ourselves.
(5) Annotated a few of the kuih for segmentation using Label Studio, thus we used our genuine original dataset, as the existing kuih datasets in dataset sharing platforms were not in segmentation format.

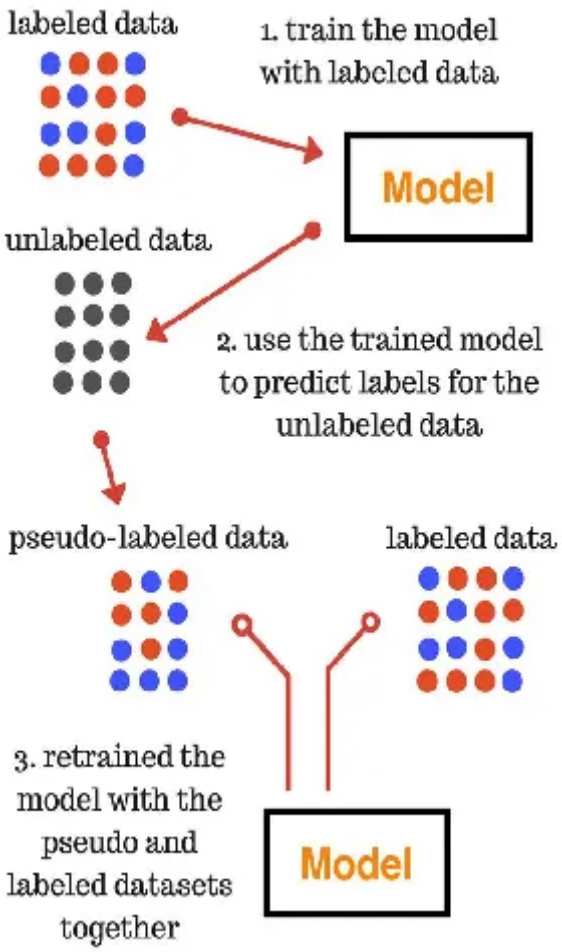
We also included high and low resolution images of kuih. Low-res images for the model to generalise better, and high-res for the attention layers to capture the minute detail. We also took images that contained lighting from awkward angles and of various intensities, as most of the photos online were taken in optimal lighting and framing (to be appealing for promotions and advertisements). This allowed us to have more variety in our data. Moreover, we also included kuih that were partially eaten.

Even Bing and Google search engines weren't really able to tell the difference between Kuih Lapis and Kek Lapis, and often got the two classes jumbled up too!

We used a dataset annotation & semi-supervised training method called **PSEUDO LABELLING**

For each class,
(1) 20-30 kuih were annotated depending on the kuih feature complexity
(2) We trained a small YOLOv11-seg of scale ‘n’ / ‘s’ segmentation model to assist & speed us up with the annotation process.
(3) The model to runs through the rest of the unannotated images in that class, and as it annotates for us, we decide whether to accept its annotation for that particular image.

(4) Trained a little bigger model combining those model-annotated images with the ones we manually annotated earlier.
(5) Repeat from step **(2)** until the entire dataset is completed.



Attached image explains pseudo-labelling:

Paper: `Lee, Dong-Hyun. (2013). Pseudo-Label : The Simple and Efficient Semi-Supervised Learning Method for Deep Neural Networks. ICML 2013 Workshop : Challenges in Representation Learning (WREPL). We propose the simple and efficient method of semi-supervised learning for deep neural networks. Basically, the proposed network is trained in a supervised fashion with labeled and unlabeled data simultaneously. For un-labeled data, Pseudo-Label s, just picking up the class which has the maximum predicted probability, are used as if they were true labels. This is in effect equivalent to Entropy Regularization. It favors a low-density separation between classes, a commonly assumed prior for semi-supervised learning. With De-noising Auto-Encoder and Dropout, this simple method outperforms conventional methods for semi-supervised learning with very small labeled data on the MNIST handwritten digit dataset.`

Eventually we were plateaued with a raw dataset of 98 images in each class that were perfectly annotated by the model for its respective class.

(6) Split per-class dataset into 90 images for train and 8 images for validation.

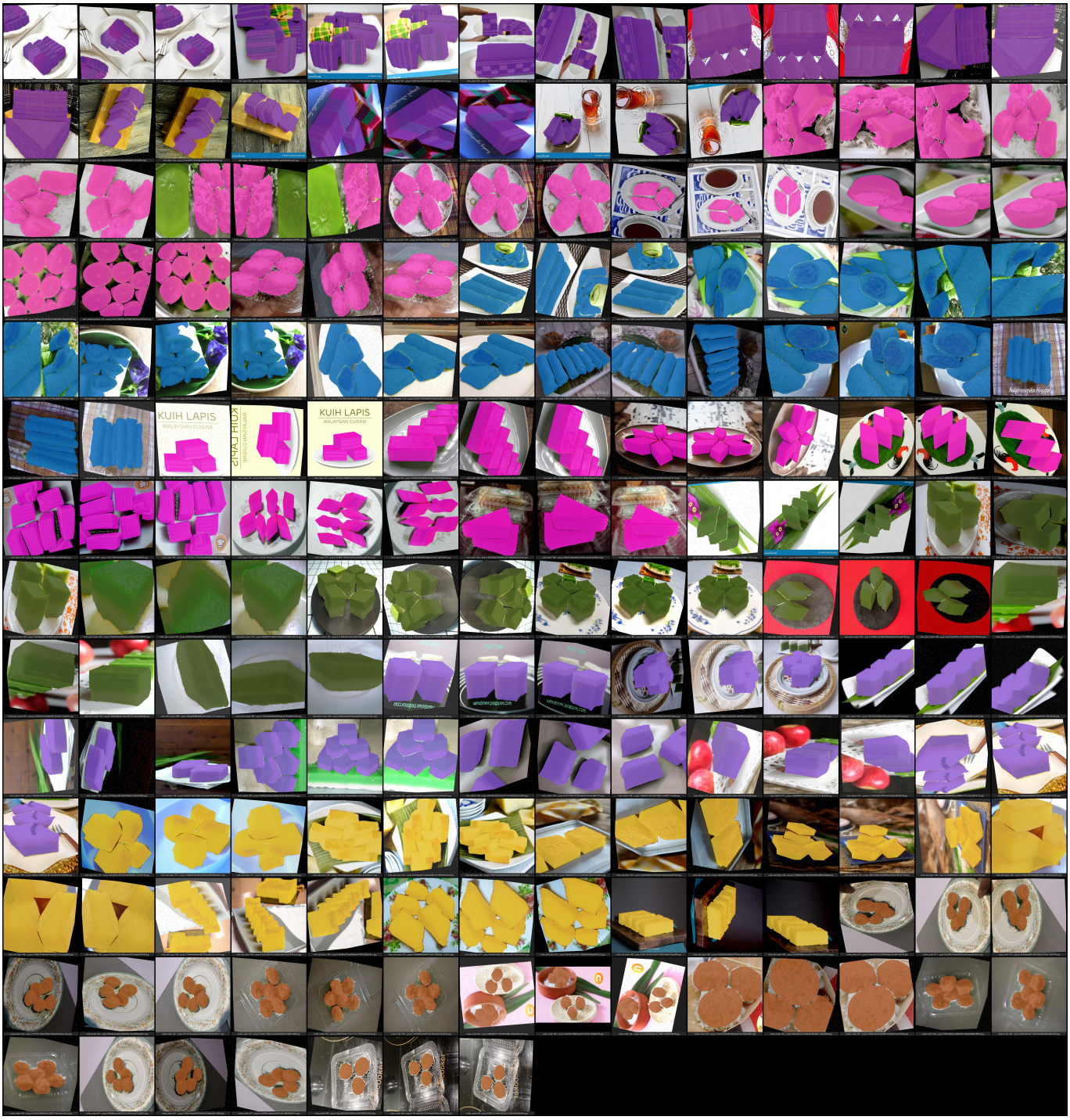
(7) We then uploaded all 784 ((90 + 8) × 8 classes) images into Roboflow for augmentation and artificially increase the dataset size. Effectively tripled the dataset size while applying augmentations.

HOWEVER we purposely left the hue and colour adjustment augmentations out because it will mess with how the model interprets the images, and due to kuih identification being very colour sensitive.

(8) Added a few non-kuih related I images with no labels into the training split to reduce False Positives

After all this, we also wrote a script to render all the segmentation annotations on top of the images and then place all of them into a grid to be neatly visualised.

Attached image below shows our rendered final validation split, the different mask colours representing the 8 different classes:




```
What was your model development process?:

# Clearly describe how you built and improved your model. Include the following:
```

```
# What models or tools did you use?

# What strategies or techniques did you try to improve performance?

# Did you try anything creative or unusual?
```

- Ran all the training and inference on our computers.
- Used CUDA to supercharge training by the GPU
- PyTorch as our training framework, and the Ultralytics library to save us a lot of coding for the metrics, loss, forward & backward propagation, etc.

Now here's the fun part:

(1) We directly edited the 'yolo11seg.yaml' model configuration file to twice the depth, width, and channel capacity of the YOLO11-seg models. We also tried adding more C3k2 blocks, increased the number of SPPF kernels, and added more C2PSA attention layers.

Those required too much computing power from our end. Yes, we tested: one epoch took 4+ hours and even the Nvidia P100 16GB VRAM GPU in Kaggle ran out of memory afterwards! Thus, we just stuck to adding more attention layers and keeping the rest as they were.

(2)
"Large models tend to overfit when trained with a small-to-medium-sized dataset"

Due to our kuih dataset being small, immediately after adding more attention layers, we trained a YOLOv11x-seg model from scratch on the full COCO 2017 dataset. Through a larger sample, the model can preconfigure all its neurons' weights and biases to be optimal, thus not too specific (un-generalisable) on the kuih dataset first to make finetuning on kuih a lot better. This is actually the main reason why pretrained models are so popular to be trained on top on, because the weights and biases will be configured nicely to prevent overfitting and are really adaptable to smaller dataset sizes.
So, we replicated that.

Normalising the exposure of test images before inference to get a more consistent light balance all over the image, giving the model less of a hard time. But that could be solved by training the model with images preprocessed to have different exposure levels.

Really, if we were given more time for R&D, our model would've been miles better at telling Malaysian traditional cookies apart!

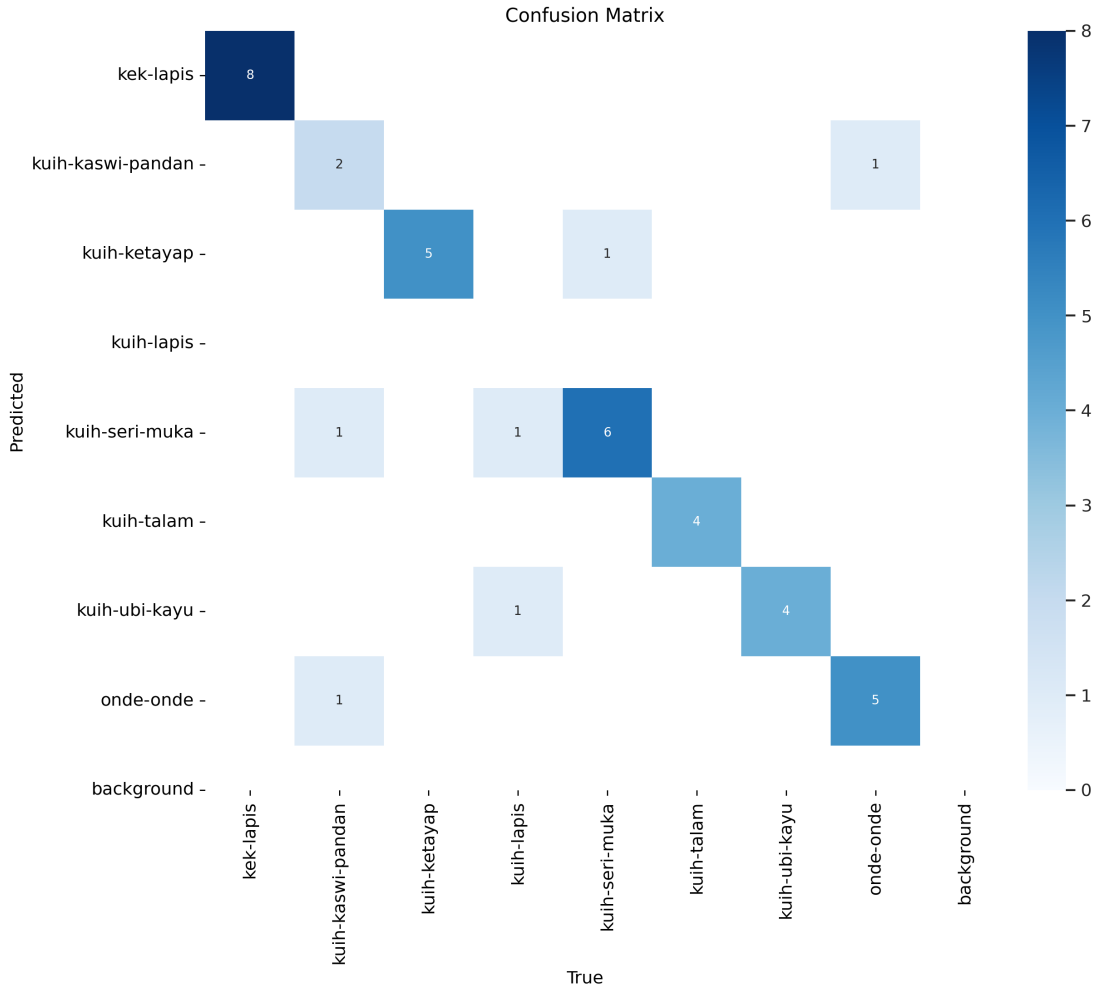
```
What is your final model and why did you choose it?:

# Clearly describe your final model and explain why you selected it. Include the
following:

# What algorithm or architecture did you use as your final model?

# Why did you choose it?
```

We chose an ensemble of a CNN segmentation model and also a Vision Transformer model.

For the CNN model:

Initially we did try to use classification models but the confusion matrix for the YOLOv11-cls models weren't at all that impressive:

Attached image shows the confusion matrix for YOLOv11m-cls model on a 50-images per class dataset:

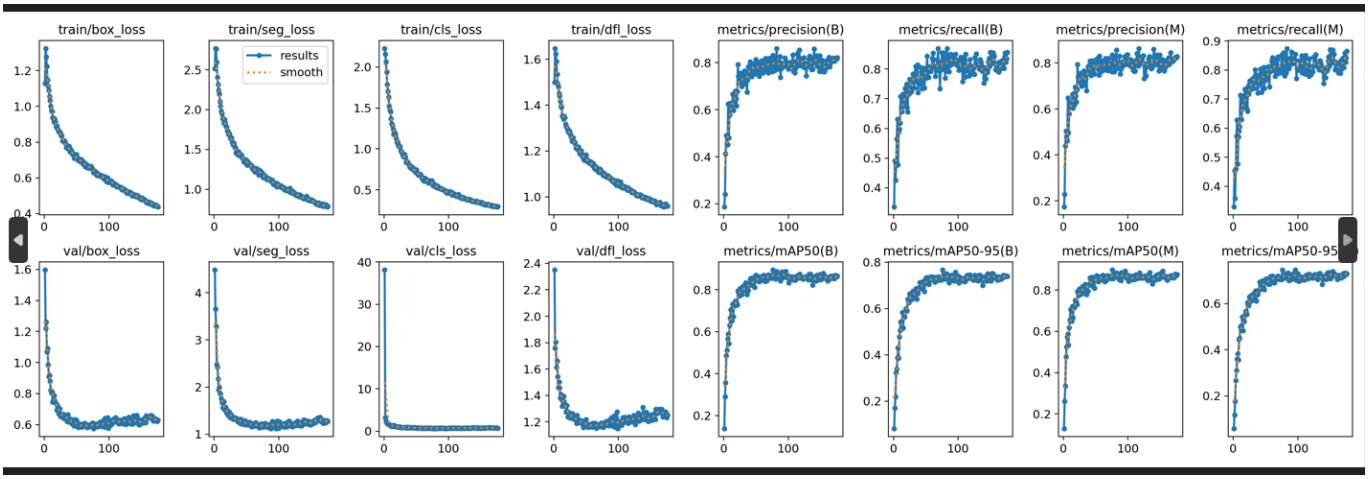


"A robust segmentation model inherently improves classification accuracy" - which is actually really true.

- Segmentation allowed the model to prioritize the core part of the image (the kuih) itself, reducing distractions from irrelevant background elements

We started training segmentation models:

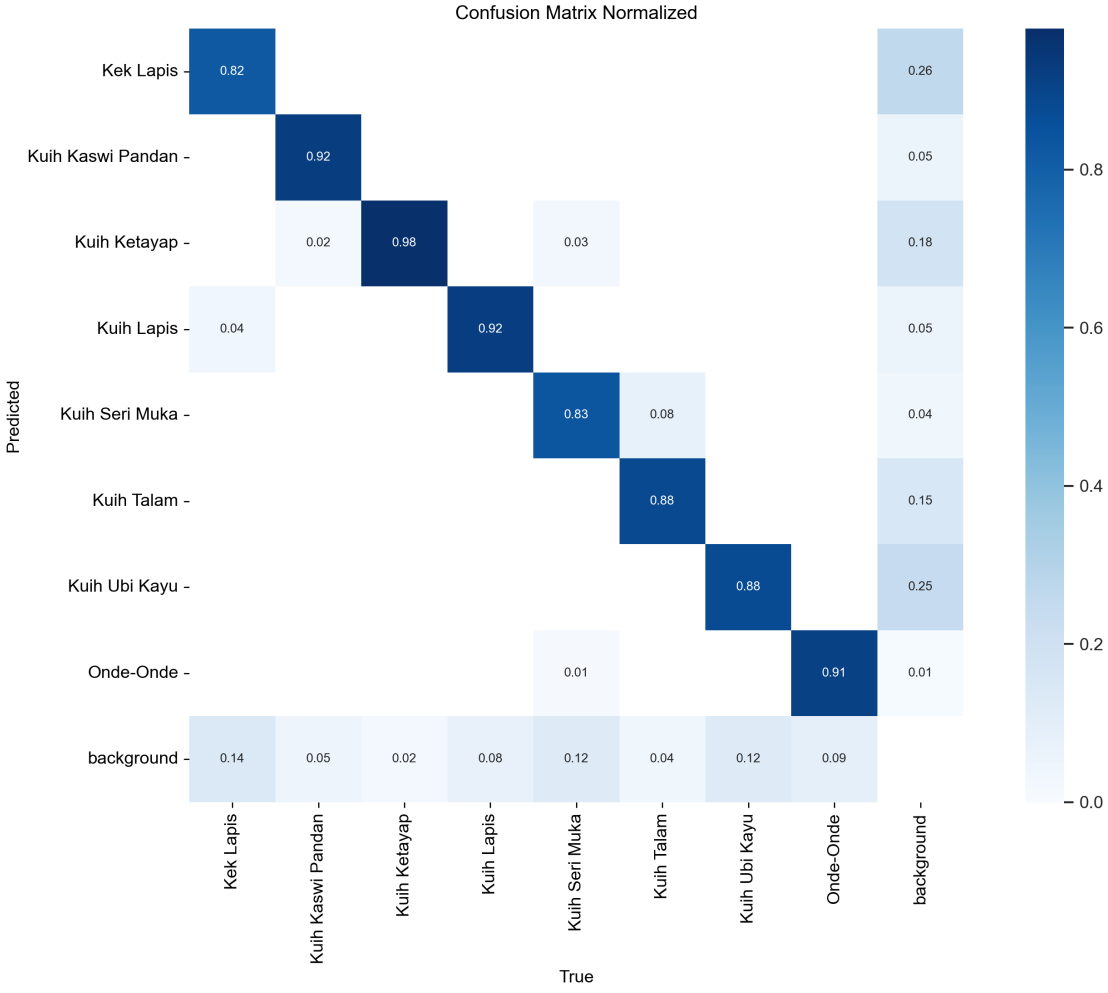
Attached image shows training & validation metrics for the YOLOv11x-seg model:



Notice how the cls_loss plummeted after only a few epochs?

True enough, the confusion matrix for the segmentation model was near-perfect.

Attached image shows the confusion matrix for YOLOv11x-seg model on an 8-images per class validation split:



For the Vision Transformer:

Vision Transformers split the input image into patches, and then "transform" the patches into tokens (like words in LLMs)

"ViTs can capture relationships across the entire image in every layer"

Kuih may look visually similar (looking at kek lapis-kuih lapis & kuih seri muka-kuih talam similarities) BUT they have different textures that cannot be easily identified when only looking at a certain part of the image.

ViTs use a 'self-attention' mechanism. That allows the model to relate long-distance relations between image regions, thus making for a suitable choice to classify kuih by **analysing the entire image together**. This means that even if a kuih looks slightly different across images, the ViT can still recognise it based on learnt patterns.

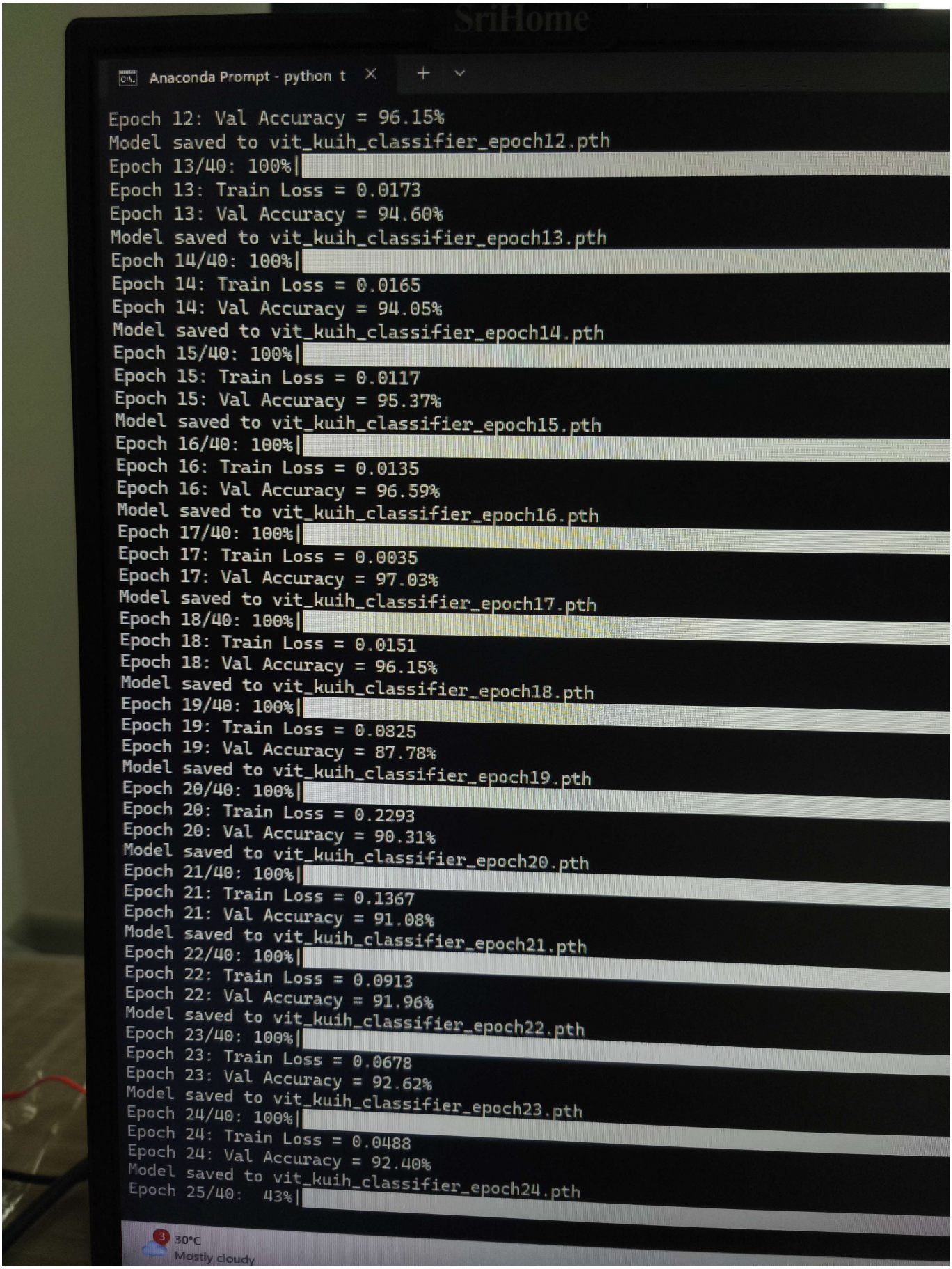
Vision Transformers perform well and more efficiently (even surpassing CNNs) only if they were pretrained on a large dataset.

We have chosen the EVA-02 model 'eva02_base_patch14_224.mim_in22k' (from Hugging Face and the Timm library) due to it being pretrained on the ImageNet 22k. Thus just like the CNN, is great at transfer learning over to smaller datasets

Attached image shows an instance of training the ViT:

Notice how fast the model converges even in the few few epochs?

Attached image below shows a separate instance of training the ViT:



ViTs plateau very fast (look at epoch 17). ViTs' fast convergence tends to bring the risk of overfitting too, thus we had to be careful and save the model at every epoch.

Finally, model ensemble

This is where we combined the outputs of the CNN and ViT to give an output.

We used a soft voting method to determine the final combined output of both models.

If both models agree with each other on the predicted class, then we just take that class.

If one model outputs a class (or NaN), another model the other, we will take the class with the highest confidence out of both models' predicted classes.

If both models give no output, then we will just leave that image blank.

This hybrid approach outperformed solo models in classification tests.