# Week 7 Report

# Wen Han Chia
# (Lizard Classification)

## Time-Log

Additionally, the time-log should include any work you've done for your *role* work (e.g., meeting management, web management, programs management, etc.)

## What did you do this week?

- Preliminary Data analysis and annotation
    - Reviewed best practices for resizing images for classification model training
        - Reviewed previous project lead's implementation done in TensorFlow
    - Annotated the rest of lizard images
- Familiarized with PACE-ICE cluster
    - Reviewed step-by-step guide to using PACE-ICE: 1) ssh into PACE cluster using IDE from local machine 2) Create virtual python environment 3) Best practices for storing huge files and running scripts

## What are you going to do next week

- Train models on PACE

## Blockers, things you want to flag, problems, etc.

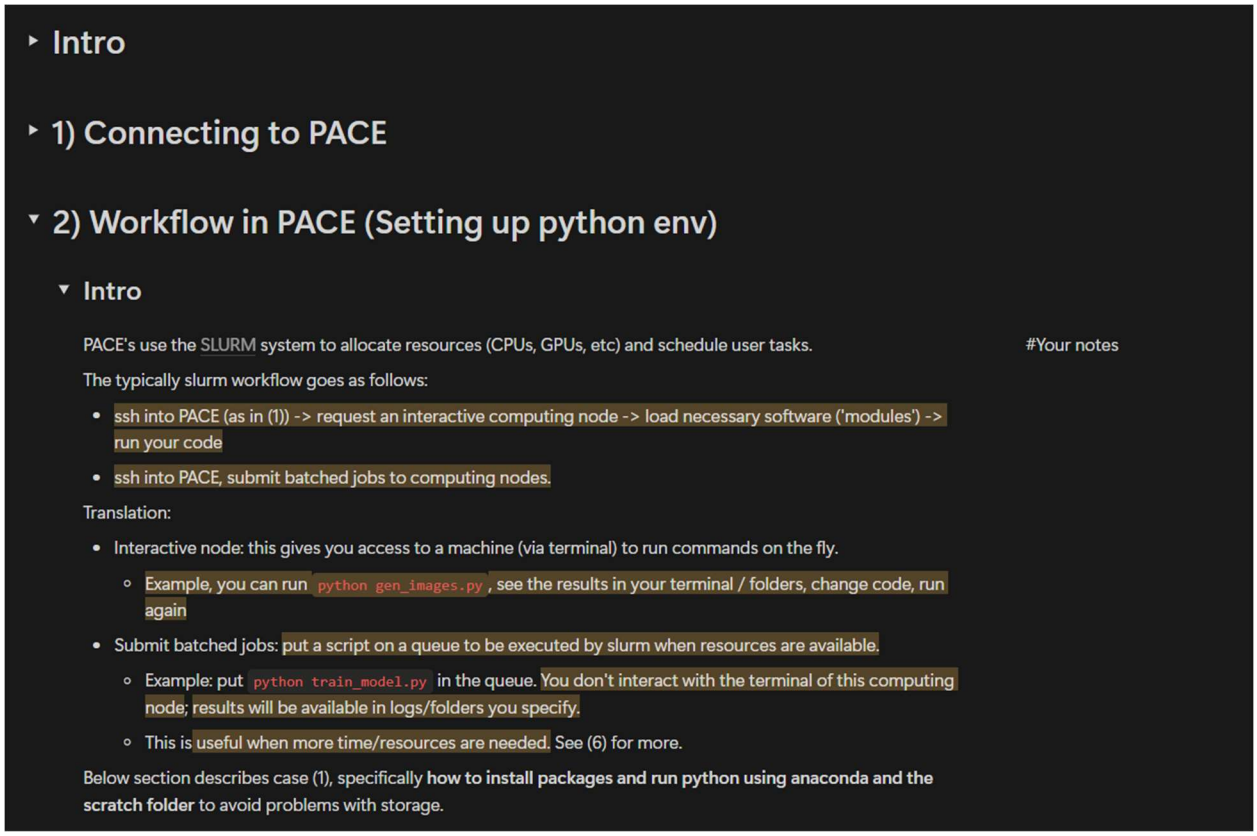
## Abstracts:

Zheng, Heliang, et al. ‘Looking for the Devil in the Details: Learning Trilinear Attention Sampling Network for Fine-Grained Image Recognition’. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, 2019, pp. 5007–16. *DOI.org (Crossref)*, https://doi.org/10.1109/CVPR.2019.00515.

Learning subtle yet discriminative features (e.g., beak and eyes for a bird) plays a significant role in fine-grained image recognition. Existing attention-based approaches localize and amplify significant parts to learn fine-grained details, which often suffer from a limited number of parts and heavy computational cost. In this paper, we propose to learn such fine-grained features from hundreds of part proposals by Trilinear Attention Sampling Network (TASN) in an efficient teacher-student manner. Specifically, TASN consists of 1) a trilinear attention module, which generates attention maps by modeling the inter-channel relationships, 2) an attention-based sampler which highlights attended parts with high resolution, and 3) a feature distiller, which distills part features into a global one by weight sharing and feature preserving strategies. Extensive experiments verify that TASN yields the best performance under the same settings with the most competitive approaches, in iNaturalist-2017, CUB-Bird, and Stanford-Cars datasets.

## What did you do and prove it

- Familiarized with PACE-ICE cluster



▸ Intro

▸ 1) Connecting to PACE

▾ 2) Workflow in PACE (Setting up python env)

▾ Intro

PACE's use the SLURM system to allocate resources (CPUs, GPUs, etc) and schedule user tasks. #Your notes

The typically slurm workflow goes as follows:

- ssh into PACE (as in (1)) -> request an interactive computing node -> load necessary software ('modules') -> run your code
- ssh into PACE, submit batched jobs to computing nodes.

Translation:

- Interactive node: this gives you access to a machine (via terminal) to run commands on the fly.
  - Example, you can run `python gen_images.py`, see the results in your terminal / folders, change code, run again
- Submit batched jobs: put a script on a queue to be executed by slurm when resources are available.
  - Example: put `python train_model.py` in the queue. You don't interact with the terminal of this computing node; results will be available in logs/folders you specify.
  - This is useful when more time/resources are needed. See (6) for more.

Below section describes case (1), specifically **how to install packages and run python using anaconda and the scratch folder** to avoid problems with storage.

- Reviewed code snippet from previous project lead's implementation on image resizing

```
22  > def download_images(filepath): ···
28
29  > def download_images_part(df): ···
51
52    def extract_label_from_filename(filename):
53        # Example: if filename is 'cat_001.jpg', the label could be 'cat'
54        return re.split('_|\.', filename)[0]
55
56    def load_and_preprocess_image(path, label):
57        image = tf.io.read_file(path)
58        image = tf.image.decode_image(image, channels=3)
59        image = tf.ensure_shape(image, [None, None, 3])
60        image = tf.image.resize(image, [256, 256])
61        if image.shape != (256,256,3):
62            print(path)
63        image = image / 255.0  # Normalize to [0,1]
64        label = tf.one_hot(label, depth=5)
65        return image, label, path
66
```