
Machine Learning HW3

ML TAs

ntu-ml-2022-spring-ta@googlegroups.com

Objective - Image Classification

1. Solve image classification with **convolutional neural networks**.
2. Improve the performance with **data augmentations**.
3. Understand popular image model techniques such as **residual**.

Task Introduction - Food Classification

- The images are collected from the food-11 dataset classified into 11 classes.
- Training set: 9866 labeled images
- Validation set: 3430 labeled images
- Testing set: 3347 images

Rules

- DO NOT attempt to find the original labels of the testing set.
- DO NOT use any external datasets.
- **DO NOT use any pretrained models.**
 - Also, do not attempt to “test how effective pretraining is” by submitting to kaggle. Pretraining is very effective and you may test it after the competition ends.
- You may use any publicly available packages/code
 - But make sure you do not use pretrained models. Most code use those.
 - You may not upload your code/checkpoints to be publicly available during the timespan of this homework.

Baseline

Simple : 0.58

Medium : 0.71 Training Augmentation + Train Longer

Strong : 0.81 Training Augmentation + Model Design + Train Loonger (+ Cross Validation + Ensemble)

Boss : >0.87549 Training Augmentation + Model Design +Test Time Augmentation + Train Loonger (+ Cross Validation + Ensemble)

Submission Format

The file should contain a header and have the following format:

```
Id,Category  
0001,1
```

Both type should be strings. Id corresponds to the jpg filenames in test. Follow the sample code if you have trouble with formatting.

Model Selection

- Visit [torchvision.models](#) for a list of model structures, or go to [timm](#) for the latest model structures.
- Pretrained weights are not allowed, specifically set `pretrained=False` to ensure that the guideline is met.

Classification

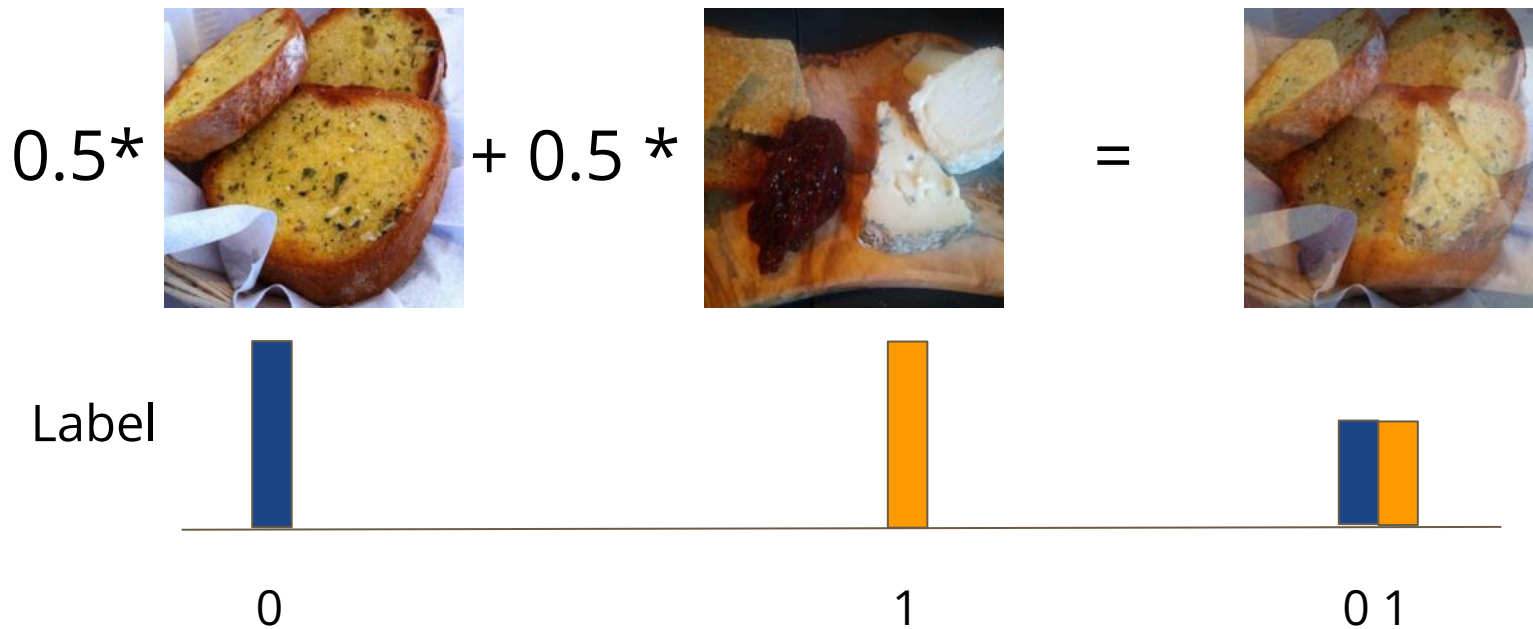
The models subpackage contains definitions for the following model architectures for image classification:

- AlexNet
- VGG
- ResNet
- SqueezeNet

Data Augmentation

- Modify the image data so non-identical inputs are given to the model each epoch, to prevent overfitting of the model
- Visit [torchvision.transforms](https://pytorch.org/vision/stable/transforms.html) for a list of choices and their corresponding effect. Diversity is encouraged! Usually, stacking multiple transformations leads to better results.
- Coding : fill in `train_tfm` to gain this effect

Advanced Data Augmentation - mixup

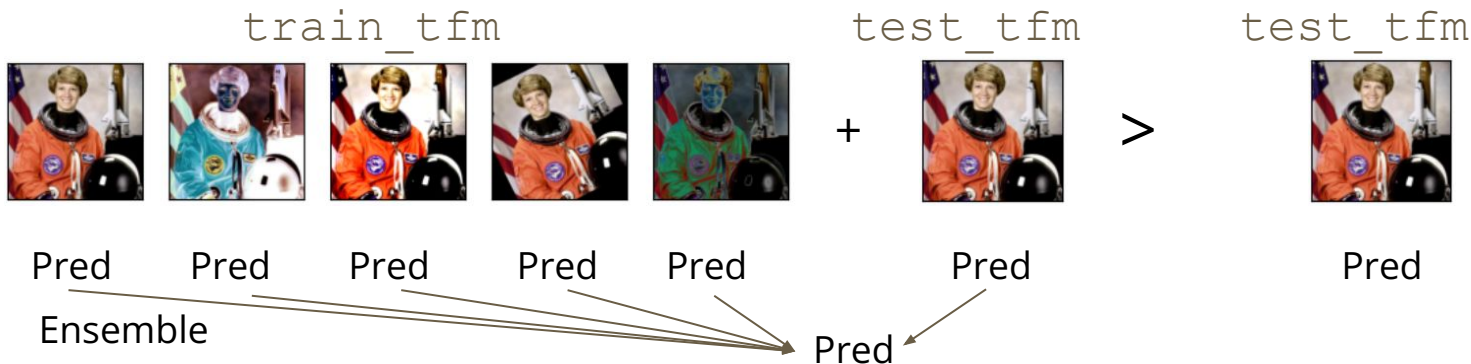


Advanced Data Augmentation - mixup

- Coding :
- In your `torch.utils.Dataset`, `__getitem__()` needs to return an image that is the linear combination of two images.
- In your `torch.utils.Dataset`, `__getitem__()` needs to return a label that is a vector, to assign probabilities to each class.
- You need to explicitly code out the math formula of the cross entropy loss, as `CrossEntropyLoss` does not support multiple labels.

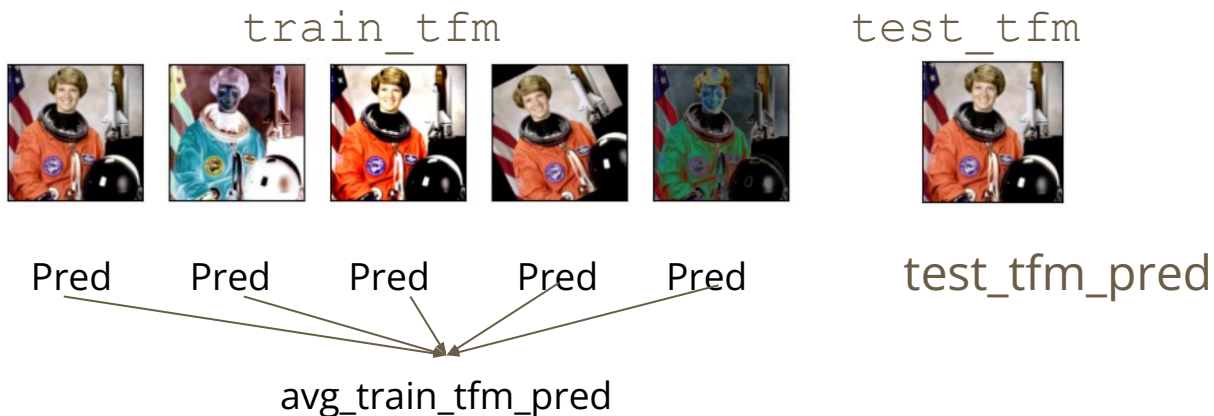
Test Time Augmentation

- The sample code tests images using a deterministic “test transformation”
- You may using the train transformation for a more diversified representation of the images, and predict with multiple variants of the test images.
- Coding : You need to fill in `train_tfm`, change the augmentation method for `test_dataset`, and modify prediction code to gain this effect



Test Time Augmentation

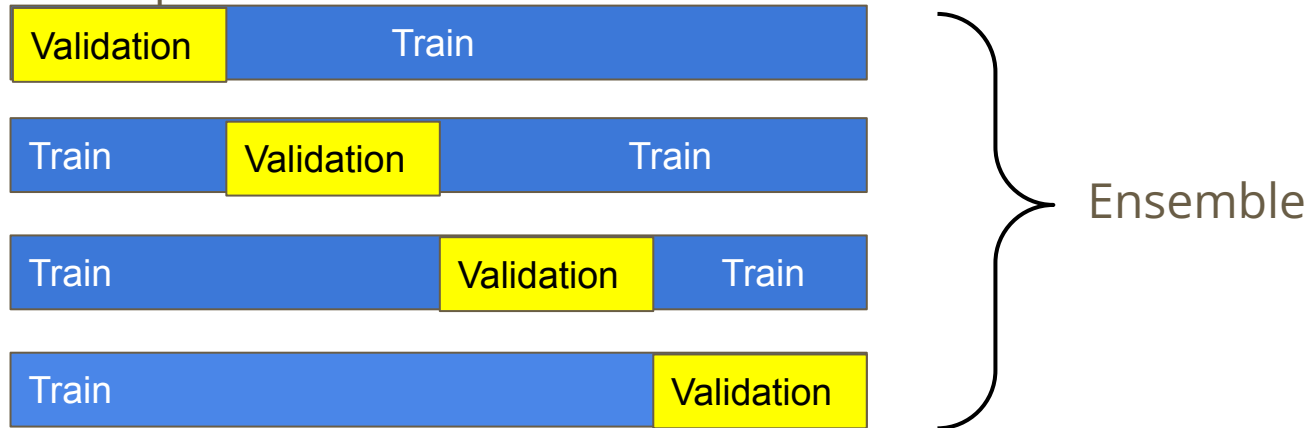
- Usually, test_tfm will produce images that are more identifiable, so you can assign a larger weight to test_tfm results for better performance.



- Ex : Final Prediction = avg_train_tfm_pred * **0.5** + test_tfm_pred* **0.5**

Cross Validation

- Cross-validation is a resampling method that uses different portions of the data to validate and train a model on different iterations. Ensembling multiple results lead to better performance.
- Coding : You need to merge the current train and validation paths, and resample from those to form new train and validation sets.



Cross Validation

- Even if you don't do cross validation, you are encouraged to resplit the train/validation set to suitable proportions.
 - Currently, train : validation $\sim 3 : 1$, more training data could be valuable.

Ensemble

- Average of logits or probability : Need to save verbose output, less ambiguous
- Voting : Easier to implement, need to break ties
- Coding : basic math operations with numpy or torch

Kaggle Tutorial

Kaggle Introduction

- Kaggle GPU : 16G NVIDIA TESLA P100
 - <https://www.kaggle.com/docs/efficient-gpu-usage>
- Faster data IO
- Easier checkpoint reusing
- Limited to 30+ hrs/week depending on usage.
- Limited to 12hrs/run
- We strongly recommend that you run with Kaggle for this homework

How to run

ML2022Spring-HW3

HW3 of Machine Learning, National Taiwan University (Course No. EE5184), taught by prof. Hung Yi Lee

[Host](#) [Overview](#) [Data](#) [Code](#) [Discussion](#) [Leaderboard](#) [Rules](#) [Team](#)

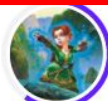
New Notebook

 Search notebooks

 Filters

[All](#) [Your Work](#) [Shared With You](#) [Bookmarks](#)

Hotness 



Sample Code-Training (Not Released)

Updated 4d ago

Private · 0 comments · ML2022Spring-HW3

How to get data : In the input section, there should already be data titled “ml2022spring-hw3”

If there isn't, click on Add data and find “ml2022spring-hw3”

The screenshot displays the Kaggle Sample Code-Training interface. The top navigation bar includes 'File', 'Edit', 'View', 'Run', 'Add-ons', and 'Help'. On the right, there are buttons for 'Share', 'Save Version' (with a count of 0), and a right arrow. Below the navigation bar is a toolbar with icons for adding, deleting, undo, redo, and running code. The main code editor area contains Python code for loading data and listing files. On the right sidebar, the 'Data' section is highlighted with a red box, showing a list of input data with 'ml2022spring-hw3' selected. Below this are sections for 'Output', 'Competitions', 'Settings' (with options for Language, Environment, Accelerator, and Internet), 'Schedule a notebook run', and 'Code Help'.

Sample Code-Training

File Edit View Run Add-ons Help

+ ✕ ↶ ↷ ▶ ▶▶ Run All Code ▾

● Draft Session off (run a cell to start) 🔌 ↺ ⋮

Data + Add data ^

Input

- ▶ ml2022spring-hw3

Output

- ▶ /kaggle/working 🔁

Competitions ▾

Settings ^

Language Python ▾

Environment Preferences

Accelerator None ▾

Internet

Schedule a notebook run ▾

Code Help ^

FIND CODE HELP

+ Code + Markdown

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        pass
        #print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

How to use gpu : Change accelerator to “gpu” when you run your code.

Since GPU time is limited, It is advised to NOT utilize GPU while debugging



The image shows a Kaggle Notebook interface. The main area contains a Python code cell with the following text:

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        pass
        #print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

At the bottom of the code editor, there are two buttons: "+ Code" and "+ Markdown".

The right-hand sidebar contains several sections:

- Data**: Includes a button "+ Add data" and a list of data sources, including "ml2022spring-hw3".
- Output**: Includes a button "+ Add output" and a list of output files, including "/kaggle/working".
- Competitions**: A section with a dropdown arrow.
- Settings**: A section with a dropdown arrow.
- Language**: A dropdown menu currently set to "Python".
- Environment**: A section with a dropdown menu currently set to "None". This section is highlighted with a red box.
- Internet**: A toggle switch currently turned on.
- Schedule a notebook run**: A section with a dropdown arrow.
- Code Help**: A section with a dropdown arrow.

How to Run interactively : The commands are very similar to google colab

Any output writing to ./ will end up here, you can download it

Sample Code-Training

File Edit View Run Add-ons Help

+ Run All

Code

Draft Session off (run a cell to start)

This Python 3 environment comes with many helpful analytics libraries installed
It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

Input data files are available in the read-only "../input/" directory
For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
 for filename in filenames:
 pass
 #print(os.path.join(dirname, filename))

You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version
You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session

+ Code

+ Markdown

Share Save Version 0

Data + Add data

Input

- nl2022spring-hw3

Output

- /kaggle/working

Competitions

Settings

- Language Python
- Environment Preferences
- Accelerator None
- Internet

Schedule a notebook run

Code Help

How to Run in background: Execute code from start to end, all results would be save **permanently**. (Output is limited to 20G, max run time = 12hrs)

Make sure your code is bug free, as any error in any code block would result in early stopping

The screenshot displays the Kaggle Code-Training environment. The top bar includes a 'Sample Code-Training' title, a menu (File, Edit, View, Run, Add-ons, Help), a 'Share' button, and a 'Save Version' button (highlighted with a red box) showing version '0'. Below the menu is a toolbar with icons for adding, deleting, undo, redo, and running code. The main area contains a code cell with Python code for loading libraries, listing input files, and printing file paths. The right sidebar shows a 'Data' section with 'ml2022spring-hw3', an 'Output' section with '/kaggle/working', and a 'Settings' section with options for Language (Python), Environment (Preferences), Accelerator (None), and Internet (checked). A 'Schedule a notebook run' button is at the bottom of the sidebar.

Sample Code-Training

File Edit View Run Add-ons Help

Share Save Version 0

+ - Undo Redo Run All Code

Draft Session off (run a cell to start)

This Python 3 environment comes with many helpful analytics libraries installed
It is defined by the kaggle/python Docker image: <https://github.com/kaggle/docker-python>
For example, here's several helpful packages to load

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
```

Input data files are available in the read-only "../input/" directory
For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

```
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        pass
        #print(os.path.join(dirname, filename))
```

You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version
You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session

Data + Add data

Input

- ml2022spring-hw3

Output

- /kaggle/working

Competitions

Settings

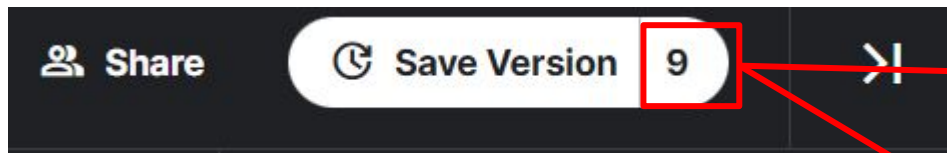
Language Python

Environment Preferences

Accelerator None

Internet ☒

Schedule a notebook run



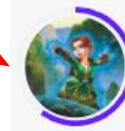
Version 2

12d ago

Save & Run All • Diff: +114 -10

Ran in 1 hour and 19 minutes

...



Version 7

11d ago

Save & Run All • Diff: +11 -9

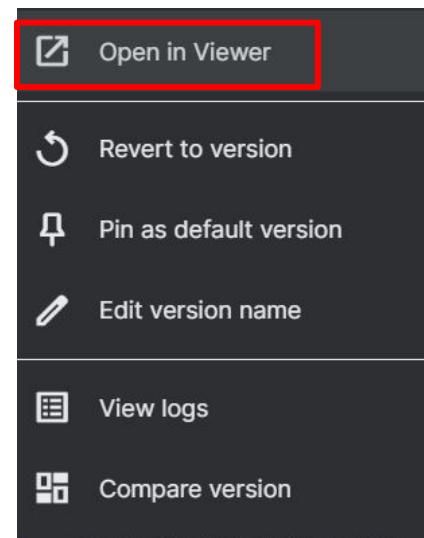
Cancelled after 12 hours

...

How to view “Run in background” results

You can view your results this way

Don't worry if your run is “cancelled”, the output will still be saved.



How to utilize your results



SPLEND1DCHAN (燦爛) +3 · 11D AGO · 47 VIEWS · PRIVATE



run_classification_best

Python · [Food-11](#)

Script

Data

Logs

Comments (0)

Settings

boss_14_0.61137.ckpt
boss_15_0.65539.ckpt
boss_1_0.51341.ckpt
boss_2_0.53557.ckpt
boss_3_0.51808.ckpt
boss_4_0.57318.ckpt
boss_5_0.58163.ckpt
boss_6_0.57114.ckpt
boss_7_0.56327.ckpt
boss_8_0.60758.ckpt
boss_9_0.59971.ckpt
boss_best.ckpt

Download All

+ New Version

+ New Notebook

Upload your model to become kaggle dataset + New Dataset
Create a new notebook with the output as input

How to train and predict

1. Run your code in background
2. Find the output data `"./submission.csv"` and upload it to the submission page

How to retrain from a checkpoint

1. Run your code in background
2. Find the output model and save it as a dataset
3. Import your dataset into the notebook via "Add data"
- 4. Modify your code to load your checkpoint**
5. Run your code in background
6. Find the output data `./submission.csv` and upload it to the submission page

Tips and tricks

Time management

- Kaggle will allocate more time for those who have utilized GPU resources in the past week. Time resets every Saturday at 08:00, Taipei Time.

=> Run any code with GPU on kaggle today (3/4) to get (possible) extra time next week.

- Time consumption is the sum of notebooks running interactively with gpu and running in background with gpu.
- Please start early

Time management

- You can go over the time limit moderately. Kaggle will not interrupt your code in the background if it is already running. If your time limit is reached, you cannot run any code with GPU.

=> 時間快用完的時候在背景跑一隻程式，等於多12小時runtime

=> 時間快用完的時候在背景跑兩隻程式，等於多24小時runtime

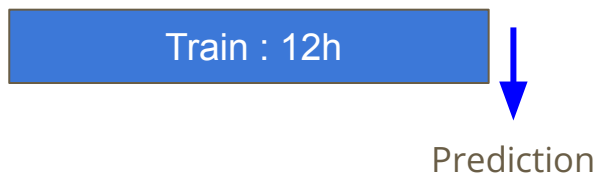
Accelerator	GPU ▾
GPU Quota	61:02 / 41 hrs

Time management - Parallelization

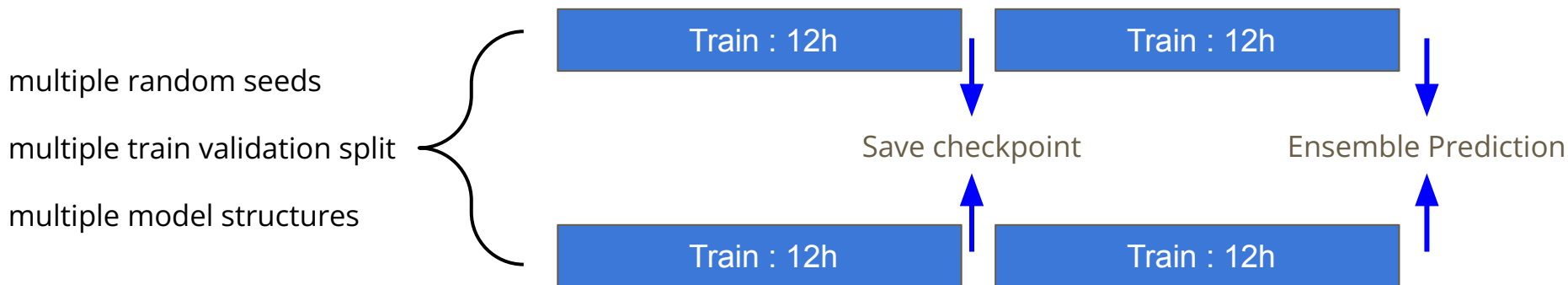
- You can run **two** codes in the background
- If you are satisfied with your code, utilize this to run **multiple random seeds/multiple train validation split/multiple model structures**, so you can ensemble

A sample procedure for beating the boss baseline

The boss baseline could be beaten with a single model trained on kaggle for 12hrs



Your procedure can be ensemble of models with parallelization



Experimental Tips

- **Augmentation** is a must to prevent overfitting. A good augmentation can carry on to the testing phase with **Test Time Augmentation**.
- If you build your own network structure and have implemented augmentation, don't be afraid to scale up your model. (Many predefined models structure are huge and perform great)
- In TA's experiment, model structures with **downsampling** work better, simply choosing the best performing models on ImageNet according to websites is not always a good idea because pretrained weights are not allowed.

Other tricks.....

- on Classification
 - Label Smoothing Cross Entropy Loss
 - FocalLoss
- on Optimization
 - Dropout
 - Gradient Accumulation
 - BatchNorm
 - Image Normalization

Running with Google Colab

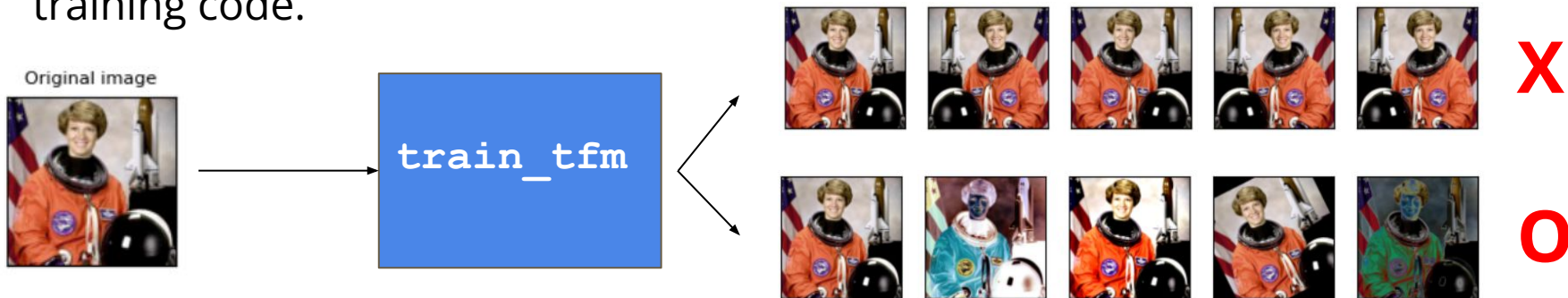
- We strongly recommend that you run with Kaggle for this homework
- If you would like to use colab, DO NOT store data in your drive and load from there, the input/output is very slow. (store at ./ instead)
- If you mount your google drive in colab : G-suite google drive now has a storage limit. Since models and data can be large, keep an eye on your used space to prevent your account being suspended.

Report Questions

Q1. Augmentation Implementation (2%)

Implement augmentation by finishing `train_tfm` in the code with image size of your choice. Copy your `train_tfm` code and paste it onto the GradeScope.

- Your `train_tfm` must be capable of producing 5+ different results when given an identical image multiple times.
- Your `train_tfm` in the report can be different from `train_tfm` in your training code.



Q2. Residual Connection Implementation (2%)

Residual Connection is widely used in CNNs such as [Deep Residual Learning for Image Recognition](#). Residual is demonstrated in the following graph.

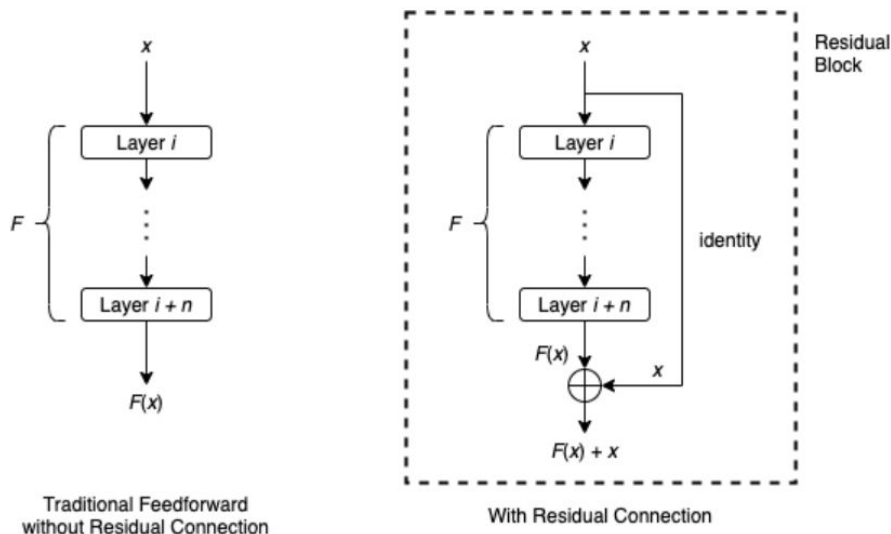


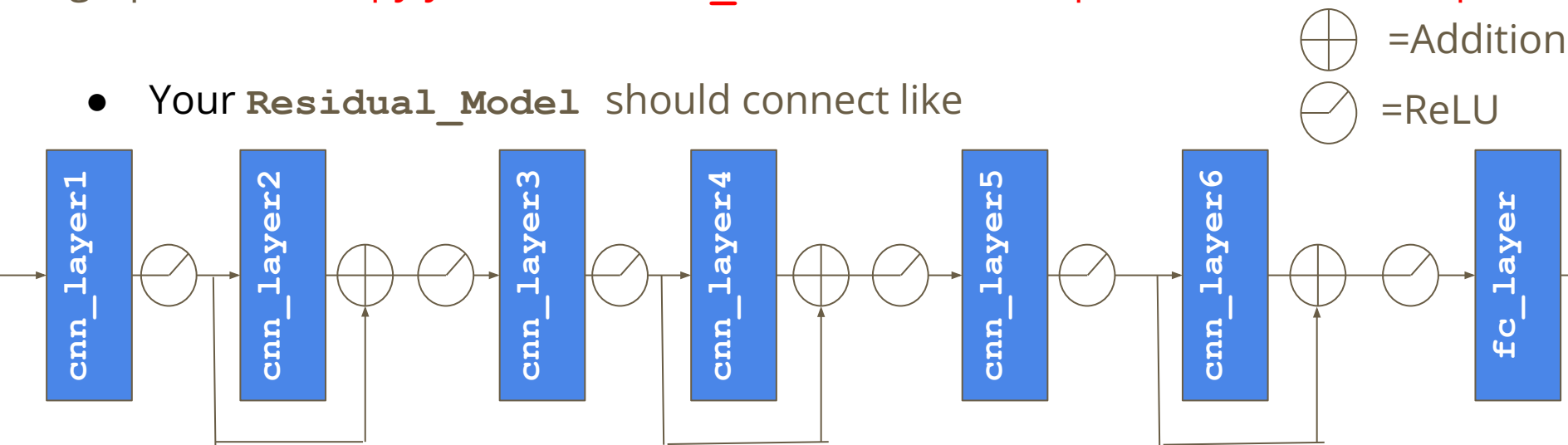
Image Source :

<https://towardsdatascience.com/what-is-residual-connection-efb07cab0d55>

Q2. Residual Connection Implementation (2%)

Implement Residual Connections in the `Residual_Model`, following the graph below. Copy your `Residual_Model` code and paste it on Gradescope.

- Your `Residual_Model` should connect like



- You should only modify the `forward` part of the model

Regulations and Grading Policy

Rules

- DO NOT attempt to find the original labels of the testing set.
- DO NOT use any external datasets.
- **DO NOT use any pretrained models.**
 - Also, do not attempt to “test how effective pretraining is” by submitting to kaggle. Pretraining is very effective and you may test it after the competition ends.
- You may use any publicly available packages/code
 - But make sure you do not use pretrained models. Most code use those.
 - You may not upload your code/checkpoints to be publicly available during the timespan of this homework.

Grading

- simple (public) +0.5 pts
- simple (private) +0.5 pts
- medium (public) +0.5 pts
- medium (private) +0.5 pts
- strong (public) +0.5 pts
- strong (private) +0.5 pts
- boss (public) +0.5 pts
- boss (private) +0.5 pts
- code submission +2 pts
- report +4 pts

Total : 10 pts

Code Submission

- NTU COOL
 - Compress your code and pack them into **.zip file**

<student_ID>_hw3.zip

- Your **.zip** file should include only
 - **Code:** either .py or .ipynb
- **Do not submit models and data**
- **File Size Limit : 25MB**
- **Submit the code that corresponds to your chosen submission in Kaggle (One of the best)**

Deadlines

- Kaggle

2022/03/25 23:59 (UTC+8)

- NTU COOL

2022/03/27 23:59 (UTC+8)

Regulations

- You should finish your homework on your own.
- You should not modify your prediction files manually
- Do not share codes or prediction files with any living creatures.
- Do not use any approaches to submit your results more than **5 times** a day.
- **Do not search or use additional data or pre-trained models.**
- Your **final grade x 0.9 and 0 pt for this HW** if you violate any of the above rules.
- Prof. Lee & TAs preserve the rights to change the rules & grades.

Links

Kaggle :

Kaggle code (join competition first) :

Colab :

Contact us if you have problems...

- **Kaggle Discussion** (Recommended for this HW)
- NTU COOL
- Email
 - mlta-2022-spring@googlegroups.com
 - The title should begin with “[hw3]”