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

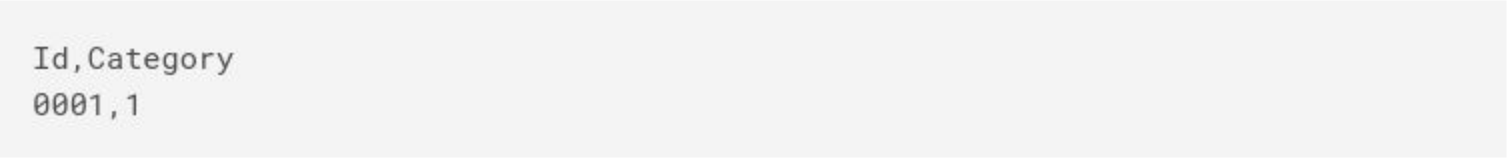
- The images are collected from the food-11 dataset classified into 11 classes.
- Training set: 9866 labeled images
- Validation set: 3430 labeled images
- Testing set: 3347 images

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.

Submission Format

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

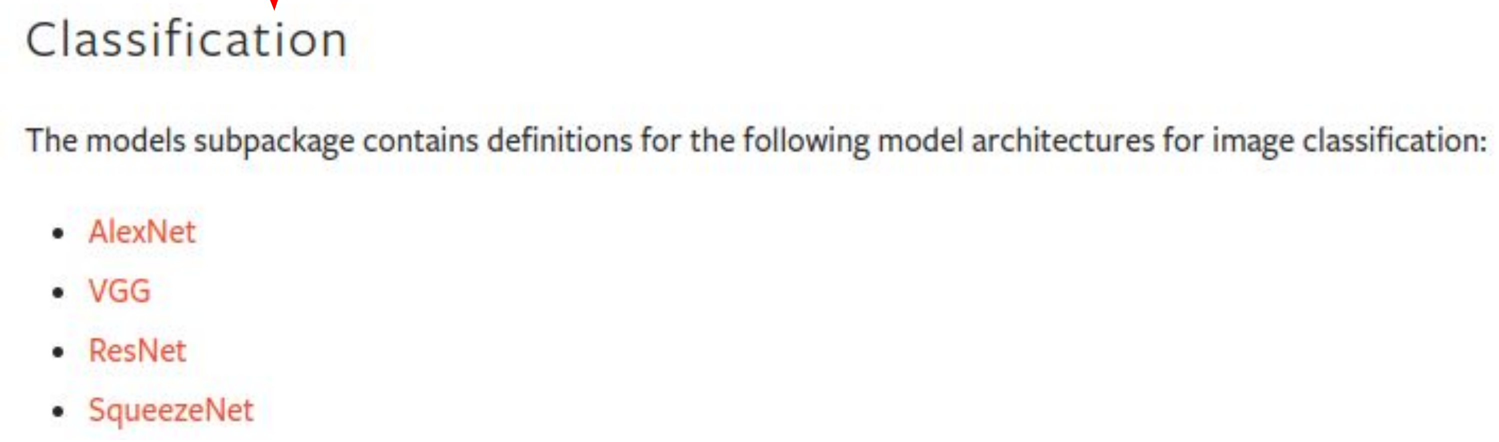


```
Id,Category
0001,1
```

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.



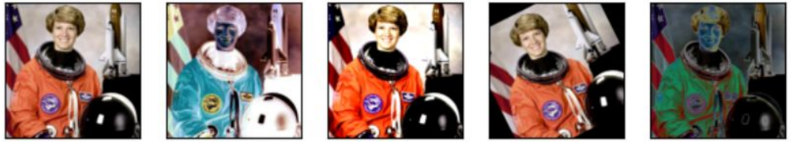
Classification

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

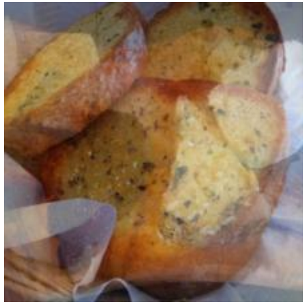
- AlexNet
- VGG
- ResNet
- SqueezeNet

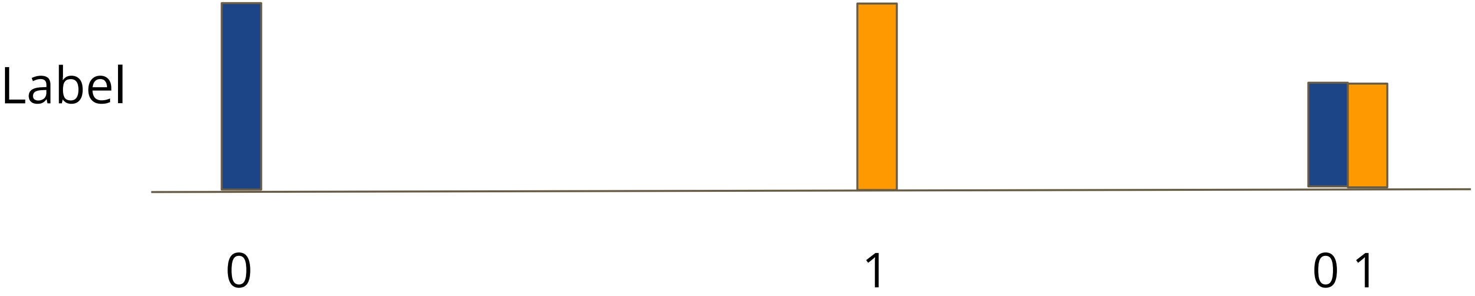
Data Augmentation

- Modify the image data so non-identical inputs are given to the model each epoch, to prevent overfitting of the model
- Visit torchvision.transforms 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

0.5* + 0.5 * = 

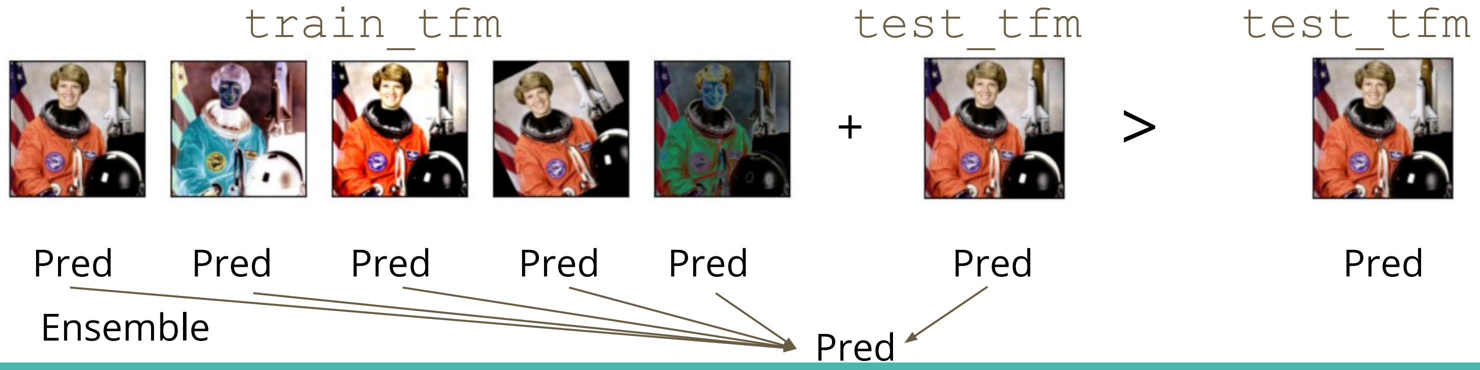


Advanced Data Augmentation - mixup

- 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



train_tfm

test_tfm

test_tfm

+

>

Pred Pred Pred Pred Pred

Pred

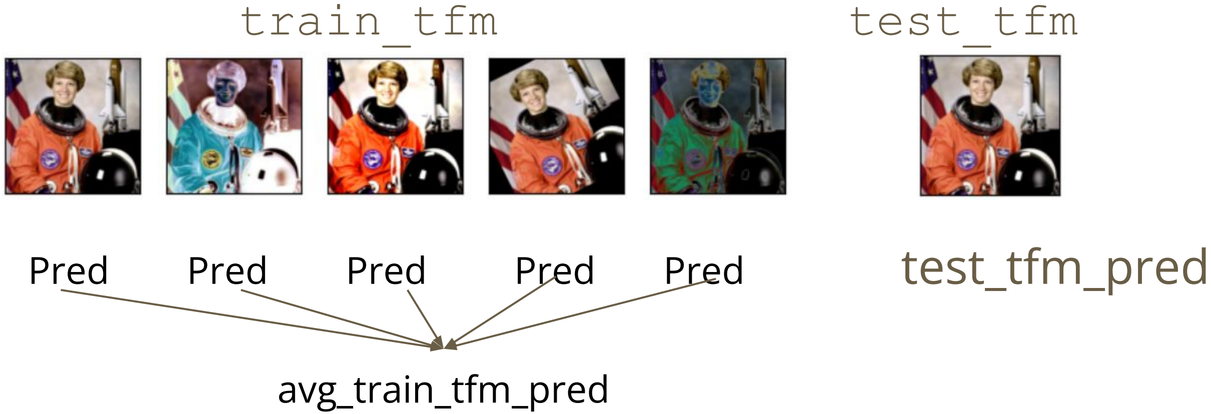
Pred

Ensemble

Pred

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.



train_tfm

test_tfm

Pred Pred Pred Pred Pred

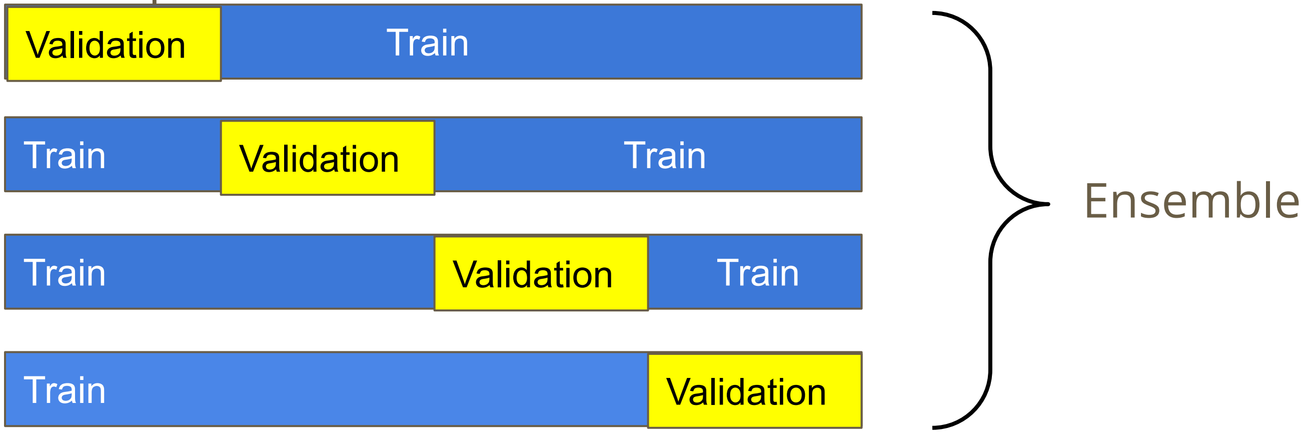
test_tfm_pred

avg_train_tfm_pred

- 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 form 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 ~ 3 : 1, more training data could be valuable.