Contents

[Approach: 1](#_Toc11412927)

[Data Preprocessing 2](#_Toc11412928)

[Loss Functions: 2](#_Toc11412929)

[Validation: 2](#_Toc11412930)

[Pre-Trained Models 2](#_Toc11412931)

[Model Training: 3](#_Toc11412932)

[Final Model: 3](#_Toc11412933)

[Takeaways: 4](#_Toc11412934)

[Pointers for participants 4](#_Toc11412935)

# Approach:

* At the start I experimented with pure Pytorch based deep learning models, but found that default FastAI models work better than similar Pytorch based models. Thus at later stages (and for my final models) I have used Pytorch.
* My approach was to multiple models and ensemble them at the end.
* The ensembling technique was slightly different from averaging the probabilities. In my final ensemble I had 7 models, and out of which I took the best model as a benchmark to be overridden only if most of the other models did not agree with its label predictions.
  + The overriding was done based on number of models that did not agree with the best model. I took 2 threholds for this:>= 5 models should not agree or >=4 models should not agree to be able to modify the benchmark’s class prediction.
  + **FINAL\_1: override if <= 2 models agree with benchmark model**
  + **FINAL\_2: override if <=3 models agree with benchmark model. This was the final submission logic**
  + Details are in the below XLS.



Data Preprocessing**:**

Initially I experimented with mainly image size and batch-size. I ran multiple models with progressive resizing but the results did not justify the run-time. Hence, I focused on keeping image size static (or within only fixed sizes), while trying to accommodate larger batches (64 Batch-size max)

* **Image Size**: I mainly worked with 224 X 224 image size for most models and 296 X 296 specifically for Efficient Net models
* **Input Normalization**: ImageNet normalized inputs. Basically each of the three channels were z normalized based on imagenet statistics
* **Augmentation**:
  + **Cut**-**out:** This is an extra transformation which is applied only to a few models
  + **Flips (horizontal & vertical)**
  + **RandomRotation (10 degrees)**
  + **MaxLighting (0.2)**
  + **MaxZoom (1.1)**
  + **MaxWarp(0.2)**
  + **Affine (0.75)**
* The above pre-processing as standard in the context of computer vision, I had already worked on multiple CV models (Kaggle competitions and work projects) earlier and that is how I understand/know about these methods.

# Loss Functions:

* I started with cross-entropy loss function but quickly shifted to Focal loss since it converged much quicker and much better

# Validation:

* Either 5% or 10% of images were kept solely for validation purposes

# Pre-Trained Models

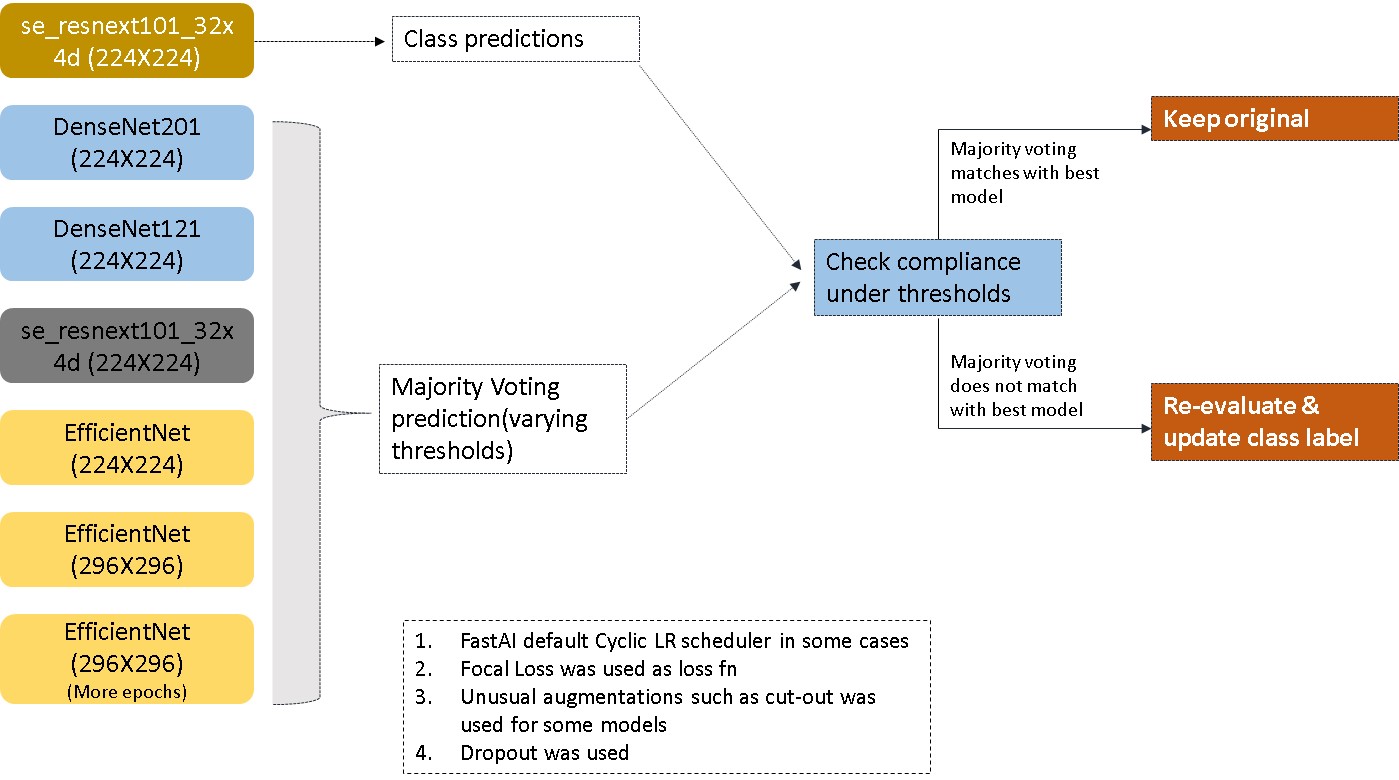
* **se\_resnext101\_32x4d:** From Cadene’s repo at this **<<**[**link**](https://github.com/Cadene/pretrained-models.pytorch)**>>**
* **EfficientNet-**b3 from this <<[**link**](https://github.com/lukemelas/EfficientNet-PyTorch) >>
* **DenseNet (121/169/201)** Models from fastai.models pretrained repo **<<**[**link**](https://docs.fast.ai/vision.models.html)**>>**
* **Note:** All pretrained models pertain to ImageNet pre-training

# Model Training:

* In some cases I started with training only the last few FC layers, in some other cases I found that training all layers unfrozen using very small learning rate worked better
* Dropout was used
* Some same Model architectures were trained on varying number of epochs and with different learning rate. If you see multiple same models (eg: EfficientNet) then they differ either in terms of total epochs or learning rate schedules or both
* In some cases FastAI’s inbuilt cyclic LR implementation is used
* All predictions are based on FastAI’s default TTA (test time augmentation)

# Final Model:

* **Final model was ensemble as described above**

****

# Takeaways:

* For me the key takeaway is that one should be able to experiment with multiple input setting in as little time as possible. Basically, playing around with image sizes, loss functions, model architectures, LR schedules etc helped me figure out a training strategy that worked for most models
* I also tried OOF based stacking but that did not work, and I think there was a bug in the code. Perhaps my fold indexes were not as stable was I though, or something else. I did not invest much time on this. But I feel this would be have results into much better results if done right.

# Pointers for participants

* Participants much explore the data as much as possible, this is true even for Deep Learning challenges.
* Explore dependencies such as class imbalance, existing bias in the input data and formulate strategies to counter those issues
* **General modeling strategy:** Start with trying to overfit a single model to your data, and then work backwards from there by trying to reduce complexity etc.
* Leverage Kaggle resources as they are very good for medium sized datasets, especially image datasets on which you can leverage power of GPUs for free
* Read papers and try to implement improvements from therein