Bristol-Myers Squibb Molecular Translation 3rd place solution

Team: kyamaro
(KF + lyakaap + camaro)

About us

- Kazuki Fujikawa (KF)
  - Data scientist at DeNA
  - Analyse company’s baseball team
  - Kaggle Grandmaster
- Shuhei Yokoo (Iyakaap)
  - Data scientist at DeNA
  - Analyse company’s baseball team
  - Kaggle Grandmaster
- Daisuke Yamamoto(Camaro)
  - Kaggle Master

Solution: Overview

Phase1:

Image Captioning

- Salt & pepper noise
- Focal loss

Phase2:

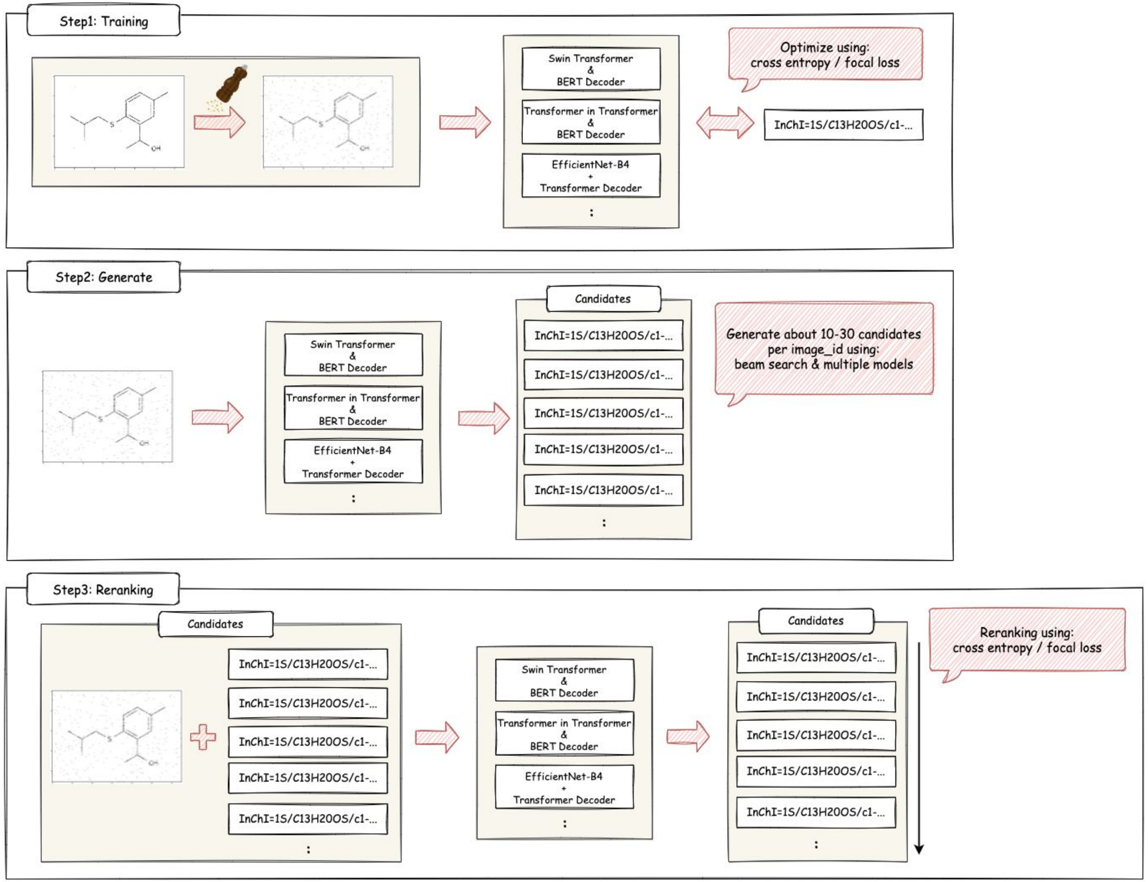
Generate InChI candidates

- Beam search
- Logit ensemble

Phase3:

Reranking InChI candidates

- Normalize with RDKit
- Rerank with multiple models



Solution: Overview

Phase1:

Image Captioning

- Salt & pepper noise
- Focal loss

Phase2:

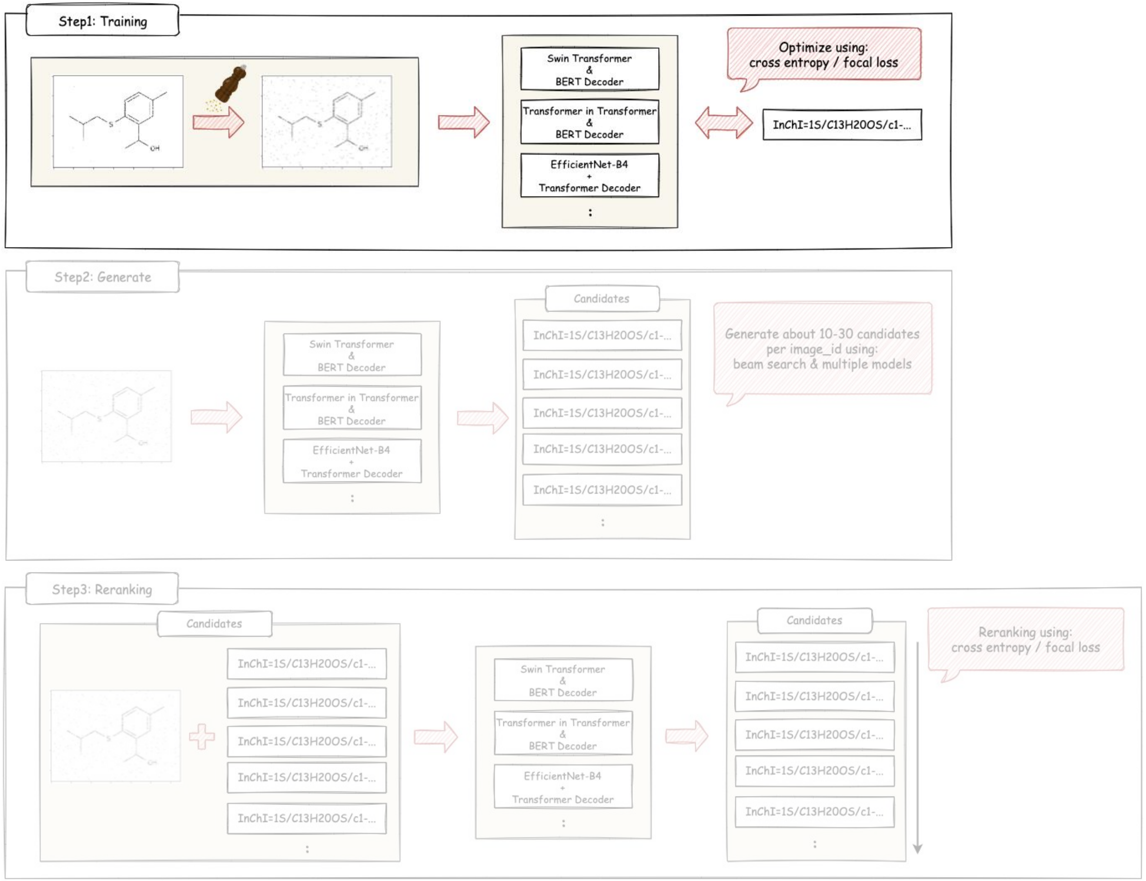
Generate InChI candidates

- Beam search
- Logit ensemble

Phase3:

Reranking InChI candidates

- Normalize with RDKit
- Rerank with multiple models



Phase1: Image captioning framework

- Generate sequences by considering InChI as a just string, in the framework of the general image captioning task
  - Surprisingly work well, although each layer has different meanings
  - Performance: Transformer based > CNN based (in our settings)
    - For this task, we need to capture "what" and "where" is in the image, so a Transformer-based Encoder would have been more advantageous.

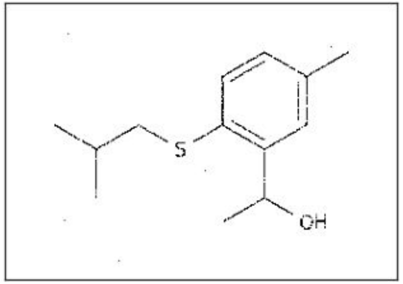
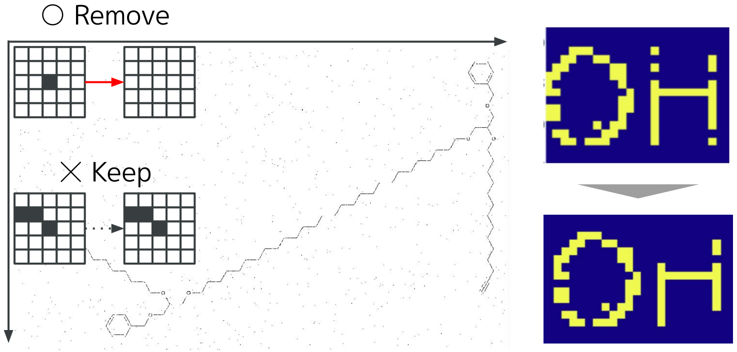


Image Encoder → **Transformer Decoder** → **InChI=1S /C13H20OS ...**

Phase1: Denoise salt and pepper（LB: 2.18→1.26）

- In a certain size kernel, if there is no dot other than center, consider it as noise and remove it.
- Even without fine tuning, CV/LB gap was drastically improved.
- CV got slightly worse maybe because it sometimes removes an useful information too.

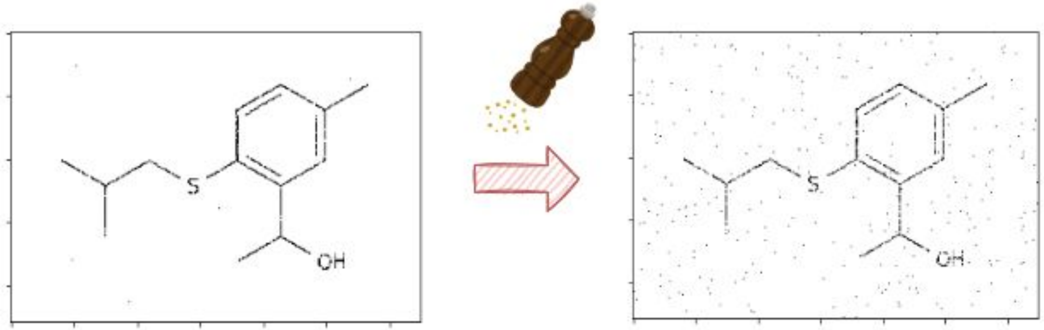
Example of denoise fliter（k=5）



Model	CV	LB
TNT	1.13	2.18
+denoise (k=3)	1.69	1.71
+denoise (k=5)	1.22	1.26
+denoise (k=7)	1.17	1.32

Phase1: Add salt and pepper（LB: 1.26→1.06）

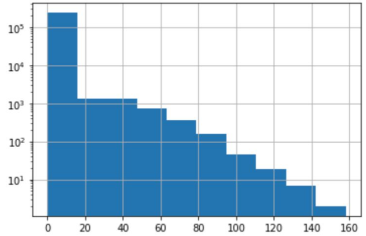
- CV/LB gap was also improved by adding salt and pepper noise at training time, even without denoise at test time.



Phase1: CE Loss → Focal Loss

- After training a few epochs, our model predicts surprisingly well and almost all samples got too easy one.
  - Ex. Swin Transformer（CV: 0.98, LB: 0.99）
    - Levenshtein=0: 87%
    - Levenshtein=1: 7%
    - …
- Using Focal Loss makes training more efficient by focusing on hard example.

Hist of Levenshtein（in Log scale）



$$\mathrm{FL}(p_t) = -(1 - p_t)^{\gamma} \log(p_t).$$

Solution: Overview

Phase1:

Image Captioning

- Swin Transformer
- Salt & pepper noise
- Focal loss
- Train longer

Phase2:

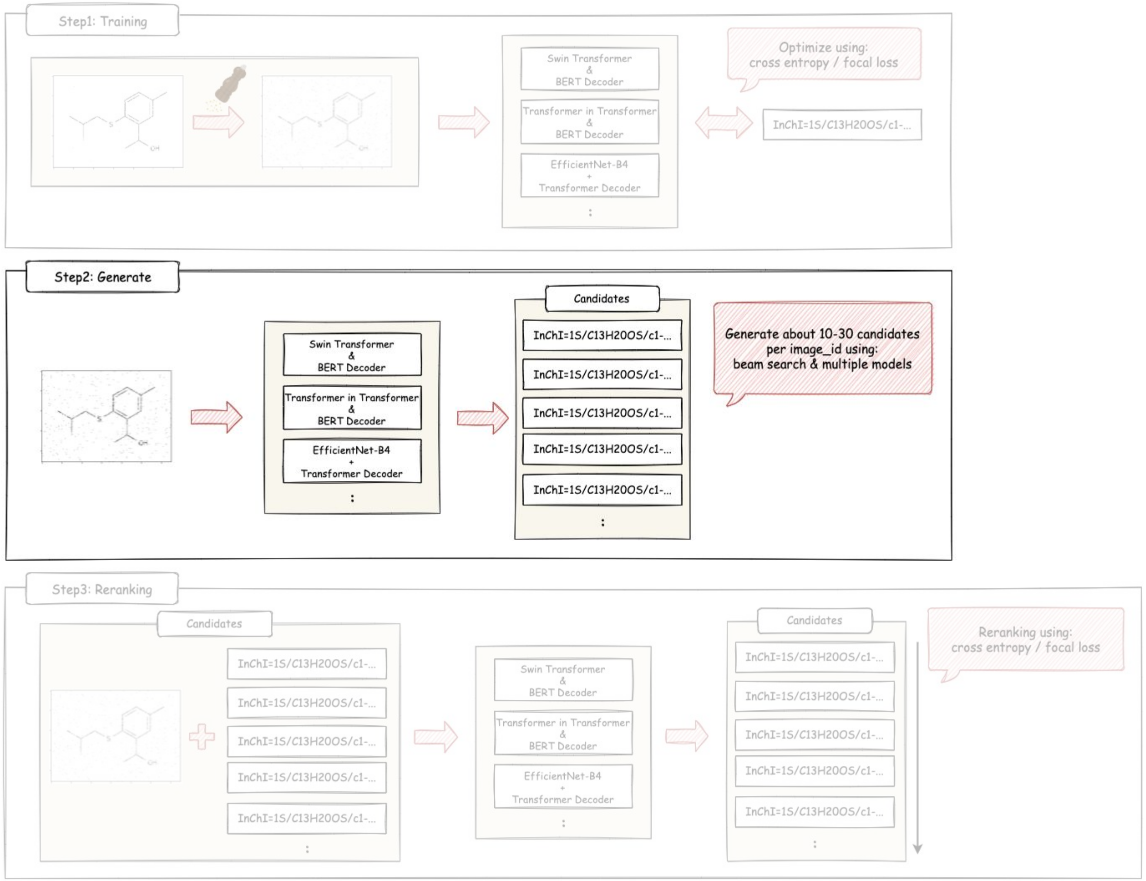
Generate InChI candidates

- Beam search
- Logit ensemble

Phase3:

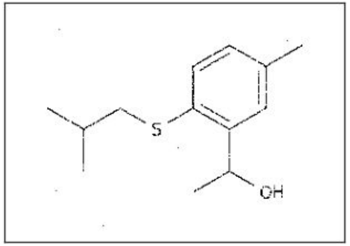
Reranking InChI candidates

- Normalize with RDKit
- Rerank with multiple models



Phase2: Beam Search

- To make reranking at phase 3 more effective, need to produce diverse and high quality candidates.
  - Combining Beam Search and Greedy method was effective
  - Use broad Beam Search (ex.k=32) only for hard examples (Iyakaap)

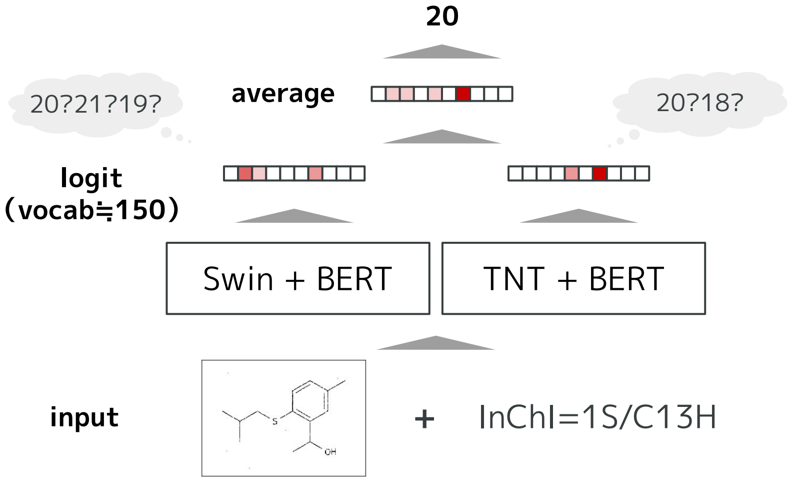


InChI=1S/C13H20OS…
InChI=1S/C13H21OS…
InChI=1S/C11H21OS…
InChI=1S/C12H20OS...

Model	CV
Swin (beam=1)	0.98
Swin (beam=4)	0.93
Swin (beam=1+4)	0.87

Phase2: Logit ensemble

- Ensemble logits at InChI generation phase
  - There is a difference in predictions even between Swin and TNT, so ensemble was effective.
  - But it was difficult to mix all of 3 members’ models due to implementation difference.



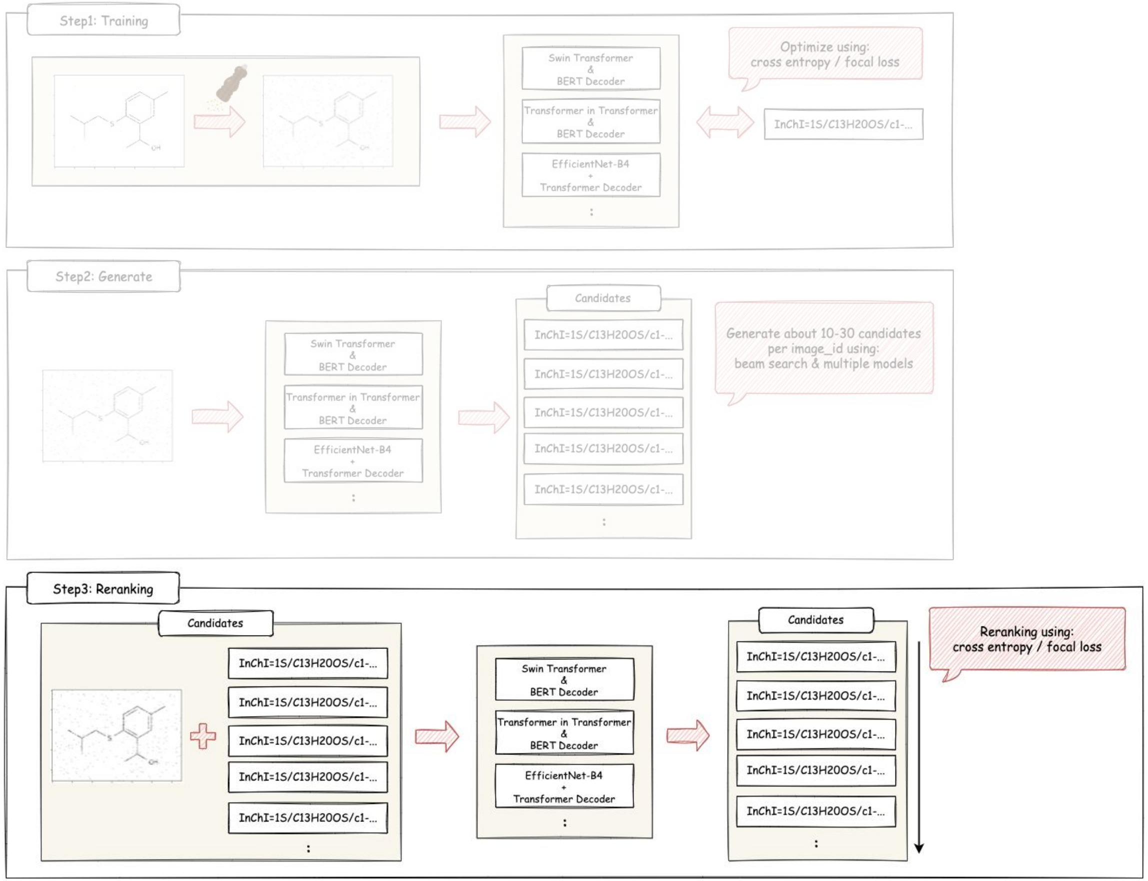
Model	CV
Swin (beam=1)	0.98
TNT (beam=1)	1.04
Swin+TNT (beam=1)	0.83

Solution: Overview

Phase1:
Image Captioning
- Swin Transformer
- Salt & pepper noise
- Focal loss
- Train longer

Phase2:
Generate InChI candidates
- Beam search
- Logit ensemble

Phase3:
Reranking InChI candidates
- Normalize with RDKit
- Rerank with multiple models



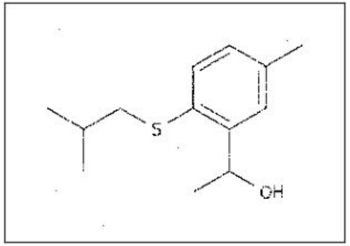
Phase3: RDKit normalization

- Rerank candidates with using RDKit normalization
  - Even though our model choose the most likely token in InChI generation, still there are many tiny mistakes.
  - RDKit normalization was effective way to fix such mistakes, but sometimes it makes score worse.
  - We re-score normalized candidates by calculating loss between generated candidates and teacher-forced model predictions.
    ※ loss is either cross entropy loss or focal loss
  - Then rerank by below logic:

```
df = df.sort_values(
    by=["is_valid", "score"],
    ascending=[False, True],
).groupby("image_id").first()
```

Phase3: Rerank by multiple models each other

- Gather candidates from models, then evaluate by themselves
  - Each model calculates loss and take average
  - Take rank average between the model groups which have a gap in obsolete value range
  - It was easy to apply to diverse models because it doesn't mind implementation difference



	Model (CE)				Model (Focal)				Average
InChI Candidates	A	B	Mean	Rank	C	D	Mean	Rank	Rank
InChI=1S/C13H20OS…	0.1	0.9	0.5	3	0.03	0.05	0.04	2	2.5
InChI=1S/C13H21OS…	0.2	0.2	0.2	1	0.02	0.02	0.02	1	**1**
InChI=1S/C12H20OS…	0.3	0.5	0.4	2	0.01	0.09	0.05	3	2.5

※ In this example we take rank average only in same image id, but actually we took globally.

Final models（part of）

Member	Model	CV	LB (greedy)	LB (reranked)
KF	Swin large (384x384) + PL	0.90	0.89	0.79
	TNT (512x1024) + PL	0.97	0.99	0.87
lyakaap	Swin base (384x384) + PL	0.92	0.85	0.74
	Swin base (384x384) + PL + focal loss	0.87	0.78	0.70
	EffNet-v2 >> ViT (448x448) + PL + focal loss	0.86	0.81	0.70
Camaro	EffNetB4 Transformer Decoder (416x736)	0.67	0.87	0.67
	EffNetB4 Transformer Decoder (416x736) + PL	0.67	0.73	0.62
	EffNetB4 Transformer Decoder (416x736) + Noise	0.65	0.84	0.62
	EffNetB4 Transformer Decoder (416x736) + Noise/Denoise	0.83	0.77	0.61

Summary

- It was the key to identify the cause of Train/Test Gap
  - It seems not so many top team found it?
- Generate many candidates by diverse models + rerank
  - The more candidates or the more models, score is the better
  - After team merging(kyakaap + camaro), score was drastically improved. Diversity wins.
- Beam search + Greedy was effective way to increase candidate
  - Beam search itself should contain greedy candidates, but using both was better somehow. Maybe due to too strong pruning?
  - Other smart search(like MCTS) may effective, though it was hard to apply due to hardware limit.

Discussions

- Why test data is a bit noisier than train data? (Is it intentional?)
- How to make use of invalid prediction in terms of InChI rules?
- How much was original expectation for winning score?
- How to merge solutions from 3 teams?
- What’s next for this competition?
    - Actually digitize old publications? Write paper?

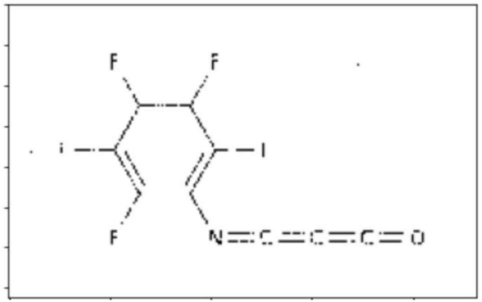
Appendix

Phase1: lyakaap part

- Encoder:
  - Swin-transformer（size=384x384）
  - EfficientNet-v2 + ViT（size=448x448）
    - CNN as Patch embedding
- Decoder:
  - 3 layers transformer
    - Increasing number of layers more makes training unstable
- PIL.Image.resize is better than cv2.resize or other resizer (reference)
- Fine tuning with pseudo labeling
  - It improves score with reducing CV/LB gap
- Training speed up by Sequence bucketing

Baseline: RDKit normalization（postprocess）

- Normalize generated InChIs by RDkit（by nofreewill）
  - Execute MolToInchi(MolFromInchi(inchi))
  - This process can correct inchis if the atoms are just mis-numbered



InChI=1S
/C9F5NO
/c10-4-5(11)7(13)9(15-2-1-3-16)8(14)6(4)12

InChI=1S
/C9F5NO
/c10-4-5(11)7(13)9(8(14)6(4)12)15-2-1-3-16

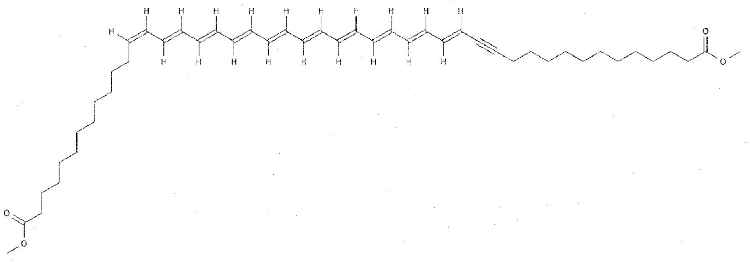
AGENDA

- Baseline
- **Solution**

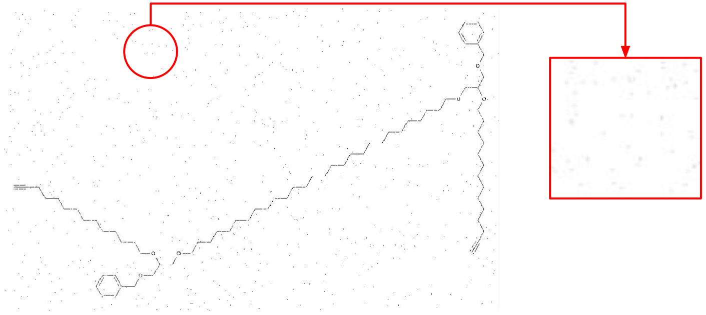
Phase1: Compare predictions between Swin and TNT

- Swin has CV-LB better CV/LB relation ship than TNT
- To dig in the gap, calculate Levenshtein between these models' predictions. Even between models, there was a gap (val: 1.02 test:2.16)
- Check images which have large Levenshtein both in valid test
  - Tend to have large image size both in val/test
  - Only in test, tend to have much salt and pepper noise

Worst Levenshtein（valid）



Worst Levenshtein（test）

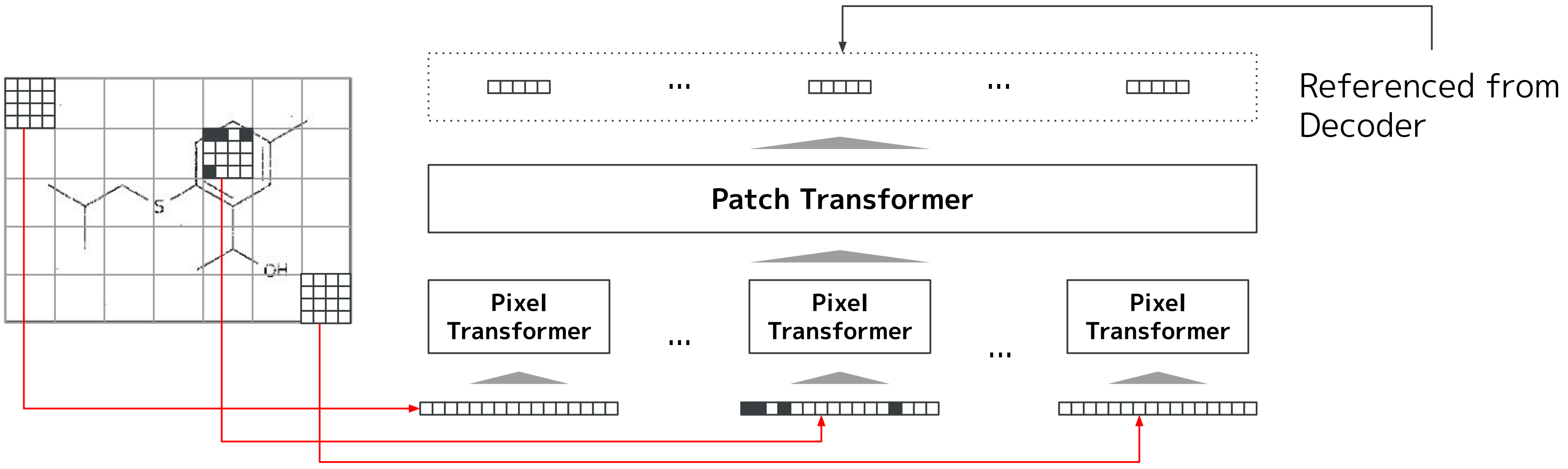


Phase1: Train longer

- There was a difference between kyakaap and camaro in CV/LB relationship.
  - Best single (Camaro): CV=0.66, LB=0.87
  - Best single (KF): CV=0.98, LB=0.99
- It can be due to train steps.
  - Camaro: 25-50 epoch (TPU 7days?)
  - KF: 10 epoch (A100 7days)
- After training kyakaap model 3 epochs more, CV improved from 0.98 to 0.91

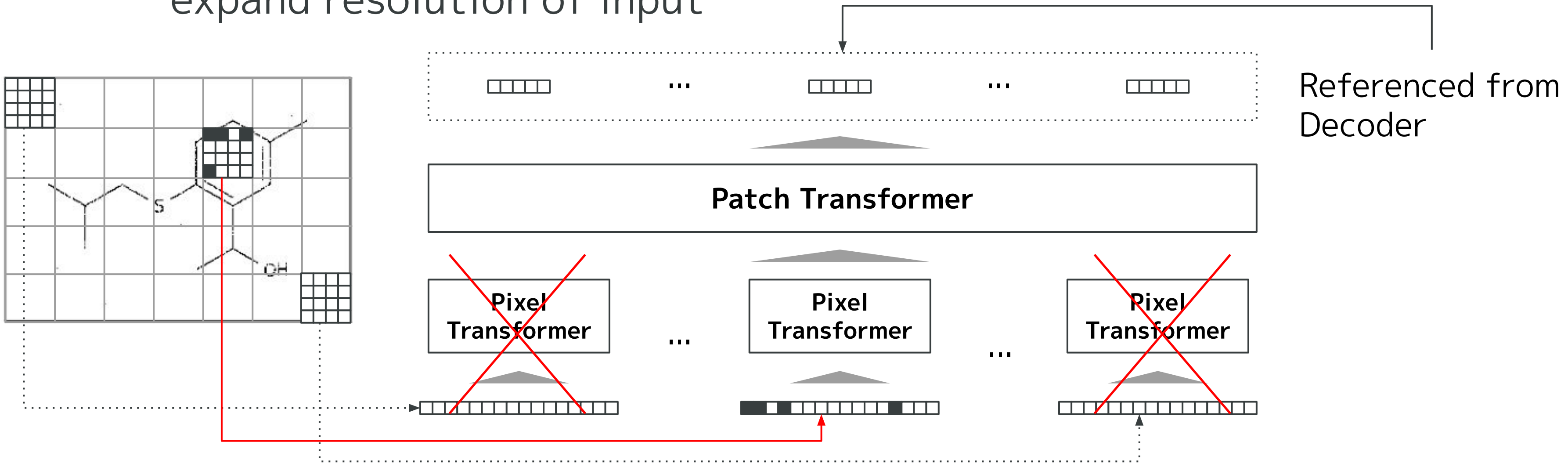
Phase1: TNT Encoder

- Use Transformer in Transformer (TNT) as Encoder
  - Reported this works well by hengck23
  - TNT encodes the relationships among pixels by Transformer, unlike ViT which encodes these by linear mapping



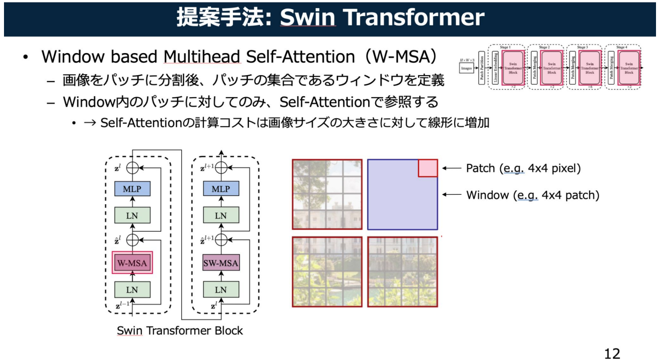
Phase1: TNT Encoder with variable patches（CV:1.30, LB:2.35）

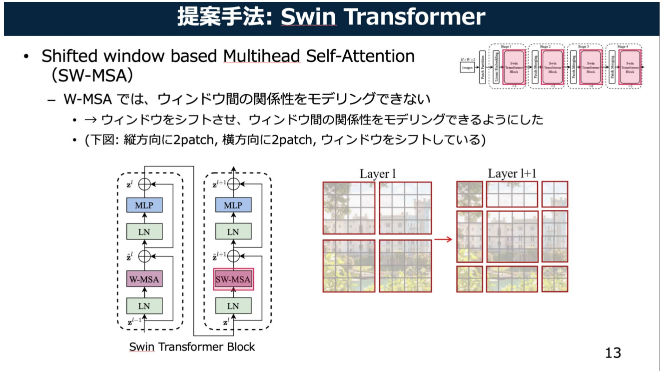
- Majority of the areas are empty in images from this competition
  - Apply TNT on only patches where not empty.（by hengck23）
  - → Reduced computational cost and GPU memory, making it easier to expand resolution of input



Phase1: Swin Transformer Encoder（LB:2.18→LB:1.43）

- Swin transformer has good CV-LB gap（CV: 1.23, LB: 1.43）
  - Better than TNT（CV: 1.30, LB: 2.18）
  - (We don’t have a enough explanation for this though…)





https://www.slideshare.net/DeepLearningJP2016/dlswin-transformer-hierarchical-vision-transformer-using-shifted-windows