Kaggle Competition: Bristol-Myers Squibb Molecular Translation 3rd place solution

Team: kyamaro
(KF + lyakaap + camaro)

2021.06. DS輪講

Kazuki Fujikawa
株式会社ディー・エヌ・エー + 株式会社 Mobility Technologies

AGENDA

- コンペティション概要
- ベースライン
- Solution

AGENDA

- **コンペティション概要**
- ベースライン
- Solution

コンペティション概要: 背景

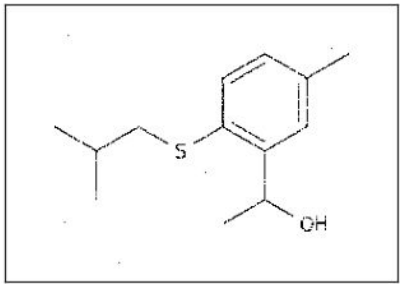
- 薬品などに有用な化学構造は論文・特許で公開される
  - 古い出版物にはInChIなどの機械で読める記述が無く、検索が困難
  - 検索が可能になると、車輪の再発明を防げたり、データマイニングで新たな化学構造を見つける助けになる
- 既存の構造式OCRツールは性能に課題がある
  - 古い出版物は画像に破損が含まれている場合が多い

破損が含まれる画像に対しても適用可能な化合物OCRを作りたい

コンペティション概要: タスク

- 与えられた化学構造画像に対するInChIを推定する

入力: モノクロ画像



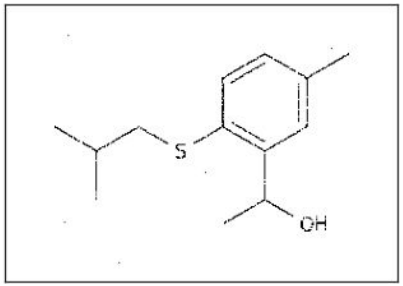
出力: InChIテキスト（改行は不要）

InChI=1S
/C13H20OS
/c1-9(2)8-15-13-6-5-10(3)7-12(13)11(4)14
/h5-7,9,11,14H,8H2,1-4H3

コンペティション概要: タスク

- 与えられた化学構造に対するInChIを推定する

入力: モノクロ画像



出力: InChIテキスト（改行は不要）

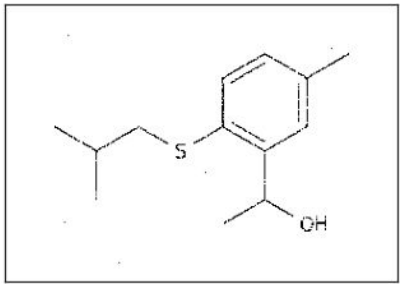
InChI=1S

InChIの規格を表すレイヤー
“1”: version
“S”: standard
（今回のデータセットは全てInChI=1S）

コンペティション概要: タスク

- 与えられた化学構造に対するInChIを推定する

入力: モノクロ画像



出力: InChIテキスト（改行は不要）

InChI=1S

/C13H20OS

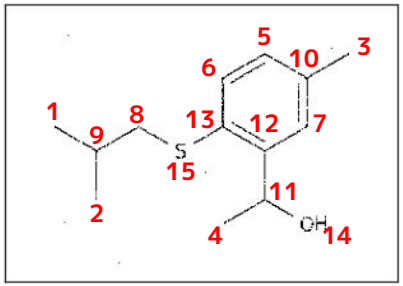
/c1-9(2)8-15-13-6-5-10(3)7-12(13)11(4)14

化学式レイヤー
分子を構成する原子とその個数を記述

コンペティション概要: タスク

- 与えられた化学構造に対するInChIを推定する

入力: モノクロ画像



出力: InChIテキスト（改行は不要）

InChI=1S
/C13H20OS

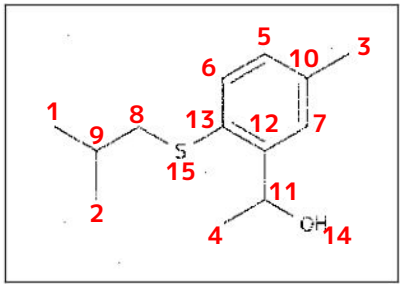
/c1-9(2)8-15-13-6-5-10(3)7-12(13)11(4)14

原子の結合情報を表すレイヤー
一貫するルールに基づいて原子に番号を振り、
分岐は括弧書きで表しながら1列の文字列で表す
e.g. 1番の次は9番、分岐して2番、戻って8番、...

コンペティション概要: タスク

- 与えられた化学構造に対するInChIを推定する

入力: モノクロ画像



出力: InChIテキスト（改行は不要）

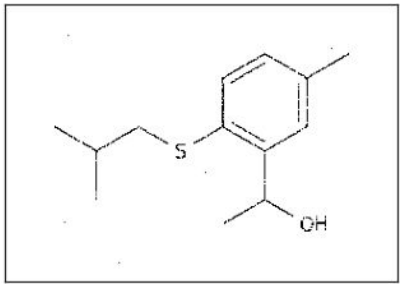
InChI=1S
/C13H20OS
/c1-9(2)8-15-13-6-5-10(3)7-12(13)11(4)14
/h5-7,9,11,14H,8H2,1-4H3

各原子が保有する水素の数を表すレイヤー
/hは接頭語、5-7, 9, 11, 14番の原子は1個の水素、
8番に2個の水素、1-4番に3個の水素が付いている
ことを表す

コンペティション概要: タスク

- 与えられた化学構造に対するInChIを推定する

入力: モノクロ画像



出力: InChIテキスト（改行は不要）

InChI=1S
/C13H20OS
/c1-9(2)8-15-13-6-5-10(3)7-12(13)11(4)14
/h5-7,9,11,14H,8H2,1-4H3

コンペティション概要: 評価

- 編集距離（Levenshtein distance）で評価
  - 推定InChIからGT InChIへ変換するのに必要な最小の編集回数
    - 編集: ”置換”, ”削除”, “追加”
    - ex. kitten → sitting: 編集距離3
      1. kitten → sitten（replace k → s）
      2. sitten → sittin（replace e → i）
      3. sittin → sitting（append g）

AGENDA

- コンペティション概要
- **ベースライン**
- Solution

ベースライン: Image captioning framework

- 一般的な Image captioning タスクのフレームワークで、InChIを単なる文字列と考えて系列生成
  - レイヤー毎に意味合いは異なるが、予想以上に上手くいく
  - EffnetなどのCNNベースよりも、TransformerベースのEncoderの方が明確に性能が良かった（弊チーム調べ）
    - 今回のタスクは、”何が” “どこに” 写っているのかを Decoder に伝える必要があったので、TransformerベースのEncoderの方が有利だった？

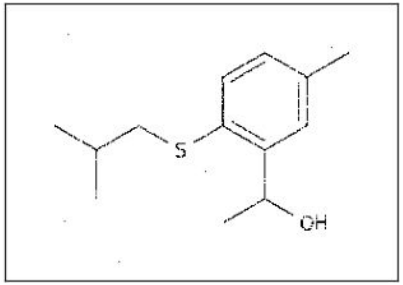
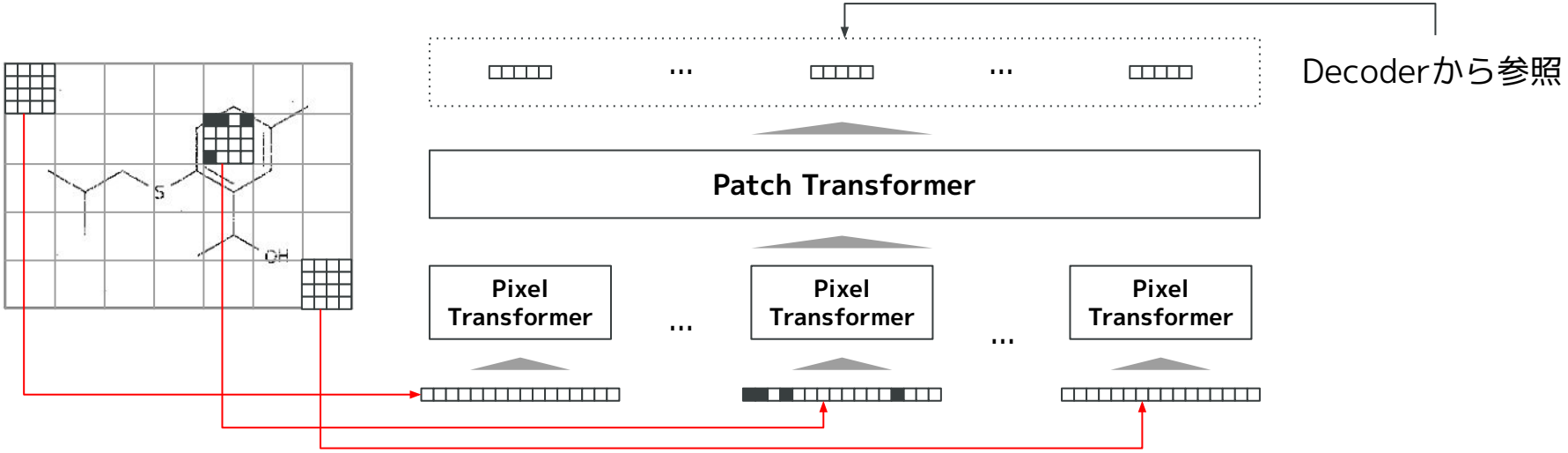


Image Encoder → Transformer Decoder → InChI=1S/C13H20OS...

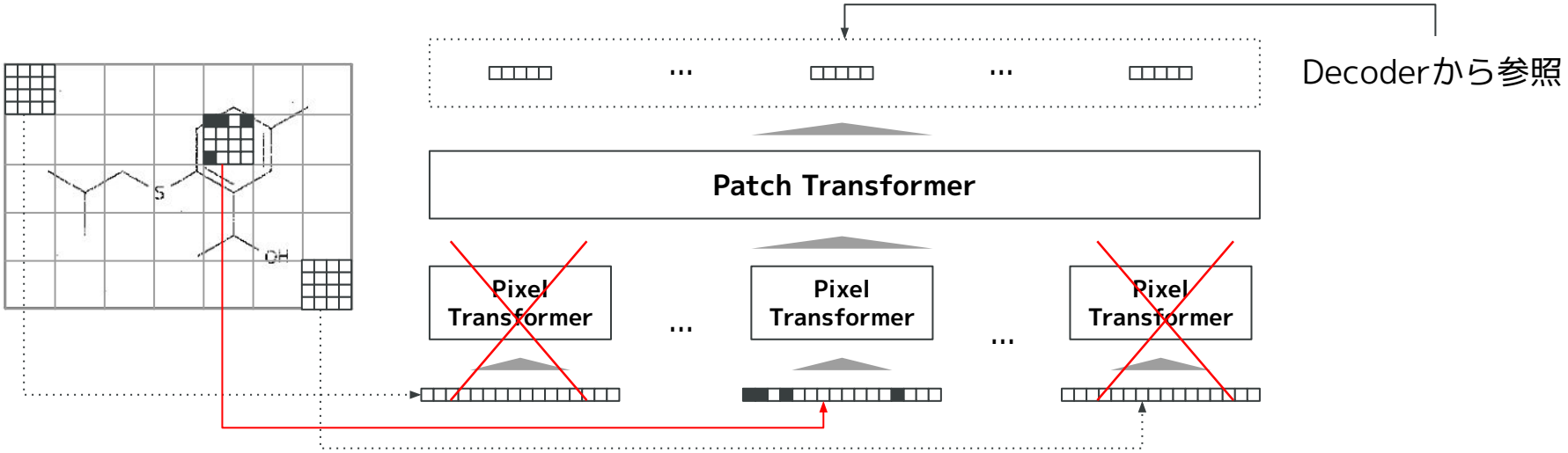
ベースライン: TNT Encoder

- Transformer in Transformer (TNT) をEncoderとして利用することが有効であることがシェアされていた（by hengck23）
  - TNT: 通常のViTのフレームワークに加え、ピクセル間の関係性もTransformerでエンコードする（ViTは線形写像）



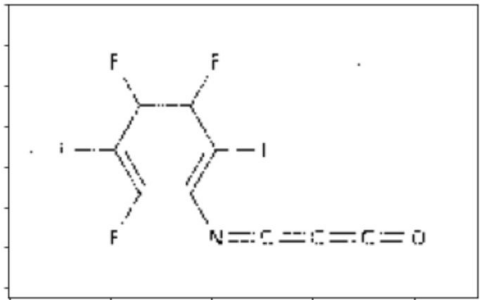
ベースライン: TNT Encoder with variable patches（CV:1.30, LB:2.35）

- 今回のタスクでは、”何も写っていない領域” を抽出しやすい
  - 何か写ったパッチに絞ってTNTを適用（by hengck23）
  - → 計算コスト・GPUメモリ削減に繋がり、解像度拡大を容易にした



ベースライン: RDKit normalization（後処理）

- 生成したInChIをRDKitで標準化（by nofreewill）
  - MolToInchi(MolFromInchi(inchi)) を実行
  - 原子に対する番号振り間違い程度であれば、この処理で標準化が可能



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

- コンペティション概要
- ベースライン
- **Solution**

Solution: Overview

Phase1: Image Captioning学習

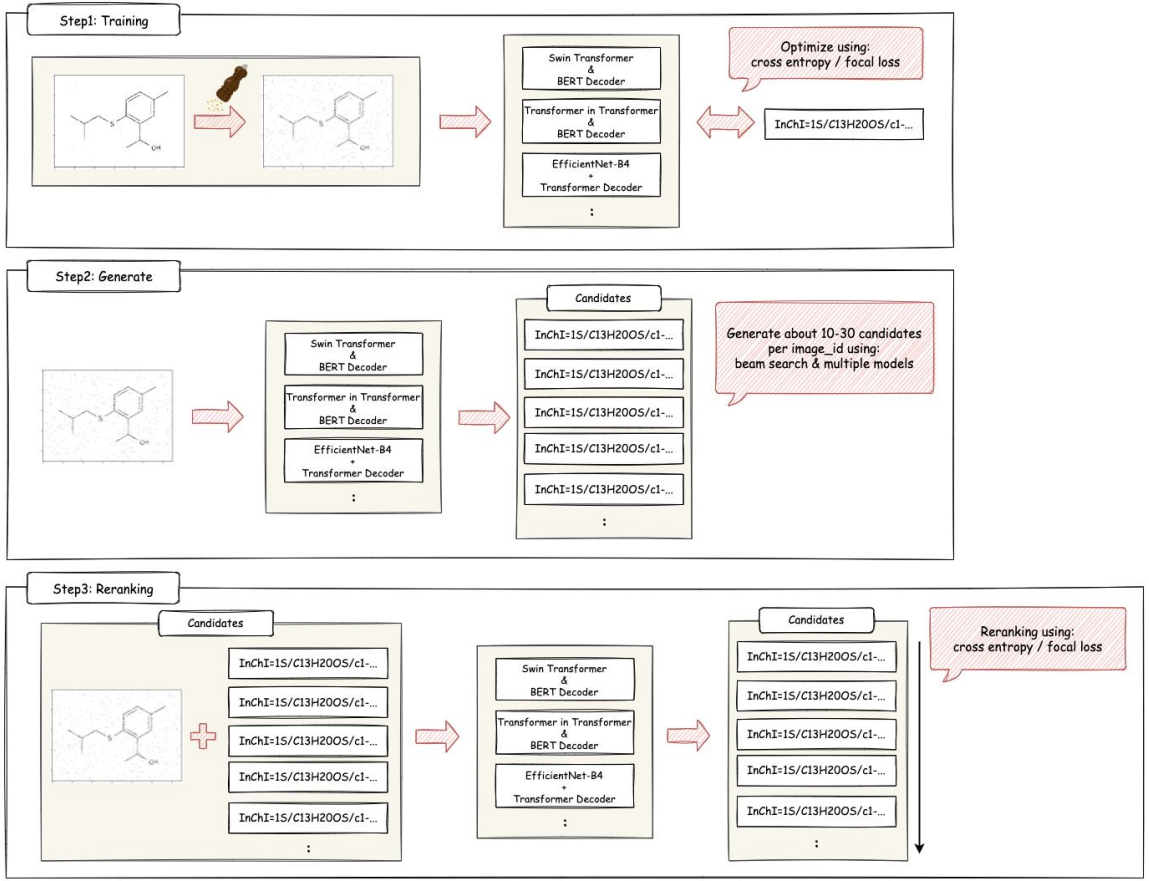
- Swin Transformer
- Salt & pepper noise
- Focal loss
- Train longer

Phase2: InChI候補生成

- Beam search
- Logit ensemble

Phase3: InChI候補リランキング

- RDKit標準化の利用
- 複数モデルで再評価



Solution: Overview

Phase1: Image Captioning学習

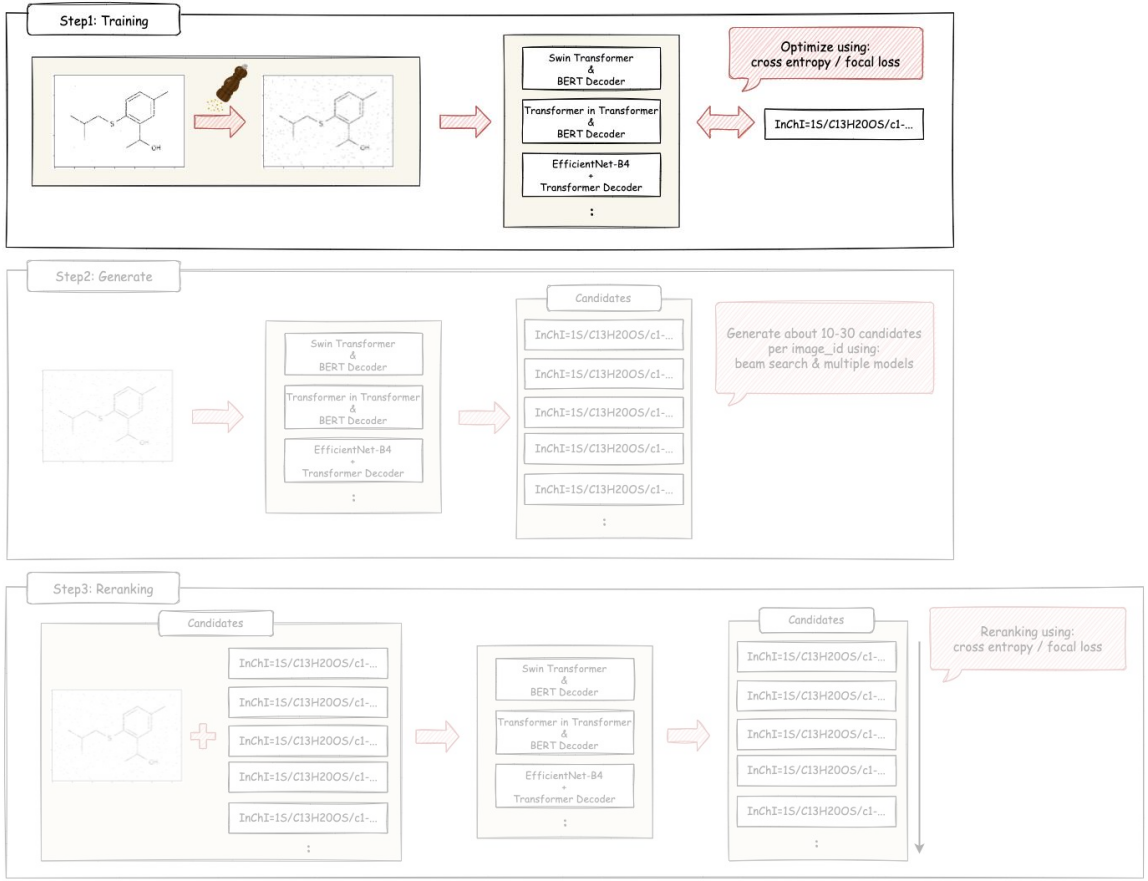
- Swin Transformer
- Salt & pepper noise
- Focal loss
- Train longer

Phase2: InChI候補生成

- Beam search
- Logit ensemble

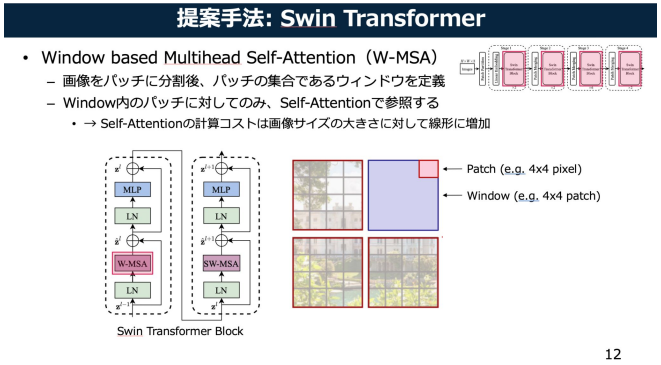
Phase3: InChI候補リランキング

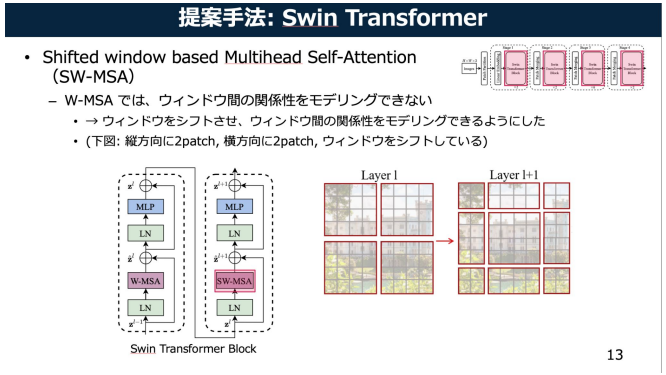
- RDKit標準化の利用
- 複数モデルで再評価



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

- CV-LBギャップが大幅に改善（CV: 1.23, LB: 1.43）
  - TNT（CV: 1.30, LB: 2.18）と比較すると顕著
  - ちょうど論文読み会の順番が回ってきたので、Swin Transformerを勉強してみたが、原因はよくわからず



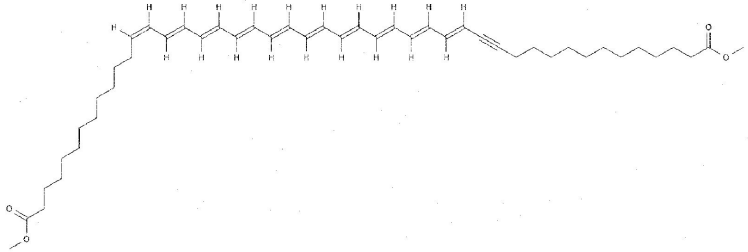


https://www.slideshare.net/DeepLearningJP2016/dlswin-transformer-hierarchical-vision-transformer-using-shifted-windows

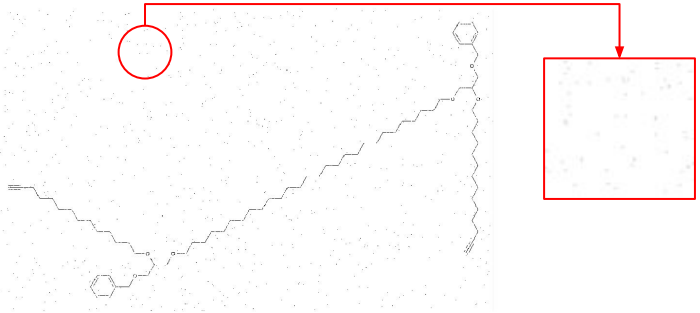
Phase1: Swin ⇔ TNT 間の生成結果の比較

- SwinはCV-LBが一貫していたため、Swinの生成結果をGTだと考えた時の、TNTとのLevenshteinを比較
  - Valid: 1.02・Test: 2.16と、2モデル間でもギャップがあることを確認
- Levenshteinが大きい順に、Valid / Test を見比べた
  - （当然だが）Valid / Test 共に、サイズの大きい化合物が目立つ
  - Testの方は、ごま塩ノイズが特に目立っていた

Levenshtein最大化合物（valid）



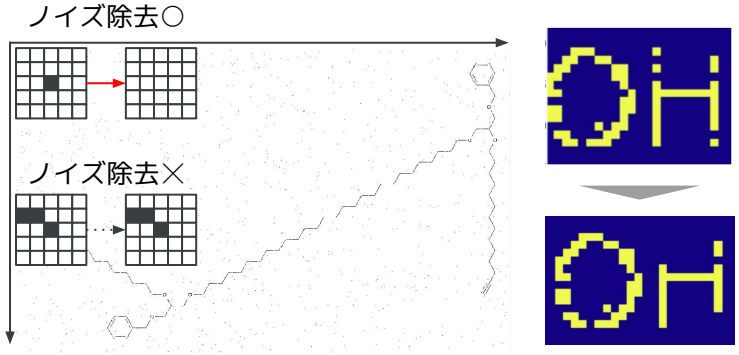
Levenshtein最大化合物（test）



Phase1: ごま塩をデノイズして予測（LB: 2.18→1.26）

- 同じ学習済みモデルに対し、testをデノイズして予測させる
  - ある点を中心とする正方形中に、一つも黒点がなければ白に置き換える
  - 再訓練していないにも関わらず、CV-LB Gapが激減
  - ただし、デノイズで重要な点を削ってしまうこともあり、CVも少し悪化していた

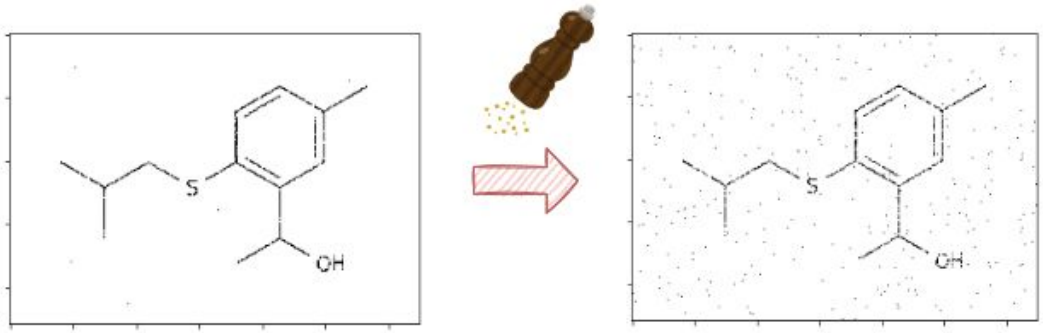
デノイズフィルタ（k=5）概要



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: 訓練時にごま塩追加（+解像度拡大）（LB: 1.26→1.06）

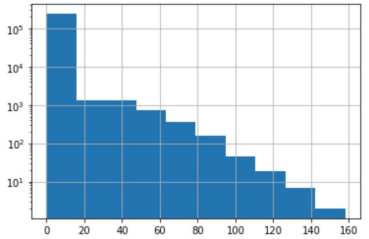
- 訓練時、ごま塩をランダムに追加して学習
  - （解像度拡大も同時にしたので差分がわかりづらいが）
    デノイズ無しでも CV-LB Gap が解消された



Phase1: CE Loss → Focal Loss

- 今回のタスクでは、大多数のサンプルが完全一致で当てられており、一部のサンプルで大きなLossが生じる状態になっていた
  - 参考: Swin Transformer（CV: 0.98, LB: 0.99）
    - Levenshtein=0: 87%
    - Levenshtein=1: 7%
    - …
- Focal Lossで、簡単なサンプルに対するLossの影響を小さくすることで、学習効率が向上
  - $\mathrm{FL}(p_t) = -(1-p_t)^{\gamma}\log(p_t).$

Levenshtein分布（Logスケール）



Phase1: Train longer

- Camaroさんとチームマージした際、CVの差に衝撃を受けた
  - Best single (Camaro): CV=0.66, LB=0.87
  - Best single (KF): CV=0.98, LB=0.99
- 主要因はepoch数ではないかと推測
  - Camaro: 25-50 epoch (TPU 7days?)
  - KF: 10 epoch (A100 7days)
  - Train/Valid lossが共通して単調減少していたので未学習気味とは思っていたが、他モデルの検証を優先して試せていなかった
- 3 epoch追加学習するとCV=0.98 → 0.91に改善
  - もっと長くするともっと伸びそう...
  - とはいえ計算コスト的に辛いので、TPU使っておけばよかったか...

Phase1: Iyakaap part

- Encoder:
  - Swin-transformer（size=384x384）
  - EfficientNet-v2の後段にViTをくっつけたもの（size=448x448）
    - Patch embeddingにCNNを使うイメージ
- Decoder:
  - 三層のtransformer
    - これ以上深くすると学習が不安定に
- 画像のresizeにPIL.Image.resizeを使う
  - cv2.resizeなどよりもPILの方が劣化が少ないらしい（参考）
- Pseudo labelを使ったfine tuning
  - CV/LBのギャップを埋めるのに貢献。CV/LBも0.3ほどスコア向上。
- Sequence bucketingによる学習の高速化

Solution: Overview

Phase1: Image Captioning学習

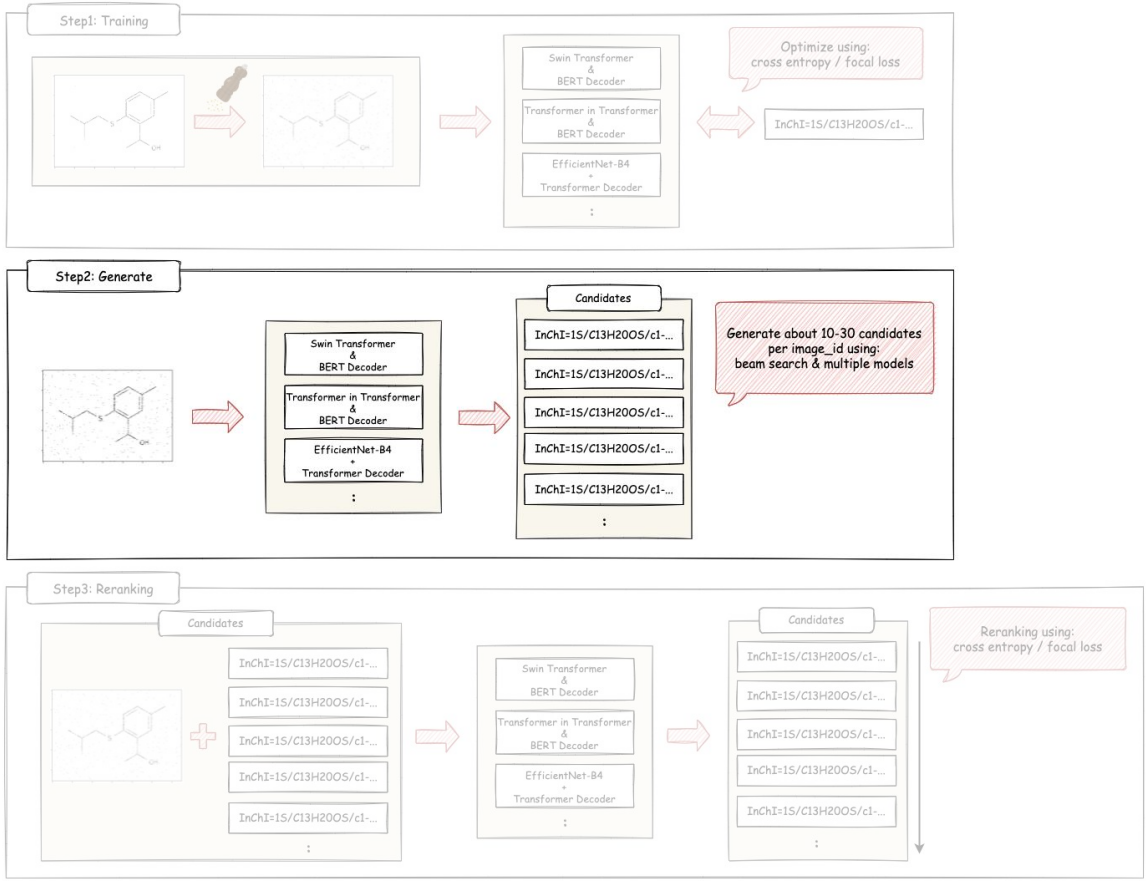
- Swin Transformer
- Salt & pepper noise
- Focal loss
- Train longer

Phase2: InChI候補生成

- Beam search
- Logit ensemble

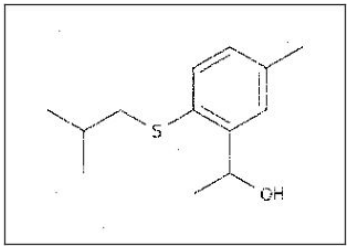
Phase3: InChI候補リランキング

- RDKit標準化の利用
- 複数モデルで再評価



Phase2: Beam Search

- Phase3でリランキングを行うため、Phase2では良質で多様な候補を生成する必要があった
  - 一つの手段としてBeam Searchを利用
  - 候補が格段に増え、Greedyの生成結果と組み合わせて Phase3 のロジックを適用することで、スコアが大きく改善された
  - 特に難しいサンプルに対してビーム幅を広げて（Beam=32など）生成することで、効率的に候補追加を行うことができた（by Iyakaap）

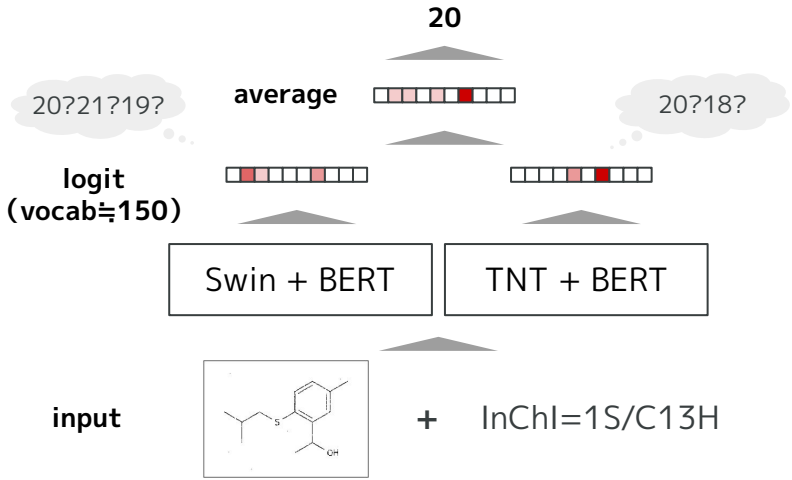


InChI=1S/C13H20OS…
InChI=1S/C13H21OS…
InChI=1S/C11H21OS…
InChI=1S/C12H20OS…

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

Phase2: Logit ensemble

- 各トークン予測に使うロジットをアンサンブル
  - Swin, TNT の2モデルだけでも割と多様性があり、アンサンブルしながら系列生成することで、より正確に予測できるようになった
  - ただし、全てのモデルが単一のプログラムで動作する必要があり、チーム内でモデルをシェアするのは難易度が高かった



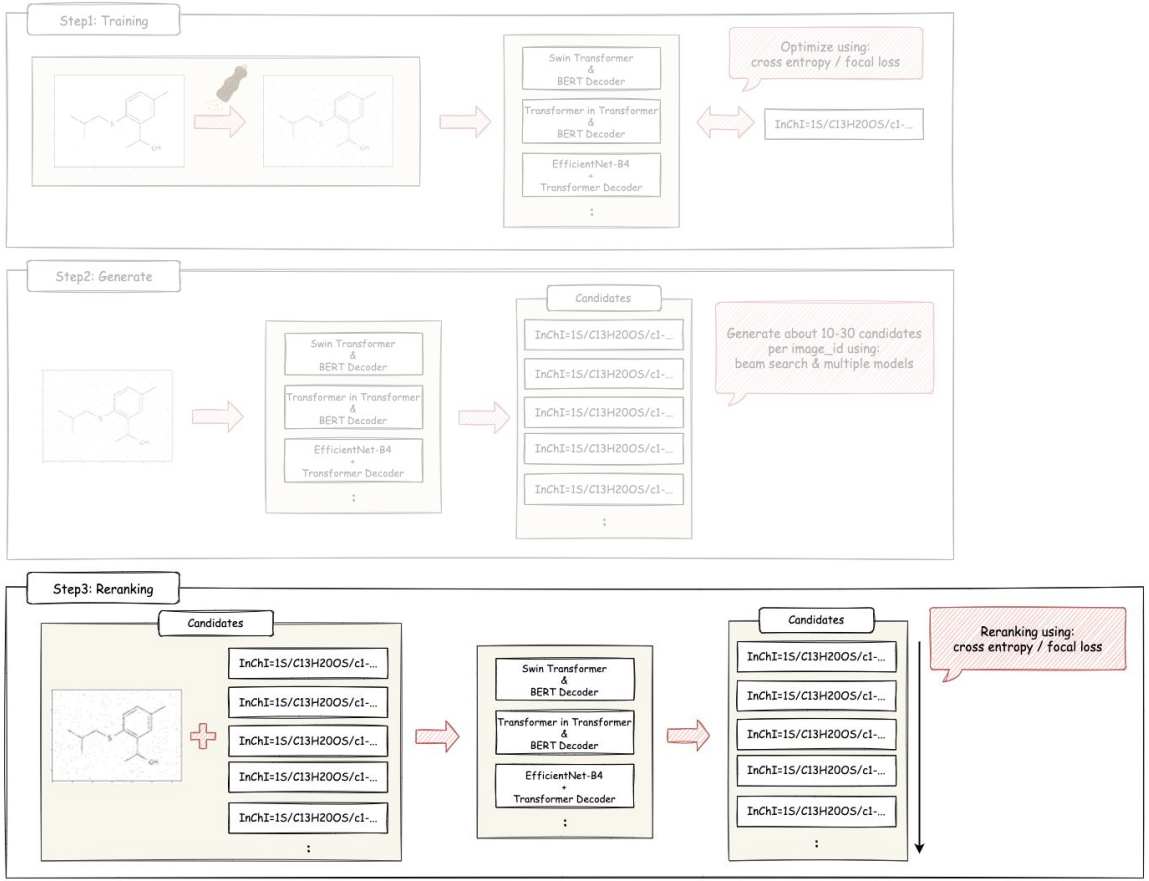
Model	CV
Swin (beam=1)	0.98
TNT (beam=1)	1.04
Swin+TNT (beam=1)	0.83

Solution: Overview

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

Phase2: InChI候補生成
- Beam search
- Logit ensemble

Phase3: InChI候補リランキング
- RDKit標準化の利用
- 複数モデルで再評価



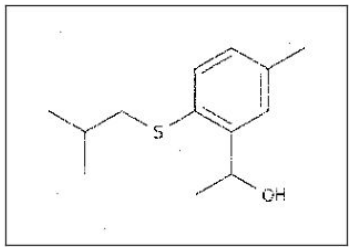
Phase3: RDKit標準化の利用

- RDKit標準化も活用し、結果をリランキングする
  - InChI生成時、基本的には尤度が大きくなるように生成しているものの、1文字など些細なミスに対して尤度は甘く出がち
  - RDKitで標準化すると逆に尤度が低くなるものも存在
  - 1文字ミスなどはInChI文法エラーになる場合も多く、RDKitでパースできるかどうか（is_valid={0, 1}）も含めた以下ロジックが強力だった
    ※ score: cross entropy loss or focal loss

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

Phase3: 複数モデルで再評価

- 複数モデルの生成結果を相互レビューさせる
  - 全InChI候補に対して各モデルでlossを計算し、平均値を利用する
  - loss値のスケールが合わないモデルに対しては、一度値をランク化した上で平均を取るようにすることで解決
  - 各サンプルに対する尤度を各モデルが推定できれば良いので、実装が異なっていても活用しやすい



	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

※ この例では同一のimageに対してRankを取っているが、実際には全imageに対するRankを取っている

最終モデル（一部）

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
Iyakaap	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

まとめ

- Train/Test Gapの発見が一つの鍵だった
  - 意外と上位でも見抜いているチームは少なかったぽい？
- データ規模が大きい場合は特にTPUが有効
  - Pytorchで不自由無く使える世の中になってほしい...
- 多様なモデルで候補生成 + リランキングの形式が上手くいった
  - 基本的に、候補・モデル共に、足せば足すほど伸びていった。
  - 特にcamaroさんと混ぜた時に急激に伸びた。多様性重要。
- ビームサーチで尤度最大になる経路（InChI）を探しているはず
  だが、別モデルの出力を持ってきた方が尤度的にも優れていた
  - 枝刈りが強すぎて有望なパスを見落としている可能性？
  - 計算コスト的に難しいが、MCTS的に探索できるとより良かったかも？