採用課題
CIFAR-10の画像分類

上智大学情報理工学科3年
浜口 創

アジェンダ

01 実験結果

02 改善施策

03 次に試すべき改善施策

04 再現性・環境整備

01

実験結果

実験設定

項目	設定内容
データセット	CIFAR-10
モデル	ResNet50
評価指標	Accuracy, Loss, Precision, Recall, F1
可視化	学習曲線, 混同行列, 誤分類サンプル
再現性	乱数シード固定 (Seed=42), cuDNN設定

４つの改善施策で精度＋30%

CIFAR-10に対してResNet50の分類モデルを実装し、改善施策を比較した。

- 転移学習（全層FT）
- スケジューラーを導入
- 正則化
- データ拡張
  - 過学習を抑え、汎化性能を改善

最終的に、Val accは

67.2%→97.7%

と大きく向上

02

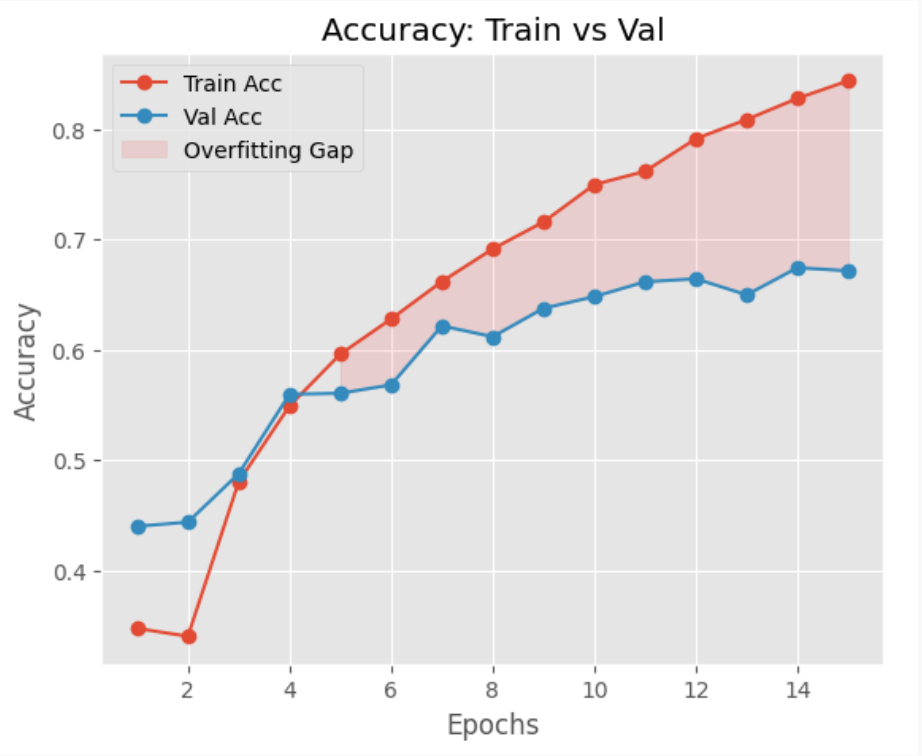
改善施策

ベースラインモデルの作成

条件

- データ拡張：正規化のみ
- Optimizer：Adam
- LR：0.001
- Epoch : 15
- Batch size : 512

結果



上記のグラフから**過学習**が発生している可能性があると考えた。

過学習の抑制実験

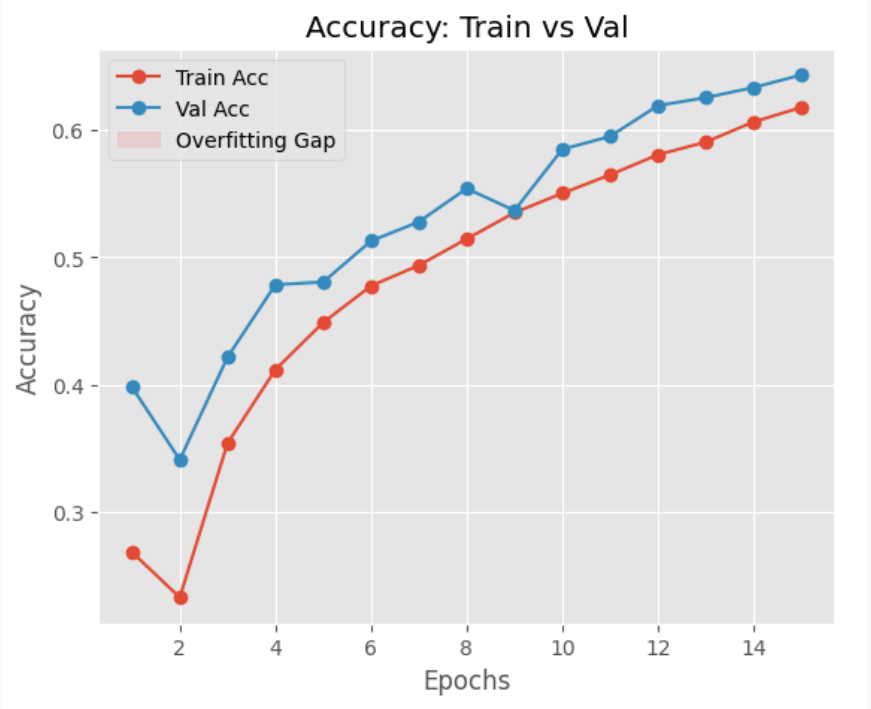
変更点

- データ拡張：

  flip/crop/colorjitterの追加

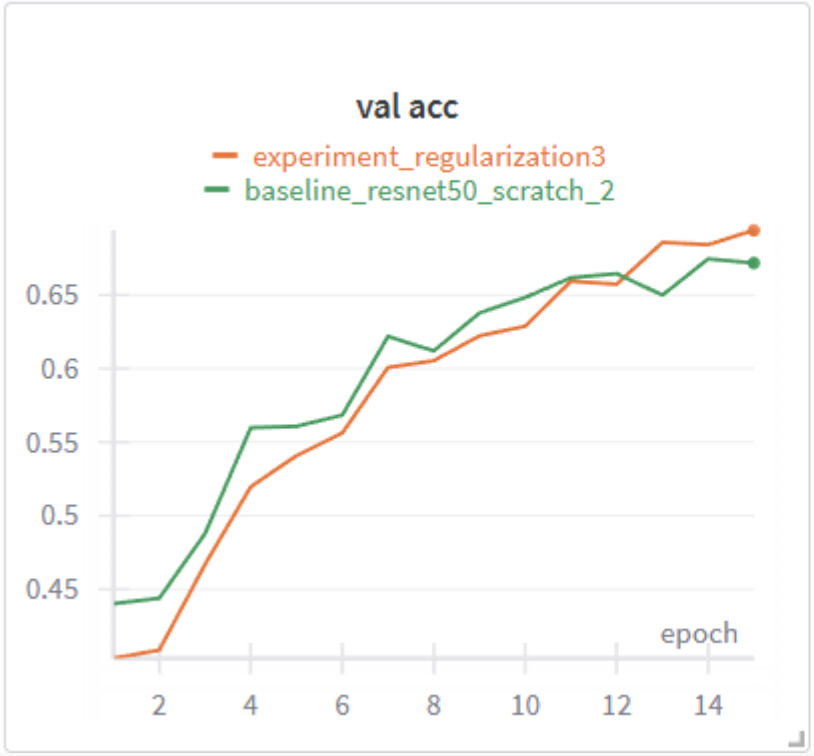
- 正則化(Weight Decay)：1e-4

結果



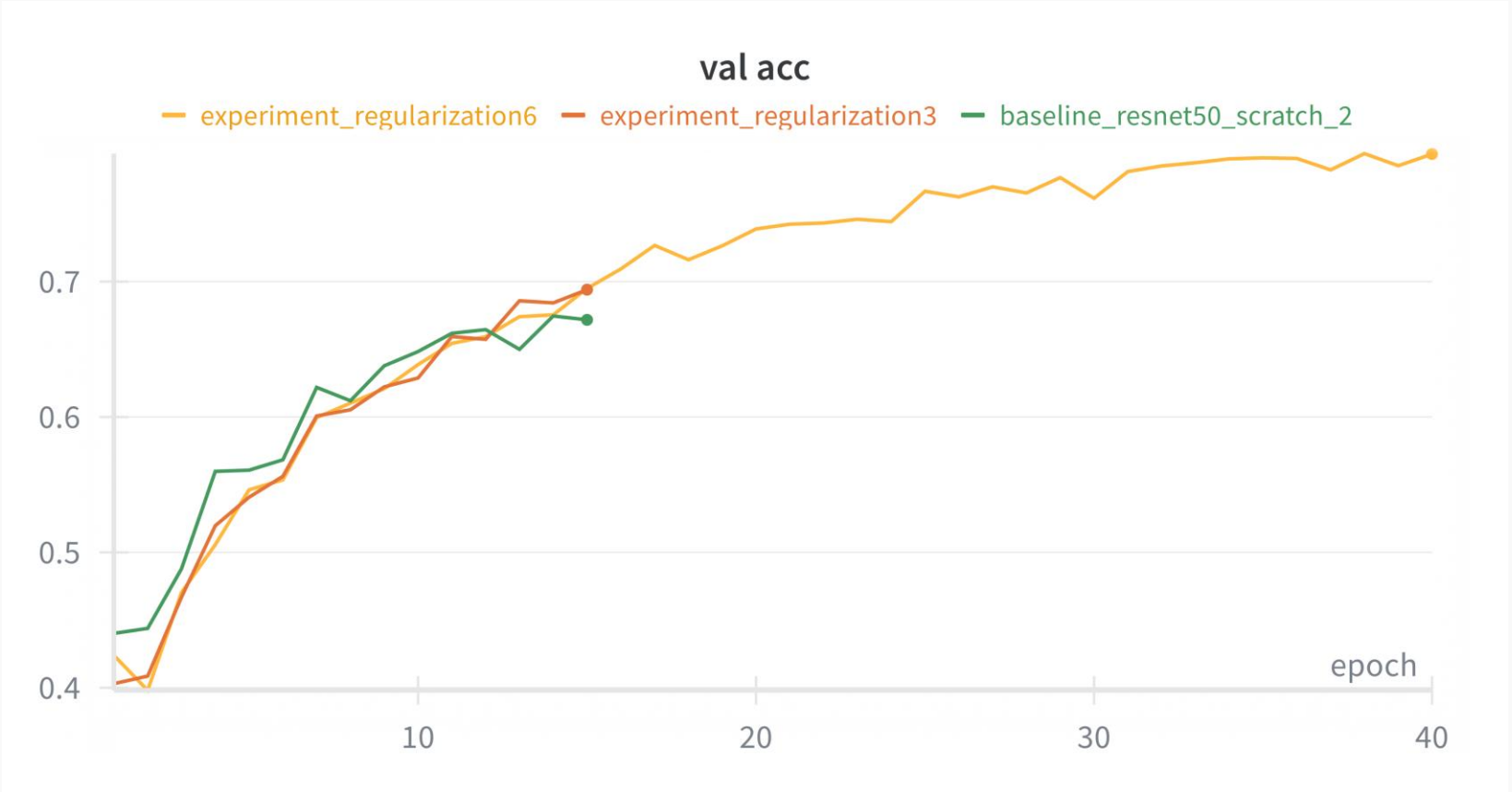
学習データの正解率とバリデーションデータの正解率を近づけることには成功。

過学習抑制実験の結果2



val lossは良くなっているが、<u>汎化性能はそこまで上がらなかった。</u>
データ拡張を緩めるなどもしたが、val accはほとんど一定。

過学習抑制実験の結果2



40エポックほどで頭打ちに見える。

誤分類サンプルを可視化

予想：plane, 正解：deer

予想：cat, 正解：frog

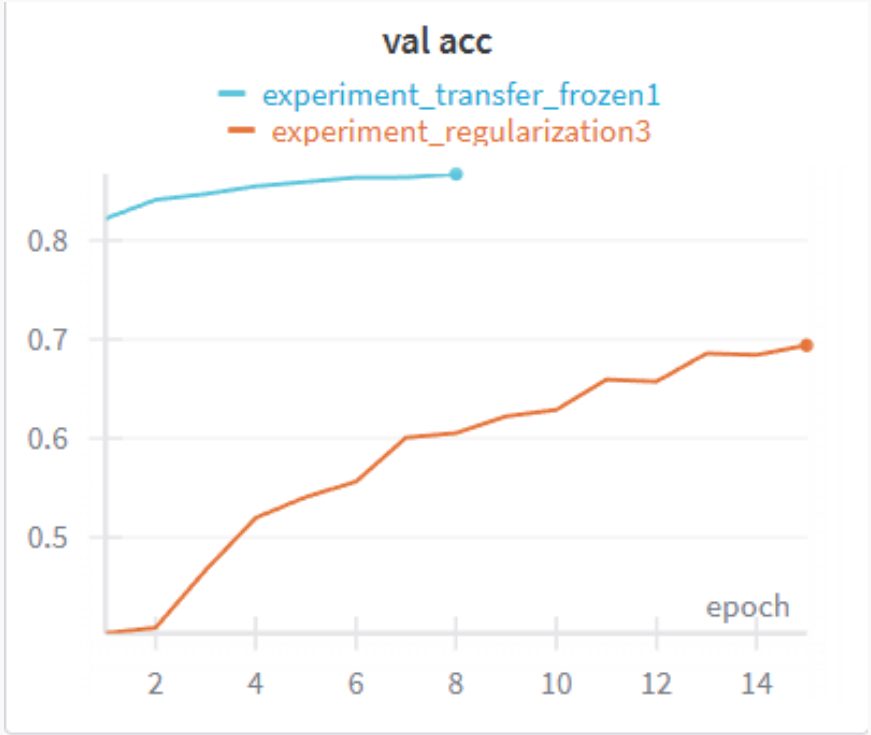
背景であったり、全体的な色であったり、雰囲気で判別しているのでは？

転移学習（全層凍結）

よって、解像度の良いImageNetで学習したResNetの重みを用いることで、
より**被写体の特徴**を抽出して推論できることを期待して、転移学習を導入する。

変更点

- 事前学習済みResNetの
  重みを用いる
- ImageNetの色平均で正規化
- 画像をresizeする。
  32×32 → 224×224



※大きく更新

時間がかかるのに驚く

Udemyのチュートリアルでは全層凍結の転移学習が一瞬で終わっていた。
逆伝搬以外にも順伝搬に時間がかかるのを把握していなかった。

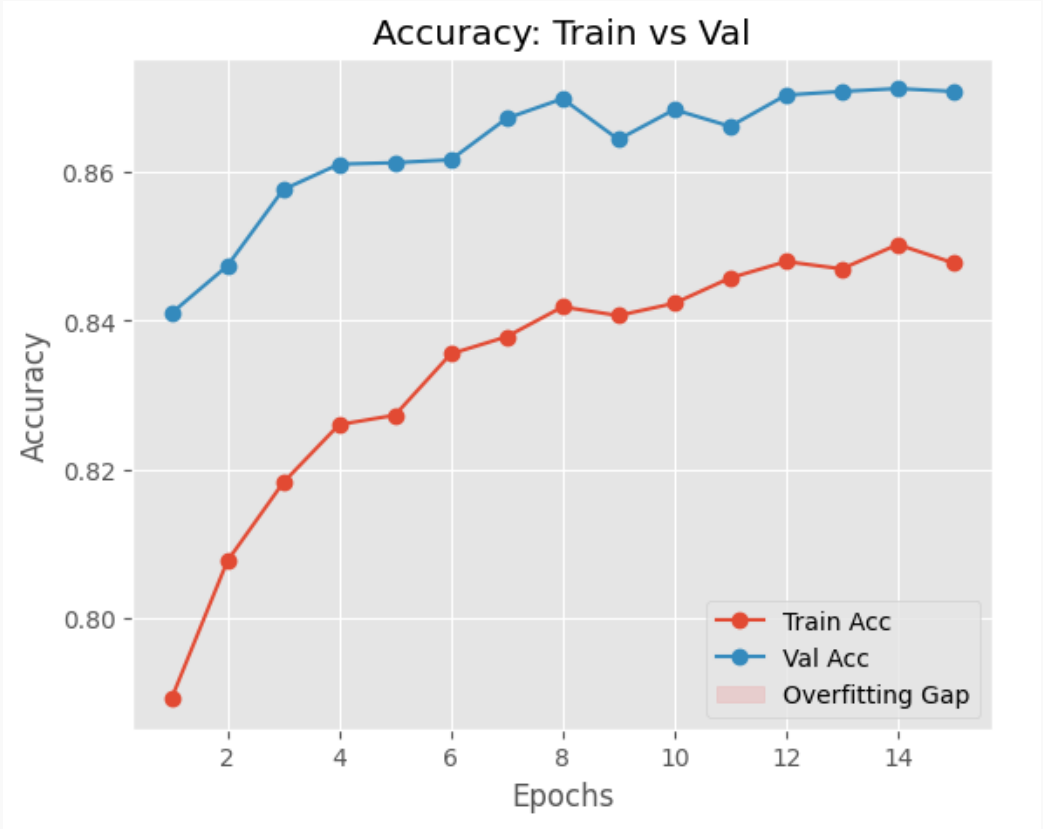
改善施策

- これはCPUでデータ拡張しているのが原因だと考えた。（特にresize）
- GPUでデータ拡張をするように変更。
- また、L4-GPUはメモリに余裕があったので、バッチサイズも増やす。

Val accの方が高い

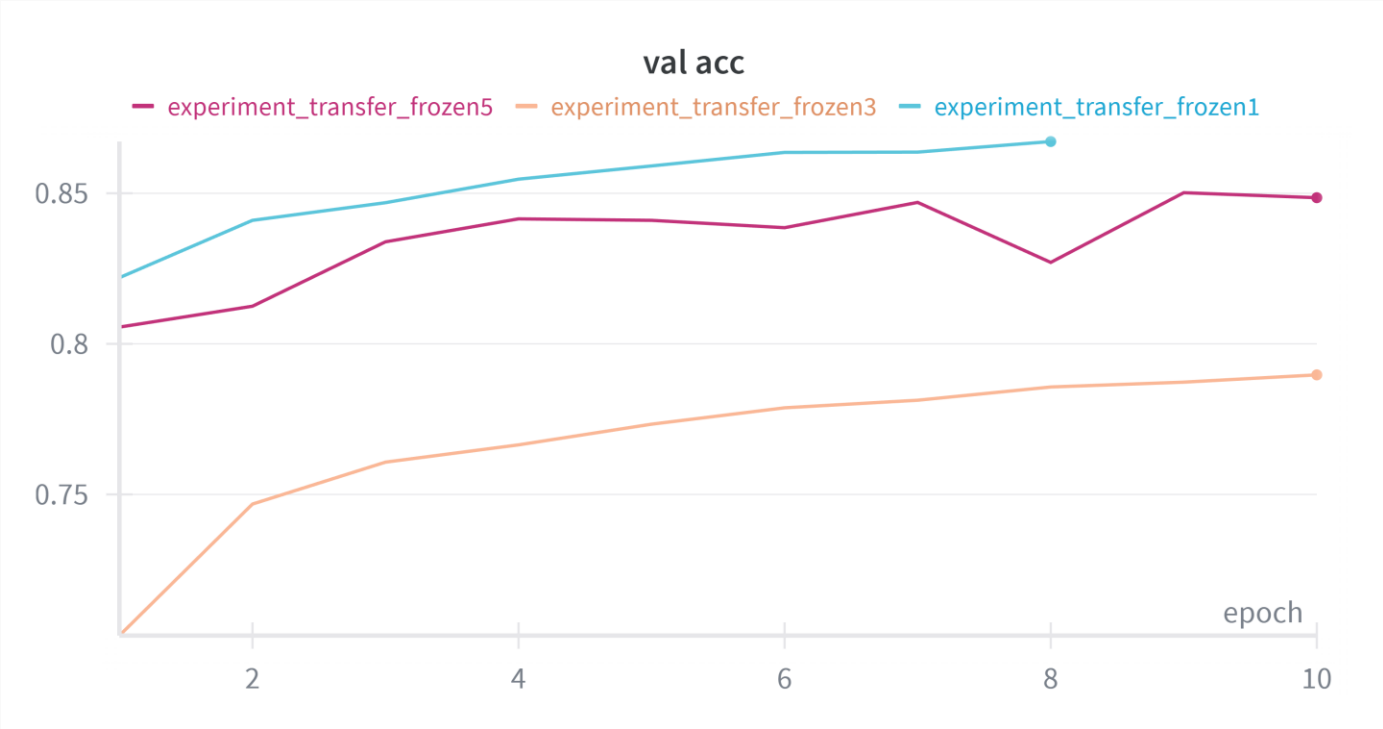
- 強いデータ拡張によって、
  学習データが難しくなってしまって
  いる可能性

- データ拡張を緩くしてみる



精度の低下…

- バッチサイズはあまり精度に関係ないと考えていたが、512にするとかなり精度が落ちた。
- また、データ拡張を強めるのも逆効果になってしまった。



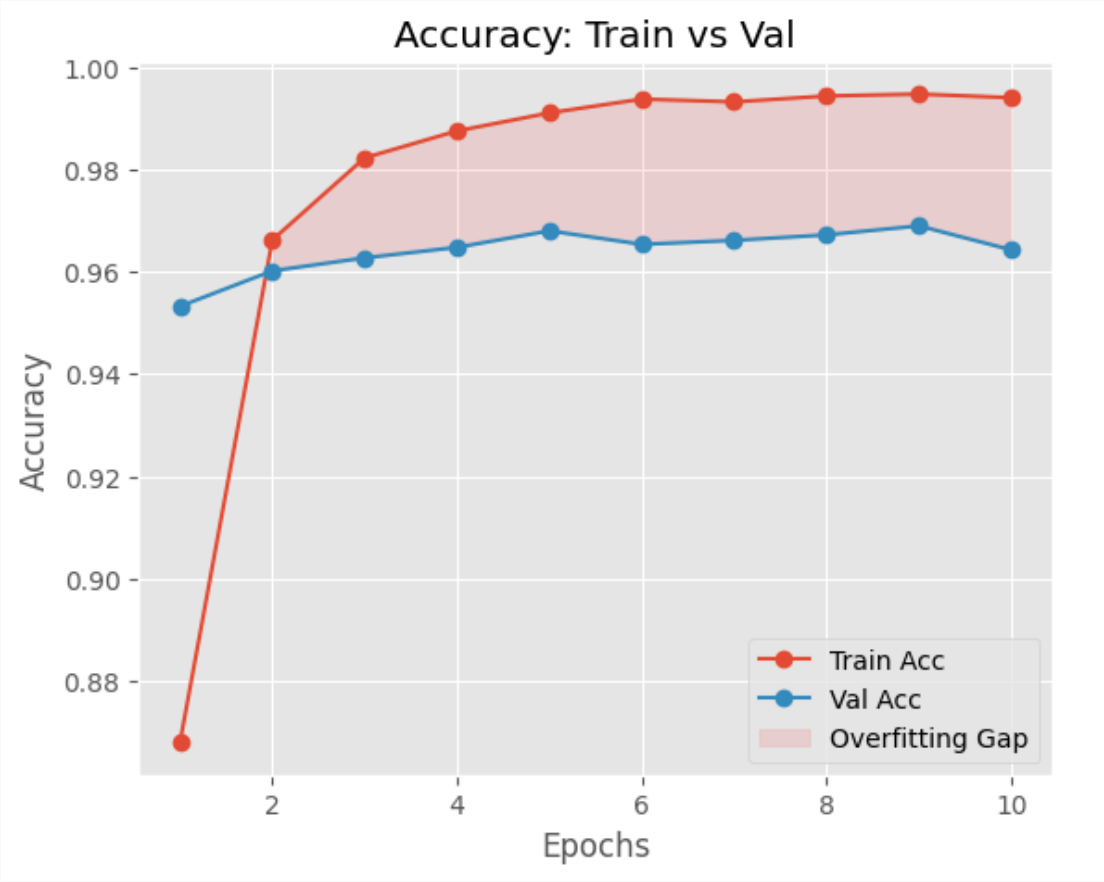
転移学習（全層FT）

学習済みResNetは細かい特徴に対して反応できるようになっている
しかし、CIFAR-10の画像は解像度が非常に粗く、
細かい情報が抜け落ちていて、この分類に対して無駄が多いのではないか

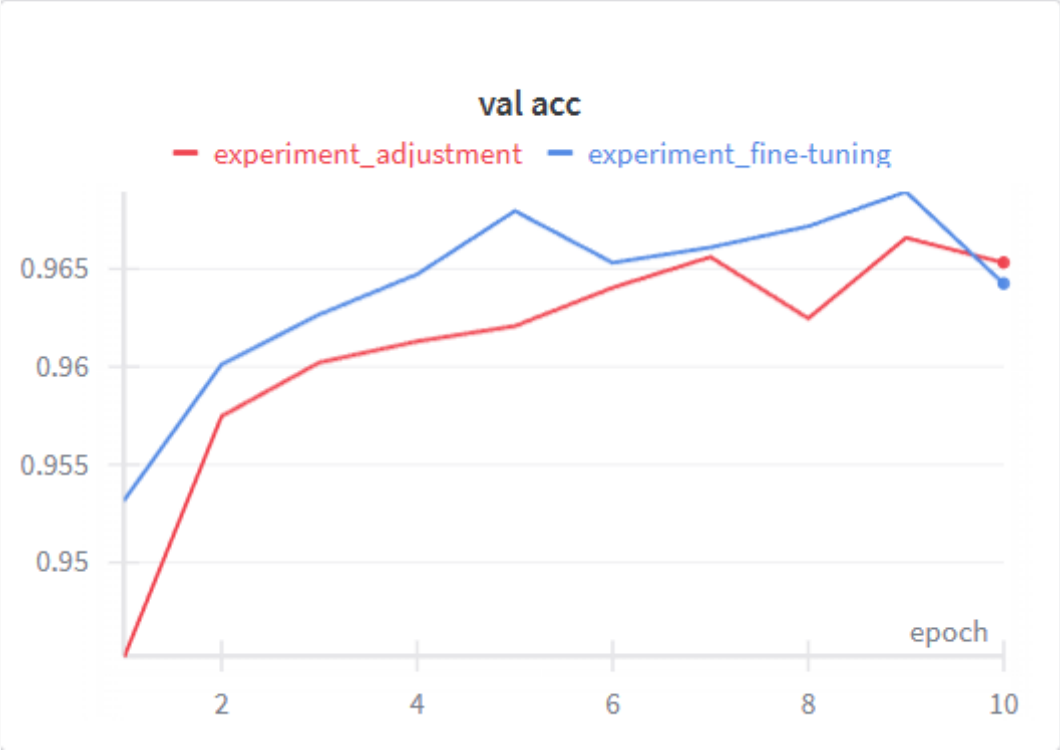
- CIFAR-10の分類で求められるのは、シルエットやパーツなどの情報を読み取ること。
- Fine-tuningすることで、学習済みResNetがよりCIFAR-10に適したモデルとなるのでは。

全層FTの結果

- 非常に良い精度が出た！
- 学習データのaccがほぼ100%
- 過学習の対策を取ることで
  より汎化性能が上がるのではないか？



実験結果



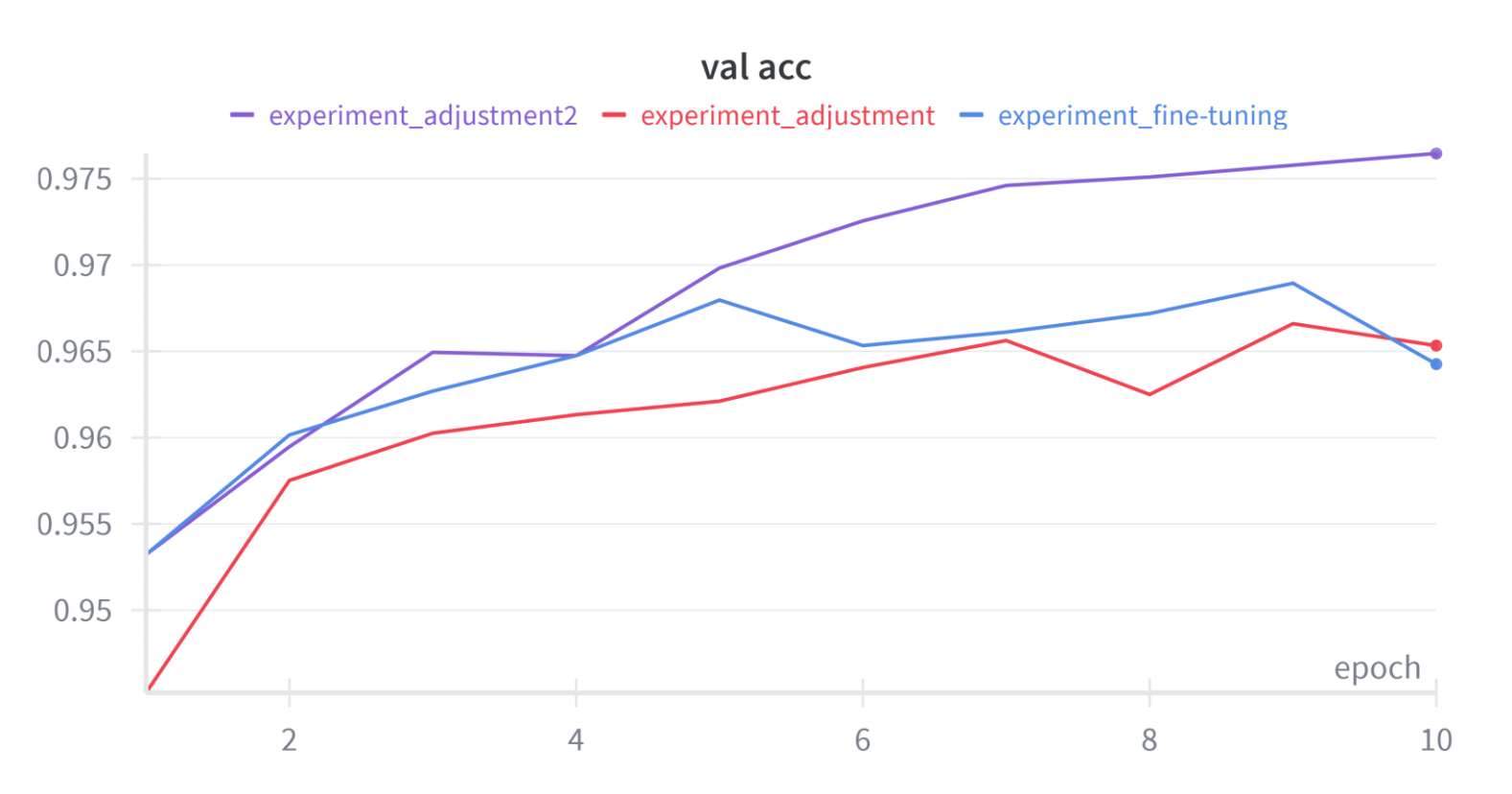
汎化性能はほとんど上がらず、lossも上がってしまう。

スケジューラーの導入

データ拡張やWDの値はいったん戻す

- 全層ファインチューニングは精度がいい。
- 学習後半で不安定になっている。
- LRを後半で小さくすることによって、収束しやすいようにする。

スケジューラー実験結果



さらなる精度の更新に成功！

Two-Stage training

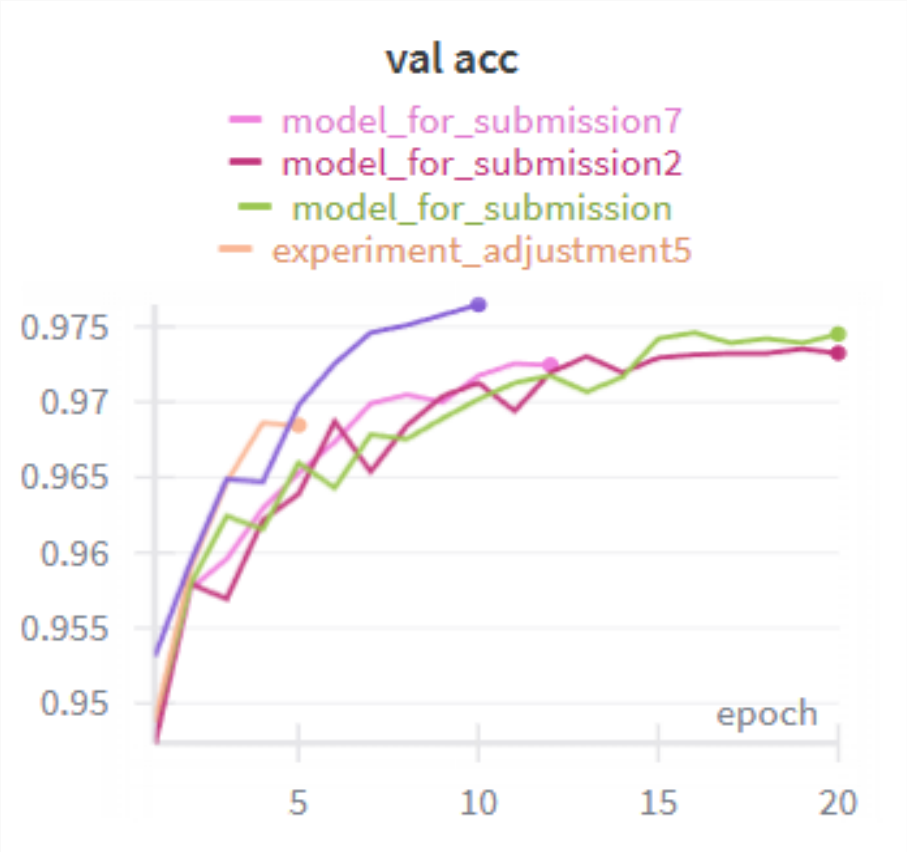
全層凍結で得た重みをベースとして、FTを行う手法

- 序盤で逆伝播する際にfc層の重みがランダムであると、大きく重みがずれてしまう可能性がある。
- 全層凍結モデルの重みを使って全層FTを行う。
- 初期の最適化が安定し、発散・過学習を抑えやすい

実験結果

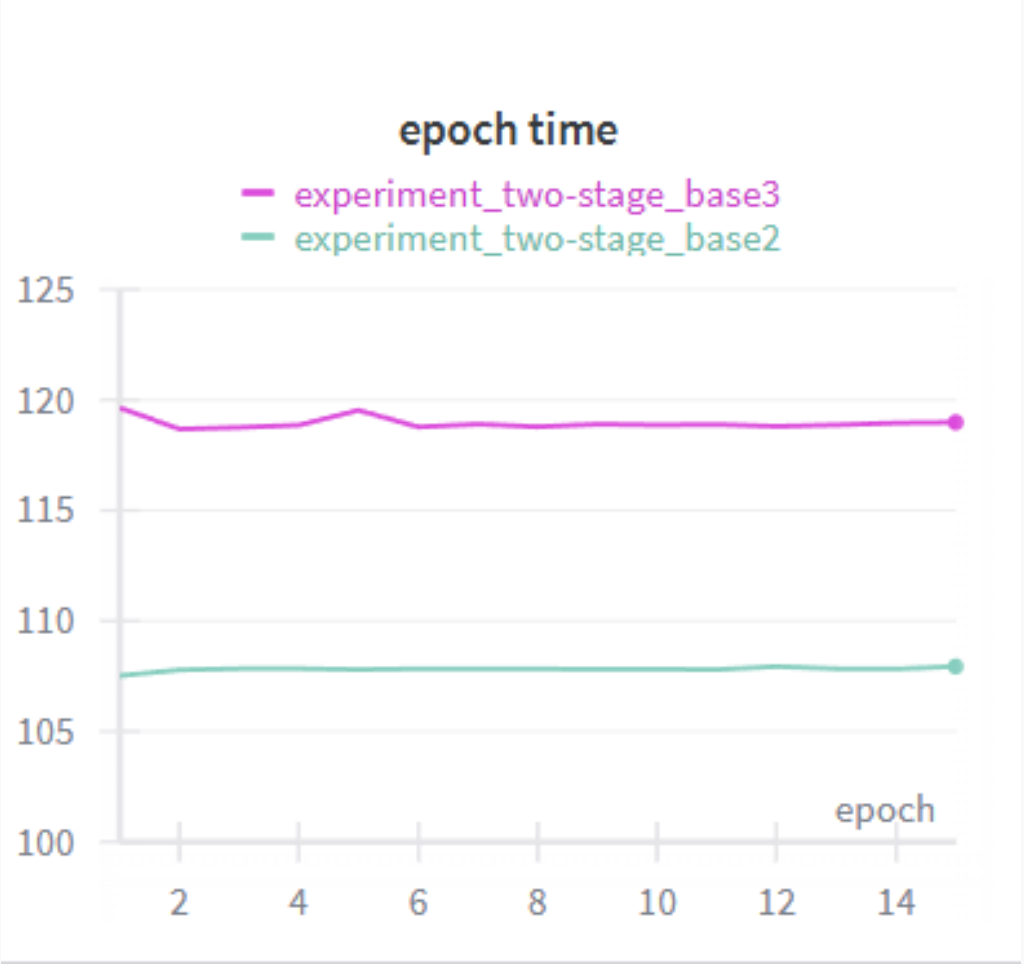
先の結果がとても良いものであった。

- 施策自体は悪くないはず
- 初期の探索の不安定さによって
  逆に良い精度にたどり着けた
- 再現性は担保しているため、
  凍結モデルを用いずに進める

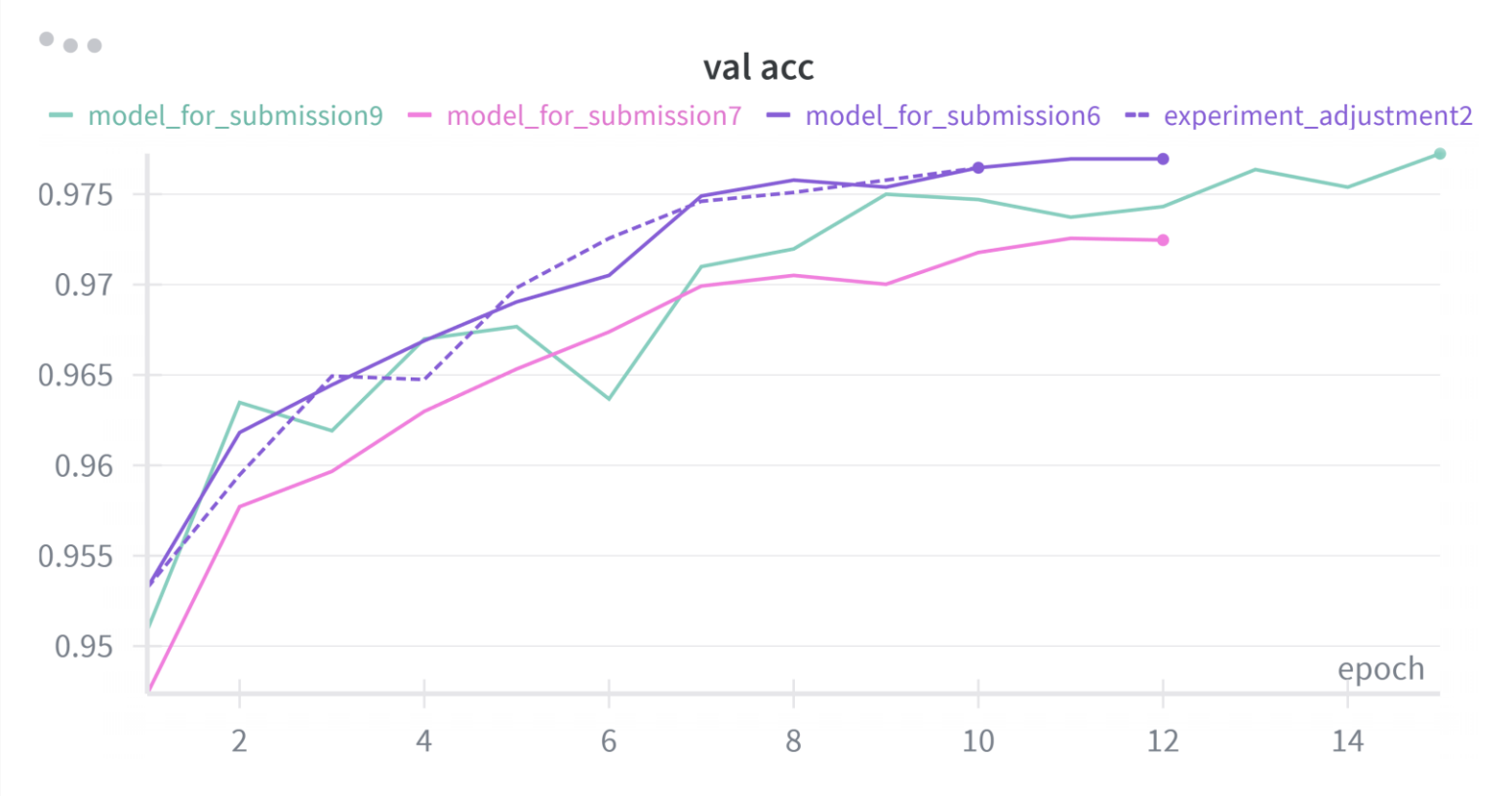


CPU/GPU前処理：実行時間差検証

- Two-stageモデルのベースとなる凍結モデルを作成する際に実行時間を測定した。
- 確実にGPU前処理の方が短いものの、**各エポック１０秒**ほどの差しかなかった。
- CPU前処理の方が精度が出やすい傾向があるので、最終的にはそちらを用いる。



エポック数など微調整を行い、最終モデル



CPU前処理・エポック数15の全層FTモデルが**97.7%**で最高

03

次に試すべき改善施策

考えられる改善施策

1. ベースとなるモデルの変更

   Efficient Net, ViTなど

2. 複雑なデータ拡張

   Mixup, CutMixなど

3. オプティマイザの変更

   AdamW, SGDなど

4. 浅い層のみ凍結したFT

   汎用的な部分は残して、細かい部分に注目する。中間層以降をFT

04

再現性・環境整備

再現性

- 学習や推論に使うファイルをすべてGitHubに追加
  - リポジトリのクローンを行うことで、同様の環境
- Seedの値を42に固定。（random/numpy/torch）
- cuDNNで計算の順番やアルゴリズムを固定化し、再現性を確保。
- requirements.txtを用意し、環境構築をスムーズに行える。

環境整備

- GitHubを見やすいように整備し、誰もがすぐに学習や推論に取り掛かれるようにした。

学習実行手順

- デフォルト実行

```
python train.py
```

- W&Bにログを記録したい場合（任意）

```
# W&Bにログイン
wandb login

python train.py --use_wandb
```

- ハイパーパラメータの変更

```
#例
python train.py --epochs 12 --batch_size 64 --seed 42
```

Out of Memoryになってしまう場合は、batch_sizeを32などに下げてお試しください。

環境整備

- argparseで、--epochsなどコマンドを入れることで、ハイパーパラメータの調整やW&Bへの記録の有無などを選択可能にし、様々な用途に対してもすぐに試せるようにした。

- また、Out of memoryによる学習のストップが起きないよう
  - DataLoader に pin_memory=True を追加
  - .to(device) に non_blocking=True を追加

などの施策を行った。

以上です
ありがとうございました！