HTRやってみた

7-1ゼミ　工藤滉青

まずこのまえ読んだ論文についてみてきた

でも知識蒸留に関してはstudentはteacherに性能が劣りやってる理由が計算効率・運用・デプロイメントにおいて利点があるのみで精度向上におけるメリットが見受けられなかったためteacherモデルのみでの構築を行った。

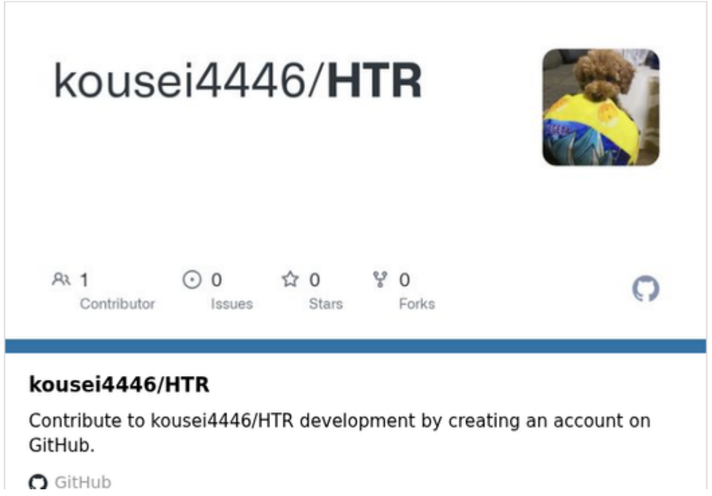
モデルは前の論文を参考にして構築した
[(Conv→ReLU→SE→MaxPooling) ×5] → Permute/Reshape → Linear → PosEnc →BiLSTM→Multi-Head Attention → Linear

損失関数：CTCロス $\mathcal{L}_{\text{ctc}} = -\log(p(\mathbf{y}|\boldsymbol{X})).$

🛠️ Project Structure

```
├ htr/                # library package
│  ├ models/          #   ├ teacher.py  / blocks.py
│  ├ datasets/        #   └ reader.py (Bentham loader)
│  └ utils/           #   └ dataloader.py
├ scripts/            # training entrypoints
│  ├ train_teacher.py
│  └ infer_demo.py    # inference demo
└ data/               # dataset root (git-ignored)
```

プロジェクト構成（README記載）



ソースコード

モデルのコード部分

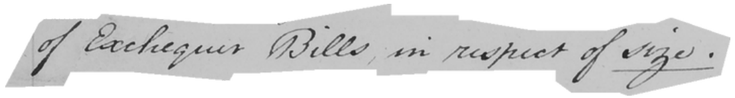
```python
class TeacherModel(nn.Module):
    """HTR-JAND *Teacher* モデル – 論文 Sec. III-B および Fig. 3 に忠実な実装。…

    def __init__(self, vocab_size: int, img_h: int = 64):
        super().__init__()

        # 同じ構造のCNNブロックを繰り返し作成するためのヘルパー関数
        def block(in_ch, out_ch, pool):
            layers = [
                FullGatedConv2d(in_ch, out_ch),
                nn.BatchNorm2d(out_ch),
                nn.ReLU(inplace=True),
                SEBlock(out_ch),
            ]
            if pool is not None:
                layers.append(nn.MaxPool2d(pool))
            return nn.Sequential(*layers)

        # ----- CNN encoder -------------------------------------------------
        self.encoder = nn.Sequential(
            block(  1,  32, (2, 2)),   # 64×W → 32×W/2
            block( 32,  64, (2, 2)),   # 32×W/2 → 16×W/4
            block( 64, 128, (2, 2)),   # 16×W/4 →  8×W/8
            block(128, 128, (2, 1)),   #  8×W/8 →  4×W/8  (height-only)
            block(128, 256, None),     #  4×W/8 →  4×W/8
        )

        feat_h = img_h // 16           # 64 → 4 after pooling schedule
        self.proj = nn.Linear(256 * feat_h, 256)

        # ----- Bi-LSTM -----------------------------------------------------
        self.lstm = nn.LSTM(
            input_size    = 256,
            hidden_size   = 128,
            num_layers    = 4,
            dropout       = 0.2,
            bidirectional = True,
            batch_first   = True,
        )

        # attention 構造
        self.pos  = PositionalEncoding(256)
        self.attn = CombinedAttention(256, heads=2)

        #分類器
        self.fc = nn.Linear(256, vocab_size)

    def forward(self, x):                # x: (B, 1, H, W)
        f = self.encoder(x)              # (B, 256, h=4, w)
        b, c, h, w = f.size()
        f = f.permute(0, 3, 1, 2).reshape(b, w, c * h)  # (B, T, 1024)
        f = self.proj(f)                                # (B, T, 256)
        f = self.pos(f)
        f, _ = self.lstm(f)                             # (B, T, 256)
        f = self.attn(f)                                # (B, T, 256)
        logits = self.fc(f)                             # (B, T, V)
        return logits.log_softmax(2)

```

データセット

Bentham データセットをとってきた
zenodoってところから

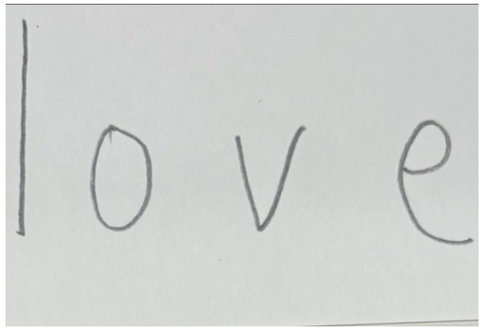
推論結果

データセット内のデータ

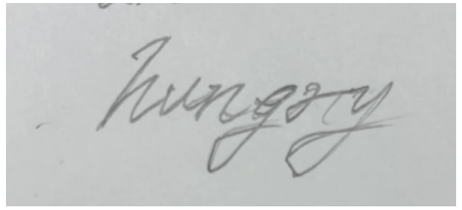


🖋 Predicted Text:
1

私の手書き英単語

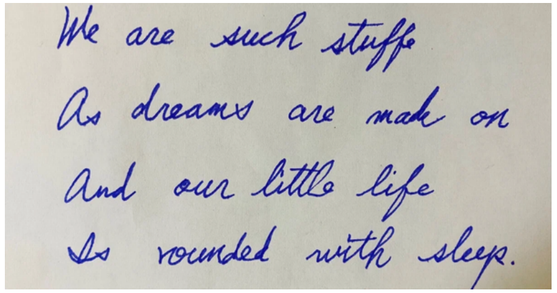


🖋 Predicted Text:
(



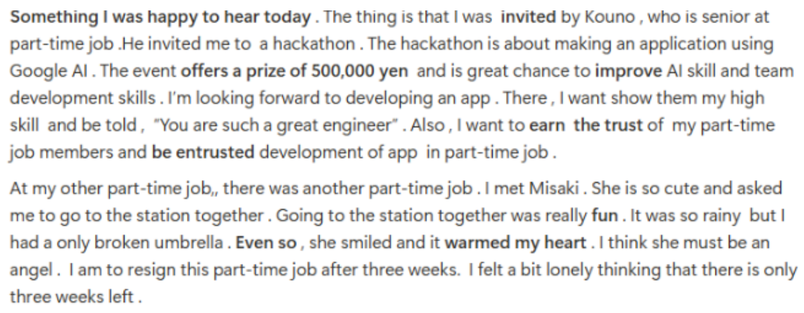
🖋 Predicted Text:
twrgy

手書き文章



🖋 Predicted Text:
§ gi .

PCによる英文・単語



🖋 Predicted Text:
4e

🖋 Predicted Text:
:or .

GPT作の英単語

🖋 Predicted Text:
turgy.

推論結果

考察
　hungryなど単語かつ筆記体の文字については一番精度としてよい
文章は全く認識できていない模様

データセットに偏りがあるまたはデータ読み込み処理がうまくできていないのでは？？

原因調査
ランダムに20件データ画像と
その画像に対応するラベルを表示
（特段変な画像やラベルと画像との
乖離はなさそう....）

今後の展望

文字ごとの画像に特化したデータセットを構築し、それを用いて
事前学習を行うことで、モデルの基礎的な文字認識能力の強化が
図れるのでは？？

てかそもそもそんなデータセットあるの？？（要検討）