AI 技術を用いた文字体系識別の可能性について*

木村　規高†

キーワード：機械学習、文字認識、EMNIST、Omniglot、文字学

1.　はじめに

現在 AI 技術の発展により、文字研究の分野でもこれらを応用した研究が見られるようになってきている。現状としてアラビア数字に関しては、画像データセットの MNIST を基に、TensorFlow や PyTorch の深層学習フレームワークを利用し、機械学習が行われてきている。

本発表では、複数の文字体系を対象とした横断的な研究の可能性を示すべく、特定の字がどの文字体系に属するかを数値化できるかといった点に着目して深層学習 (ディープラーニング) を行う。ここでは実験的に、一般的に同じ文字の系統とみなされ、字形も類似しているギリシア文字とラテン文字を研究対象とし、どのような結果が得られるかを検証する。

2.　機械学習の手順

本発表では Pyhton を用い、Google Colab 上でプログラミングを作成する。またライブラリに PyTorch[1]を使用する。手順としては、まずラテン文字識別の機械学習とギリシア文字の機械学習に分け、それぞれで畳み込みニューラルネットワークのアルゴリズムを用いて学習を行い、学習後のデータを基に新たに確立を求めるプログラムを実行する。ラテン文字のデータセットとしては EMNIST が知られているが、本発表では GitHub で公開されている Omniglot[2]を使用する。これにはラテン文字、ギリシア文字をはじめとする約 50 の表音文字体系が含まれており、ラテン文字とギリシア文字は小文字が収められている。

2.1　ラテン文字の学習

Omniglot は 1 クラス当たりのデータ数が 20 個であり、これは機械学習、特に深層学習のモデルを使用する場合には十分な数ではない。そこで本発表では画像の回転や拡大によって画像データを水増しして深層学習に用いることとする。以下にコードを示す。それぞれのコードの簡単な処理目的は “#” にて示し、項数の関係で一部過程は省略する。

```
# データ拡張の定義
transform = transforms.Compose([
    transforms.RandomHorizontalFlip(), # ランダムに画像を水平回転
    transforms.RandomRotation(10), # ランダムに画像を最大 10 度回転
    transforms.RandomResizedCrop(28, scale=(0.8, 1.0)), # ランダムに画像をクロップしてリサイズ
```

*本発表は JSPS 科研費 24KJ0477 の助成を受けた。

†筑波大学大学院人文社会科学研究群人文学学位プログラム言語学サブプログラム博士後期課程

[1] PyTorch は、Facebook の人工知能グループにより開発されたライブラリである。

[2] 引用情報は次の通りである。作成者：brendenlake、タイトル：omniglot、公開年：2018 年、URL：https://github.com/brendenlake/omniglot

```
    transforms.ToTensor()
])

# 元データセットのパス
dataset_path = '/content/drive/My Drive/images_background/Latin'

# データ拡張をして保存
[省略]
```

続いて、データセットの画像に対して、グレースケールへの変換、リサイズ、テンソル形式の変換、テンソル形式の画像の正規化といった前処理を行う。またデータセットの画像のうち、70%をトレーニングデータに、残りの 30%をテストデータに充てる。

```
# データ変換の定義
transform = transforms.Compose([
    transforms.Grayscale(num_output_channels=1),  # グレースケールに変換
    transforms.Resize((28, 28)),  # サイズを 28x28 にリサイズ
    transforms.ToTensor(), # PIL Image を Tensor に変換
    transforms.Normalize((0.5,), (0.5,))
])

# データセットの読み込み
dataset_path = '/content/drive/My Drive/images_background/Latin'
dataset = datasets.ImageFolder(root=dataset_path, transform=transform)

train_size = int(0.7 * len(dataset))  # 70%をトレーニングデータにする
test_size = len(dataset) - train_size  # 残りをテストデータにする
train_dataset, test_dataset = random_split(dataset, [train_size,
test_size])

# データローダーの定義
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
```

続いてニューラルネットワークのモデルの定義と、損失関数と最適化手法の定義を行う。最適化関数 (optimizer、ここでは Adam を使用) は、モデルの重みとバイアスを更新して、損失関数の値を最小値にするための手法であり、損失関数 (criterion、ここでは nn.CrossEntropyLoss を使用) はモデルの予測と実施のラベルの誤差を計算する。またモデルの定義では、少ない画像数に対する過学習を防止するために、層の数やニューロン数を少なく設定している。

```
# モデルの定義
class LatinNet(nn.Module):
    def __init__(self):
        super(LatinNet, self).__init__()
```

```
        # 畳み込み層
        self.conv1 = nn.Conv2d(1, 16, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(16, 32, kernel_size=3, padding=1)
        # プーリング層
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)
        # 全結合層
        self.fc1 = nn.Linear(32 * 7* 7, 64)
        self.fc2 = nn.Linear(64, 26)
        # 正則化（過学習の防止）
        self.dropout = nn.Dropout(p=0.5)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 32 * 7 * 7)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x

# モデルのインスタンス化
model = LatinNet()

# 最適化関数と損失関数の定義
optimizer = optim.Adam(model.parameters(), lr=0.001)
criterion = nn.CrossEntropyLoss()
```

最後に学習を実行する。今回は 10 回のエポックを設定し、その学習を 3 度繰り返し行った。実行したコードと最後の 10 回分の経過のアウトプットを以下に示す。最終的には学習誤差 (損失: Loss) を 0.0012 まで減らすことができた。

```
# 学習
[省略]
# 結果の表示
    print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {run-
ning_loss/len(train_loader):.4f}')
```

[Output]

Epoch [1/10], Loss: 0.0443

Epoch [2/10], Loss: 0.0466

Epoch [3/10], Loss: 0.0645

Epoch [4/10], Loss: 0.0274

Epoch [5/10], Loss: 0.0154

Epoch [6/10], Loss: 0.0032

Epoch [7/10], Loss: 0.0020

Epoch [8/10], Loss: 0.0017
Epoch [9/10], Loss: 0.0014
Epoch [10/10], Loss: 0.0012

2.2　ギリシア文字の機械学習

ギリシア文字の機械学習も 2.1 節のラテン文字と同様に実行した。その結果、最終的な学習誤差は以下のようになり、学習誤差をこちらも 0.0025 まで減らすことができた。

[Output]
Epoch [1/10], Loss: 0.0082
Epoch [2/10], Loss: 0.0093
Epoch [3/10], Loss: 0.0075
Epoch [4/10], Loss: 0.0057
Epoch [5/10], Loss: 0.0057
Epoch [6/10], Loss: 0.0052
Epoch [7/10], Loss: 0.0037
Epoch [8/10], Loss: 0.0034
Epoch [9/10], Loss: 0.0035
Epoch [10/10], Loss: 0.0025

2.3　文字識別の機械学習

本発表の目的は、特定の字がどちらの文字体系に属すかを検討するものであるため、最終的には、ギリシア文字に属す確率、及びラテン文字に属す確率をそれぞれ明確にしたい。

既に学習済みのモデルを使用するため、2.1 節と 2.2 節のモデルの重みを保存し、新たに Google Colab のノートブックで読み込むこととする。

初めに、ロードするモデルと同じモデルクラスを再び定義する。続いてモデルのインスタンスを作成し、保存済みのモデルを読み込む。また、ドロップアウトを評価モードで動作させるために評価モードに設定する。

続いて、画像パスを入力として受け取り、ギリシア文字とラテン文字の予測確率を計算する関数を定義する。ソフトマックス関数を用いて、出力をクラス確率に変換している。またその次に、各モデルの予測確率の中で、最大の確率とそのクラスインデックスを取得している。

```
# モデルの定義
[省略]
保存した重みの読み込み
[省略]
# 予測関数の定義
def predict_character(image_path):
    # 画像の読み込みと前処理
    img = Image.open(image_path)
    img = transform(img)
    img = img.unsqueeze(0)  # バッチ次元の追加

    # 予測
```

```
    with torch.no_grad(): # 勾配計算の無効化
        latin_output = latin_model(img)
        greek_output = greek_model(img)

    # 出力をソフトマックスで確率に変換
    greek_probs = torch.softmax(greek_output, dim=1)[0]
    latin_probs = torch.softmax(latin_output, dim=1)[0]

    # 最大確率を持つクラス(文字)を取得
    greek_max_prob, greek_pred = torch.max(greek_probs, 0)
    latin_max_prob, latin_pred = torch.max(latin_probs, 0)

    return greek_max_prob.item(), latin_max_prob.item(), img.squeeze(0)
```

3. 機械学習の結果の検証

本節では、2.3 節で学習したモデルを用い、個別の字がどちらも文字体系に属すのかについて、画像のパスを指定して予測を実行することにより検証を行った結果を述べる。特に①ラテン文字とギリシア文字で類似している字、②ラテン文字にのみにみられる字、③ギリシア文字にのみみられる字に分けて、それぞれ 2 字ずつ検証した結果を述べ、表 1 に示す。検証する画像は Omniglot に収められている 20 個のデータを使用する。

表 1 機械学習の結果

No/字	o, *o*		k, *κ*		f		g		*β*		*ζ*	
	ラテン	ギリシア	ラテン	ギリシア	ラテン	ギリシア	ラテン	ギリシア	ラテン	ギリシア	ラテン	ギリシア
1	51.13%	98.67%	71.32%	99.96%	99.70%	89.11%	99.99%	75.18%	82.64%	98.64%	56.92%	100%
2	99.68%	99.90%	98.19%	99.73%	99.98%	99.69%	99.86%	97.93%	94.33%	99.87%	99.58%	99.16%
3	99.63%	99.31%	96.48%	99.41%	100%	43.10%	99.97%	61.81%	53.02%	98.11%	89.36%	100%
4	84.45%	99.94%	94.44%	99.36%	99.78%	87.99%	99.89%	49.44%	100%	99.95%	80.03%	99.98%
5	99.97%	100%	99.84%	99.17%	100%	99.97%	99.48%	77.87%	99.06%	100%	51.85%	99.96%
6	99.97%	100%	99.77%	84.00%	99.94%	67.82%	99.97%	64.09%	75.72%	100%	99.59%	99.99%
7	56.80%	98.64%	99.96%	99.66%	100%	99.43%	100%	97.44%	100%	99.59%	63.19%	99.83%
8	99.32%	99.95%	66.84%	100%	100%	99.84%	100%	100%	88.72%	99.79%	77.03%	100%
9	100%	100%	74.73%	99.98%	99.99%	98.69%	99.99%	58.24%	99.75%	99.75%	99.79%	99.56%
10	100%	99.67%	99.99%	99.76%	99.80%	99.91%	99.98%	91.44%	98.60%	99.96%	95.07%	100%
11	99.07%	99.99%	93.16%	99.99%	100%	99.75%	100%	99.69%	63.64%	100%	99.98%	99.75%
12	96.26%	100%	91.64%	99.79%	92.64%	80.70%	99.93%	64.73%	99.66%	99.98%	99.96%	100%
13	99.57%	99.98%	54.95%	99.91%	87.74%	60.43%	99.53%	95.46%	75.79%	99.93%	95.38%	100%
14	98.48%	100%	87.62%	99.90%	99.97%	94.21%	100%	75.86%	87.25%	97.75%	53.54%	99.98%
15	99.27%	99.99%	31.15%	50.89%	100%	83.39%	99.99%	50.84%	56.35%	99.81%	83.77%	99.77%
16	99.95%	99.62%	72.55%	99.95%	99.93%	37.63%	100%	99.99%	91.66%	99.89%	84.42%	100%
17	99.97%	98.63%	77.95%	99.86%	100%	95.63%	99.92%	70.43%	96.26%	100%	76.28%	100%
18	98.67%	99.99%	85.30%	99.72%	100%	79.87%	100%	92.76%	95.29%	100%	97.85%	100%
19	99.93%	99.13%	60.99%	99.71%	100%	99.89%	100%	99.07%	63.53%	100%	99.99%	100%
20	84.21%	99.97%	50.62%	99.91%	100%	94.56%	99.99%	98.52%	95.78%	99.40%	90.60%	100%

まず、ラテン文字とギリシア文字で字形の変化が起きていないと考えられる “o” と “ο” について検証を行った結果、ほとんどの画像で 100%近い数値を記録した。一方で “k” と “κ” については、一部の画像がギリシア文字に属す確率が高くなった。これは “κ” の曲線が強調している画像に対してみられる傾向であったため、正確な結果が出ているとも言える。

ラテン文字にのみ見られる字については、“f” と “g” を検証した結果、そのほとんどでラテン文字に属す確率が高く出た。しかし、ギリシア文字でも画像によっては高い確率が検出された。

ギリシア文字にのみ見られる “β” と “ζ” を検証した結果、ギリシア文字はほぼ 100%に近い数値が出た。ラテン文字は 60%近い確率のものをあった一方で、半分は 100%に近い数値が出てしまったため、半分が正確に判定できなかった。

4. 本発表の応用可能性と課題について

とある字がいずれの文字体系に属すかといった研究は、特に未解読文字や、かすれていたり汚れている文字に対して、その字や文字体系を特定することに役立つことが考えられる。そのほかにも、文字体系自体の類似度や系統関係の研究にも応用可能である。例えば、ギリシア文字とラテン文字の確立がそれぞれ 100%に近い確率である字が存在するとすれば、その字はいずれの文字体系にも属す字であると言える。それだけでなく、2 つの字が類似していることを示すデータともなり、少なからず系統関係があるということを明らかにできるかもしれない。本発表では既に系統関係の明らかになっている文字体系同士を対象としたが、これを漢字の祖先文字とも言われる西安の遺跡から出土した文字であったり、独自の文字と言われる彝文字、ヴァイ文字などの研究に応用することで、系統関係に関する研究に寄与することができると考える。

ただ、現時点ではその精度に大きな問題が見られ、改善の余地があると言える。また、字形以外の要素については触れることができていないため、形音義 (字形・音価・意味) すべてを考慮した枠組みを構築する必要がある。

【参考文献】

Baldominos A, Saez Y, Isasi P.(2019) A Survey of Handwritten Character Recognition with MNIST and EMNIST. *Applied Sciences*; 9(15), 3169: 1-16.

Lake B. M, Salakhutdinov R, Tenenbaum J. B. (2019) The Omniglot challenge: a 3-year progress report. *Current Opinion in Behavioral Sciences*; 29: 97-104.

川島賢 (2019)「今すぐ試したい！機械学習・深層学習 (ディープラーニング) 画像認識プログラミングレシピ」秀和システム.