進捗報告

表 1: 実験の設定

dataset	cifar10
n data	16,000 / model
task	5, 7, 10 クラス識別
input	image(3x32x32)
output	class(5, 7, 10)
model	CNN(16 層)
optim	SDG (lr=0.001, moment=0.9)
loss	Cross Entropy Loss
batch size	64
epoch	100

1　今週やったこと

- 結合モデルの実験

2　モデルの構築

前回, 5 クラス識別器を 2 つ利用したモデルの構築をしたが, 10 クラスの識別が 50%であったため, より適したパラメータを探索し, 再び実験して精度の向上を目指した.

2.1　5 クラス x2

cifar10 に含まれる 10 クラスを 5 クラスずつに分割した。インデックスの前半 (airplane, mobile, bird, cat, deer) と後半 (dog, frog, horse, ship, truck) で分けることにした。生成した部分データセットを、それぞれモデル A, B で学習した。5 クラス分類ができるモデル A, B を持つ結合モデル A + B を作成し、10 クラス分類の精度を計測した。このモデルの学習と相互関係を図 1 に示した。

結合モデル A + B は 10 クラスのデータセットを A, B に入力し、得られた出力をクラスインデックス順に結合して、出力とする。結合では特別な処理を行わず、そのままのデータを連結した.

表 2: モデルごとの 7 クラスの振り分け

model	0	1	2	3	4	5	6	7	8	9
A	o	o	o	o	o	o	o			
B	o	o	o	o				o	o	o
C	o				o	o	o	o	o	o

2.2　7 クラス x3

さらに今回は, 細川君に指摘してもらったアイデアでも実験を行った. 7 クラス分類器を 3 つ組み合わせて,10 クラスの分類を行った. 表 2 のように, 2 つ以上の分類器で各クラスを推定するように, クラスを振り分けた.

2.3　結果

5 クラス分類の精度を図 2 に, 7 クラス分類の精度を図 3 に示した.

5 クラス分類の場合, 80% 〜 90%の正答率で前回よりも 10%程向上した. 前回のデータ数 2000 よりもデータ数を増やしたことで精度がよくなった. 結合した結果も, 70%(前回+20%) となった.

7 クラス分類では, 正答率が平均 80%程度で, 結合した結果 10 クラス分類では 86.7%となった.

3　考察

データ数とバッチサイズを増やして, 精度の向上と学習の安定化ができた. 特に 5 クラス分類ではあるが, 正答率 9 割を超えることができた. データオーギュメントはしていないので, さらに精度を上げることはできると思われる.

図 2, 3 ともにインデックスが前半のクラスを持つモデルでは, 正答率が低い傾向が見られた. 誤差の影響ではなく, 困難なクラスの分類によって精度が下がっていると考えられる. これはクラスを単純に分割したことによる偏りに原因がある.

分割する組み合わせを自由に替える実装ができたので, 様々な組み合わせパターンで実験して, その差異を

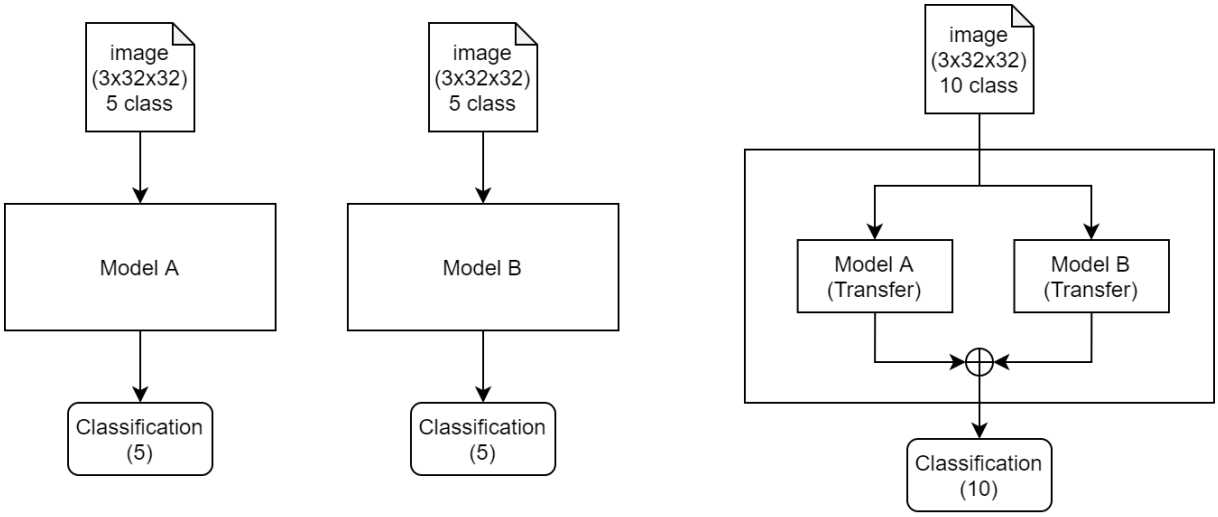

図 1: モデルの簡略図　() 内はデータの次元数

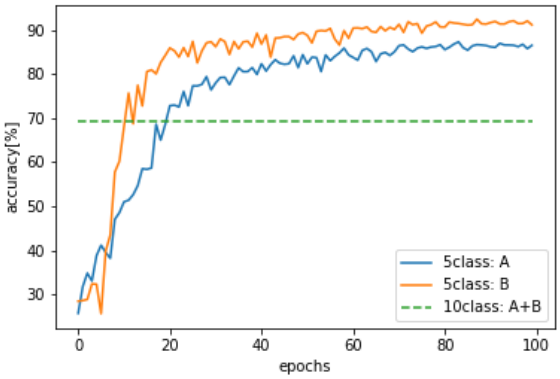

図 2: 5 クラス分類の正答率

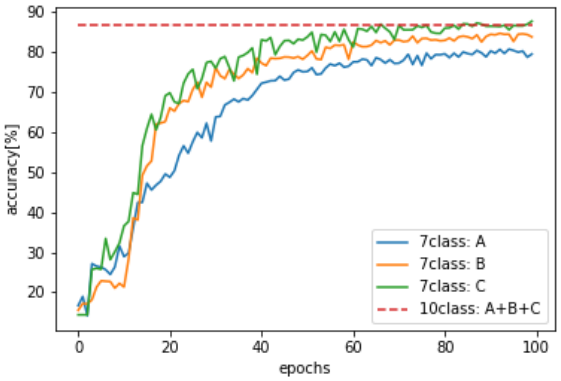

図 3: 7 クラス分類の正答率

見たい. 特にモデル間の正答率の差が目立ったため, これを埋めるような組み合わせを探したい.

4　今後の予定

- クラスの組み合わせの改良

5　ソースコード

結合モデルの Pytorch での実装をコード 1 に示す.

Listing 1: concat

```
class catCNN(nn.Module):
  def __init__(self, models, device,
      n_class=10):
    super(catCNN, self).__init__()
    self.models = models
    self.n_class = n_class
    self.device = device

def forward(self, x):
  batch_size = x.shape[0]
  output = torch.zeros(batch_size, self.
      n_class).to(self.device)
  count = torch.zeros(self.n_class).to(
      self.device)
  for idx, model in self.models:
    idx = torch.tensor(idx).to(self.
        device)
    output.index_add_(1, idx, model(x))
    count[idx] += 1

```

```
17      output = output / count
18      return output
```