【適用事例】 畳み込みニューラルネットワーク（CNN）を用いた数字画像認識

《学修項目》

- 画像データと対応する教師データの読み込み
- CNNの構成と学習法の設定
- CNNの学習と評価

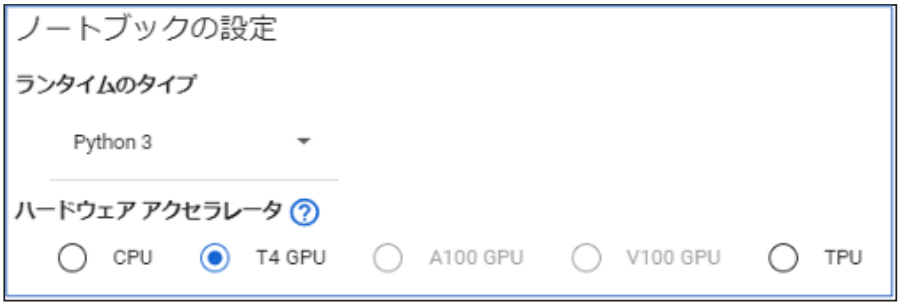
《キーワード》

> TensorFlow、Keras、数字画像認識、MNIST、畳み込み層、Conv2D、活性化関数、プーリング層、テンソル要素のベクトル化、出力層、ソフトマックス関数、クロスエントロピー誤差、識別精度(accuracy)、エポック(epoch)

準備：Google Colaboratory環境におけるGPUの利用

CNNでは、学習データが多くなると、学習に膨大な時間がかかってしまう場合がある。そこで、Google Colaboratoryでは、必要に応じて、GPUを使用して、処理の高速化をすることが可能である。

設定は、次のとおりである。 メニューの「編集」から「ノートブックの設定」を選択、「ハードウェアアクセラレータ」から「T4 GPU」などを選択すればよい。



Google Colaboratory環境におけるGPUの設定パネル

1. 各種モジュールの読み込みと数字画像データの読み込み

前述の適用例では、単純な3層型ニューラルネットワークを用いて数字画像認識を行ってみたが、ここでは、同様の画像データについて、畳み込みニューラルネットワークを用いた学習と認識を行ってみよう（本プログラムは、TensorFlowのチュートリアルページ の内容を参考としている）。

> https://www.tensorflow.org/tutorials/images/intro_to_cnns?hl=ja

In [1]:
```
import tensorflow as tf  #TensorFlowモジュールのインポート
from keras import datasets, layers, models #Kerasモジュール内の関数名をインポート
(train_images, train_labels), (test_images, test_labels) = datasets.mnist.load_data(
train_images = train_images.reshape((60000, 28, 28, 1)) #学習データの形状を設定(60000枚の
test_images = test_images.reshape((10000, 28, 28, 1)) #テストデータの形状設定(10000枚の画像
train_images, test_images = train_images / 255.0, test_images / 255.0 #各画素値を0-1の
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mni
st.npz
11490434/11490434 [==============================] - 2s 0us/step
```

1行目では、TensorFlowモジュールをインポートし、2行目では、Kerasモジュール内の各種関数を直接呼び出せるように設定をしている。3行目は、使用する数字画像データベース(MNIST)のデータを読み込んでいる。ここで、train_imagesは学習用のデータ、train_lablesはその教師データ（対応する画像のクラス、0から9の整数で表現）、test_imagesは評価用のデータ、test_labelsはその教師データをそれぞれ示している。4行目は行列形状を設定しており、train_imagesは画像枚数が60,000枚、各画像は28×28画素、各画素はグレースケールであることを意味している。5行目は、test_imagesの形状であり、画像枚数が10,000枚、各画像は28×28画素、各画素はグレースケールである。そして、最後の6行目では、各画像の各画素が0から255のグレースケールとなっているので、この値を0から1に正規化している。

2. CNNの構成

図41に、数字画像認識に使用する畳み込みニューラルネットワークの構造を示し、その下に、この構造を実現するためのプログラムを示す。以下、プログラム行と図を対照しながら、各部の説明をする。

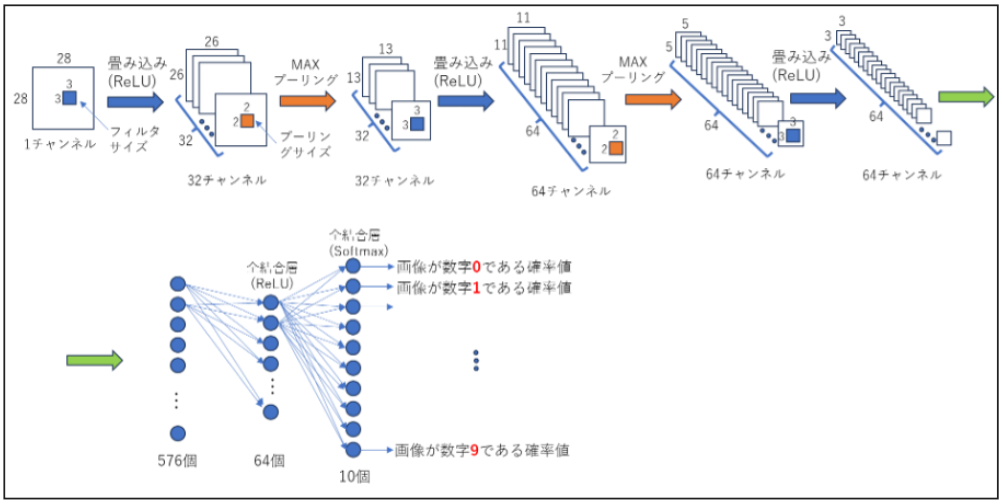


図 41　CNNの構造

プログラムのmodel.addメソッドは、ネットワークに各層を追加している。

In [2]:
```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1))) #畳
model.add(layers.MaxPooling2D((2, 2)))                    #MAXプーリング:サイズ2x2
model.add(layers.Conv2D(64, (3, 3), activation='relu'))   #畳み込み層:フィルタ数が64、フィル
model.add(layers.MaxPooling2D((2, 2)))                    #MAXプーリング:サイズ2x2
model.add(layers.Conv2D(64, (3, 3), activation='relu'))   #畳み込み層:フィルタ数が64、フィル
model.add(layers.Flatten())                               #前段の出力(テンソル)の要素値を1
model.add(layers.Dense(64, activation='relu'))            #全結合層:ユニット数が64、活性化関
model.add(layers.Dense(10, activation='softmax'))         #全結合層(出力層):ユニット数が10、
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```
1  model = models.Sequential()
2  model.add(layers.Conv2D(32, (3, 3), activation='relu',
input_shape=(28, 28, 1)))
```

まず、２行目では、layers.Conv2Dを使って、畳み込み層を追加している。32は、フィルタ数を示し、(3, 3)はフィルタのサイズ、activationは活性化関数の指定（ここでは、ReLUを指定）、input_shapeは入力画像のサイズが28×28画素でグレースケール画像であることを意味している。図41で考える

と、入力画像について、3×3のフィルタを適用して畳み込みを行い、結果として、26×26の大きさの32チャンネルの画像(行列)を生成していることになる。

```
3  model.add(layers.MaxPooling2D((2, 2)))
```

3行目では、layers.MaxPooling2Dを使って、MAXプーリング層を追加している。引数の(2, 2)はプーリングサイズを意味している。その結果、26×26画素の画像は半分のサイズの13×13画素の画像となる。チャンネル数は32のままである。

```
4  model.add(layers.Conv2D(64, (3, 3), activation='relu'))
5  model.add(layers.MaxPooling2D((2, 2)))
6  model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

以下同様に、4行目で畳み込み層、5行目でMAXプーリング、6行目で畳み込み層をそれぞれ追加している。図41に示すように、結果として、3×3画素の画像が64枚（64チャンネル）生成されることになる。

```
7  model.add(layers.Flatten())
```

7行目のlayers.Flattenでは、上記で生成したテンソルの要素値を1列に並べてベクトル化する操作を行う。具体的には、3×3の画像が64枚あるので、その画像の各画素を1列に並べることになるので、3×3×64=576の数値列（ベクトル）となる。これが、全結合層の入力ベクトルとなる。

```
8  model.add(layers.Dense(64, activation='relu'))
```

8行目のlayers.Denseは、全結合層の追加を意味する。引数の64は、ユニットの個数、activationは活性化関数の種類を示す。よって、図41の下段に示すように、入力層576個のユニットから全結合された64個のユニットで構成され、このユニットからの出力には活性化関数としてReLUが適用される。

```
9  model.add(layers.Dense(10, activation='softmax'))
```

9行目も全結合層の追加であり、これが出力層となる。ここでは、数字クラス（数字の0から9）を表現するために10個のユニットを用意し、活性化関数はソフトマックス関数を指定している。これにより、図41に示すように各ユニットからは、各数字クラスである確率値が出力されることになる（例えば、一番上の出力ユニットからは、入力画像が数字の0である確率値が出力される）。

```
10  model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

最後に10行目のmodel.compileで、最適化アルゴリズムをAdam、誤差関数としてクラス出力に関するクロスエントロピー誤差を採用、学習途中の評価のための尺度として識別精度(accuracy)を指定している。

3. CNNの学習と評価

上記の手順で作成したCNNを用いて、学習データについて学習を行うためには、次のようにプログラムを記述する。

```
In [9]: model.fit(train_images, train_labels, epochs=5) #学習データの学習、学習回数は5回
```

```
Epoch 1/5
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0088 - accuracy:
0.9972
Epoch 2/5
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0072 - accuracy:
0.9979
Epoch 3/5
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0070 - accuracy:
0.9979
Epoch 4/5
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0069 - accuracy:
0.9978
Epoch 5/5
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0054 - accuracy:
0.9981
```

```
Out[9]: <keras.src.callbacks.History at 0x7ae2ecb12b00>
```

ここで、epochsを5としているので、数字画像全体(60,000枚)を一通り学習する計算を5回繰り返していることになる。学習結果は次のとおりである。なお、学習時のフィルタの値、バイアス値、重みの値の初期値は乱数によって設定されるため、学習ごとに微妙に結果の値が異なる点に留意されたい。

これより、学習データについては、99.81%の認識率が得られていることがわかる。次にテストデータについての認識率を評価してみよう。

```
In [10]: test_loss, test_acc = model.evaluate(test_images, test_labels) #テストデータの評価:誤差と
```

```
313/313 [==============================] - 1s 3ms/step - loss: 0.0396 - accuracy: 0.9
928
```

結果から、テストデータの認識率は99.28%となっていることがわかる。

4. MLPとCNNの性能比較

以上の結果を前述の「3層型ニューラルネットワークを用いた数字認識」の結果と比較してみよう。

前回の結果は表9より、各種パラメータを調整しても、テストデータに対する認識率は97.97%にとどまっていた。それに比べると、CNNを用いた手法では、99%以上の認識率を達成している。

chap. A3, 表9 MLPの認識率（再掲）

中間層1:ユニット数	活性化	中間層2:ユニット数	活性化	最適化	dropout	epoch	認識率:学習	認識率:テスト
64	sigmoid	----	----	SDG	----	5	89.08%	89.81%
64	ReLU	----	----	SDG	----	5	93.25%	93.55%
128	ReLU	----	----	SDG	----	5	93.58%	93.79%
128	ReLU	----	----	Adam	----	5	98.03%	97.44%
128	ReLU	64	ReLU	Adam	----	5	98.68%	97.90%
128	ReLU	----	----	Adam	----	10	99.51%	97.97%
128	ReLU	----	----	Adam	0.2	10	97.64%	97.87%
128	ReLU	----	----	Adam	0.4	10	96.54%	97.46%

このように、CNNでは、畳み込み層とプーリング層を組み合わせて適用することにより、画像から適切な特徴量を抽出することができ、それを全結合層のニューラルネットワークに入力することで、全

結合層のみで構成される単純なニューラルネットワークに比べて、画像認識の精度を上げることができることがわかる。

memo