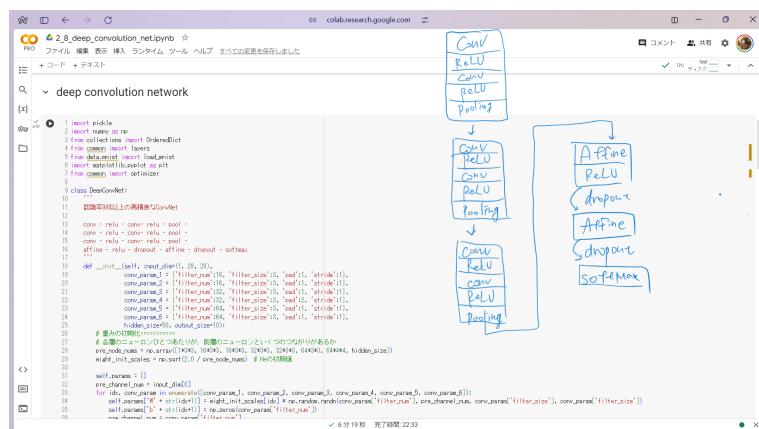


深層学習 day2 実装演習キャプチャ 3

1 2_8_deep_convolution_net

This screenshot shows the initial setup phase of a Python notebook titled '2_8_deep_convolution_net.ipynb'. The code block contains imports for numpy, tensorflow, and various utility functions. It also includes logic to check if the notebook is running on Google Colab and sets up the file system by mounting the Google Drive. A note at the bottom indicates that the code is based on DNN code from the 'DNN_code/DNN_code.colab.day2' folder.

```
# Notebook Colab での実行が可能です
# ドライブのマウント
# ノートブックファイルと同じフォルダへの移動
# Googleドライブのマイドライブを基準にDNN_code/DNN_code.colab.day2フォルダを置くことを修正しています。必要に応じて、パスを変更してください。
# !ls
# !ls -l /content/drive/MyDrive/Colab_Notebooks/DNN_code/DNN_code.colab.day2
# !ls -l /content/drive/MyDrive/Colab_Notebooks/DNN_code/DNN_code.colab.day2/notebook
# !ls -l /content/drive/MyDrive/Colab_Notebooks/DNN_code/DNN_code.colab.day2/notebook/notebooks
```



This screenshot shows the continuation of the '2_8_deep_convolution_net.ipynb' notebook. The code defines the forward pass of the network, including the prediction function which takes input data and applies it to the layers. It also defines a gradient function for training, which performs backpropagation through the network. The code uses various helper functions like 'dot' and 'softmax'.

```
def predict(self, x, train=False):
    for i in range(len(self.layers) - 1):
        if train:
            x = layer.forward(x, train)
        else:
            x = layer.forward(x)
    return x

def forward(self, x, d):
    v = self.predict(x, train=True)
    return self.last_layer.forward(v, d)

def accuracy(self, x, batch_size=100):
    if len(x) > 1:
        d = np.argmax(x, axis=1)
    else:
        d = np.array([np.argmax(x, axis=1)])
    v = self.predict(x, train=False)
    v = np.argmax(v, axis=1)
    acc = np.sum(d == v) / len(d)
    return acc
```

```
2.8_deep_convolution.net.ipynb
```

```
71     self.last_layer = layers.SoftmaxWithLoss()
72
73     def predict(self, x, train_flg=False):
74         for layer in self.layers:
75             x = layer.forward(x)
76             if layer == self.last_layer:
77                 x *= layer.forward(train_flg)
78
79         return x
80
81     def loss(self, x, d):
82         return self.last_layer.loss(x, d)
83
84     def backward(self, x, d):
85         for layer in reversed(self.layers):
86             d = layer.backward(x, d)
87
88         if self.last_layer == layer:
89             d *= self.last_layer.backward(d)
90
91         return d
92
93     def gradient(self, x, d):
94         for layer in reversed(self.layers):
95             d = layer.gradient(x, d)
96
97         if self.last_layer == layer:
98             d *= self.last_layer.gradient(x)
99
100        return d
101
102    def accuracy(self, x, d, batch_size=100):
103        if d.ndim != 1 or d.shape[0] != x.shape[0]:
104            raise ValueError("dim mismatch")
105
106        acc = 0.0
107
108        for i in range(0, x.shape[0], batch_size):
109            tx = x[i:i+batch_size]
110            td = d[i:i+batch_size]
111            ty = self.predict(tx, train_flg=False)
112            y = np.argmax(ty, axis=1)
113            acc += np.sum(y == td)
114
115        acc /= x.shape[0]
116
117        return acc
118
119    def __init__(self, layers, optimizer):
120        self.layers = layers
121        self.optimizer = optimizer
122        self.loss_fn = layers[-1].loss
123
124        self.train_loss, self.test_loss = 0, 0
125        self.train_accuracy, self.test_accuracy = 0, 0
126
127        self.train_loss_list = []
128        self.test_loss_list = []
129        self.train_accuracy_list = []
130        self.test_accuracy_list = []
131
132        self.batch_size = 100
133
134        self.x_train, self.d_train, self.x_test, self.d_test = load_mnist(fashion=False)
```

```
2.8_deep_convolution.net.ipynb
```

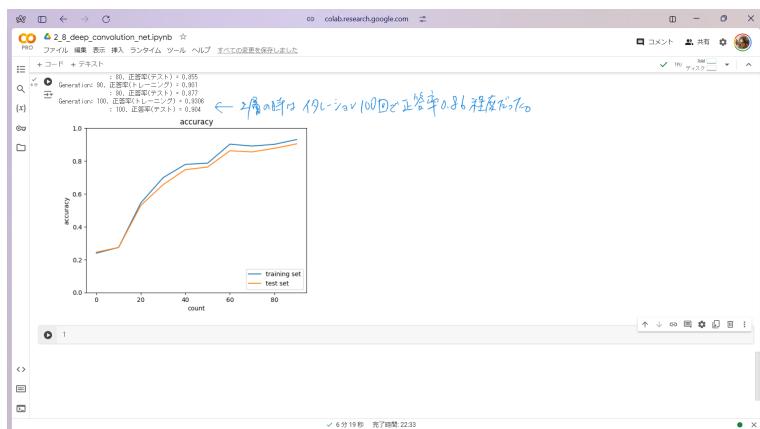
```
89
90     def __init__(self, layers, optimizer):
91         self.layers = layers
92         self.optimizer = optimizer
93
94         self.loss_fn = layers[-1].loss
95
96         self.train_loss, self.test_loss = 0, 0
97         self.train_accuracy, self.test_accuracy = 0, 0
98
99         self.train_loss_list = []
100        self.test_loss_list = []
101        self.train_accuracy_list = []
102        self.test_accuracy_list = []
103
104        self.batch_size = 100
105
106        self.x_train, self.d_train, self.x_test, self.d_test = load_mnist(fashion=True)
107
108        print("データ読み込み完了")
109
110        network = DeepConvNet()
111        optimizer = optimizer.Adam()
112
113        self.train_size = 100
114        train_size = x_train.shape[0]
115        batch_size = 100
116
117        train_loss, test_loss = [], []
118        accuracies_train, accuracies_test = [], []
119
120        for i in range(0, train_size, batch_size):
121            tx = x_train[i:i+batch_size]
122            td = d_train[i:i+batch_size]
123            ty = network.predict(tx, train_flg=True)
124            loss = network.loss(tx, td)
125            network.backward(tx, td)
126            network.gradient(tx, td)
127            network.update(optimizer, tx, td)
128
129            train_loss.append(loss)
130            accuracies_train.append(network.accuracy(tx, td))
131
132            if i % batch_size == 0:
133                test_loss.append(network.loss(x_test, d_test))
134                accuracies_test.append(network.accuracy(x_test, d_test))
135
136        self.train_loss_list.append(train_loss)
137        self.test_loss_list.append(test_loss)
138        self.train_accuracy_list.append(accuracies_train)
139        self.test_accuracy_list.append(accuracies_test)
140
141        print("訓練終了")
142
143        print("test loss: " + str(test_loss[-1]))
144        print("train loss: " + str(train_loss[-1]))
145        print("test accuracy: " + str(accuracies_test[-1]))
146        print("train accuracy: " + str(accuracies_train[-1]))
```

```
2.8_deep_convolution.net.ipynb
```

```
1    x_train, d_train, x_test, d_test = load_mnist(fashion=False)
2
3    # データの特徴の数を取得する
4    x_train, d_train = x_train[:1000], d_train[:1000]
5    x_test, d_test = x_test[:1000], d_test[:1000]
6
7    print("データ読み込み完了")
8
9    network = DeepConvNet()
10   optimizer = optimizer.Adam()
11
12   self.train_size = 100
13   train_size = x_train.shape[0]
14   batch_size = 100
15
16   train_loss, test_loss = [], []
17   accuracies_train, accuracies_test = [], []
18
19   for i in range(0, train_size, batch_size):
20       tx = x_train[i:i+batch_size]
21       td = d_train[i:i+batch_size]
22       ty = network.predict(tx, train_flg=True)
23       loss = network.loss(tx, td)
24       network.backward(tx, td)
25       network.gradient(tx, td)
26
27       if i % batch_size == 0:
28           test_loss.append(network.loss(x_test, d_test))
29           accuracies_test.append(network.accuracy(x_test, d_test))
30
31       train_loss.append(loss)
32       accuracies_train.append(network.accuracy(tx, td))
33
34   self.train_loss_list.append(train_loss)
35   self.test_loss_list.append(test_loss)
36   self.train_accuracy_list.append(accuracies_train)
37   self.test_accuracy_list.append(accuracies_test)
38
39   print("訓練終了")
40
41   print("test loss: " + str(test_loss[-1]))
42   print("train loss: " + str(train_loss[-1]))
43   print("test accuracy: " + str(accuracies_test[-1]))
44   print("train accuracy: " + str(accuracies_train[-1]))
```

```

22 for i in range(100):
23     batch_size = np.random.choice(train_size, batch_size)
24     x_batch = x_train[batch_index]
25     d_batch = d_train[batch_index]
26     grad = network.backprop(x_batch, d_batch)
27     optimizer.update(network, grad)
28     loss = network.loss(x_batch, d_batch)
29     train_loss += loss / batch_size
30
31     if (i+1) % print_interval == 0:
32         wcrf.train = network.accuracy(x_train, d_train)
33         acc_train, acc_test = wcrf.test(x_train, d_train)
34         acc_train, acc_test = wcrf.accuracy(x_train, d_train)
35         acc_train, acc_test = wcrf.accuracy(x_test, d_test)
36         acc_train, acc_test = wcrf.accuracy(x_test, d_test)
37         print('Generation: ' + str(i+1) + ', 正確率(トレーニング): ' + str(acc_train))
38         print('Generation: ' + str(i+1) + ', 正確率(テスト): ' + str(acc_test))
39
40         plt.title("accuracy")
41         plt.xlabel("count")
42         plt.ylabel("accuracy")
43         plt.plot(count, acc_train, "blue", label="training set")
44         plt.plot(count, acc_test, "orange", label="test set")
45         plt.legend()
46         plt.show()
47
48         f.write(str(i+1) + ", " + str(wcrf.train) + ", " + str(acc_train) + ", " + str(acc_test) + "\n")
49
50 f.close()
51 print("訓練終了")
52 print("正確率(トレーニング): " + str(wcrf.train))
53 print("正確率(テスト): " + str(wcrf.accuracy(x_test, d_test)))
54
```



2 2_9_regularization



The screenshot shows a Google Colab notebook interface. The title bar indicates the file is named '2.9_regularization.ipynb'. The code cell contains the following Python code:

```
1 import tensorflow as tf
2 import numpy as np

[4]: [1] (x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()
      2 x_train = x_train / 255
      3 x_test = x_test / 255
      4 x_train = np.expand_dims(x_train, axis=-1)
      5 x_test = np.expand_dims(x_test, axis=-1)
      6 x_train = np.reshape(x_train, (y_train.shape[0], depth=10))
      7 x_test = np.reshape(x_test, (y_test.shape[0], depth=10))

    8 print(x_train.shape)
    9 print(y_train.shape)
   10 print(x_test.shape)
   11 print(y_test.shape)

[5]: Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170480717198901 [=====] 100% |██████████| 3s 0.0MB/s
100000 (100000/100000) [=====] 100% |██████████| 0s 0.0MB/s
(50000, 10)
(50000, 10)
(10000, 10)
(10000, 10)

[6]: [1] (label, _,
      2 _): 'airplane',
      3 _): 'automobile',
      4 _): 'bird',
      5 _): 'boat',
      6 _): 'car',
      7 _): 'deer',
      8 _): 'dog',
      9 _): 'horse',
     10 _): 'truck'
```

index2label ... 0~9の数字に対してラベルが解説されています。

```
index2label = [
    0: "airplane",
    1: "automobile",
    2: "bird",
    3: "boat",
    4: "car",
    5: "dog",
    6: "frog",
    7: "horse",
    8: "motorcycle",
    9: "truck"
]
```

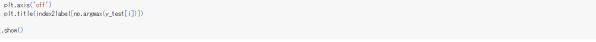
```
import tensorflow as tf
import random

index = 0
count = 50
n = 10

plt.figure(figsize=(10, 10))

for i in range(count):
    pth = os.path.join('fashion-mnist/images', index)
    pth_sub = os.path.join(pth, i + 1)
    os.makedirs(pth_sub, exist_ok=True)
    os.makedirs(os.path.join(pth_sub, 'train'), exist_ok=True)
    os.makedirs(os.path.join(pth_sub, 'val'), exist_ok=True)
    os.makedirs(os.path.join(pth_sub, 'test'), exist_ok=True)

    os.listdir(index2label[re.random.randint(0, 9)])
```



airplane	truck	dog	horse	truck	ship	frog	automobile	horse	cat	automobile
										

A grid of 100 small images arranged in a 10x10 pattern. Each image is a sample from the CIFAR-10 dataset, showing a different object or animal. The images include various types of cars (sedans, SUVs, trucks), different breeds of dogs, cats, frogs, birds, and other animals like deer and horses. The images are color photographs with varying backgrounds.

```


1 epochs = 5
2 batch_size = 256
3
4 def create_model(input_shape, classes):
5     model = tf.keras.models.Sequential([
6         tf.keras.layers.Conv2D(32, (3, 3), padding='same', input_shape=input_shape[1], activation='relu'),
7         tf.keras.layers.MaxPooling2D(),
8         tf.keras.layers.Flatten(),
9         tf.keras.layers.Dense(512, activation='relu'),
10        tf.keras.layers.Dense(classes, activation='softmax')
11    ])
12    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc'])
13
14    return model
15
16 model = create_model(x_train.shape[1])
17 model.summary()


```

```

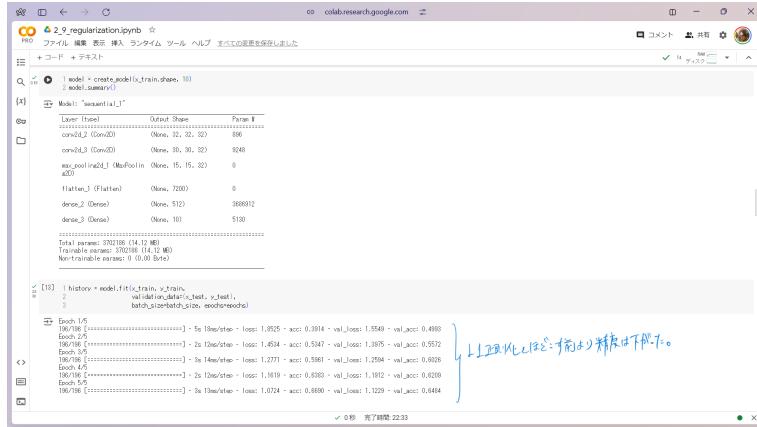

1 model = create_model(x_train.shape[1])
2 model.summary()
3
4
5 Epoch 0/5
6
7 100/100 [=====] - 8s/step - loss: 1.5011 - acc: 0.4670 - val_loss: 1.1681 - val_acc: 0.5738
8 100/100 [=====] - 8s/step - loss: 1.0955 - acc: 0.6171 - val_loss: 1.0789 - val_acc: 0.6242
9 100/100 [=====] - 2s/step - loss: 0.9398 - acc: 0.6740 - val_loss: 0.9397 - val_acc: 0.6513
10 Epoch 1/5
11 100/100 [=====] - 2s/step - loss: 0.8191 - acc: 0.7361 - val_loss: 0.8185 - val_acc: 0.6719
12 100/100 [=====] - 2s/step - loss: 0.7022 - acc: 0.7587 - val_loss: 0.6569 - val_acc: 0.6751
13
14


```

```


1 def create_model(input_shape, classes):
2     model = tf.keras.models.Sequential([
3         tf.keras.layers.Conv2D(32, (3, 3), padding='same', input_shape=input_shape[1], activation='relu'),
4         tf.keras.layers.Activation('relu'),
5         tf.keras.layers.MaxPooling2D(),
6         tf.keras.layers.Flatten(),
7         tf.keras.layers.Dense(512, activation='relu', activity_regularizer=tf.keras.regularizers.L1(0.001)),
8         tf.keras.layers.Dense(classes, activation='softmax')
9     ])
10
11    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc'])
12
13    return model
14
15 model = create_model(x_train.shape[1])
16 model.summary()

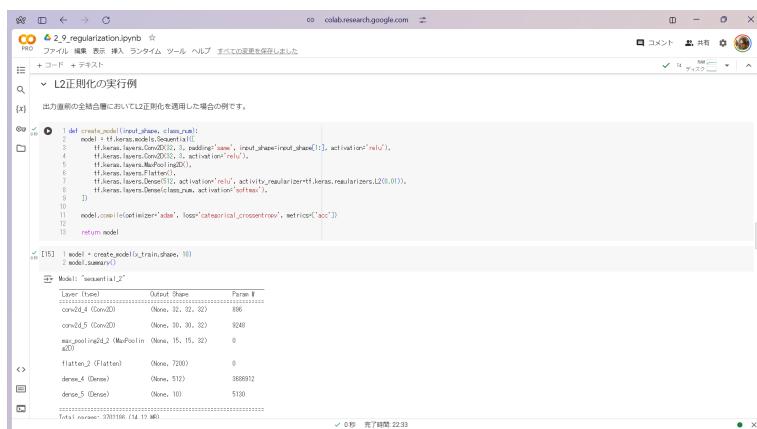

```



```
# In [1]: model = create_model(x_train, y_train, 10)
# In [2]: model.summary()

# In [3]: history = model.fit(x_train, y_train,
#                         validation_data=(x_test, y_test),
#                         batch_size=batch_size, epochs=epochs)

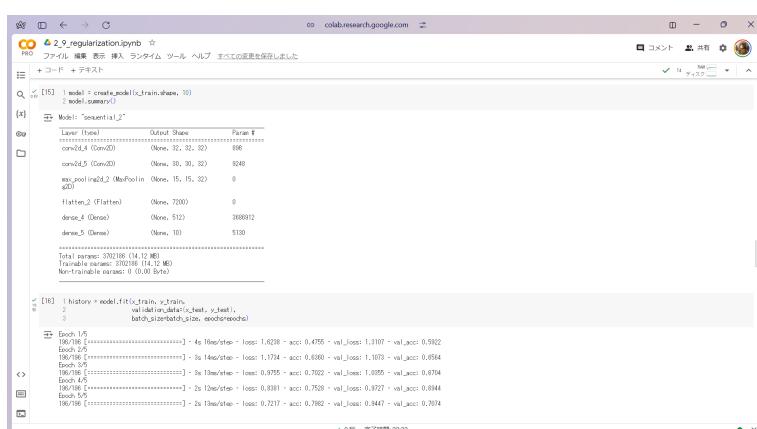
# In [4]: Epoch 1/5
# 196/196 [=====] - 4s 18ms/step - loss: 1.0925 - acc: 0.3014 - val_loss: 1.0540 - val_acc: 0.4989
# 196/196 [=====] - 2s 18ms/step - loss: 1.6314 - acc: 0.5347 - val_loss: 1.0857 - val_acc: 0.5972
# 196/196 [=====] - 3s 18ms/step - loss: 1.2771 - acc: 0.5881 - val_loss: 1.0294 - val_acc: 0.6028
# 196/196 [=====] - 2s 18ms/step - loss: 1.1619 - acc: 0.6088 - val_loss: 1.0192 - val_acc: 0.6209
# 196/196 [=====] - 3s 18ms/step - loss: 1.0724 - acc: 0.6890 - val_loss: 1.0129 - val_acc: 0.6484
```



```
# In [1]: def create_model(x_train, y_train):
#     model = Sequential()
#     model.add(Conv2D(32, (3, 3), padding='same', input_shape=x_train.shape[1:], activation='relu'))
#     model.add(Activation('relu'))
#     model.add(MaxPooling2D(pool_size=(2, 2)))
#     model.add(Dropout(0.2))
#     model.add(Dense(512, activation='relu', activity_regularizer=lens_regularizerL2(0.01)))
#     model.add(Dense(10, activation='softmax'))
#     model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc'])
#     return model

# In [2]: model = create_model(x_train, y_train, 10)
# In [3]: model.summary()

# In [4]: Model: "sequential1"
# Layer (type)                 Output Shape              Param #   ...
# conv2d_4 (Conv2D)            (None, 32, 32, 32)       968
# conv2d_5 (Conv2D)            (None, 16, 16, 32)      9248
# max_pooling2d_2 (MaxPooling2D) (None, 16, 16, 32)      0
# flatten_2 (Flatten)          (None, 7200)             0
# dense_2 (Dense)              (None, 512)            368912
# dense_3 (Dense)              (None, 10)             510
```



```
# In [1]: model = create_model(x_train, y_train, 10)
# In [2]: model.summary()

# In [3]: Model: "sequential1"
# Layer (type)                 Output Shape              Param #   ...
# conv2d_4 (Conv2D)            (None, 32, 32, 32)       968
# conv2d_5 (Conv2D)            (None, 16, 16, 32)      9248
# max_pooling2d_2 (MaxPooling2D) (None, 16, 16, 32)      0
# flatten_2 (Flatten)          (None, 7200)             0
# dense_2 (Dense)              (None, 512)            368912
# dense_3 (Dense)              (None, 10)             510
# ...
# In [4]: history = model.fit(x_train, y_train,
#                         validation_data=(x_test, y_test),
#                         batch_size=batch_size, epochs=epochs)

# In [5]: Epoch 1/5
# 196/196 [=====] - 4s 18ms/step - loss: 1.0218 - acc: 0.4956 - val_loss: 1.0107 - val_acc: 0.5922
# 196/196 [=====] - 2s 18ms/step - loss: 1.1714 - acc: 0.5680 - val_loss: 1.0173 - val_acc: 0.6594
# 196/196 [=====] - 3s 18ms/step - loss: 0.9375 - acc: 0.7022 - val_loss: 1.0295 - val_acc: 0.6704
# 196/196 [=====] - 2s 18ms/step - loss: 0.8181 - acc: 0.7328 - val_loss: 0.9372 - val_acc: 0.6944
# 196/196 [=====] - 2s 18ms/step - loss: 0.7217 - acc: 0.7982 - val_loss: 0.9447 - val_acc: 0.7074
```

2.9_regularization.ipynb

ファイル フォルダ 検索 リスト ランタイム ツール ヘルプ すべての変数を保存しました

+ コード + テキスト

Elasti Net の実例

```
1 def create_model(input_shape, classes=2):
2     model = Sequential()
3     model.add(Flatten())
4     model.add(Dense(300, 3, padding='same', input_shape=input_shape), activation='relu'),
5     model.add(Dense(300, 3, activation='relu'),
6     model.add(Flatten())
7     model.add(Dense(100, activation='relu'), activity_regularizer=keras.regularizers.L1L2(0.01)),
8     model.add(Dense(100, activation='relu'), activation='softmax'),
9     model.add(Dense(2))
10    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc'])
11
12    return model
```

[1] In [1] model = create_model(input_shape=(100, 100, 3), classes=2)
[2] In [1] model.summary()

Model: "sequential_3"

Layer (type)	Output Shape	Param #
Flatten	(None, 10000)	0
conv2d_1 (Conv2D)	(None, 32, 32, 32)	1056
conv2d_2 (Conv2D)	(None, 30, 30, 32)	9248
max_pooling2d_1 (MaxPooling2D)	(None, 15, 15, 32)	0
flatten_3 (Flatten)	(None, 7200)	0
dense_3 (Dense)	(None, 512)	368912
dense_4 (Dense)	(None, 10)	5130

Total params: 3702188 (14.12 MB)
Trainable params: 3702188 (14.12 MB)

コメント 共有

Elasti Net
↓
L1正則化項 / L1W²
L2正則化項 / L2W²
多角化させた結果の特徴量を定義する方法。

Elasti Net
 \downarrow
 L1 正則化項 $\lambda_1 \|w\|_1$
 L2 正則化項 $\lambda_2 \|w\|^2$
 等於岭回归系数的常数项方法

```
2.9_regularization.ipynb ☆
ファイル フォルダ 新規 フォルダ ランタイム ツール ヘルプ すべての変更を保存しました
+ コード + テスト

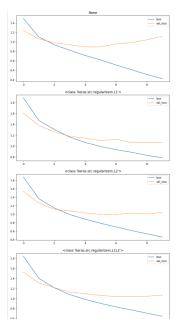
[1]: [1] model = create_model((x_train.shape[1], 10))
      2) model.summary()

[2]: Model: "sequential_1"
   ___________________________________
   Layer (type)                 Output Shape              Param #   
   =================================================================
   flatten_3 (Flatten)          (None, 7200)             0
   dense_9 (Dense)              (None, 512)              3888012  
   dense_7 (Dense)              (None, 10)               5130
   ___________________________________
   Total params: 39,181,032
   Trainable params: 38,691,512 (98.4%)
   Non-trainable params: 0 (0.00% Byte)

[3]: history = model.fit(x_train, y_train,
      epochs=10,
      validation_data=(x_test, y_test),
      batch_size=batch_size, verbose=1)

Epoch 1/5
100/100 [=====] - 0s/step - loss: 1.0414 - acc: 0.3906 - val_loss: 1.5175 - val_acc: 0.5168
100/100 [=====] - 0s/step - loss: 1.4032 - acc: 0.3550 - val_loss: 1.2227 - val_acc: 0.5801
100/100 [=====] - 0s/step - loss: 1.2196 - acc: 0.4264 - val_loss: 1.2104 - val_acc: 0.6238
100/100 [=====] - 0s/step - loss: 1.0892 - acc: 0.4927 - val_loss: 1.1107 - val_acc: 0.6913
100/100 [=====] - 0s/step - loss: 0.9596 - acc: 0.7009 - val_loss: 1.1067 - val_acc: 0.6975
```

✓ 0 秒 完了時間: 22:33



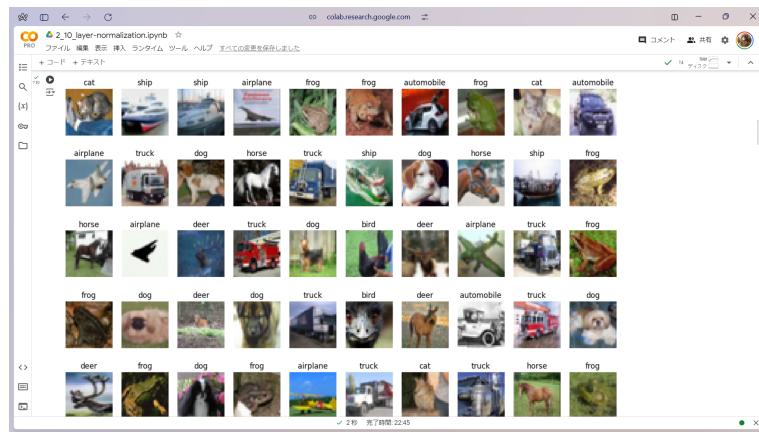
横軸: パラメータ α の誤差
縦軸: パラメータ α の誤差
(横軸: 視覚・視覚)

3 2_10_layer_normalization

layer-normalization.ipynb

```
1 import tensorflow as tf
2 import numpy as np
3
4 index = 0
5 count = 50
6
7 plt.figure(figsize=(10, 10))
8
9 for i in range(5):
10     plt.subplot(5, 10, index+1)
11     plt.axis('off')
12     plt.title(str(index+1))
13     plt.imshow(arasari.y_test[i])
14
15 plt.show()
```

cat	ship	ship	airplane	frog	frog	automobile	frog	cat	automobile



```

2_10_layer-normalization.ipynb
+ コード + テキスト
Q [x] 1 epochs = 5
2 batch_size = 256
3
4 def create_model(input_shape, class_num):
5     model = tf.keras.models.Sequential([
6         tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape),
7         tf.keras.layers.MaxPool2D((2, 2), activation='relu'),
8         tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
9         tf.keras.layers.MaxPool2D((2, 2), activation='relu'),
10        tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
11        tf.keras.layers.MaxPool2D((2, 2), activation='relu'),
12        tf.keras.layers.Flatten(),
13        tf.keras.layers.Dense(128, activation='relu'),
14        tf.keras.layers.Dense(class_num, activation='softmax'),
15    ])
16    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc'])
17    return model
18
19 model = create_model(x_train.shape, 10)
20 model.summary()
Model: "sequential"
Layer (type)                 Output Shape              Param #   
=================================================================
conv1 (Conv2D)                (None, 32, 32, 32)       96        
conv1_1 (Conv2D)               (None, 16, 16, 32)      9248      
conv1_2 (Conv2D)               (None, 16, 16, 32)      9248      
max_pooling2d (MaxPooling2D)  (None, 16, 16, 32)      0         
flatten (Flatten)             (None, 7200)            0         
dense (Dense)                (None, 512)             3689012  
dense_1 (Dense)               (None, 10)              5100      
Total params: 3702108 (14.12 MB)
Trainable params: 3689012 (14.09 MB)
Non-trainable params: 0 (0.00 MB)

```

```

2_10_layer-normalization.ipynb
+ コード + テキスト
Q [x] 1 model = create_model(x_train.shape, 10)
2 model.summary()
Model: "sequential"
Layer (type)                 Output Shape              Param #   
=================================================================
conv1 (Conv2D)                (None, 32, 32, 32)       96        
conv1_1 (Conv2D)               (None, 16, 16, 32)      9248      
conv1_2 (Conv2D)               (None, 16, 16, 32)      9248      
max_pooling2d (MaxPooling2D)  (None, 16, 16, 32)      0         
flatten (Flatten)             (None, 7200)            0         
dense (Dense)                (None, 512)             3689012  
dense_1 (Dense)               (None, 10)              5100      
Total params: 3702108 (14.12 MB)
Trainable params: 3689012 (14.09 MB)
Non-trainable params: 0 (0.00 MB)

```

[10]:

```

1 history = model.fit(x_train, y_train,
2                      validation_data=(x_test, y_test),
3                      batch_size=batch_size,
4                      epochs=10)
Epoch 0/5: [........................] - 8s/step - loss: 1.5981 - acc: 0.4840 - val_loss: 1.2138 - val_acc: 0.5713
Epoch 1/5: [........................] - 8s/step - loss: 1.0982 - acc: 0.6118 - val_loss: 1.0654 - val_acc: 0.6106
Epoch 2/5: [........................] - 8s/step - loss: 0.9465 - acc: 0.6721 - val_loss: 0.8952 - val_acc: 0.6552
Epoch 3/5: [........................] - 8s/step - loss: 0.8251 - acc: 0.7107 - val_loss: 0.8446 - val_acc: 0.6704
Epoch 4/5: [........................] - 8s/step - loss: 0.7025 - acc: 0.7471 - val_loss: 0.8060 - val_acc: 0.6970
Epoch 5/5: [........................] - 8s/step - loss: 0.6741 - acc: 0.7645 - val_loss: 0.7471 - val_acc: 0.7070

```

正規化レイヤーあり

Batch正規化

1 層目のCNNの時にBatch正規化を適用した場合の例。

```
def create_model(input_shape, classes):
    model = Sequential()
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(BatchNormalization())
    model.add(Dense(128, activation='relu'))
    model.add(BatchNormalization())
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dense(128, activation='relu'))
    model.add(Dense(classes, activation='softmax'))
    return model
```

[12] 1 model = create_model((train.shape[1]), 10)
2 model.summary()

Model: "sequential1"

Layer (Type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 32, 32, 32)	960
batch_normalization (Batch Normalization)	(None, 32, 32, 32)	128
conv2d_2 (Conv2D)	(None, 30, 30, 32)	9248
max_pooling2d_1 (MaxPooling2D)	(None, 15, 15, 32)	0

2_10_layer-normalization.py

```
# モデルの構成と学習用データの準備
model = create_model(trn_sharp)
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# モデルの学習
history = model.fit(x_train, y_train,
                     validation_data=(x_val, y_val),
                     epochs=100, batch_size=128)

# モデルの評価
score = model.evaluate(x_val, y_val, batch_size=128)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
# 2_10_layer-normalization.pyrb ☆
ファイル フォルダ 表示 検索 ランタイム タール ヘルプ すべての変更を保存しました
+ コード + テスト
Q [x] model = create_model(x_train_shas, 1)
[x] Model: "sequential_1"
Layer (type)          Output Shape         Param #      ...
[...]  
conv2d_4 (Conv2D)     (None, 32, 32, 32)    896  
instance_normalization (In (None, 32, 32, 32)) 64  
shallow_maxpooling1 (MaxPooling2D)  
conv2d_5 (Conv2D)     (None, 32, 32, 32)    8240  
separable_conv2d_2 (SeparableConv2D)  
dense_4 (Dense)      (None, 512)           368892  
dense_5 (Dense)      (None, 10)            510  
Total params: 370252 (14.12 MB)  
Trainable params: 370252 (14.12 MB)  
Non-trainable params: 0 (0.00 MB)  
-----  
[x] history = model.fit(x_train, y_train,  
                      validation_data=(x_test, y_test),  
                      batch_size=64, epochs=10)  
Epoch 1/5  
1/100 [=====] - 7s 2ms/step - loss: 1.284 - acc: 0.390 - val_loss: 1.289 - val_acc: 0.389  
Epoch 2/5  
1/100 [=====] - 7s 2ms/step - loss: 1.281 - acc: 0.395 - val_loss: 1.159 - val_acc: 0.581  
Epoch 3/5  
1/100 [=====] - 7s 2ms/step - loss: 0.870 - acc: 0.851 - val_loss: 1.005 - val_acc: 0.647  
Epoch 4/5  
1/100 [=====] - 7s 2ms/step - loss: 0.707 - acc: 0.740 - val_loss: 1.009 - val_acc: 0.627  
Epoch 5/5  
1/100 [=====] - 7s 2ms/step - loss: 0.583 - acc: 0.826 - val_loss: 1.112 - val_acc: 0.644  
-----  
+ Instance正规化
```

```
# 2_10_layer-normalization.pyrb ☆
ファイル フォルダ 表示 検索 ランタイム タール ヘルプ すべての変更を保存しました
+ コード + テスト
Q [x] model = create_model(x_train_shas, 1)
[x] Model: "sequential_1"
Layer (type)          Output Shape         Param #      ...
[...]  
conv2d_8 (Conv2D)     (None, 32, 32, 32)    896  
instance_normalization (In (None, 32, 32, 32)) 64  
shallow_maxpooling1 (MaxPooling2D)  
conv2d_9 (Conv2D)     (None, 32, 32, 32)    8240  
separable_conv2d_3 (SeparableConv2D)  
dense_6 (Dense)      (None, 512)           368892  
dense_7 (Dense)      (None, 10)            510  
Total params: 370252 (14.12 MB)  
Trainable params: 370252 (14.12 MB)  
Non-trainable params: 0 (0.00 MB)  
-----  
[x] history = model.fit(x_train, y_train,  
                      validation_data=(x_test, y_test),  
                      batch_size=64, epochs=10)  
Epoch 1/5  
1/100 [=====] - 7s 2ms/step - loss: 1.040 - acc: 0.469 - val_loss: 1.250 - val_acc: 0.548  
Epoch 2/5  
1/100 [=====] - 7s 2ms/step - loss: 1.005 - acc: 0.415 - val_loss: 1.029 - val_acc: 0.644  
Epoch 3/5  
1/100 [=====] - 7s 2ms/step - loss: 0.937 - acc: 0.843 - val_loss: 0.949 - val_acc: 0.870  
Epoch 4/5  
1/100 [=====] - 7s 2ms/step - loss: 0.878 - acc: 0.724 - val_loss: 0.901 - val_acc: 0.805  
Epoch 5/5  
1/100 [=====] - 7s 2ms/step - loss: 0.768 - acc: 0.792 - val_loss: 0.882 - val_acc: 0.897  
-----  
+ 2秒 完了時間: 22:45
```

```
# 2_10_layer-normalization.pyrb ☆
ファイル フォルダ 表示 検索 ダウンロード ランタイム タール ヘルプ すべての変更を保存しました
+ コード + テスト
Q [x] def create_model(x_train_shas, classes, norm_layer):
  if norm_layer == 'None':
    model = tf.keras.models.Sequential([
      tf.keras.layers.Conv2D(32, 3, padding='same', input_shape=input_shape[1:], activation='relu'),
      tf.keras.layers.Conv2D(32, 3, activation='relu'),
      tf.keras.layers.MaxPooling2D(),
      tf.keras.layers.Flatten(),
      tf.keras.layers.Dense(128, activation='relu'),
      tf.keras.layers.Dense(classes, activation='softmax'),
    ])
  else:
    model = tf.keras.models.Sequential([
      tf.keras.layers.Conv2D(32, 3, padding='same', input_shape=input_shape[1:], activation='relu'),
      tf.keras.layers.BatchNormalization(),
      tf.keras.layers.MaxPooling2D(),
      tf.keras.layers.Flatten(),
      tf.keras.layers.Dense(128, activation='relu'),
      tf.keras.layers.Dense(classes, activation='softmax'),
    ])
  model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc'])
  return model
-----  
[x] 1 epochs = 10  
2 batch_size = 256  
3 norm_layers = [  
4   None,  
5   tf.keras.layers.BatchNormalization,  
6   tf.keras.layers.LayerNormalization,  
7   tf.keras.layers.InstanceNormalization  
8 ]  
-----  
+ 2秒 完了時間: 22:45
```

