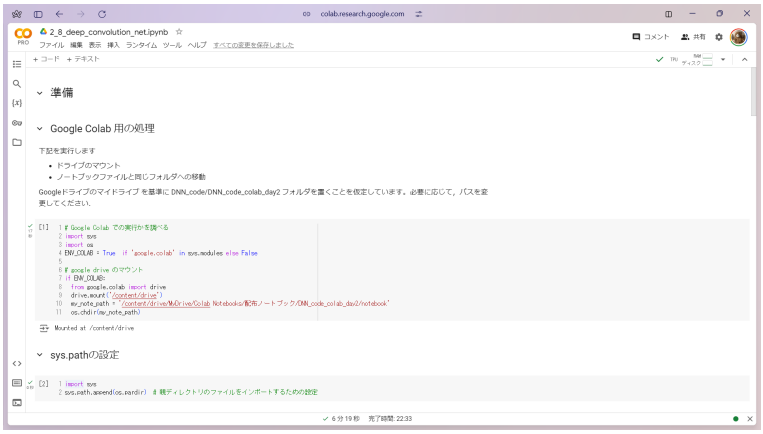
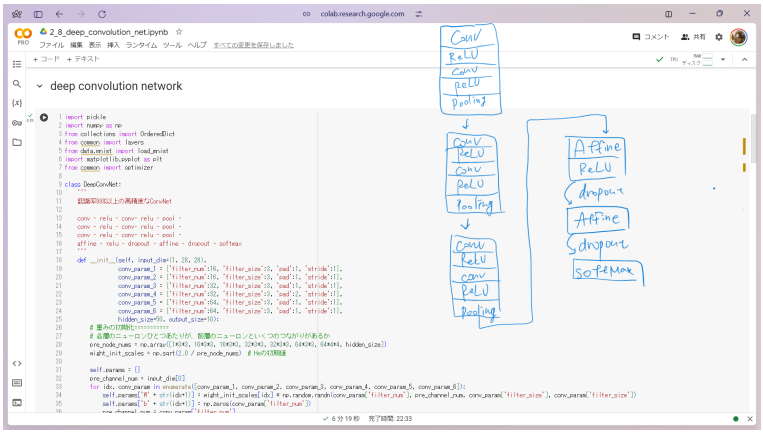
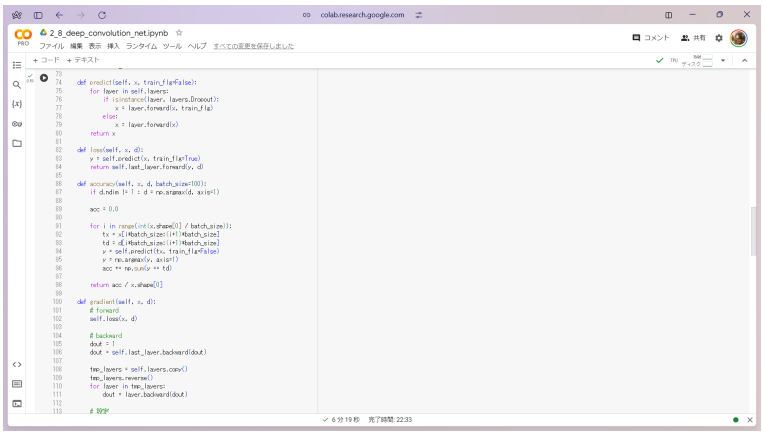
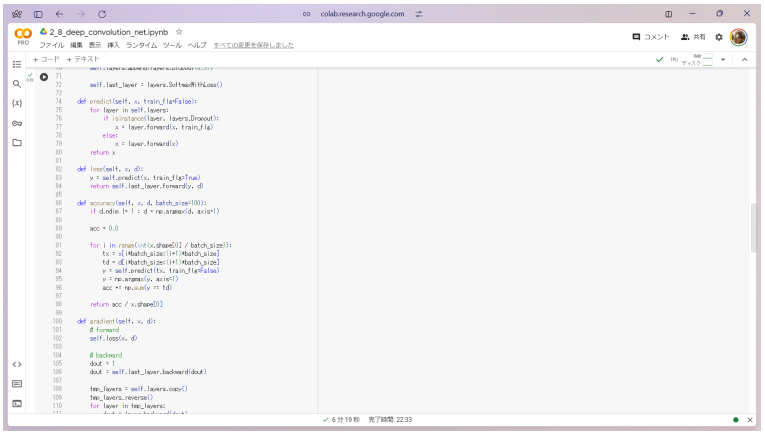
深層学習 day2 実装演習キャプチャ 3

1　2_8_deep_convolution_net

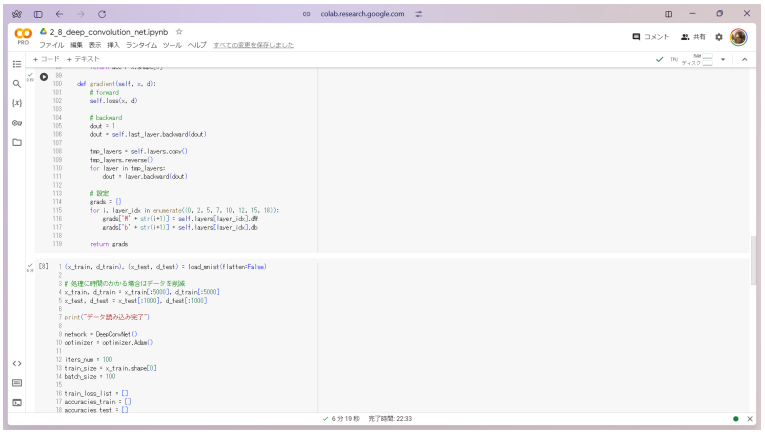






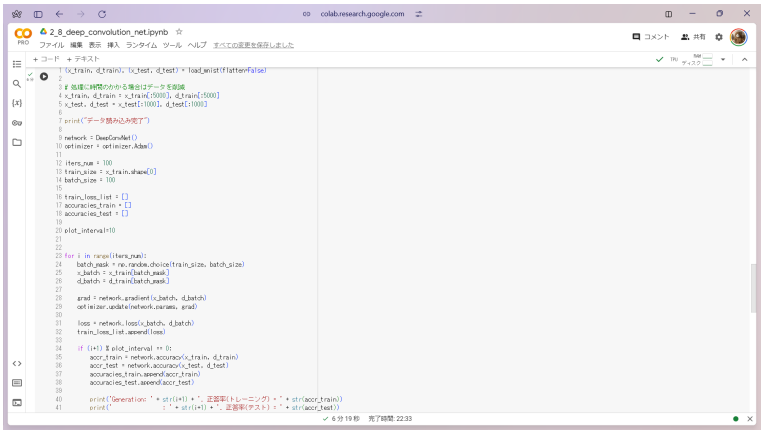


```
        self.last_layer = layers.SoftmaxWithLoss()

    def predict(self, x, train_flg=False):
        for layer in self.layers:
            if isinstance(layer, layers.Dropout):
                x = layer.forward(x, train_flg)
            else:
                x = layer.forward(x)
        return x

    def loss(self, x, d):
        y = self.predict(x, train_flg=True)
        return self.last_layer.forward(y, d)

    def accuracy(self, x, d, batch_size=100):
        if d.ndim != 1 : d = np.argmax(d, axis=1)

        acc = 0.0

        for i in range(int(x.shape[0] / batch_size)):
            tx = x[i*batch_size:(i+1)*batch_size]
            td = d[i*batch_size:(i+1)*batch_size]
            y = self.predict(tx, train_flg=False)
            y = np.argmax(y, axis=1)
            acc += np.sum(y == td)

        return acc / x.shape[0]

    def gradient(self, x, d):
        # forward
        self.loss(x, d)

        # backward
        dout = 1
        dout = self.last_layer.backward(dout)

        tmp_layers = self.layers.copy()
        tmp_layers.reverse()
        for layer in tmp_layers:
```

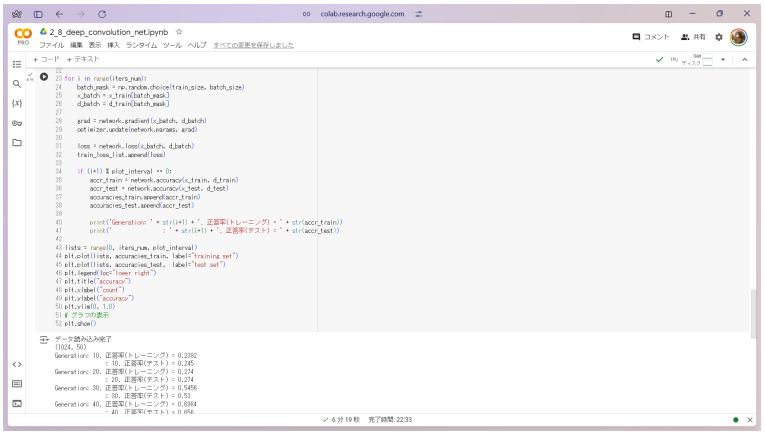


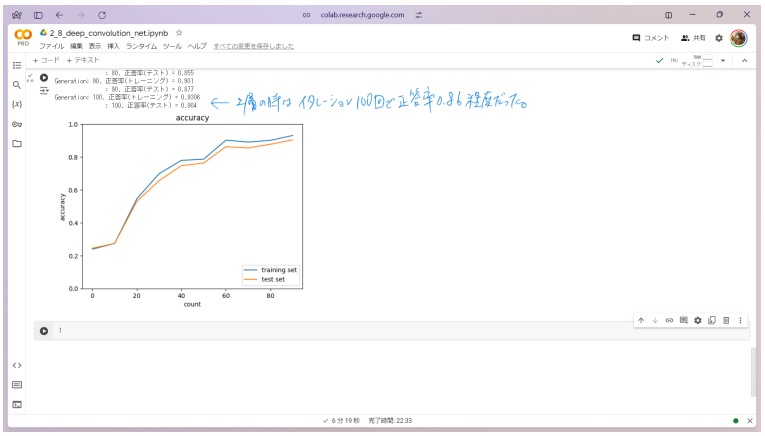
```
    def gradient(self, x, d):
        # forward
        self.loss(x, d)

        # backward
        dout = 1
        dout = self.last_layer.backward(dout)

        tmp_layers = self.layers.copy()
        tmp_layers.reverse()
        for layer in tmp_layers:
            dout = layer.backward(dout)

        # 設定
        grads = {}
        for i, layer_idx in enumerate((0, 2, 5, 7, 10, 12, 15, 18)):
            grads['W' + str(i+1)] = self.layers[layer_idx].dW
            grads['b' + str(i+1)] = self.layers[layer_idx].db

        return grads
```

```
(x_train, d_train), (x_test, d_test) = load_mnist(flatten=False)

# 処理に時間のかかる場合はデータを削減
x_train, d_train = x_train[:5000], d_train[:5000]
x_test, d_test = x_test[:1000], d_test[:1000]

print("データ読み込み完了")

network = DeepConvNet()
optimizer = optimizer.Adam()

iters_num = 100
train_size = x_train.shape[0]
batch_size = 100

train_loss_list = []
accuracies_train = []
accuracies_test = []
```

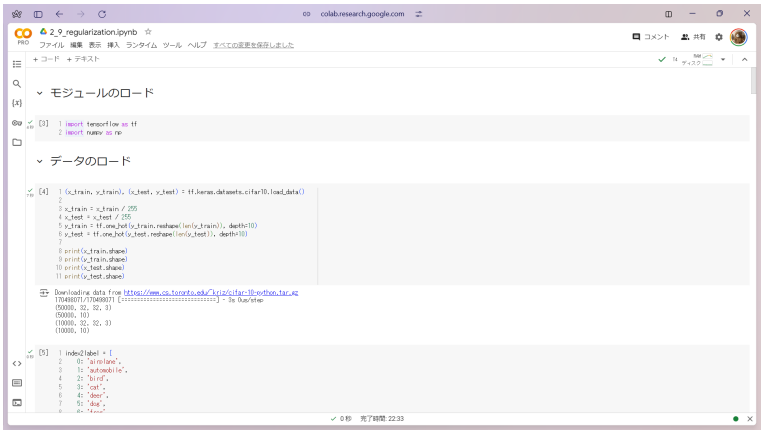


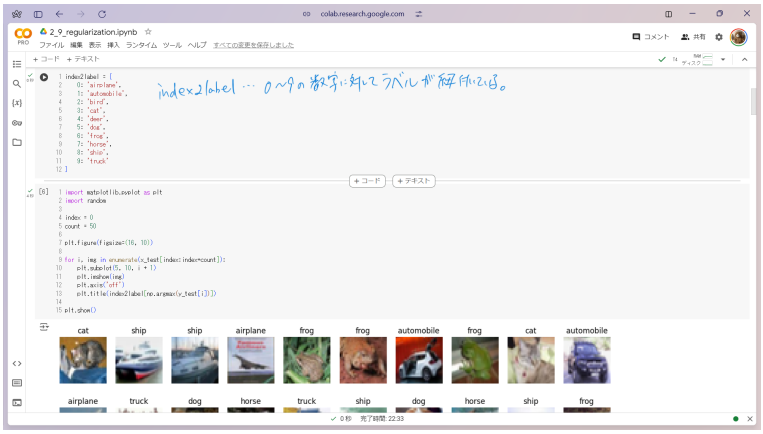
```
(x_train, d_train), (x_test, d_test) = load_mnist(flatten=False)

# 処理に時間のかかる場合はデータを削減
x_train, d_train = x_train[:5000], d_train[:5000]
x_test, d_test = x_test[:1000], d_test[:1000]

print("データ読み込み完了")

network = DeepConvNet()
optimizer = optimizer.Adam()

iters_num = 100
train_size = x_train.shape[0]
batch_size = 100

train_loss_list = []
accuracies_train = []
accuracies_test = []

plot_interval=10


for i in range(iters_num):
    batch_mask = np.random.choice(train_size, batch_size)
    x_batch = x_train[batch_mask]
    d_batch = d_train[batch_mask]

    grad = network.gradient(x_batch, d_batch)
    optimizer.update(network.params, grad)

    loss = network.loss(x_batch, d_batch)
    train_loss_list.append(loss)

    if (i+1) % plot_interval == 0:
        accr_train = network.accuracy(x_train, d_train)
        accr_test = network.accuracy(x_test, d_test)
        accuracies_train.append(accr_train)
        accuracies_test.append(accr_test)

        print("Generation: " + str(i+1) + ". 正答率(トレーニング) : " + str(accr_train))
        print("                  : " + str(i+1) + ". 正答率(テスト) : " + str(accr_test))
```

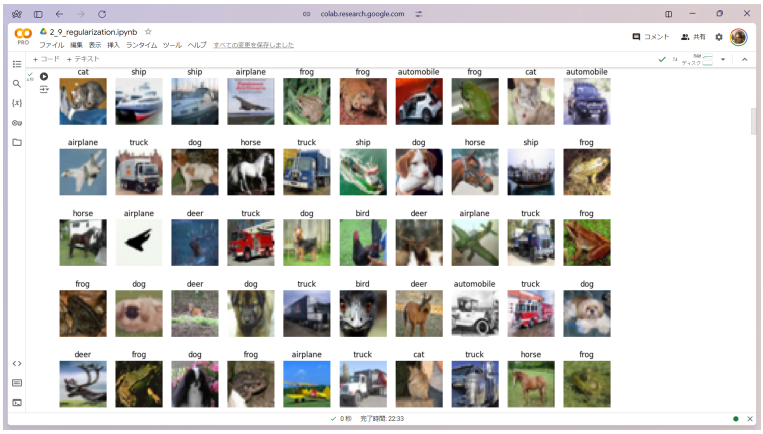

colab.research.google.com
2_8_deep_convolution_net.ipynb
ファイル 編集 表示 挿入 ランタイム ツール ヘルプ すべての変更を保存しました
コメント 共有
+ コード + テキスト
for i in range(iters_num):
batch_mask = np.random.choice(train_size, batch_size)
x_batch = x_train[batch_mask]
d_batch = d_train[batch_mask]
grad = network.gradient(x_batch, d_batch)
optimizer.update(network.params, grad)
loss = network.loss(x_batch, d_batch)
train_loss_list.append(loss)
if (i+1) % plot_interval == 0:
accr_train = network.accuracy(x_train, d_train)
accr_test = network.accuracy(x_test, d_test)
accuracies_train.append(accr_train)
accuracies_test.append(accr_test)
print("Generation: " + str(i+1) + ". 正答率(トレーニング) = " + str(accr_train))
print("   : " + str(i+1) + ". 正答率(テスト) = " + str(accr_test))
lists = range(0, iters_num, plot_interval)
plt.plot(lists, accuracies_train, label="training set")
plt.plot(lists, accuracies_test, label="test set")
plt.legend(loc="lower right")
plt.title("accuracy")
plt.xlabel("count")
plt.ylabel("accuracy")
plt.ylim(0, 1.0)
# グラフの描画
plt.show()
データ読み込み完了
(1024, 50)
Generation: 10. 正答率(トレーニング) = 0.2102
: 10. 正答率(テスト) = 0.245
Generation: 20. 正答率(トレーニング) = 0.274
: 20. 正答率(テスト) = 0.274
Generation: 30. 正答率(トレーニング) = 0.5456
: 30. 正答率(テスト) = 0.53
Generation: 40. 正答率(トレーニング) = 0.8984
: 40. 正答率(テスト) = 0.858
6 分 19 秒 完了時間: 22:33

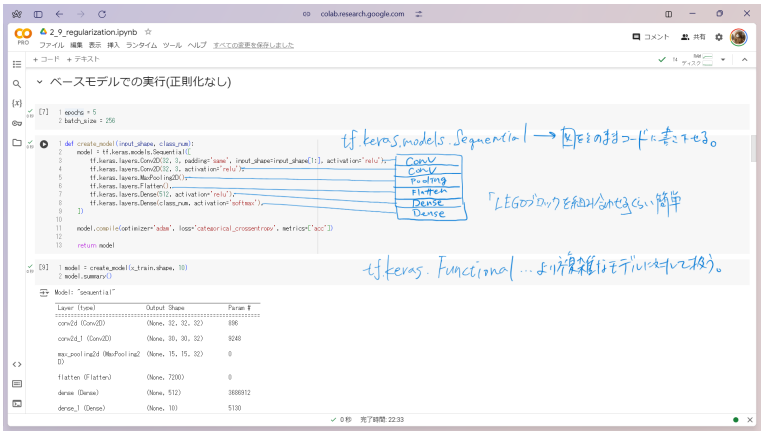

colab.research.google.com
2_8_deep_convolution_net.ipynb
ファイル 編集 表示 挿入 ランタイム ツール ヘルプ すべての変更を保存しました
コメント 共有
+ コード + テキスト
: 100. 正答率(テスト) = 0.955
Generation: 90. 正答率(トレーニング) = 0.991
: 90. 正答率(テスト) = 0.977
Generation: 100. 正答率(トレーニング) = 0.9886
: 100. 正答率(テスト) = 0.934
← 2層の時は イタレーション100回で正答率0.86程度だった。
accuracy
accuracy
count
training set
test set
1
6 分 19 秒 完了時間: 22:33

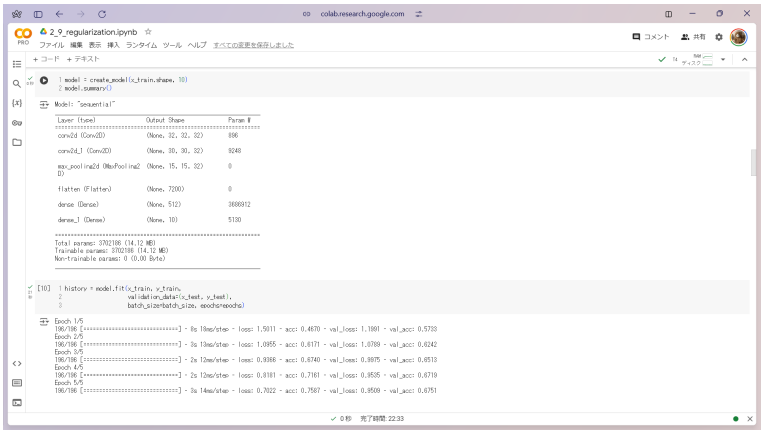
2　2_9_regularization

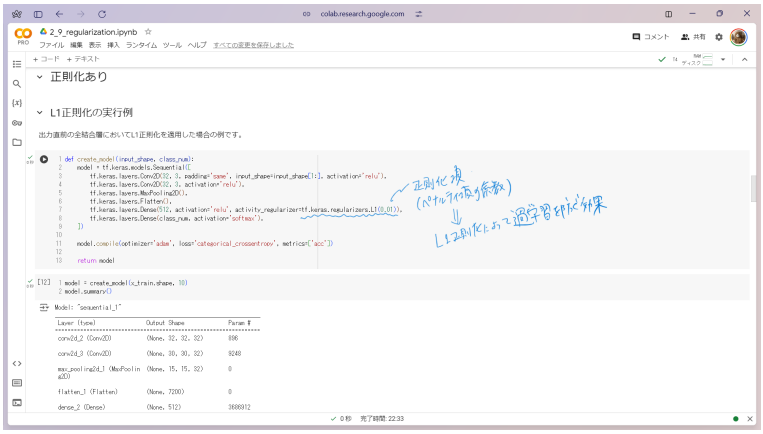


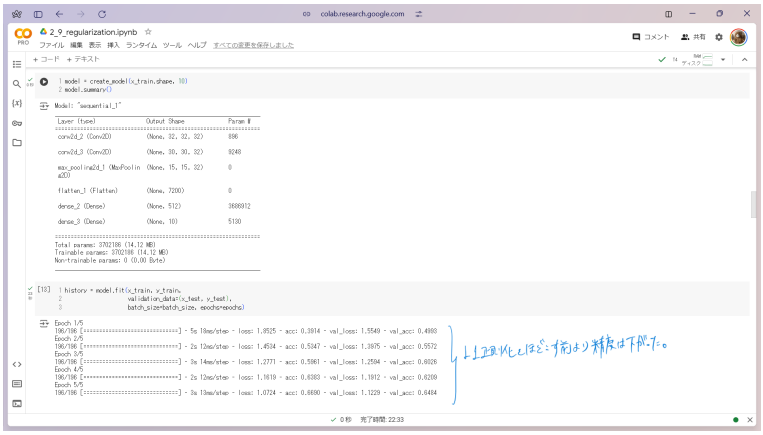


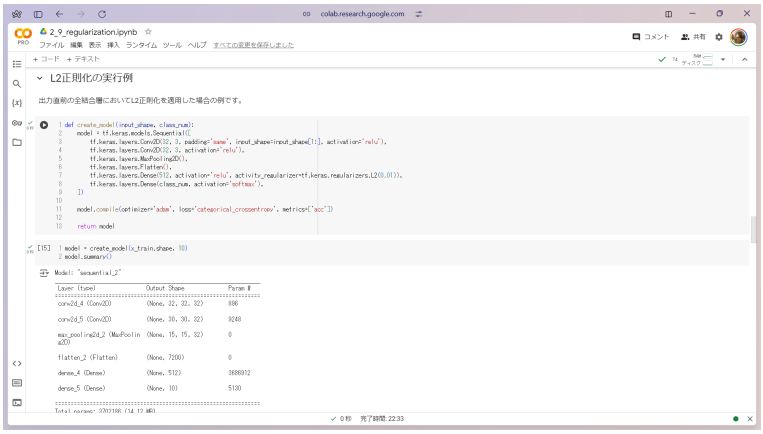


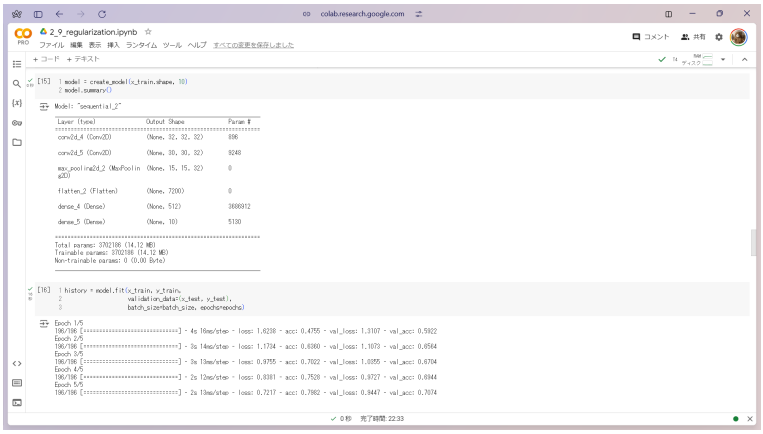

colab.research.google.com
2_9_regularization.ipynb
ファイル 編集 表示 挿入 ランタイム ツール ヘルプ すべての変更を保存しました
コメント 共有
+ コード + テキスト
ベースモデルでの実行(正則化なし)
[7] 1 epochs = 5
2 batch_size = 256
1 def create_model(input_shape, class_num):
2 model = tf.keras.models.Sequential([
3 tf.keras.layers.Conv2D(32, 3, padding='same', input_shape=input_shape[1:], activation='relu'),
4 tf.keras.layers.Conv2D(32, 3, activation='relu'),
5 tf.keras.layers.MaxPooling2D(),
6 tf.keras.layers.Flatten(),
7 tf.keras.layers.Dense(512, activation='relu'),
8 tf.keras.layers.Dense(class_num, activation='softmax'),
9])
10
11 model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc'])
12
13 return model
tf.keras.models.Sequential → 直感的なコードに書ける。
Conv
Conv
Pooling
Flatten
Dense
Dense
「LEGOブロックを組み合わせるような簡単
tf.keras.Functional …より複雑なモデルに対して使う。
[9] 1 model = create_model(x_train.shape, 10)
2 model.summary()
Model: "sequential"
Layer (type) Output Shape Param #
conv2d (Conv2D) (None, 32, 32, 32) 896
conv2d_1 (Conv2D) (None, 30, 30, 32) 9248
max_pooling2d (MaxPooling2D) (None, 15, 15, 32) 0
flatten (Flatten) (None, 7200) 0
dense (Dense) (None, 512) 3688912
dense_1 (Dense) (None, 10) 5130
0秒 完了時間: 22:33

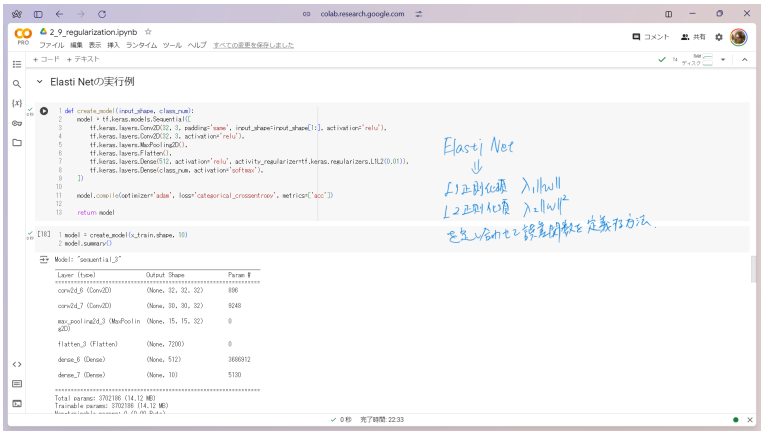

colab.research.google.com
2_9_regularization.ipynb
ファイル 編集 表示 挿入 ランタイム ツール ヘルプ すべての変更を保存しました
コメント 共有
+ コード + テキスト
[9] 1 model = create_model(x_train.shape, 10)
2 model.summary()
Model: "sequential"
Layer (type) Output Shape Param #
conv2d (Conv2D) (None, 32, 32, 32) 896
conv2d_1 (Conv2D) (None, 30, 30, 32) 9248
max_pooling2d (MaxPooling2D) (None, 15, 15, 32) 0
flatten (Flatten) (None, 7200) 0
dense (Dense) (None, 512) 3688912
dense_1 (Dense) (None, 10) 5130
Total params: 3702186 (14.12 MB)
Trainable params: 3702186 (14.12 MB)
Non-trainable params: 0 (0.00 Byte)
[10] 1 history = model.fit(x_train, y_train,
2 validation_data=(x_test, y_test),
3 batch_size=batch_size, epochs=epochs)
Epoch 1/5
196/196 [==========] - 8s 19ms/step - loss: 1.5011 - acc: 0.4870 - val_loss: 1.1091 - val_acc: 0.5733
Epoch 2/5
196/196 [==========] - 3s 13ms/step - loss: 1.0955 - acc: 0.6171 - val_loss: 1.0789 - val_acc: 0.6242
Epoch 3/5
196/196 [==========] - 2s 12ms/step - loss: 0.9366 - acc: 0.6740 - val_loss: 0.9975 - val_acc: 0.6513
Epoch 4/5
196/196 [==========] - 2s 12ms/step - loss: 0.8181 - acc: 0.7161 - val_loss: 0.9535 - val_acc: 0.6719
Epoch 5/5
196/196 [==========] - 3s 14ms/step - loss: 0.7022 - acc: 0.7587 - val_loss: 0.9509 - val_acc: 0.6751
0秒 完了時間: 22:33

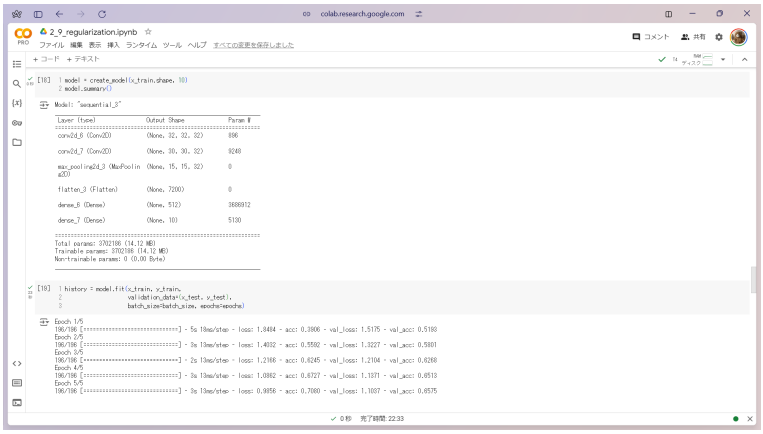

colab.research.google.com
2_9_regularization.ipynb
ファイル 編集 表示 挿入 ランタイム ツール ヘルプ すべての変更を保存しました
コメント 共有
+ コード + テキスト
正則化あり
L1正則化の実行例
出力直前の全結合層においてL1正則化を適用した場合の例です。
1 def create_model(input_shape, class_num):
2 model = tf.keras.models.Sequential([
3 tf.keras.layers.Conv2D(32, 3, padding='same', input_shape=input_shape[1:], activation='relu'),
4 tf.keras.layers.Conv2D(32, 3, 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(0.01)),
8 tf.keras.layers.Dense(class_num, activation='softmax'),
9])
10
11 model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc'])
12
13 return model
正則化項
(ペナルティ項)
係数
L1正則化による過学習抑制効果
[12] 1 model = create_model(x_train.shape, 10)
2 model.summary()
Model: "sequential_1"
Layer (type) Output Shape Param #
conv2d_2 (Conv2D) (None, 32, 32, 32) 896
conv2d_3 (Conv2D) (None, 30, 30, 32) 9248
max_pooling2d_1 (MaxPooling2D) (None, 15, 15, 32) 0
flatten_1 (Flatten) (None, 7200) 0
dense_2 (Dense) (None, 512) 3688912
0秒 完了時間: 22:33

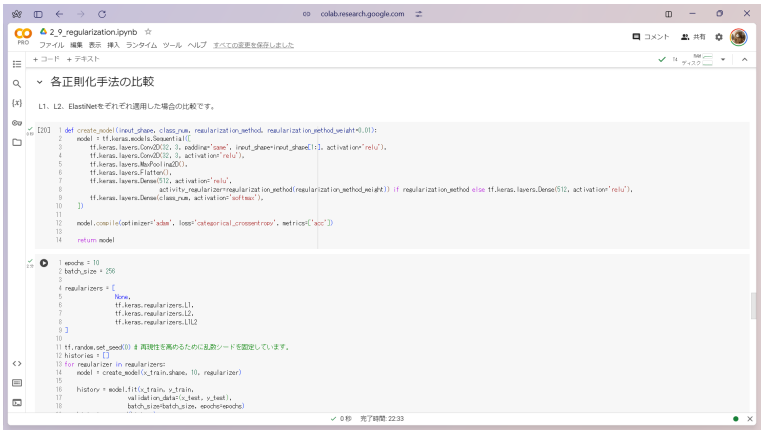

2_9_regularization.ipynb
ファイル 編集 表示 挿入 ランタイム ツール ヘルプ すべての変更を保存しました
+ コード + テキスト
model = create_model(x_train.shape, 10)
model.summary()
Model: "sequential_1"
Layer (type) Output Shape Param #
conv2d_2 (Conv2D) (None, 32, 32, 32) 896
conv2d_3 (Conv2D) (None, 30, 30, 32) 9248
max_pooling2d_1 (MaxPooling2D) (None, 15, 15, 32) 0
flatten_1 (Flatten) (None, 7200) 0
dense_2 (Dense) (None, 512) 3686912
dense_3 (Dense) (None, 10) 5130
Total params: 3702186 (14.12 MB)
Trainable params: 3702186 (14.12 MB)
Non-trainable params: 0 (0.00 Byte)
[18] history = model.fit(x_train, y_train,
validation_data=(x_test, y_test),
batch_size=batch_size, epochs=epochs)
Epoch 1/5
196/196 [==========] - 5s 19ms/step - loss: 1.8525 - acc: 0.3814 - val_loss: 1.5549 - val_acc: 0.4988
Epoch 2/5
196/196 [==========] - 2s 12ms/step - loss: 1.4534 - acc: 0.5347 - val_loss: 1.3875 - val_acc: 0.5572
Epoch 3/5
196/196 [==========] - 3s 14ms/step - loss: 1.2771 - acc: 0.5961 - val_loss: 1.2594 - val_acc: 0.6026
Epoch 4/5
196/196 [==========] - 2s 12ms/step - loss: 1.1619 - acc: 0.6383 - val_loss: 1.1912 - val_acc: 0.6269
Epoch 5/5
196/196 [==========] - 3s 13ms/step - loss: 1.0724 - acc: 0.6690 - val_loss: 1.1229 - val_acc: 0.6484
L2正則化したほど：精度は下がった。
0秒 完了時間: 22:33

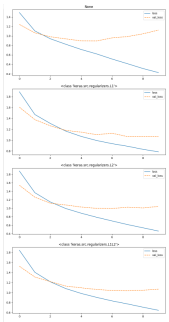

2_9_regularization.ipynb
ファイル 編集 表示 挿入 ランタイム ツール ヘルプ すべての変更を保存しました
+ コード + テキスト
L2正則化の実行例
出力直前の全結合層においてL2正則化を適用した場合の例です。
def create_model(input_shape, class_num):
model = tf.keras.models.Sequential([
tf.keras.layers.Conv2D(32, 3, padding='same', input_shape=input_shape[1:], activation='relu'),
tf.keras.layers.Conv2D(32, 3, activation='relu'),
tf.keras.layers.MaxPooling2D(),
tf.keras.layers.Flatten(),
tf.keras.layers.Dense(512, activation='relu', activity_regularizer=tf.keras.regularizers.L2(0.01)),
tf.keras.layers.Dense(class_num, activation='softmax'),
])
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc'])
return model
[15] model = create_model(x_train.shape, 10)
model.summary()
Model: "sequential_2"
Layer (type) Output Shape Param #
conv2d_4 (Conv2D) (None, 32, 32, 32) 896
conv2d_5 (Conv2D) (None, 30, 30, 32) 9248
max_pooling2d_2 (MaxPooling2D) (None, 15, 15, 32) 0
flatten_2 (Flatten) (None, 7200) 0
dense_4 (Dense) (None, 512) 3686912
dense_5 (Dense) (None, 10) 5130
Total params: 3702186 (14.12 MB)
0秒 完了時間: 22:33


2_9_regularization.ipynb
ファイル 編集 表示 挿入 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 + テキスト
[15] model = create_model(x_train.shape, 10)
model.summary()
Model: "sequential_2"
Layer (type) Output Shape Param #
conv2d_4 (Conv2D) (None, 32, 32, 32) 896
conv2d_5 (Conv2D) (None, 30, 30, 32) 9248
max_pooling2d_2 (MaxPooling2D) (None, 15, 15, 32) 0
flatten_2 (Flatten) (None, 7200) 0
dense_4 (Dense) (None, 512) 3686912
dense_5 (Dense) (None, 10) 5130
Total params: 3702186 (14.12 MB)
Trainable params: 3702186 (14.12 MB)
Non-trainable params: 0 (0.00 Byte)
[16] history = model.fit(x_train, y_train,
validation_data=(x_test, y_test),
batch_size=batch_size, epochs=epochs)
Epoch 1/5
196/196 [==========] - 4s 16ms/step - loss: 1.8238 - acc: 0.4755 - val_loss: 1.3107 - val_acc: 0.5922
Epoch 2/5
196/196 [==========] - 3s 14ms/step - loss: 1.1724 - acc: 0.6060 - val_loss: 1.1073 - val_acc: 0.6564
Epoch 3/5
196/196 [==========] - 3s 13ms/step - loss: 0.9755 - acc: 0.7022 - val_loss: 1.0055 - val_acc: 0.6704
Epoch 4/5
196/196 [==========] - 2s 12ms/step - loss: 0.8381 - acc: 0.7528 - val_loss: 0.9727 - val_acc: 0.6944
Epoch 5/5
196/196 [==========] - 2s 13ms/step - loss: 0.7217 - acc: 0.7982 - val_loss: 0.9447 - val_acc: 0.7074
0秒 完了時間: 22:33


2_9_regularization.ipynb
Elasti Netの実行例
Elasti Net
⇓
L1正則化項 λ1||w||
L2正則化項 λ2||w||²
を足し合わせた誤差関数を定義する方法。
Model: "sequential_3"
Total params: 3702186 (14.12 MB)
Trainable params: 3702186 (14.12 MB)
Non-trainable params: 0 (0.00 Byte)


2_9_regularization.ipynb
Model: "sequential_3"
Total params: 3702186 (14.12 MB)
Trainable params: 3702186 (14.12 MB)
Non-trainable params: 0 (0.00 Byte)
Epoch 1/5
196/196 [==============================] - 5s 19ms/step - loss: 1.8484 - acc: 0.3906 - val_loss: 1.5175 - val_acc: 0.5188
Epoch 2/5
196/196 [==============================] - 3s 13ms/step - loss: 1.4032 - acc: 0.5502 - val_loss: 1.3227 - val_acc: 0.5801
Epoch 3/5
196/196 [==============================] - 2s 13ms/step - loss: 1.2166 - acc: 0.6245 - val_loss: 1.2104 - val_acc: 0.6268
Epoch 4/5
196/196 [==============================] - 3s 13ms/step - loss: 1.0862 - acc: 0.6727 - val_loss: 1.1371 - val_acc: 0.6513
Epoch 5/5
196/196 [==============================] - 3s 13ms/step - loss: 0.9856 - acc: 0.7080 - val_loss: 1.1097 - val_acc: 0.6575

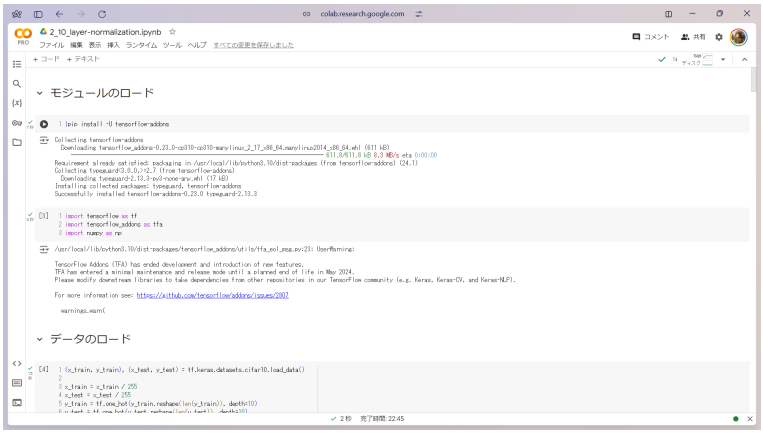

2_9_regularization.ipynb
各正則化手法の比較
L1、L2、ElastiNetをそれぞれ適用した場合の比較です。

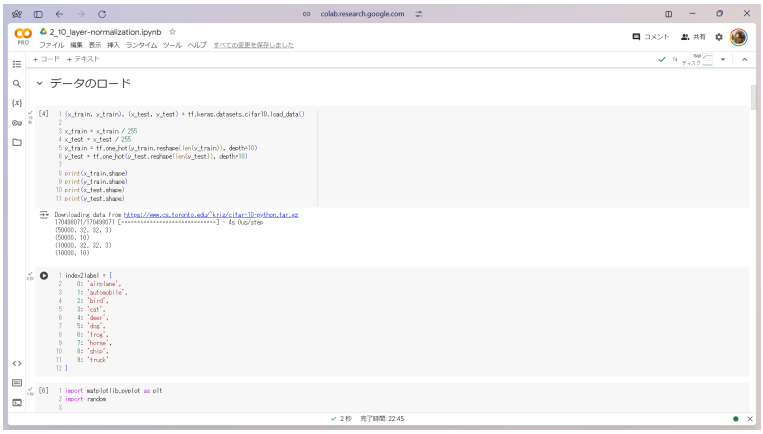


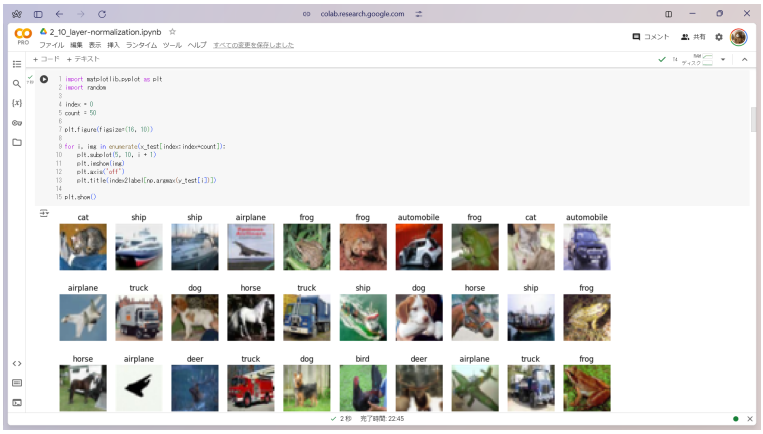
青線：トレーニング時の誤差
赤線：バリデーションデータの誤差

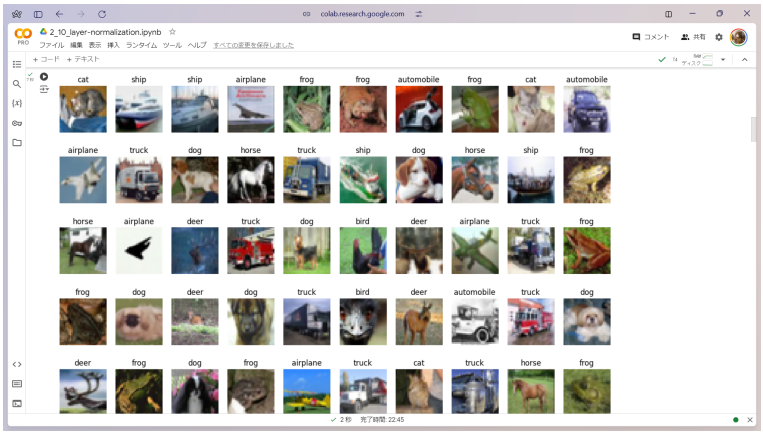
（横軸：イテレーション数）

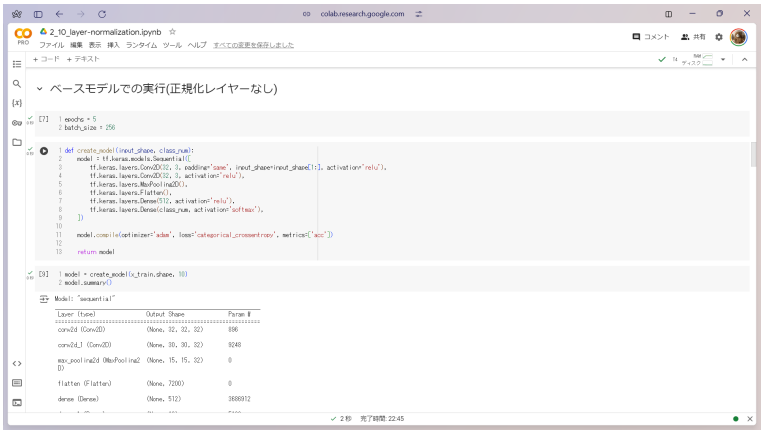
3　2_10_layer_normalization

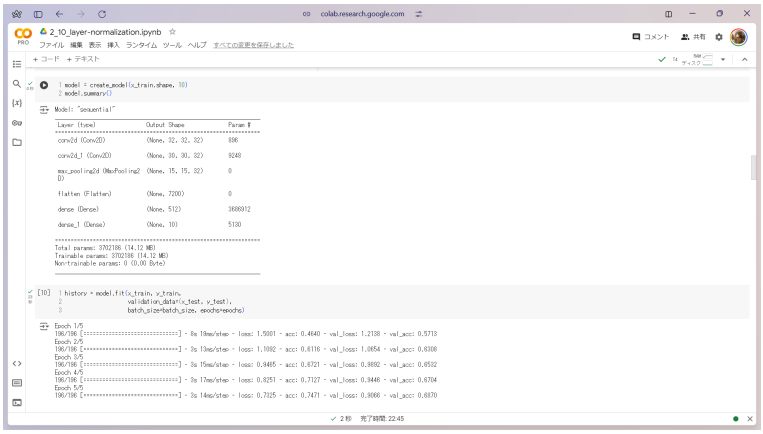


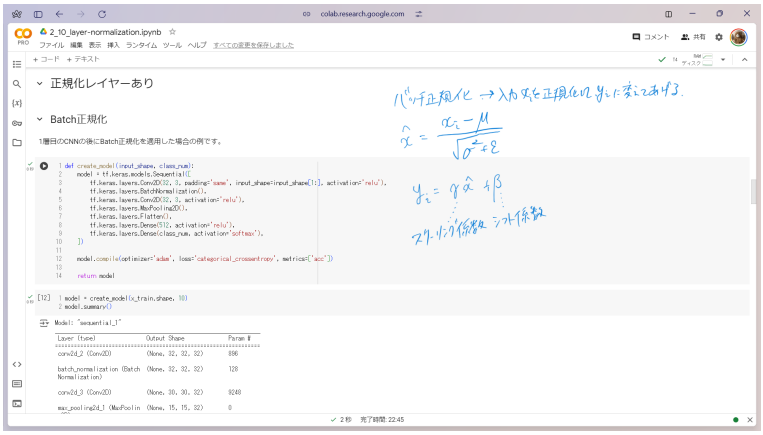


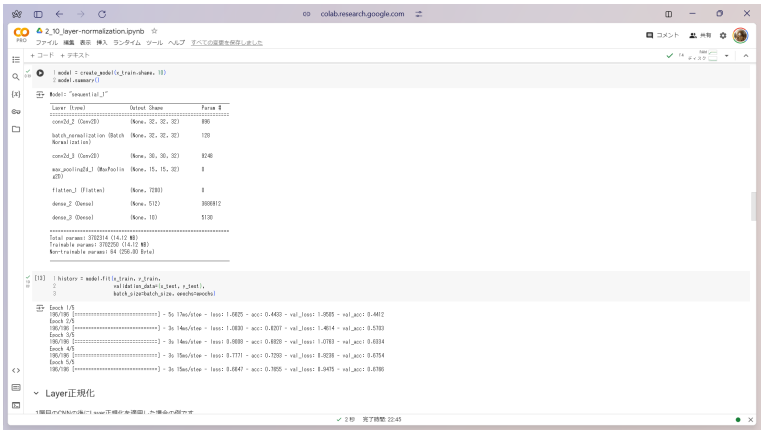


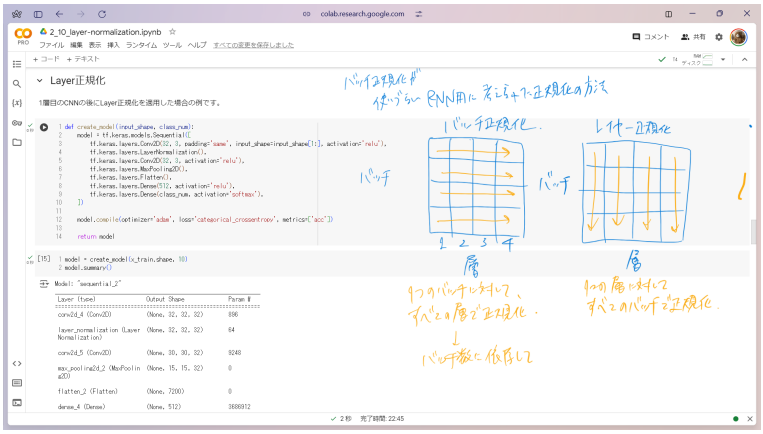

colab.research.google.com
2_10_layer-normalization.ipynb
cat ship ship airplane frog frog automobile frog cat automobile
airplane truck dog horse truck ship dog horse ship frog
horse airplane deer truck dog bird deer airplane truck frog
frog dog deer dog truck bird deer automobile truck dog
deer frog dog frog airplane truck cat truck horse frog

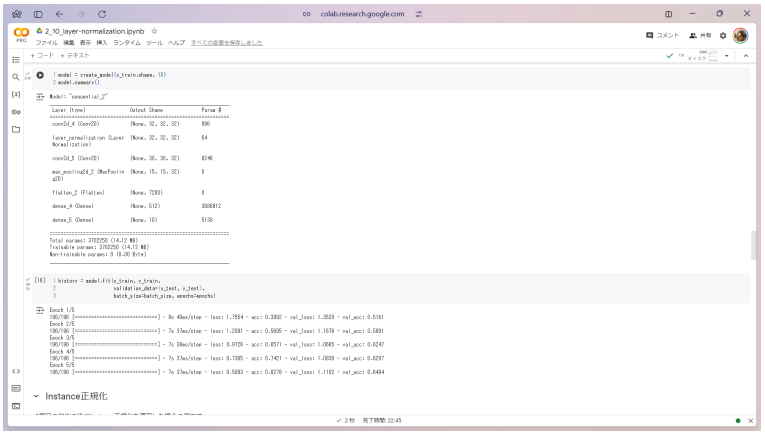

colab.research.google.com
2_10_layer-normalization.ipynb
ベースモデルでの実行(正規化レイヤーなし)
epochs = 5
batch_size = 256
def create_model(input_shape, class_num):
model = tf.keras.models.Sequential([
tf.keras.layers.Conv2D(32, 3, padding='same', input_shape=input_shape[1:], activation='relu'),
tf.keras.layers.Conv2D(32, 3, activation='relu'),
tf.keras.layers.MaxPool2D(),
tf.keras.layers.Flatten(),
tf.keras.layers.Dense(512, activation='relu'),
tf.keras.layers.Dense(class_num, activation='softmax'),
])
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc'])
return model
model = create_model(x_train.shape, 10)
model.summary()
Model: "sequential"
Layer (type) Output Shape Param #
conv2d (Conv2D) (None, 32, 32, 32) 896
conv2d_1 (Conv2D) (None, 30, 30, 32) 9248
max_pooling2d (MaxPooling2D) (None, 15, 15, 32) 0
flatten (Flatten) (None, 7200) 0
dense (Dense) (None, 512) 3688912

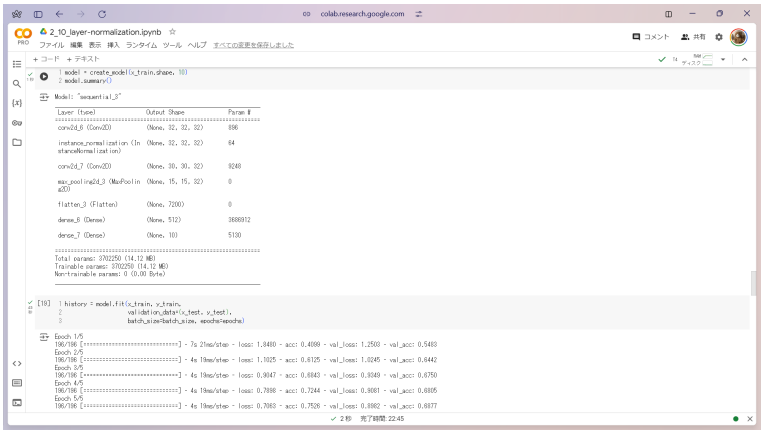

colab.research.google.com
2_10_layer-normalization.ipynb
model = create_model(x_train.shape, 10)
model.summary()
Model: "sequential"
Layer (type) Output Shape Param #
conv2d (Conv2D) (None, 32, 32, 32) 896
conv2d_1 (Conv2D) (None, 30, 30, 32) 9248
max_pooling2d (MaxPooling2D) (None, 15, 15, 32) 0
flatten (Flatten) (None, 7200) 0
dense (Dense) (None, 512) 3688912
dense_1 (Dense) (None, 10) 5130
Total params: 3702186 (14.12 MB)
Trainable params: 3702186 (14.12 MB)
Non-trainable params: 0 (0.00 Byte)
history = model.fit(x_train, y_train,
validation_data=(x_test, y_test),
batch_size=batch_size, epochs=epochs)
Epoch 1/5
196/196 [==========] - 8s 19ms/step - loss: 1.5001 - acc: 0.4640 - val_loss: 1.2138 - val_acc: 0.5718
Epoch 2/5
196/196 [==========] - 3s 13ms/step - loss: 1.1092 - acc: 0.6116 - val_loss: 1.0854 - val_acc: 0.6308
Epoch 3/5
196/196 [==========] - 3s 15ms/step - loss: 0.9465 - acc: 0.6721 - val_loss: 0.9892 - val_acc: 0.6592
Epoch 4/5
196/196 [==========] - 3s 17ms/step - loss: 0.8251 - acc: 0.7127 - val_loss: 0.9446 - val_acc: 0.6704
Epoch 5/5
196/196 [==========] - 3s 14ms/step - loss: 0.7125 - acc: 0.7471 - val_loss: 0.9066 - val_acc: 0.6870

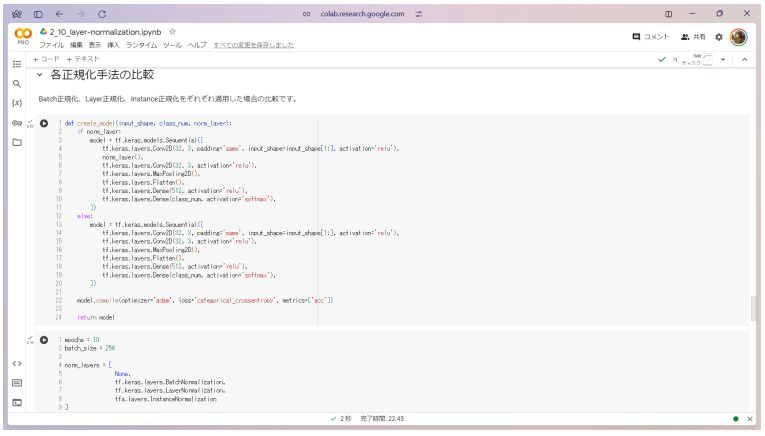

colab.research.google.com
2_10_layer-normalization.ipynb
正規化レイヤーあり
Batch正規化
1層目のCNNの後にBatch正規化を適用した場合の例です。
バッチ正規化 → 入力を正規化して学習に変えてみる。
$\hat{x} = \frac{x - \mu}{\sqrt{\sigma^2 + \varepsilon}}$
$y = \gamma\hat{x} + \beta$
γ: スケール係数 β: シフト係数
model = create_model(x_train.shape, 10)
model.summary()
Model: "sequential_1"
Layer (type) Output Shape Param #
conv2d_2 (Conv2D) (None, 32, 32, 32) 896
batch_normalization (BatchNormalization) (None, 32, 32, 32) 128
conv2d_3 (Conv2D) (None, 30, 30, 32) 9248
max_pooling2d_1 (MaxPooling2D) (None, 15, 15, 32) 0
完了時間 22:45

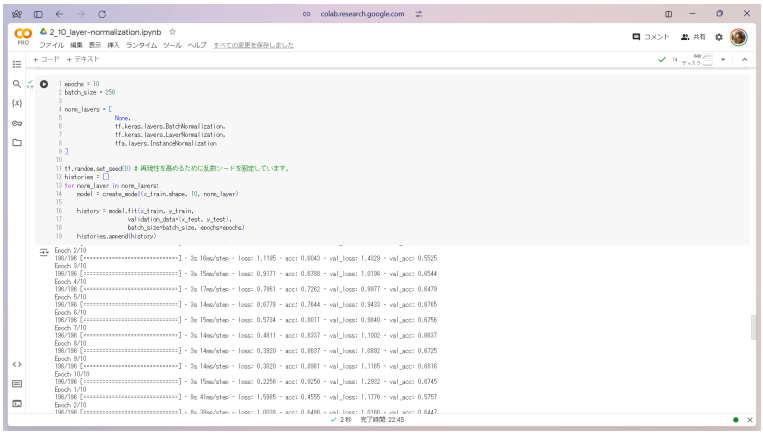

colab.research.google.com
2_10_layer-normalization.ipynb
model = create_model(x_train.shape, 10)
model.summary()
Model: "sequential_1"
Total params: 3702314 (14.12 MB)
Trainable params: 3702250 (14.12 MB)
Non-trainable params: 64 (256.00 Byte)
history = model.fit(x_train, y_train,
validation_data=(x_test, y_test),
batch_size=batch_size, epochs=epochs)
Epoch 1/5
1563/1563 [==========] - 5s 17ms/step - loss: 1.6825 - acc: 0.4433 - val_loss: 1.3585 - val_acc: 0.4412
Epoch 2/5
1563/1563 [==========] - 3s 14ms/step - loss: 1.4830 - acc: 0.4207 - val_loss: 1.4014 - val_acc: 0.5303
Epoch 3/5
1563/1563 [==========] - 3s 14ms/step - loss: 0.9698 - acc: 0.6828 - val_loss: 1.0783 - val_acc: 0.6314
Epoch 4/5
1563/1563 [==========] - 3s 15ms/step - loss: 0.7171 - acc: 0.7293 - val_loss: 0.8236 - val_acc: 0.6754
Epoch 5/5
1563/1563 [==========] - 3s 15ms/step - loss: 0.6647 - acc: 0.7855 - val_loss: 0.9415 - val_acc: 0.6786
Layer正規化
完了時間 22:45

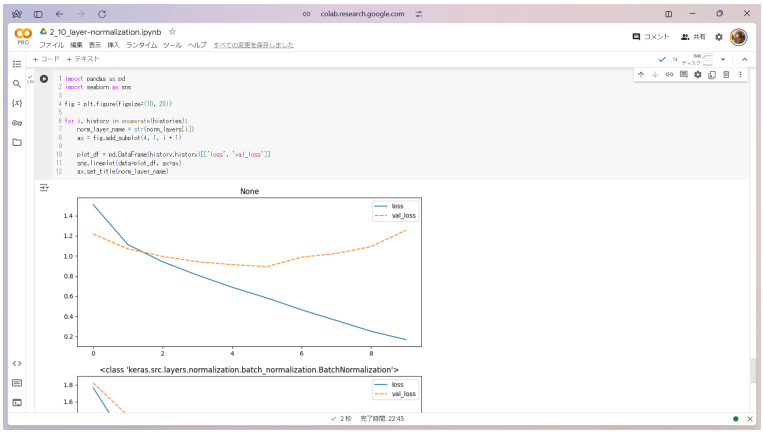

colab.research.google.com
2_10_layer-normalization.ipynb
Layer正規化
1層目のCNNの後にLayer正規化を適用した場合の例です。
バッチ正規化 → RNN用に考えられた正規化の方法
バッチ正規化
レイヤー正規化
バッチ
バッチ
層
層
1つのバッチに対して、すべての層で正規化
バッチ数に依存しない
1つの層に対してすべてのバッチで正規化
model = create_model(x_train.shape, 10)
model.summary()
Model: "sequential_2"
Layer (type) Output Shape Param #
conv2d_4 (Conv2D) (None, 32, 32, 32) 896
layer_normalization (LayerNormalization) (None, 32, 32, 32) 64
conv2d_5 (Conv2D) (None, 30, 30, 32) 9248
max_pooling2d_2 (MaxPooling2D) (None, 15, 15, 32) 0
flatten_2 (Flatten) (None, 7200) 0
dense_4 (Dense) (None, 512) 3686912
完了時間 22:45


colab.research.google.com
2_10_layer-normalization.ipynb
ファイル 編集 表示 挿入 ランタイム ツール ヘルプ すべての変更を保存しました
コメント 共有
+ コード + テキスト
1 model = create_model(x_train.shape, 10)
2 model.summary()
Model: "sequential_2"
Layer (type) Output Shape Param #
conv2d_4 (Conv2D) (None, 32, 32, 32) 896
layer_normalization (Layer Normalization) (None, 32, 32, 32) 64
conv2d_5 (Conv2D) (None, 30, 30, 32) 9248
max_pooling2d_2 (MaxPooling2D) (None, 15, 15, 32) 0
flatten_2 (Flatten) (None, 7200) 0
dense_4 (Dense) (None, 512) 3686912
dense_5 (Dense) (None, 10) 5130
Total params: 3702250 (14.12 MB)
Trainable params: 3702250 (14.12 MB)
Non-trainable params: 0 (0.00 Byte)
[18] 1 history = model.fit(x_train, y_train,
2 validation_data=(x_test, y_test),
3 batch_size=batch_size, epochs=epochs)
Epoch 1/5
196/196 [==========] - 9s 48ms/step - loss: 1.7554 - acc: 0.3992 - val_loss: 1.3528 - val_acc: 0.5181
Epoch 2/5
196/196 [==========] - 7s 37ms/step - loss: 1.2091 - acc: 0.5695 - val_loss: 1.1674 - val_acc: 0.5891
Epoch 3/5
196/196 [==========] - 7s 38ms/step - loss: 0.9726 - acc: 0.6571 - val_loss: 1.0065 - val_acc: 0.6247
Epoch 4/5
196/196 [==========] - 7s 37ms/step - loss: 0.7385 - acc: 0.7421 - val_loss: 1.0039 - val_acc: 0.6297
Epoch 5/5
196/196 [==========] - 7s 37ms/step - loss: 0.5883 - acc: 0.8278 - val_loss: 1.1182 - val_acc: 0.6494
Instance正規化
2秒 完了時間 22:45


colab.research.google.com
2_10_layer-normalization.ipynb
ファイル 編集 表示 挿入 ランタイム ツール ヘルプ すべての変更を保存しました
コメント 共有
+ コード + テキスト
1 model = create_model(x_train.shape, 10)
2 model.summary()
Model: "sequential_3"
Layer (type) Output Shape Param #
conv2d_6 (Conv2D) (None, 32, 32, 32) 896
instance_normalization (InstanceNormalization) (None, 32, 32, 32) 64
conv2d_7 (Conv2D) (None, 30, 30, 32) 9248
max_pooling2d_3 (MaxPooling2D) (None, 15, 15, 32) 0
flatten_3 (Flatten) (None, 7200) 0
dense_6 (Dense) (None, 512) 3686912
dense_7 (Dense) (None, 10) 5130
Total params: 3702250 (14.12 MB)
Trainable params: 3702250 (14.12 MB)
Non-trainable params: 0 (0.00 Byte)
[18] 1 history = model.fit(x_train, y_train,
2 validation_data=(x_test, y_test),
3 batch_size=batch_size, epochs=epochs)
Epoch 1/5
196/196 [==========] - 7s 21ms/step - loss: 1.8480 - acc: 0.4099 - val_loss: 1.2500 - val_acc: 0.5488
Epoch 2/5
196/196 [==========] - 4s 19ms/step - loss: 1.1025 - acc: 0.6125 - val_loss: 1.0245 - val_acc: 0.6442
Epoch 3/5
196/196 [==========] - 4s 19ms/step - loss: 0.8047 - acc: 0.8043 - val_loss: 0.8349 - val_acc: 0.6750
Epoch 4/5
196/196 [==========] - 4s 19ms/step - loss: 0.7898 - acc: 0.7244 - val_loss: 0.8081 - val_acc: 0.6805
Epoch 5/5
196/196 [==========] - 4s 19ms/step - loss: 0.7063 - acc: 0.7526 - val_loss: 0.8982 - val_acc: 0.6877
2秒 完了時間 22:45


colab.research.google.com
2_10_layer-normalization.ipynb
ファイル 編集 表示 挿入 ランタイム ツール ヘルプ すべての変更を保存しました
コメント 共有
+ コード + テキスト
各正規化手法の比較
Batch正規化、Layer正規化、Instance正規化をそれぞれ適用した場合の比較です。
def create_model(input_shape, class_num, norm_layer):
if norm_layer:
model = tf.keras.models.Sequential([
tf.keras.layers.Conv2D(32, 3, padding='same', input_shape=input_shape[1:], activation='relu'),
norm_layer(),
tf.keras.layers.Conv2D(32, 3, activation='relu'),
tf.keras.layers.MaxPooling2D(),
tf.keras.layers.Flatten(),
tf.keras.layers.Dense(512, activation='relu'),
tf.keras.layers.Dense(class_num, 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.Conv2D(32, 3, activation='relu'),
tf.keras.layers.MaxPooling2D(),
tf.keras.layers.Flatten(),
tf.keras.layers.Dense(512, activation='relu'),
tf.keras.layers.Dense(class_num, activation='softmax'),
])
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc'])
return model
epochs = 10
batch_size = 256
norm_layers = [
None,
tf.keras.layers.BatchNormalization,
tf.keras.layers.LayerNormalization,
tfa.layers.InstanceNormalization
]
2秒 完了時間 22:45


colab.research.google.com
2_10_layer-normalization.ipynb
epochs = 10
batch_size = 256
norm_layers = [
None,
tf.keras.layers.BatchNormalization,
tf.keras.layers.LayerNormalization,
tfa.layers.InstanceNormalization
]
tf.random.set_seed(0) # 再現性を高めるために乱数シードを固定しています。
histories = []
for norm_layer in norm_layers:
model = create_model(x_train.shape, 10, norm_layer)
history = model.fit(x_train, y_train,
validation_data=(x_test, y_test),
batch_size=batch_size, epochs=epochs)
histories.append(history)
Epoch 2/10
196/196 [==============================] - 3s 16ms/step - loss: 1.1195 - acc: 0.6043 - val_loss: 1.4329 - val_acc: 0.5525
Epoch 3/10
196/196 [==============================] - 3s 15ms/step - loss: 0.9171 - acc: 0.6788 - val_loss: 1.0198 - val_acc: 0.6544
Epoch 4/10
196/196 [==============================] - 3s 17ms/step - loss: 0.7861 - acc: 0.7262 - val_loss: 0.9977 - val_acc: 0.6479
Epoch 5/10
196/196 [==============================] - 3s 14ms/step - loss: 0.6778 - acc: 0.7644 - val_loss: 0.9433 - val_acc: 0.6765
Epoch 6/10
196/196 [==============================] - 3s 15ms/step - loss: 0.5734 - acc: 0.8011 - val_loss: 0.9840 - val_acc: 0.6756
Epoch 7/10
196/196 [==============================] - 3s 14ms/step - loss: 0.4811 - acc: 0.8337 - val_loss: 1.1002 - val_acc: 0.6637
Epoch 8/10
196/196 [==============================] - 3s 14ms/step - loss: 0.3920 - acc: 0.8637 - val_loss: 1.0892 - val_acc: 0.6725
Epoch 9/10
196/196 [==============================] - 3s 14ms/step - loss: 0.3020 - acc: 0.8961 - val_loss: 1.1165 - val_acc: 0.6816
Epoch 10/10
196/196 [==============================] - 3s 15ms/step - loss: 0.2256 - acc: 0.9250 - val_loss: 1.2892 - val_acc: 0.6745
Epoch 1/10
196/196 [==============================] - 8s 41ms/step - loss: 1.5985 - acc: 0.4555 - val_loss: 1.3776 - acc: 0.5757
Epoch 2/10
2秒 完了時間 22:45


colab.research.google.com
2_10_layer-normalization.ipynb
import pandas as pd
import seaborn as sns
fig = plt.figure(figsize=(10, 20))
for i, history in enumerate(histories):
norm_layer_name = str(norm_layers[i])
ax = fig.add_subplot(4, 1, i + 1)
plot_df = pd.DataFrame(history.history)[['loss', 'val_loss']]
sns.lineplot(data=plot_df, ax=ax)
ax.set_title(norm_layer_name)
None
loss
val_loss
<class 'keras.src.layers.normalization.batch_normalization.BatchNormalization'>
loss
val_loss
2秒 完了時間 22:45