HW3 Report_ CNN Implementation_109064518_高聖哲

1. Model Architecture

1.Model 所需要使用到的 Layer 以及 Function, 如:Conv2D,FC……等等

1. 將其他動作寫成其他 py 檔, 在從 colab 中引入進來

階層一

hw3_109064518.py

1. Dataset reading&splitting
2. def to_categorical(one hot function)
3. def train(計算 train& validation function
4. def test(計算 test accuracy

階層二

layer.py

model.py

utils.py

1. 做 forward 以及 backward 運算
2. 取得並設置 params

1. im2col 和 col2im function(優化卷積層計算)
2. 其餘額外需要用的 function

階層三

Con2D Layer

AdamGD

AvgPool Layer

Softmax

FC Layer

CrossEntropy

ReLU

LeNet5

Con2D

AdamG

AvgPool

Softmax

FC Layer

CrossEn

ReLU

set params

get params

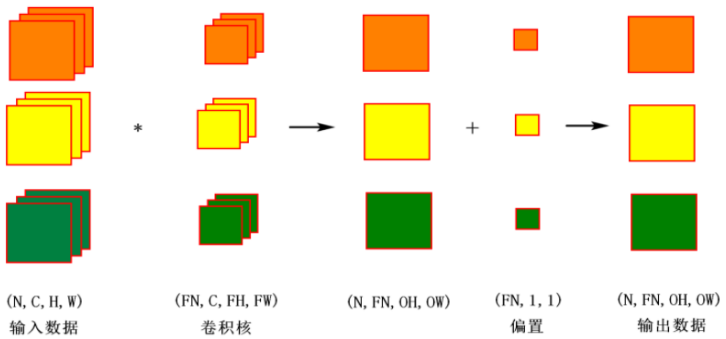
dataloader

im2col

save_params_to_file

col2im

load_params_from_file



Con2D Layer Parameter

Parameters ----------

N：批數目，C：通道數，H：輸入資料高，W：輸入資料，stride : 步幅 ，pad : 填充

out_h: 輸出資料的高，out_w 輸出資料的長，filter_w : 卷積核的長

2. Loss function

a. AdamGD Function

```
class AdamGD():

    def __init__(self, lr, beta1, beta2, epsilon, params):
        self.lr = lr
        self.beta1 = beta1
        self.beta2 = beta2
        self.epsilon = epsilon
        self.params = params

        self.momentum = {}
        self.rmsprop = {}

        for key in self.params:
            self.momentum['vd' + key] = np.zeros(self.params[key].shape)
            self.rmsprop['sd' + key] = np.zeros(self.params[key].shape)

    def update_params(self, grads):

        for key in self.params:
            # Momentum update.
            self.momentum['vd' + key] = (self.beta1 * self.momentum['vd' + key]) + (1 - self.beta1) * grads['d' + key]
            # RMSprop update.
            self.rmsprop['sd' + key] =  (self.beta2 * self.rmsprop['sd' + key]) + (1 - self.beta2) * (grads['d' + key]**2)
            # Update parameters.
            self.params[key] = self.params[key] - (self.lr * self.momentum['vd' + key]) / (np.sqrt(self.rmsprop['sd' + key]) + self.epsilon)

        return self.params
```

b. CrossEntropy Function

```
class CrossEntropyLoss():

    def __init__(self):
        pass

    def get(self, y_pred, y):
        """
            Return the negative log likelihood and the error at the last layer.

            Parameters:
            - y_pred: model predictions.
            - y: ground truth labels.
        """
        loss = -np.sum(y * np.log(y_pred))
        return loss
```

c. Set AdamGD Param and calculate loss

```
cost = CrossEntropyLoss()
lr = 0.0005

optimizer = AdamGD(lr = lr, beta1 = 0.9, beta2 = 0.999, epsilon = 1e-8, params = model.get_params())
```

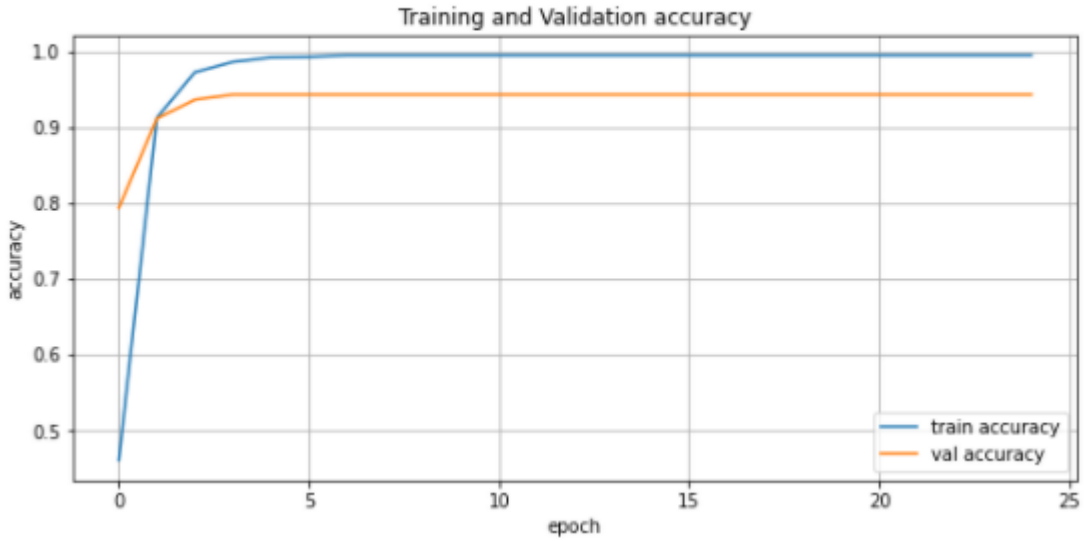
```
for i, (X_batch, y_batch) in zip(pbar, train_loader):

    y_pred = model.forward(X_batch)
    loss = cost.get(y_pred, y_batch)

    grads = model.backward(y_pred, y_batch)
    params = optimizer.update_params(grads)
    model.set_params(params)

    train_loss += loss * BATCH_SIZE
```

```
for i, (X_batch, y_batch) in zip(pbar, val_loader):

    y_pred = model.forward(X_batch)
    loss = cost.get(y_pred, y_batch)

    grads = model.backward(y_pred, y_batch)
    params = optimizer.update_params(grads)
    model.set_params(params)

    val_loss += loss * BATCH_SIZE
```

3. Result and image

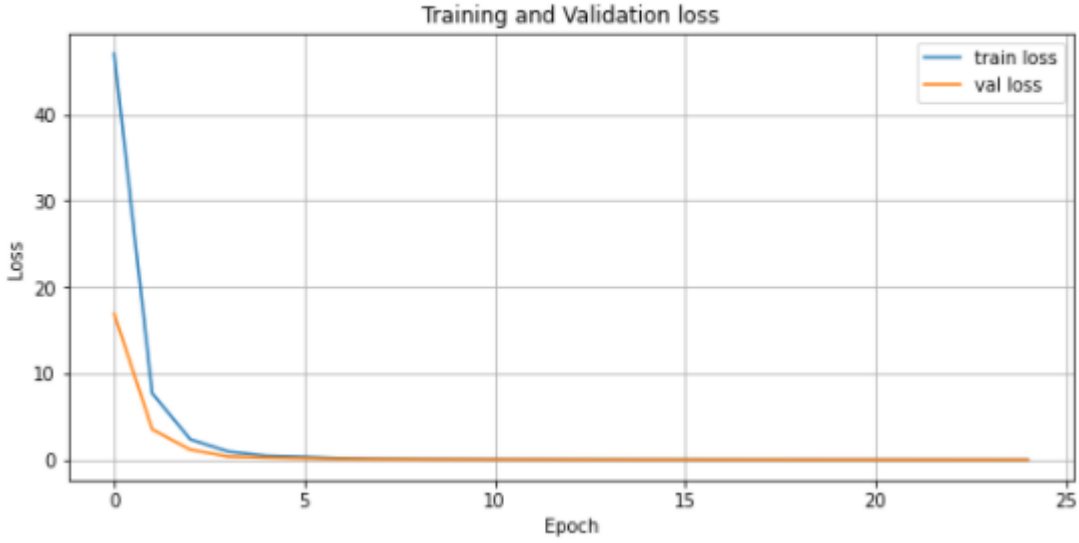
a. Training and Validation Accuracy

```
epoch 24
Train accuracy: 0.9951409135082604
Val accuracy: 0.9433106575963719
```



b. Training and Validation Loss

```
Epoch 24
Train Loss: 0.00905474025332699
Validation Loss: 0.006522690766322416
```

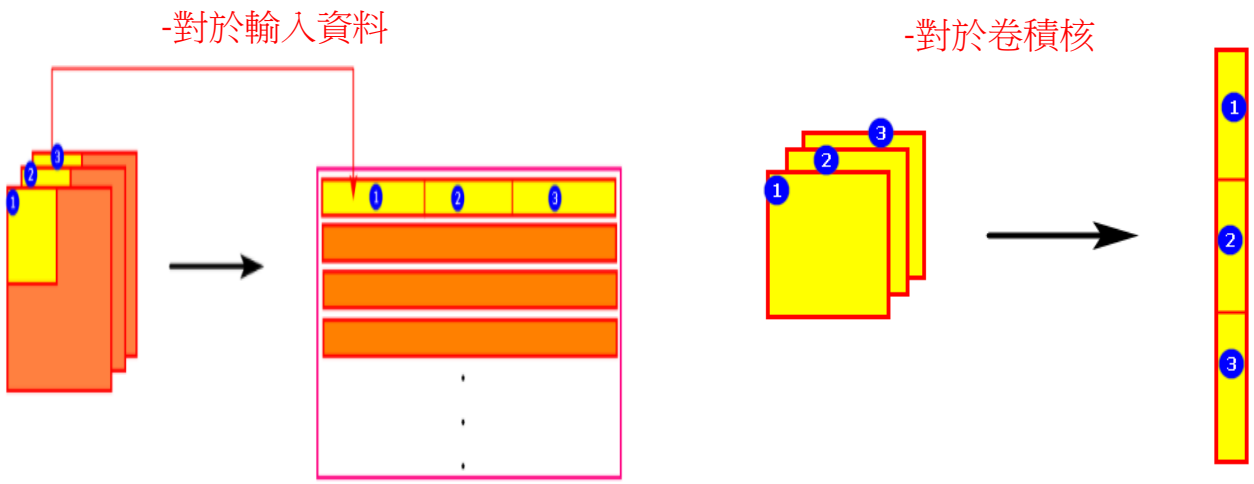


c. Test accuracy and loss

```
test-loss: 1.065898 | test-acc: 0.956
```

4. Describe the major problem you encountered and how did you deal with it.
   a. Problem
      i. 原本在計算 Conv2D 使用 with nested for loops 的方法計算. 一個 epoch 所需要的時間為 4 小時
   b. Solution
      i. 使用網路上所查到的 im2col/col2im 的方法
      ii. im2col:再輸入數據上,根據卷積核大小，將四個通道依次展開為一維數組，然後連接為一個長的一維數組，再根據步幅，將輸入數據中每個應用卷積核的地方都會生成一個一維數組，如下圖所示

```
def im2col(X, HF, WF, stride, pad):
    """
        Transforms our input image into a matrix.

        Parameters:
        - X: input image.
        - HF: filter height.
        - WF: filter width.
        - stride: stride value.
        - pad: padding value.

        Returns:
        -cols: output matrix.
    """
    # Padding
    X_padded = np.pad(X, ((0,0), (0,0), (pad, pad), (pad, pad)), mode='constant')
    i, j, d = get_indices(X.shape, HF, WF, stride, pad)
    # Multi-dimensional arrays indexing.
    cols = X_padded[:, d, i, j]
    cols = np.concatenate(cols, axis=-1)
    return cols
```



輸入資料展開以適合卷積核（權重）

- 輸入資料，將應用卷積核的區域（4 維資料）橫向展開為一行
- 卷積核，縱向展開為 1 列
- 計算乘積即可

iii. colim2>>與 im2col 反方向動作，將原本輸出還原

```
def col2im(dX_col, X_shape, HF, WF, stride, pad):
    """
        Transform our matrix back to the input image.

        Parameters:
        - dX_col: matrix with error.
        - X_shape: input image shape.
        - HF: filter height.
        - WF: filter width.
        - stride: stride value.
        - pad: padding value.

        Returns:
        -x_padded: input image with error.
    """
    # Get input size
    N, D, H, W = X_shape
    # Add padding if needed.
    H_padded, W_padded = H + 2 * pad, W + 2 * pad
    X_padded = np.zeros((N, D, H_padded, W_padded))

    # Index matrices, necessary to transform our input image into a matrix.
    i, j, d = get_indices(X_shape, HF, WF, stride, pad)
    # Retrieve batch dimension by spliting dX_col N times: (X, Y) => (N, X, Y)
    dX_col_reshaped = np.array(np.hsplit(dX_col, N))
    # Reshape our matrix back to image.
    # slice(None) is used to produce the [::] effect which means "for every elements".
    np.add.at(X_padded, (slice(None), d, i, j), dX_col_reshaped)
    # Remove padding from new image if needed.
    if pad == 0:
        return X_padded
    elif type(pad) is int:
        return X_padded[pad:-pad, pad:-pad, :, :]
```

c. Result

i. 將一個 epoch 所需要的時間降為 6 分鐘