binary & bmp file 분석 및 CNN

https://youtu.be/CyZdOlRzibY

한성대학교 HANSUNG UNIVERSITY

CryptoCraft LAB
https://crypto.modoo.at

Contents

binary file (hex file) 분석

binary to bmp

bmp file → CNN

CryptoCraft LAB

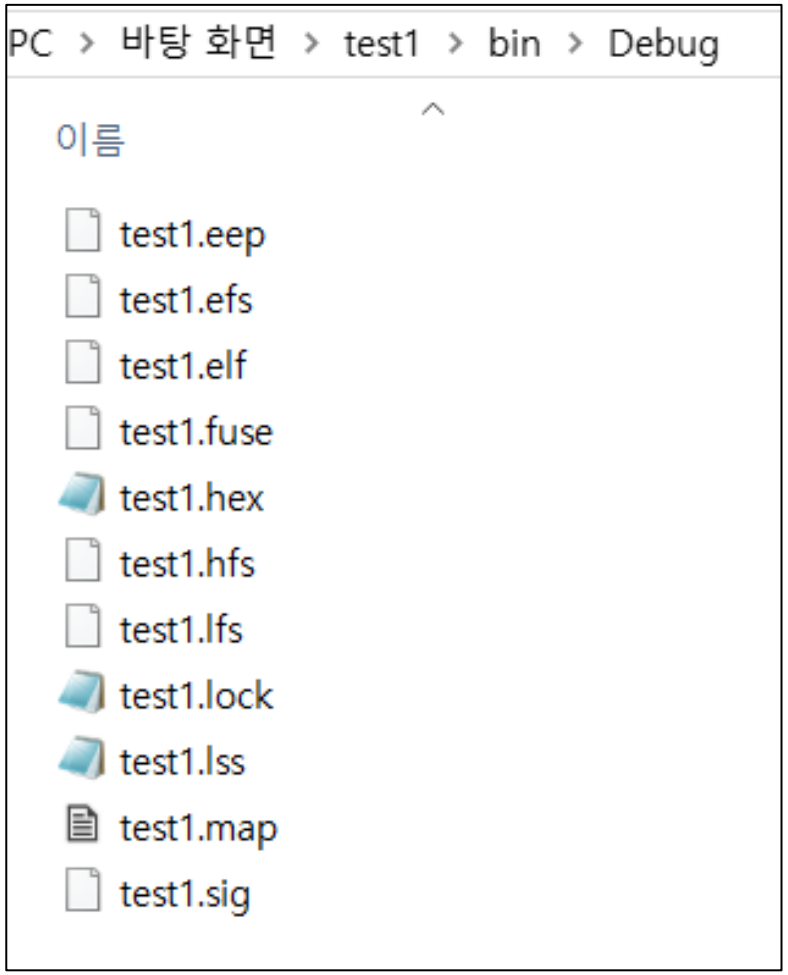
binary file

❖ **c → assembly → opcode → embedded**

- **opcode**
  - ➢ 수행할 어셈블리 명령어를 나타내는 코드
- object file
  - ➢ 컴파일러 또는 어셈블러를 통해 변환된 파일(.o)

❖ 임베디드 시스템의 경우, 빌드된 binary file을 MCU에 다운로드

- .bin .hex파일을 주로 사용



binary file (lss file)

```
0000008c <__ctors_end>:
   8c:   11 24           eor     r1, r1
   8e:   1f be           out     0x3f, r1    ; 63
   90:   cf ef           ldi     r28, 0xFF ; 255
   92:   d0 e1           ldi     r29, 0x10 ; 16
   94:   de bf           out     0x3e, r29 ; 62
   96:   cd bf           out     0x3d, r28 ; 61

00000098 <__do_copy_data>:
   98:   11 e0           ldi     r17, 0x01 ; 1
   9a:   a0 e0           ldi     r26, 0x00 ; 0
   9c:   b1 e0           ldi     r27, 0x01 ; 1
   9e:   e0 e1           ldi     r30, 0x10 ; 16
   a0:   f1 e0           ldi     r31, 0x01 ; 1
   a2:   00 e0           ldi     r16, 0x00 ; 0
   a4:   0b bf           out     0x3b, r16 ; 59
   a6:   02 c0           rjmp    .+4                 ; 0x
   a8:   07 90           elpm    r0, Z+
   aa:   0d 92           st      X+, r0
   ac:   a0 30           cpi     r26, 0x00 ; 0
   ae:   b1 07           cpc     r27, r17
   b0:   d9 f7           brne    .-10        ; 0xa8 <__do_copy_data+0x10>
   b2:   0e 94 5f 00     call    0xbe        ; 0xbe <main>
   b6:   0c 94 86 00     jmp     0x10c       ; 0x10c <_exit>
```

```
000000be <main>:

#include <avr/io.h>

int main(void)
{
   be:   df 93           push    r29
   c0:   cf 93           push    r28
   c2:   00 d0           rcall   .+0                 ; 0xc4 <main+0x6>
   c4:   00 d0           rcall   .+0                 ; 0xc6 <main+0x8>
   c6:   cd b7           in      r28, 0x3d ; 61
   c8:   de b7           in      r29, 0x3e ; 62

    // Insert code
    int a=13, b=86;
   ca:   8d e0           ldi     r24, 0x0D ; 13
   cc:   90 e0           ldi     r25, 0x00 ; 0
   ce:   9c 83           std     Y+4, r25  ; 0x04
   d0:   8b 83           std     Y+3, r24  ; 0x03
   d2:   86 e5           ldi     r24, 0x56 ; 86
   d4:   90 e0           ldi     r25, 0x00 ; 0
   d6:   9a 83           std     Y+2, r25  ; 0x02
   d8:   89 83           std     Y+1, r24  ; 0x01
```

* .lss

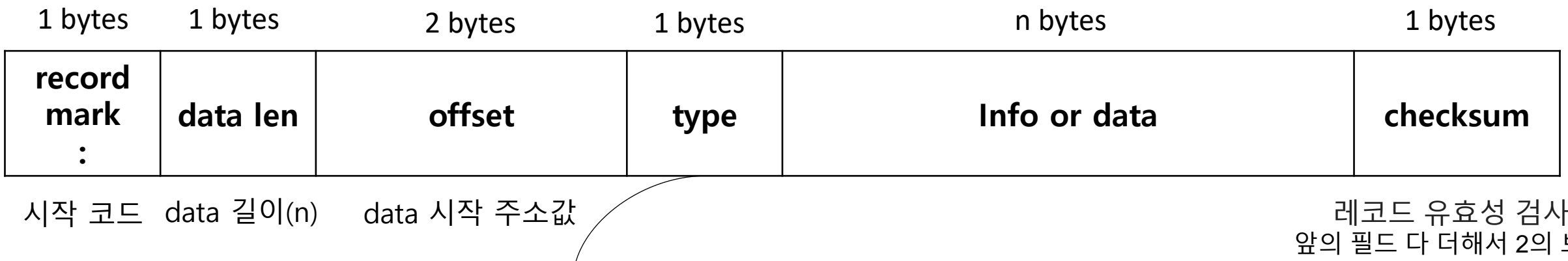
opcode

instruction

.hex

```
:1000E0009C81822793279A83898329813A818B81F6
:1000F0009C81822793279A83898329813A818B81E6
:100100009C81822793279A838983E7CFF894FFCF36
:00000001FF
```

intel hex file format



00 : Data Record

0l : End of File Record >> data field X

02 : Extended Segment Address Record

03 : Start Segment Address Record

04 : Extended Linear Address Record

05 : Start Linear Address Record

- I8 HEX file : 00, 01 레코드만 사용 (16비트 주소지정)
- I16 HEX file : 00 ~ 03 레코드만 사용 (20비트 주소지정)
- I32HEX file : 00, 01, 04, 05 레코드만 사용 (32비트 주소지정)

binary file (lss / hex)

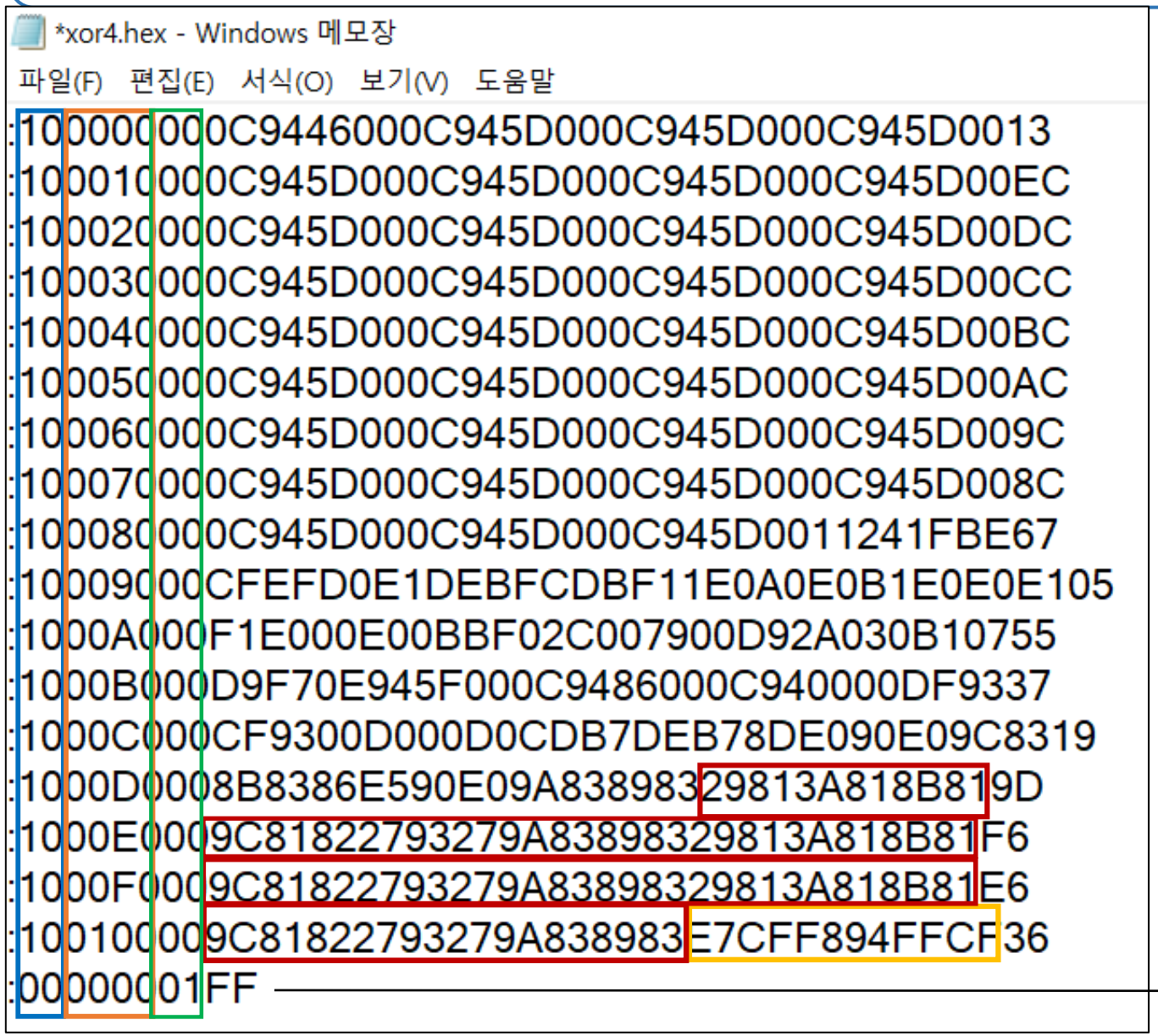
```
    while(1){
        b = a^b;
  da:   29 81     ldd     r18, Y+1   ; 0x01
  dc:   3a 81     ldd     r19, Y+2   ; 0x02
  de:   8b 81     ldd     r24, Y+3   ; 0x03
  e0:   9c 81     ldd     r25, Y+4   ; 0x04
  e2:   82 27     eor     r24, r18
  e4:   93 27     eor     r25, r19
  e6:   9a 83     std     Y+2, r25   ; 0x02
  e8:   89 83     std     Y+1, r24   ; 0x01
        b = a^b;
  ea:   29 81     ldd     r18, Y+1   ; 0x01
  ec:   3a 81     ldd     r19, Y+2   ; 0x02
  ee:   8b 81     ldd     r24, Y+3   ; 0x03
  f0:   9c 81     ldd     r25, Y+4   ; 0x04
  f2:   82 27     eor     r24, r18
  f4:   93 27     eor     r25, r19
  f6:   9a 83     std     Y+2, r25   ; 0x02
  f8:   89 83     std     Y+1, r24   ; 0x01
        b = a^b;
  fa:   29 81     ldd     r18, Y+1   ; 0x01
  fc:   3a 81     ldd     r19, Y+2   ; 0x02
  fe:   8b 81     ldd     r24, Y+3   ; 0x03
 100:   9c 81     ldd     r25, Y+4   ; 0x04
 102:   82 27     eor     r24, r18
 104:   93 27     eor     r25, r19
 106:   9a 83     std     Y+2, r25   ; 0x02
 108:   89 83     std     Y+1, r24   ; 0x01
 10a:   e7 cf     rjmp    .-50       ; 0xda <main+0x1c>

0000010c <_exit>:
 10c:   f8 94     cli

0000010e <__stop_program>:
 10e:   ff cf     rjmp    .-2        ; 0x10e <__stop_program>
```

*xor4.hex - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말

```
:10000000C9446000C945D000C945D000C945D0013
:10001000C945D000C945D000C945D000C945D00EC
:10002000C945D000C945D000C945D000C945D00DC
:10003000C945D000C945D000C945D000C945D00CC
:10004000C945D000C945D000C945D000C945D00BC
:10005000C945D000C945D000C945D000C945D00AC
:10006000C945D000C945D000C945D000C945D009C
:10007000C945D000C945D000C945D000C945D008C
:10008000C945D000C945D000C945D0011241FBE67
:10009000CFEFD0E1DEBFCDBF11E0A0E0B1E0E0E105
:1000A000F1E000E00BBF02C007900D92A030B10755
:1000B000D9F70E945F000C9486000C940000DF9337
:1000C000CF9300D000D0CDB7DEB78DE090E09C8319
:1000D0008B8386E590E09A838983829813A818B819D
:1000E0009C818227932793A8389832981 3A818B81F6
:1000F0009C818227932793A8389832981 3A818B81E6
:1001000 09C818227932793A838983E7CFF894FFCF36
:00000001FF
```

binary file (lss / hex)



: 10 0000 00 0C94 ~ 0013

: 10 0010 00 0C94~ 00EC

파란색 : data len → 16 bytes

주황색 : offset이 0010(16bytes)씩 증가

초록색 : data type → 일반 data

빨간색 : data → opcode

보라색 : checksum

*record의 마지막은 항상 00000001FF

data type 01 → end of file record

binary file(.hex) → bmp

1. hex file → main함수 내부에서 xor연산을 수행한 부분만 추출

```
29813A818B819C81822793279A83898329813A818B819C81822793279A83898329813A818B819C81822793279A838983
```

2. txt to bmp convert code (https://github.com/Hamz-a/txt2bmp) 통해 bmp file로 변환 가능

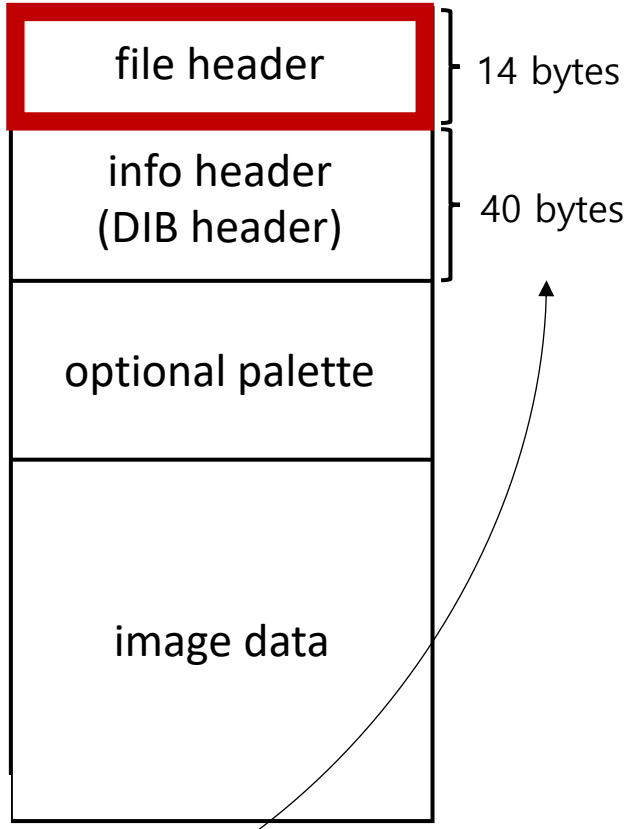
```
C:₩Users₩CryptoCraftLab₩dl₩data₩binarydata>python txt2bmp.py -f test1.txt -b testbmp.bmp
Saved as bitmap in: testbmp.bmp
```

bmp file format

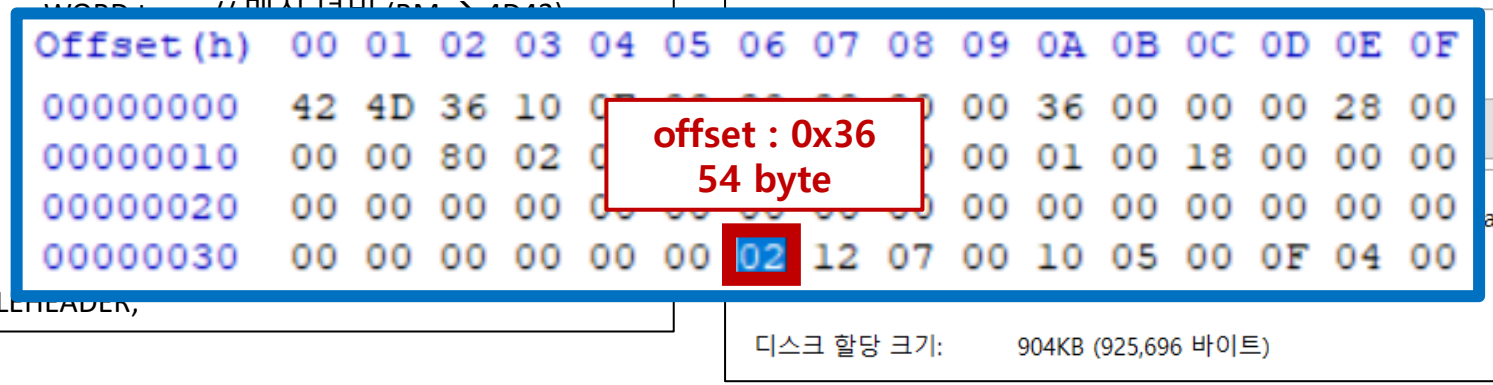
❖ **BMP file에 대한 일반적 정보 (14 bytes)**

- 매직 넘버 : 42 4D (BMP 파일임을 식별) : 2 bytes
- 파일 전체 크기(헤더 포함) → 리틀 엔디언 표기 : 4 bytes
- 예약된 값 → 사용 x → 00 00 : 4 bytes
- 데이터 시작 위치 (offset) → 리틀 엔디언 표기 : 4 bytes

14 bytes




```
typedef struct tagBITMAPFILEHEADER {
    WORD    bfType;      // 매직 넘버 (BM → 4D42)
    DWORD   bfSize;      // 파일 전체 크기
    WORD    bfReserved1; // 예약된 값
    WORD    bfReserved2; // 예약된 값
    DWORD   bfOffBits;   // 데이터 시작 위치 (offset)
} BITMAPFILEHEADER;
```




```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000   42 4D 36 10 0E 00 00 00 00 00 36 00 00 00 28 00
00000010   00 00 80 02 00 00 E0 01 00 00 01 00 18 00 00 00
00000020   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000030   00 00 00 00 00 00 02 12 07 00 10 05 00 0F 04 00
```

offset : 0x36
54 byte

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000   42 4D 36 10 0E 00 00 00 00 00 36 00 00 00 28 00   BM6.....6...(.
```

00 0E 10 36 → 921654

00 00 00 36 → 54 → offset 0x36부터 data 시작

bmp file format

❖ **data에 대한 구체적인 정보 (40 bytes)**

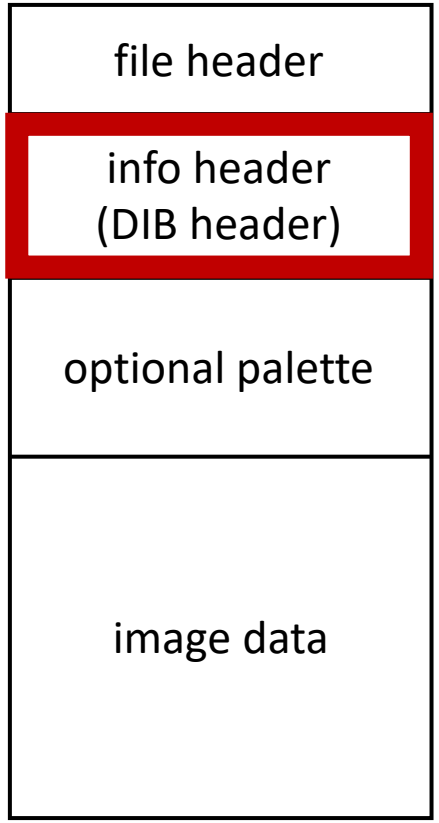
- offset 0x0E부터 시작
- 아래 구조체와 같은 정보들을 담고 있음

```
typedef struct tagBITMAPINFOHEADER {
    DWORD       biSize;          // info header size
    LONG        biWidth;         // image width
    LONG        biHeight;        // image height
    WORD        biplanes;        // color plane의 수(항상 1)
    WORD        biBitCount;      // bit / pixel (컬러, 흑백 구별)
    DWORD       biCompression;   // 압축유무
    DWORD
    LONG
    LONG
    DWORD
    DWORD
} BITMAPINFOHEADER;
```

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000   42 4D 36 10 0E 00 00 00 00 00 36 00 00 00 28 00
00000010   00 00 80 02 00 00 E0 01 00 00 01 00 18 00 00 00
00000020   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000030   00 00 00 00 00 00 02 12 07 00 10 05 00 0F 04 00
```

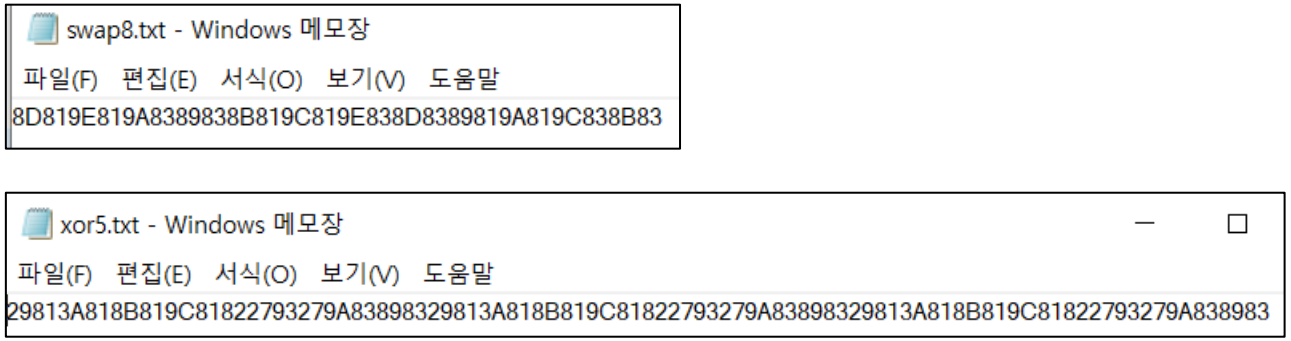
0x28 → 40 bytes
(info header size)

offset : 0x36
data 시작

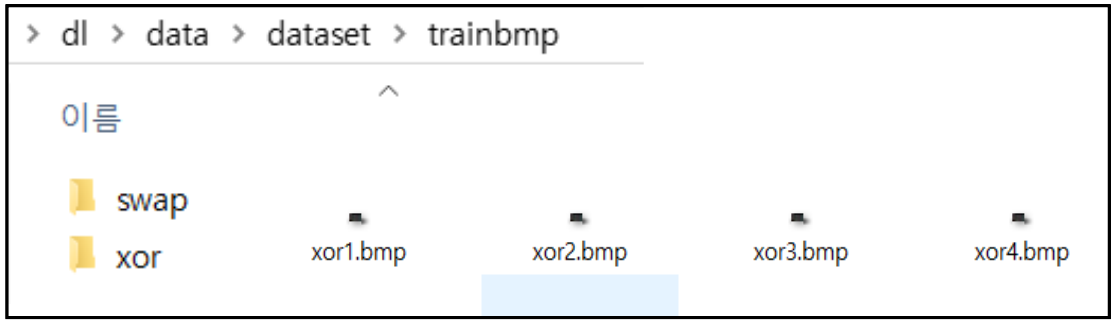


bmp → convolutional neural network

- binary → bmp 과정 반복하여 dataset 만든 후 CNN 수행
  - 간단한 연산으로 분류 실험 (xor, swap)
  - 연산에 해당하는 부분의 opcode만



연산 부분의 opcode



train set of data set

bmp → convolutional neural network

- 이미 학습된 가중치를 가져와서 **전이학습**
- 그 뒤에 **layer 3개 추가**하여 사용
- class 2개로만 실험 → **sigmoid**
- **optimizer = RMSprop**

```
from tensorflow.keras.optimizers import RMSprop


x = layers.Flatten()(last_output)

x = layers.Dense(1024, activation = 'relu')(x)

x = layers.Dense(1, activation = 'sigmoid')(x)

model = Model( pre_trained_model.input, x)

model.compile(optimizer=RMSprop(lr=0.001),
              loss = 'binary_crossentropy',
              metrics = ['acc'])
model.summary()
```

```
import os

from tensorflow.keras import layers
from tensorflow.keras import Model
!wget --no-check-certificate ₩
    https://storage.googleapis.com/mledu-datasets/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5 ₩
    -O /tmp/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5

from tensorflow.keras.applications.inception_v3 import InceptionV3

# 1.
local_weights_file = '/tmp/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5'
pre_trained_model = InceptionV3(input_shape = (150, 150, 3),
                                include_top = False,
                                weights = None)
pre_trained_model.load_weights(local_weights_file)

for layer in pre_trained_model.layers:
  layer.trainable = False

# Uncomment the code below to see the (huge) model summary
pre_trained_model.summary()

# 2.
last_layer = pre_trained_model.get_layer('mixed7')
print('last layer output shape: ', last_layer.output_shape)
last_output = last_layer.output
```

가져온 모델이 훈련하면서 **학습한 parameter를 잠금 (layer.trainable = False)**
해당 파라미터는 내가 사용할 데이터로 갈아줘도 훈련이 되지 않고 그냥 사용

bmp → convolutional neural network

- **데이터 구성 (train / validation / test)**

  colab 사용 → 아래 코드 통해서 구글 드라이브 경로로

```
from google.colab import drive

drive.mount('/content/gdrive')
```

- **train set을 나눠서 validation set**으로 사용
- ImageDataGenerator

  → 이미지 전처리위한 클래스 (이미지 변형 및 정규화)

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

import os

# Define our example directories and files
base_dir = '/content/gdrive/My Drive/dataset'

train_dir = os.path.join( base_dir, 'trainbmp')
validation_dir = os.path.join( base_dir, 'valbmp')
test_dir = os.path.join( base_dir, 'testbmp')

train_xor_dir = os.path.join(train_dir, 'xor')
train_swap_dir = os.path.join(train_dir, 'swap')
validation_xor_dir = os.path.join(validation_dir, 'xor')
validation_swap_dir = os.path.join(validation_dir, 'swap')
test_xor_dir = os.path.join(test_dir, 'xor')
test_swap_dir = os.path.join(test_dir, 'swap')

train_xor_fnames = os.listdir(train_xor_dir)
train_swap_fnames = os.listdir(train_swap_dir)


train_datagen = ImageDataGenerator(rescale = 1./255.)

test_datagen = ImageDataGenerator( rescale = 1.0/255. )


train_generator = train_datagen.flow_from_directory(train_dir,
                                                    batch_size = 20,
                                                    class_mode = 'binary',
                                                    target_size = (150, 150))


validation_generator =  test_datagen.flow_from_directory( validation_dir,
                                                          batch_size  = 20,
                                                          class_mode  = 'binary',
                                                          target_size = (150, 150))
```

bmp → convolutional neural network

- **학습 진행**

  1epoch당 50번씩 전체 데이터를 총 20번 학습

```
Epoch 1/20
100/100 - 4s
Epoch 2/20
100/100 - 3s
```

```
Epoch 19/20
100/100 - 3s
Epoch 20/20
100/100 - 3s
```

- train set / validation set 학습

```
history = model.fit_generator(
            train_generator,
            validation_data = validation_generator,
            steps_per_epoch = 50,
            epochs = 20,
            validation_steps = 50,
            verbose = 2)
```

CNN 세미나에 좀 더 자세히..나와있습니다..
https://youtu.be/HFAHK5orJjg

bmp → convolutional neural networ

- **데이터 변형 x & dropout x**(for 오버피팅 방지) 모델
- **성능**

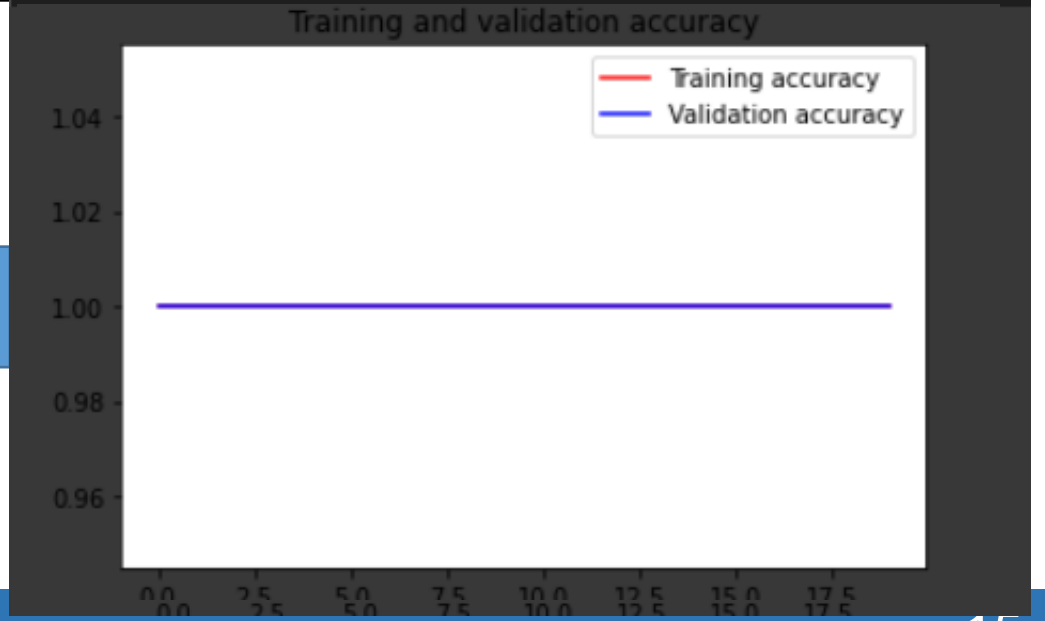
  train & validation set의 정확도는 100센트 달성

  그러나 **데이터가 매우 적고 loss가 줄어들지 않음**
- 간단한 연산, opcode 추출

  → 바이너리 파일의 내용이 거의 비슷해서 정확도가 높게 나온 것 같음

```
Epoch 20/20
50/50 - 2s - loss: 6.3331e-13 - acc: 1.0000 - val_loss: 1.4428e-12 - val_acc: 1.0000
```

```
import matplotlib.pyplot as plt
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'r', label='Training accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend(loc=0)
plt.figure()


plt.show()
```



bmp → convolutional neural network

- **학습에 사용하지 않은 test data으로 예측**

```
output = model.predict_generator(test_generator,steps=5)
print(test_generator.class_indices)
print(output)
```

```
{'swap': 0, 'xor': 1}
[[1.]
 [1.]
 [1.]
 [1.]
 [1.]]
```

→ xor 파일 test 결과 1.00 으로 예측

감사합니다