프로젝트 최종

2024.06.21

8조
201920907 박해웅
201920871 윤상훈

Neural network 프로그램 분석

Mnist.ino

```
tflite::InitializeTarget();

model = tflite::GetModel(g_person_detect_model_data);
if (model->version() != TFLITE_SCHEMA_VERSION) {
  MicroPrintf(
    "Model provided is schema version %d not equal "
    "to supported version %d.",
    model->version(), TFLITE_SCHEMA_VERSION);
  return;
}

static tflite::MicroMutableOpResolver<10> micro_op_resolver;
micro_op_resolver.AddShape();
micro_op_resolver.AddStridedSlice();
micro_op_resolver.AddPack();
micro_op_resolver.AddMaxPool2D();
micro_op_resolver.AddFullyConnected();
micro_op_resolver.AddAveragePool2D();
micro_op_resolver.AddConv2D();
micro_op_resolver.AddDepthwiseConv2D();
micro_op_resolver.AddReshape();
micro_op_resolver.AddSoftmax();

static tflite::MicroInterpreter static_interpreter(
  model, micro_op_resolver, tensor_arena, kTensorArenaSize, nullptr,  &micro_profiler);
interpreter = &static_interpreter;

TfLiteStatus allocate_status = interpreter->AllocateTensors();
if (allocate_status != kTfLiteOk) {
  MicroPrintf("AllocateTensors() failed");
  return;
}
input = interpreter->input(0);
```

System_setup.cpp

```
void InitializeTarget() {
  DEBUG_SERIAL_OBJECT.begin(9600);
  ulong start_time = millis();
  while (!DEBUG_SERIAL_OBJECT) {
    // allow for Arduino IDE Serial Monitor synchronization
    if (millis() - start_time > kSerialMaxInitWait) {
      break;
    }
  }
}
```

- system_setup.cpp 파일의 InitializeTarget() 함수를 통해 tensorflow lite micro의 시리얼 통신을 설정
- 지정된 시간 내에 포트가 준비되지 않으면 초기화 중단
- model 변수에 모델 저장

Neural network 프로그램 분석

All_ops_resolver.cpp

```cpp
AllOpsResolver::AllOpsResolver() {
  // Please keep this list of Builtin Operators in alphabetical order.
  AddAbs();
  AddAdd();
  AddAddN();
  AddArgMax();
  AddArgMin();
  AddAssignVariable();
  AddAveragePool2D();
  AddBatchToSpaceNd();
  AddBroadcastArgs();
  AddBroadcastTo();
  AddCallOnce();
  AddCast();
  AddCeil();
  AddCircularBuffer();
  AddConcatenation();
  AddConv2D();
  AddCos();
  AddCumSum();
  AddDepthToSpace();
  AddDepthwiseConv2D();
  AddDequantize();
  AddDetectionPostprocess();
  AddDiv();
  AddElu();
  AddEqual();
  AddEthosU();
  AddExp();
```

micro_op_resolver.h

```cpp
MicroInterpreter(const Model* model, const MicroOpResolver& op_resolver,
                 uint8_t* tensor_arena, size_t tensor_arena_size,
                 MicroResourceVariables* resource_variables = nullptr,
                 MicroProfilerInterface* profiler = nullptr);

// Create an interpreter instance using an existing MicroAllocator instance.
// This constructor should be used when creating an allocator that needs to
// have allocation handled in more than one interpreter or for recording
// allocations inside the interpreter. The lifetime of the allocator must be
// as long as that of the interpreter object.
MicroInterpreter(const Model* model, const MicroOpResolver& op_resolver,
                 MicroAllocator* allocator,
                 MicroResourceVariables* resource_variables = nullptr,
                 MicroProfilerInterface* profiler = nullptr);

~MicroInterpreter();
```

- micro_op_resolver.h 파일을 통해 MicroInterpreter의 생성자 인자 확인
- profiler 인자를 확인하여 profiler 추가하여 실행
- op_resolver 인자에 사용할 오퍼레이터 전달
- all_ops_resolver.cpp 파일에서 오퍼레이터 확인 가능

Neural network 프로그램 분석

mnist.ino

```
float x_test[kInputTensorSize] = { 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ···
for (int i = 0; i < kInputTensorSize; i++) {
  input->data.f[i] = x_test[i];
}

uint32_t event_handle = micro_profiler.BeginEvent("Invoke");
// Run the model on this input and make sure it succeeds.
unsigned long start_time = micros();  // 코드 실행 시작 시간 기록
if (kTfLiteOk != interpreter->Invoke()) {
  MicroPrintf("Invoke failed.");
}
unsigned long end_time = micros();  // 코드 실행 종료 시간 기록
micro_profiler.EndEvent(event_handle);

TfLiteTensor *output = interpreter->output(0);
```

micro_profiler.cpp

```
uint32_t MicroProfiler::BeginEvent(const char* tag) {
  if (num_events_ == kMaxEvents) {
    num_events_ = 0;
  }

  tags_[num_events_] = tag;
  start_ticks_[num_events_] = GetCurrentTimeTicks();
  end_ticks_[num_events_] = start_ticks_[num_events_] - 1;
  return num_events_++;
}

void MicroProfiler::EndEvent(uint32_t event_handle) {
  TFLITE_DCHECK(event_handle < kMaxEvents);
  end_ticks_[event_handle] = GetCurrentTimeTicks();
}
```

micro_profiler.cpp

```
void MicroProfiler::Log() const {
#if !defined(TF_LITE_STRIP_ERROR_STRINGS)
  for (int i = 0; i < num_events_; ++i) {
    uint32_t ticks = end_ticks_[i] - start_ticks_[i];
    MicroPrintf("%s took %" PRIu32 " ticks (%d ms).", tags_[i], ticks,
                TicksToMs(ticks));
  }
#endif
}
```

micro_profiler.log

```
Invoke took 1u ticks (89293 ms).
SHAPE took 1u ticks (15 ms).
STRIDED_SLICE took 1u ticks (35 ms).
PACK took 1u ticks (20 ms).
RESHAPE took 1u ticks (42 ms).
CONV_2D took 1u ticks (78929 ms).
MAX_POOL_2D took 1u ticks (6228 ms).
RESHAPE took 1u ticks (113 ms).
FULLY_CONNECTED took 1u ticks (3779 ms).
```

- 테스트할 이미지를 넣고 profiler를 사용하여 각 계층별 수행 시간 확인
- 모델의 구조에 대한 정보도 확인 가능

Neural network 프로그램 분석

```
if (kTfLiteOk != interpreter->Invoke())
```

micro_interpreter.cpp

```
TfLiteStatus MicroInterpreter::Invoke() {
  if (initialization_status_ != kTfLiteOk) {
    MicroPrintf("Invoke() called after initialization failed\n");
    return kTfLiteError;
  }

  if (!tensors_allocated_) {
    TF_LITE_ENSURE_OK(&context_, AllocateTensors());
  }

  InvokeSubgraph_num++;
  unsigned long start = micros();
  TfLiteStatus status = graph_.InvokeSubgraph(0);
  unsigned long end = micros();
  InvokeSubgraph_time = end - start;
  return status;
}
```

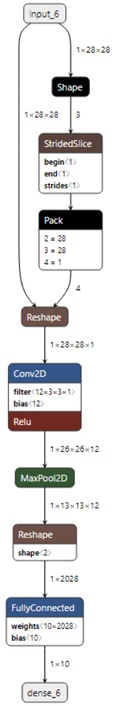
- 추론할 이미지를 저장하고 본격적으로 추론을 수행하는 Invoke() 함수 확인
- micro_interpreter.cpp 함수의 Invoke() 함수 구조 파악
- InvokeSubgraph() 함수에서 각 계층에 대한 연산을 수행하는 것을 확인

Neural network 프로그램 분석

micro_graph.cpp

```cpp
TfLiteStatus MicroGraph::InvokeSubgraph(int subgraph_idx) {
  int previous_subgraph_idx = current_subgraph_index_;
  current_subgraph_index_ = subgraph_idx;

  if (static_cast<size_t>(subgraph_idx) >= subgraphs_->size()) {
    MicroPrintf("Accessing subgraph %d but only %d subgraphs found",
                subgraph_idx, subgraphs_->size());
    return kTfLiteError;
  }
  uint32_t operators_size = NumSubgraphOperators(model_, subgraph_idx);


  for (size_t i = 0; i < operators_size; ++i) {
    TfLiteNode* node =
        &(subgraph_allocations_[subgraph_idx].node_and_registrations[i].node);
    const TfLiteRegistration* registration = subgraph_allocations_[subgraph_idx]
                                                 .node_and_registrations[i]
                                                 .registration;


#if !defined(TF_LITE_STRIP_ERROR_STRINGS)
    ScopedMicroProfiler scoped_profiler(
        OpNameFromRegistration(registration),
        reinterpret_cast<MicroProfilerInterface*>(context_->profiler));
#endif

    TFLITE_DCHECK(registration->invoke);
    TfLiteStatus invoke_status = registration->invoke(context_, node);

    allocator_->ResetTempAllocations();

    if (invoke_status == kTfLiteError) {
      MicroPrintf("Node %s (number %d) failed to invoke with status %d",
                  OpNameFromRegistration(registration), i, invoke_status);
      return kTfLiteError;
    } else if (invoke_status != kTfLiteOk) {
      return invoke_status;
    }
  }
```

```cpp
TFLITE_DCHECK(registration->invoke);
TfLiteStatus invoke_status = registration->invoke(context_, node);
```




```
src
|----tensorflow/lite
|    |----micro
|    |    |----kernels
|    |    |    |----cmsis_nn
|    |    |    |    |----conv.cpp
|    |    |    |    |----fully_connected.cpp
|    |    |    |    |----polling.cpp
|    |    |    |    |          ...
|    |    |    |----shape.cpp
|    |    |    |----strided_slice.cpp
|    |    |    |----reshape.cpp
|    |    |    |----shape.cpp
|    |    |    |          ...
```

서브그래프 연산에 사용되는 파일들의 위치

프로파일링 방법

time_measurements.cpp

```cpp
#include "time_measurements.h"
#include <cstdio>


unsigned long InvokeSubgraph_time;
unsigned long InvokeSubgraph_num;
unsigned long InvokeSubgraph_NumSubgraphOperators_time;
unsigned long InvokeSubgraph_NumSubgraphOperators_num;
unsigned long InvokeSubgraph_for_time;
unsigned long InvokeSubgraph_for_num;
unsigned long InvokeSubgraph_declare_node_time;
unsigned long InvokeSubgraph_declare_node_num;
unsigned long InvokeSubgraph_declare_registration_time;
unsigned long InvokeSubgraph_declare_registration_num;
unsigned long InvokeSubgraph_error_priflier_time;
unsigned long InvokeSubgraph_error_priflier_num;
unsigned long InvokeSubgraph_operation_time[8];
unsigned long InvokeSubgraph_operation_num;
unsigned long InvokeSubgraph_ResetTempAllocations_time;
unsigned long InvokeSubgraph_ResetTempAllocations_num;
unsigned long InvokeSubgraph_kTfLiteError_time;
unsigned long InvokeSubgraph_kTfLiteError_num;



//초기화
void reset_measurements() {
    InvokeSubgraph_time = 0;
    InvokeSubgraph_num = 0;
    InvokeSubgraph_NumSubgraphOperators_time = 0;
    InvokeSubgraph_NumSubgraphOperators_num = 0;
    InvokeSubgraph_for_time = 0;
    InvokeSubgraph_for_num = 0;
    InvokeSubgraph_declare_node_time = 0;
    InvokeSubgraph_declare_node_num = 0;
    InvokeSubgraph_declare_registration_time = 0;
    InvokeSubgraph_declare_registration_num = 0;
    InvokeSubgraph_error_priflier_time = 0;
    InvokeSubgraph_error_priflier_num = 0;
    InvokeSubgraph_operation_time[8] = {0};
    InvokeSubgraph_operation_num = 0;
    InvokeSubgraph_ResetTempAllocations_time = 0;
    InvokeSubgraph_ResetTempAllocations_num = 0;
    InvokeSubgraph_kTfLiteError_time = 0;
    InvokeSubgraph_kTfLiteError_num = 0;
}
```

time_measurements.h

```cpp
#ifndef TIME_MEASUREMENTS_H
#define TIME_MEASUREMENTS_H
#include <Arduino.h>

// 여러 함수의 수행 시간을 저장할 전역 변수 선언
//micro_interpreter.cpp
extern unsigned long InvokeSubgraph_time;
extern unsigned long InvokeSubgraph_num;

extern unsigned long InvokeSubgraph_NumSubgraphOperators_time;
extern unsigned long InvokeSubgraph_NumSubgraphOperators_num;

extern unsigned long InvokeSubgraph_for_time;
extern unsigned long InvokeSubgraph_for_num;

extern unsigned long InvokeSubgraph_declare_node_time;
extern unsigned long InvokeSubgraph_declare_node_num;

extern unsigned long InvokeSubgraph_declare_registration_time;
extern unsigned long InvokeSubgraph_declare_registration_num;

extern unsigned long InvokeSubgraph_error_priflier_time;
extern unsigned long InvokeSubgraph_error_priflier_num;

extern unsigned long InvokeSubgraph_operation_time[8];
extern unsigned long InvokeSubgraph_operation_num;

extern unsigned long InvokeSubgraph_ResetTempAllocations_time;
extern unsigned long InvokeSubgraph_ResetTempAllocations_num;

extern unsigned long InvokeSubgraph_kTfLiteError_time;
extern unsigned long InvokeSubgraph_kTfLiteError_num;



void reset_measurements();



#endif // TIME_MEASUREMENTS_H
```

```cpp
void profile_print(const char* func_name, unsigned long func_time, unsigned long func_num) {
  char buf[80];
  sprintf(buf, "%-37s : %-14lu | %-8lu | %-8lu | %-8lf", func_name, func_time, func_num, func_time*func_num);
  Serial.println(buf);
}
```

-> profile_print 함수 선언

시리얼 포트 출력으로 확인!

프로파일링 결과

main.ino

```
// Run the model on this input and make sure it succeeds.
unsigned long start_time = micros();  //추론 시작
if (kTfLiteOk != interpreter->Invoke()) {
  MicroPrintf("Invoke failed.");
}
unsigned long end_time = micros();  //추론 끝
```

- 프로파일링 할 함수의 앞 뒤에

  Arduino.h 의 micros() 함수 사용

  Invoke 함수 추적 ---------- >

micro_interpreter.cpp

```
TfLiteStatus MicroInterpreter::Invoke() {
  if (initialization_status_ != kTfLiteOk) {
    MicroPrintf("Invoke() called after initialization failed\n");
    return kTfLiteError;
  }

  // Ensure tensors are allocated before the interpreter is invoked to avoid
  // difficult to debug segfaults.
  if (!tensors_allocated_) {
    TF_LITE_ENSURE_OK(&context_, AllocateTensors());
  }

  InvokeSubgraph_num++;
  unsigned long start = micros();
  TfLiteStatus status = graph_.InvokeSubgraph(0);
  unsigned long end = micros();
  InvokeSubgraph_time = end - start;
  return status;
}
```

InvokeSubgraph 추적 !

프로파일링 결과

micro_graph.cpp

```
TfLiteStatus MicroGraph::InvokeSubgraph(int subgraph_idx) { //subgraph_idx = 0
  int previous_subgraph_idx = current_subgraph_index_;  //임시 저장
  current_subgraph_index_ = subgraph_idx;

  if (static_cast<size_t>(subgraph_idx) >= subgraphs_->size()) { //서브그래프 인덱스가 유효
    MicroPrintf("Accessing subgraph %d but only %d subgraphs found",
                subgraph_idx, subgraphs_->size());
    return kTfLiteError;
  }

  InvokeSubgraph_NumSubgraphOperators_num++;
  unsigned long start1 = micros();
  uint32_t operators_size = NumSubgraphOperators(model_, subgraph_idx); //subgraph_idx=0
  unsigned long end1 = micros();
  InvokeSubgraph_NumSubgraphOperators_time = end1 - start1;


  //Serial.println(operators_size);

  unsigned long start2 = micros();
  InvokeSubgraph_for_num++;
  for (size_t i = 0; i < operators_size; ++i) {          //operatiors_size = 8

    unsigned long start3 = micros();
    TfLiteNode* node =                                  //현재 오퍼레이션의 노드 갖고옴
        &(subgraph_allocations_[subgraph_idx].node_and_registrations[i].node);
    unsigned long end3 = micros();
    InvokeSubgraph_declare_node_time = (end3 - start3);
    InvokeSubgraph_declare_node_num++;


    unsigned long start4 = micros();
    const TfLiteRegistration* registration = subgraph_allocations_[subgraph_idx]     //현재
                                                 .node_and_registrations[i]
                                                 .registration;
    unsigned long end4 = micros();
    InvokeSubgraph_declare_registration_time = end4 - start4;
    InvokeSubgraph_declare_registration_num++;

// This ifdef is needed (even though ScopedMicroProfiler itself is a no-op with
// -DTF_LITE_STRIP_ERROR_STRINGS) because the function OpNameFromRegistration is
// only defined for builds with the error strings.
    unsigned long start5 = micros();
#if !defined(TF_LITE_STRIP_ERROR_STRINGS)
    ScopedMicroProfiler scoped_profiler(
        OpNameFromRegistration(registration),
        reinterpret_cast<MicroProfilerInterface*>(context_->profiler));
#endif
    unsigned long end5 = micros();
    InvokeSubgraph_error_priflier_time = end5 - start5;
    InvokeSubgraph_error_priflier_num++;


    unsigned long start6 = micros();
    TFLITE_DCHECK(registration->invoke);
    TfLiteStatus invoke_status = registration->invoke(context_, node);     //오퍼레이션 실행
    unsigned long end6 = micros();
    InvokeSubgraph_operation_time[i] = end6-start6;
    InvokeSubgraph_operation_num++;

    unsigned long start7 = micros();
    allocator_->ResetTempAllocations();     //할당초기화
    unsigned long end7 = micros();
    InvokeSubgraph_ResetTempAllocations_time = end7 - start7;
    InvokeSubgraph_ResetTempAllocations_num++;

    unsigned long start8 = micros();
    if (invoke_status == kTfLiteError) {
      MicroPrintf("Node %s (number %d) failed to invoke with status %d",
                  OpNameFromRegistration(registration), i, invoke_status);
      return kTfLiteError;
    } else if (invoke_status != kTfLiteOk) {
      return invoke_status;
    }
    unsigned long end8 = micros();
    InvokeSubgraph_kTfLiteError_time = end8 - start8;
    InvokeSubgraph_kTfLiteError_num++;
  }
  unsigned long end2 = micros();
  InvokeSubgraph_for_time = end2 - start2;

  current_subgraph_index_ = previous_subgraph_idx;
  return kTfLiteOk;
}
```

프로파일링 결과

출력 시리얼 모니터 ×

Message (Enter to send message to 'Arduino Nano 33 BLE' on 'COM8')

```
19:55:18.943 ->
19:55:18.943 -> ******************************************************************************
19:55:18.943 -> predicated_class : 7
19:55:18.943 -> total time : 89788
19:55:18.943 ->
19:55:18.943 -> ==============================================================================
19:55:18.943 -> function                              : (micros/call)  | call     | time
19:55:18.989 -> ------------------------------------------------------------------------------
19:55:18.989 -> InvokeSubgraph                        : 89770          | 1        | 89770
19:55:18.989 -> InvokeSubgraph_NumSubgraphOperators   : 10             | 1        | 10
19:55:18.989 -> InvokeSubgraph_for                    : 89733          | 1        | 89733
19:55:18.989 -> InvokeSubgraph_declare_node           : 9              | 8        | 72
19:55:18.989 -> InvokeSubgraph_declare                : 8              | 8        | 64
19:55:18.989 -> InvokeSubgraph_error_priflier         : 10             | 8        | 80
19:55:18.989 -> InvokeSubgraph_operation              : 88850          | 1        | 88850
19:55:18.989 ->   >>  InvokeSubgraph_operation[0]     : 12             | 1        | 12
19:55:18.989 ->   >>  InvokeSubgraph_operation[1]     : 50             | 1        | 50
19:55:18.989 ->   >>  InvokeSubgraph_operation[2]     : 19             | 1        | 19
19:55:18.989 ->   >>  InvokeSubgraph_operation[3]     : 41             | 1        | 41
19:55:18.989 ->   >>  InvokeSubgraph_operation[4]     : 78838          | 1        | 78838
19:55:18.989 ->   >>  InvokeSubgraph_operation[5]     : 6224           | 1        | 6224
19:55:18.989 ->   >>  InvokeSubgraph_operation[6]     : 84             | 1        | 84
19:55:18.989 ->   >>  InvokeSubgraph_operation[7]     : 3582           | 1        | 3582
19:55:18.989 -> InvokeSubgraph_ResetTempAllocations   : 37             | 8        | 296
19:55:18.989 -> InvokeSubgraph_kTfLiteError           : 9              | 8        | 72
19:55:18.989 -> ==============================================================================
19:55:18.989 -> ******************************************************************************
```

전체 수행 시간 : 89,788 μs

InvokeSubpraph 함수 : 89,770 μs

-> for 반복문 : 89,733 μs

-> operation 실행 : 88,850 μs

```
unsigned long start6 = micros();
TFLITE_DCHECK(registration->invoke);
TfLiteStatus invoke_status = registration->invoke(context_, node);     //오퍼레이션 실행
unsigned long end6 = micros();
InvokeSubgraph_operation_time[i] = end6-start6;
InvokeSubgraph_operation_num++;
```

프로파일링 결과

micro_graph.cpp

```
for (size_t i = 0; i < operators_size; ++i) {
```

```
unsigned long start6 = micros();
TFLITE_DCHECK(registration->invoke);
TfLiteStatus invoke_status = registration->invoke(context_, node);    //오퍼레이션 실행
unsigned long end6 = micros();
InvokeSubgraph_operation_time[i] = end6-start6;
InvokeSubgraph_operation_num++;
```

```
InvokeSubgraph_operation              : 88850     | 1       | 88850
  >>  InvokeSubgraph_operation[0]     : 12        | 1       | 12
  >>  InvokeSubgraph_operation[1]     : 50        | 1       | 50
  >>  InvokeSubgraph_operation[2]     : 19        | 1       | 19
  >>  InvokeSubgraph_operation[3]     : 41        | 1       | 41
  >>  InvokeSubgraph_operation[4]     : 78838     | 1       | 78838
  >>  InvokeSubgraph_operation[5]     : 6224      | 1       | 6224
  >>  InvokeSubgraph_operation[6]     : 84        | 1       | 84
  >>  InvokeSubgraph_operation[7]     : 3582      | 1       | 3582
```

micro_profiler.log

```
Invoke took lu ticks (89293 ms).
SHAPE took lu ticks (15 ms).
STRIDED_SLICE took lu ticks (35 ms).
PACK took lu ticks (20 ms).
RESHAPE took lu ticks (42 ms).
CONV_2D took lu ticks (78929 ms).
MAX_POOL_2D took lu ticks (6228 ms).
RESHAPE took lu ticks (113 ms).
FULLY_CONNECTED took lu ticks (3779 ms).
```

int8 적용

int8 적용

Int8 tflite -> c array

```
# xxd를 사용할 수 없을 경우, 설치한다.
!apt-get -qq install xxd
# 파일을 C 소스파일로 저장
!xxd -i mnist_model_quant.tflite > model.cc
# 소스파일을 출력
!cat model.cc
```

기존 float형으로 입력하던 데이터

int형으로 변환하여 모델에 전달

mnist.ino

```
// 전처리: 입력 데이터를 signed int8로 양자화
int8_t quantized_input[kInputTensorSize];
preprocess(x_test, quantized_input, kInputTensorSize, input->params.scale, input->params.zero_point);
memcpy(input->data.int8, quantized_input, kInputTensorSize * sizeof(int8_t));

uint32_t event_handle = micro_profiler.BeginEvent("Invoke");
// Run the model on this input and make sure it succeeds.
unsigned long start_time = micros();  // 코드 실행 시작 시간 기록
if (kTfLiteOk != interpreter->Invoke()) {
  MicroPrintf("Invoke failed.");
}
unsigned long end_time = micros();  // 코드 실행 종료 시간 기록
micro_profiler.EndEvent(event_handle);
```

time_measurements.cpp

```
// 전처리 함수: float32를 signed int8로 변환하고 시리얼 모니터에 출력
void preprocess(const float* input_data, int8_t* quantized_data, int size, float scale, int zero_point) {
  for (int i = 0; i < size; ++i) {
    // float32 값을 signed int8로 변환
    int quantized_value = static_cast<int>(input_data[i] / scale + zero_point);
    // signed int8 범위로 클램핑
    if (quantized_value < -128) quantized_value = -128;
    if (quantized_value > 127) quantized_value = 127;
    quantized_data[i] = static_cast<int8_t>(quantized_value);
  }
```

int8 적용

정확도 또한 int로

```
Output probabilities (int8):
Class 0: 15
Class 1: 24
Class 2: 36
Class 3: 56
Class 4: 6
Class 5: 7
Class 6: -50
Class 7: 112
Class 8: 33
Class 9: 37
```

```
****************************************************************************
predicated_class : 7
total time : 29672

============================================================================
function                              : (micros/call)  | call     | time
----------------------------------------------------------------------------
InvokeSubgraph                        : 29654          | 1        | 29654
InvokeSubgraph_NumSubgraphOperators   : 10             | 1        | 10
InvokeSubgraph_for                    : 29616          | 1        | 29616
InvokeSubgraph_declare_node           : 9              | 8        | 72
InvokeSubgraph_declare                : 9              | 8        | 72
InvokeSubgraph_error_priflier         : 19             | 8        | 152
InvokeSubgraph_operation              : 28582          | 1        | 28582
  >>  InvokeSubgraph_operation[0]     : 13             | 1        | 13
  >>  InvokeSubgraph_operation[1]     : 34             | 1        | 34
  >>  InvokeSubgraph_operation[2]     : 19             | 1        | 19
  >>  InvokeSubgraph_operation[3]     : 20             | 1        | 20
  >>  InvokeSubgraph_operation[4]     : 25101          | 1        | 25101
  >>  InvokeSubgraph_operation[5]     : 1767           | 1        | 1767
  >>  InvokeSubgraph_operation[6]     : 31             | 1        | 31
  >>  InvokeSubgraph_operation[7]     : 1597           | 1        | 1597
InvokeSubgraph_ResetTempAllocations   : 10             | 8        | 80
InvokeSubgraph_kTfLiteError           : 9              | 8        | 72
============================================================================
Invoke took 1u ticks (29690 ms).
SHAPE took 1u ticks (77 ms).
STRIDED_SLICE took 1u ticks (98 ms).
PACK took 1u ticks (83 ms).
RESHAPE took 1u ticks (85 ms).
CONV_2D took 1u ticks (25165 ms).
MAX_POOL_2D took 1u ticks (1832 ms).
RESHAPE took 1u ticks (96 ms).
FULLY_CONNECTED took 1u ticks (1661 ms).
****************************************************************************
```

최적화 시작

기존 프로파일링 결과

micro_profiler.log

```
Invoke took 1u ticks (89293 ms).
SHAPE took 1u ticks (15 ms).
STRIDED_SLICE took 1u ticks (35 ms).
PACK took 1u ticks (20 ms).
RESHAPE took 1u ticks (42 ms).
CONV_2D took 1u ticks (78929 ms).
MAX_POOL_2D took 1u ticks (6228 ms).
RESHAPE took 1u ticks (113 ms).
FULLY_CONNECTED took 1u ticks (3779 ms).
```

Int 8 적용

micro_profiler.log

```
Invoke took 1u ticks (29690 ms).
SHAPE took 1u ticks (77 ms).
STRIDED_SLICE took 1u ticks (98 ms).
PACK took 1u ticks (83 ms).
RESHAPE took 1u ticks (85 ms).
CONV_2D took 1u ticks (25165 ms).
MAX_POOL_2D took 1u ticks (1832 ms).
RESHAPE took 1u ticks (96 ms).
FULLY_CONNECTED took 1u ticks (1661 ms).
```

convolution

arm_convolve_wrapper_s8.c

```
else if ((input_dims->n == 1) && (filter_dims->w == 3) && (filter_dims->h == 3) && (conv_params->dilation.w == 1) && (conv_params->dilation.h == 1))
{   //here

    return arm_convolve_3x3_s8(ctx,
                               conv_params,
                               quant_params,
                               input_dims,
                               input_data,
                               filter_dims,
                               filter_data,
                               bias_dims,
                               bias_data,
                               output_dims,
                               output_data);
}
else
{
    return arm_convolve_s8(ctx,
                           conv_params,
                           quant_params,
                           input_dims,
                           input_data,
                           filter_dims,
                           filter_data,
                           bias_dims,
                           bias_data,
                           output_dims,
                           output_data);
}
```

필터 크기 3x3 과 입력 채널 1(흑백 사진)에
맞는 함수 선택 가능하도록
조건 만듦

arm_convolve_3x3_s8

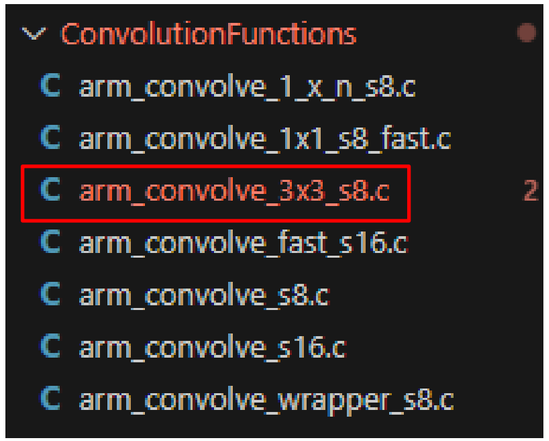
convolution

arm_nn_functions.h

```
arm_cmsis_nn_status arm_convolve_s8(const cmsis_nn_context *ctx,
                                    const cmsis_nn_conv_params *conv_params,
                                    const cmsis_nn_per_channel_quant_params *quant_params,
                                    const cmsis_nn_dims *input_dims,
                                    const q7_t *input_data,
                                    const cmsis_nn_dims *filter_dims,
                                    const q7_t *filter_data,
                                    const cmsis_nn_dims *bias_dims,
                                    const int32_t *bias_data,
                                    const cmsis_nn_dims *output_dims,
                                    q7_t *output_data);

arm_cmsis_nn_status arm_convolve_3x3_s8(const cmsis_nn_context *ctx,
                                    const cmsis_nn_conv_params *conv_params,
                                    const cmsis_nn_per_channel_quant_params *quant_params,
                                    const cmsis_nn_dims *input_dims,
                                    const q7_t *input_data,
                                    const cmsis_nn_dims *filter_dims,
                                    const q7_t *filter_data,
                                    const cmsis_nn_dims *bias_dims,
                                    const int32_t *bias_data,
                                    const cmsis_nn_dims *output_dims,
                                    q7_t *output_data);
```

arm_convolve_3x3_s8 함수 호출을 위해 헤더파일에도 추가

arm_convolve_3x3_s8.c 파일 제작



convolution

arm_convolve_s8.c (기존)

```
for (i_out_y = 0; i_out_y < output_y; i_out_y++)
{
    for (i_out_x = 0; i_out_x < output_x; i_out_x++)
    {
        const int32_t base_idx_y = stride_y * i_out_y - pad_y;
        const int32_t base_idx_x = stride_x * i_out_x - pad_x;

        for (i_ker_y = 0; i_ker_y < kernel_y; i_ker_y++)
        {
            for (i_ker_x = 0; i_ker_x < kernel_x; i_ker_x++)
            {
                const int32_t k_y = base_idx_y + dilation_y * i_ker_y;
                const int32_t k_x = base_idx_x + dilation_x * i_ker_x;

                if (k_y < 0 || k_y >= input_y || k_x < 0 || k_x >= input_x)
                {
                    /* Filling 0 for out-of-bound paddings */
                    memset(two_column_buf, 0, sizeof(q15_t) * input_ch);
                }
                else
                {
                    /* Copying the pixel data to column */
                    arm_q7_to_q15_with_offset(
                        input_data + (k_y * input_x + k_x) * input_ch, two_column_buf, input_ch, input_offset);
                }
                two_column_buf += input_ch;
            }
        }

        /* Computation is filed for every 2 columns */
        if (two_column_buf == buffer_a + 2 * input_ch * kernel_y * kernel_x)
        {
            out = arm_nn_mat_mult_kernel_s8_s16(filter_data,
                                                buffer_a,
                                                output_ch,
                                                output_shift,
                                                output_mult,
                                                out_offset,
                                                out_activation_min,
                                                out_activation_max,
                                                input_ch * kernel_y * kernel_x,
                                                bias_data,
                                                out);

            /* counter reset */
            two_column_buf = buffer_a;
        }
```

im2col (이미지 -> 열) : 2열 버퍼에 저장

```
for (i_out_y = 0; i_out_y < output_y; i_out_y++)
{
    for (i_out_x = 0; i_out_x < output_x; i_out_x++)
    {
        const int32_t base_idx_y = stride_y * i_out_y - pad_y;
        const int32_t base_idx_x = stride_x * i_out_x - pad_x;

        for (i_ker_y = 0; i_ker_y < kernel_y; i_ker_y++)
        {
            for (i_ker_x = 0; i_ker_x < kernel_x; i_ker_x++)
            {
                const int32_t k_y = base_idx_y + dilation_y * i_ker_y;
                const int32_t k_x = base_idx_x + dilation_x * i_ker_x;

                if (k_y < 0 || k_y >= input_y || k_x < 0 || k_x >= input_x)
                {
                    /* Filling 0 for out-of-bound paddings */
                    memset(two_column_buf, 0, sizeof(q15_t) * input_ch);
                }
                else
                {
                    /* Copying the pixel data to column */
                    arm_q7_to_q15_with_offset(
                        input_data + (k_y * input_x + k_x) * input_ch, two_column_buf, input_ch, input_offset);
                }
                two_column_buf += input_ch;
            }
        }
```

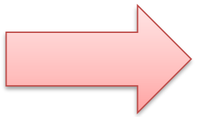
매트릭스 곱셈

```
        /* Computation is filed for every 2 columns */
        if (two_column_buf == buffer_a + 2 * input_ch * kernel_y * kernel_x)
        {
            out = arm_nn_mat_mult_kernel_s8_s16(filter_data,
                                                buffer_a,
                                                output_ch,
                                                output_shift,
                                                output_mult,
                                                out_offset,
                                                out_activation_min,
                                                out_activation_max,
                                                input_ch * kernel_y * kernel_x,
                                                bias_data,
                                                out);

            /* counter reset */
            two_column_buf = buffer_a;
        }
```

convolution

arm_convolve_s8.c (상단)

```
for (i_out_y = 0; i_out_y < output_y; i_out_y++)
{
    for (i_out_x = 0; i_out_x < output_x; i_out_x++)
    {
        const int32_t base_idx_y = stride_y * i_out_y - pad_y;
        const int32_t base_idx_x = stride_x * i_out_x - pad_x;

        for (i_ker_y = 0; i_ker_y < kernel_y; i_ker_y++)
        {
            for (i_ker_x = 0; i_ker_x < kernel_x; i_ker_x++)
            {
                const int32_t k_y = base_idx_y + dilation_y * i_ker_y;
                const int32_t k_x = base_idx_x + dilation_x * i_ker_x;

                if (k_y < 0 || k_y >= input_y || k_x < 0 || k_x >= input_x)
                {
                    /* Filling 0 for out-of-bound paddings */
                    memset(two_column_buf, 0, sizeof(q15_t) * input_ch);
                }
                else
                {
                    /* Copying the pixel data to column */
                    arm_q7_to_q15_with_offset(
                        input_data + (k_y * input_x + k_x) * input_ch, two_column_buf, input_ch, input_offset);
                }
                two_column_buf += input_ch;
            }
        }
```



arm_convolve_3x3_s8.c (상단 1)

```
for (i_out_y = 0; i_out_y < output_y; i_out_y++) // 26
{
    for (i_out_x = 0; i_out_x < output_x; i_out_x++) // 26
    {
        // 현재 필터의 시작위치
        const int32_t base_idx_y = stride_y * i_out_y - pad_y;
        const int32_t base_idx_x = stride_x * i_out_x - pad_x;

        // 3*3 필터 크기만큼 반복    //
        const int32_t k_y1 = base_idx_y;
        const int32_t k_y2 = k_y1 + dilation_y;
        const int32_t k_y3 = k_y2 + dilation_y;

        const int32_t k_x1 = base_idx_x;
        const int32_t k_x2 = k_x1 + dilation_x;
        const int32_t k_x3 = k_x2 + dilation_x;

        const q7_t *src_ptrs[9] = {
            input_data + (k_y1 * input_x + k_x1),
            input_data + (k_y1 * input_x + k_x2),
            input_data + (k_y1 * input_x + k_x3),
            input_data + (k_y2 * input_x + k_x1),
            input_data + (k_y2 * input_x + k_x2),
            input_data + (k_y2 * input_x + k_x3),
            input_data + (k_y3 * input_x + k_x1),
            input_data + (k_y3 * input_x + k_x2),
            input_data + (k_y3 * input_x + k_x3)};

        q15_t *dst_ptr = two_column_buf;
```

3x3 필터 루프 전부 언롤링

루프 언롤링에 필요한 변수 추가 선언

convolution

arm_convolve_s8.c (상단)

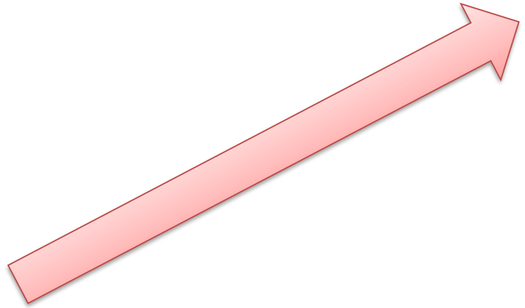
```
else
{
    /* Copying the pixel data to column */
    arm_q7_to_q15_with_offset(
        input_data + (k_y * input_x + k_x) * input_
}
```

arm_q7_to_15_with_offset.c

```
while (block_cnt > 0)
{
    *dst++ = (q15_t)*src++ + offset;

    // Decrement the loop counter
    block_cnt--;
}
```

```
*dst++ = (q15_t)*src++ + offset;
```

1번 반복



arm_convolve_3x3_s8.c (상단 2)

```
q15_t *dst_ptr = two_column_buf;

q7_t src1 = (*src_ptrs[0]) & mask1;
q7_t src2 = (*src_ptrs[1]) & mask1;
q7_t src3 = (*src_ptrs[2]) & mask1;
q7_t src4 = (*src_ptrs[3]) & mask1;
q7_t src5 = (*src_ptrs[4]) & mask1;
q7_t src6 = (*src_ptrs[5]) & mask1;
q7_t src7 = (*src_ptrs[6]) & mask1;
q7_t src8 = (*src_ptrs[7]) & mask1;

uint32_t val1 = src1 + (src2 << 16);
uint32_t val2 = src3 + (src4 << 16);
uint32_t val3 = src5 + (src6 << 16);
uint32_t val4 = src7 + (src8 << 16);

result1 = __UQADD16(val1, summand);
result2 = __UQADD16(val2, summand);
result3 = __UQADD16(val3, summand);
result4 = __UQADD16(val4, summand);

*(dst_ptr++) = result1 & mask2;
*(dst_ptr++) = (result1 >> 16) & mask2;
*(dst_ptr++) = result2 & mask2;
*(dst_ptr++) = (result2 >> 16) & mask2;
*(dst_ptr++) = result3 & mask2;
*(dst_ptr++) = (result3 >> 16) & mask2;
*(dst_ptr++) = result4 & mask2;
*(dst_ptr++) = (result4 >> 16) & mask2;

// 남은 하나의 데이터 처리
*dst_ptr++ = (q15_t)(*src_ptrs[8]++) + input_offset;

two_column_buf += 9;

uint16_t num_col_a = input_ch * kernel_y * kernel_x; // 9
```

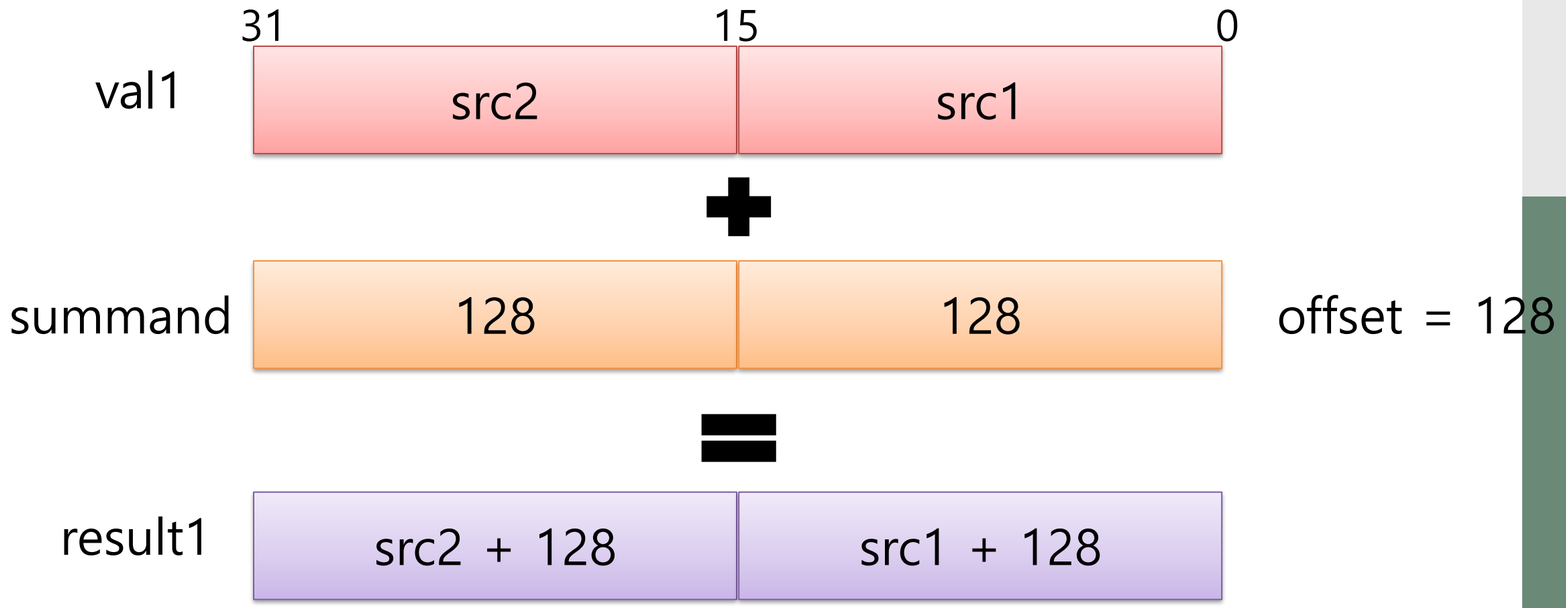
convolution

arm_convolve_3x3_s8.c (상단 2)

```
q15_t *dst_ptr = two_column_buf;

q7_t src1 = (*src_ptrs[0]) & mask1;
q7_t src2 = (*src_ptrs[1]) & mask1;
q7_t src3 = (*src_ptrs[2]) & mask1;
q7_t src4 = (*src_ptrs[3]) & mask1;
q7_t src5 = (*src_ptrs[4]) & mask1;
q7_t src6 = (*src_ptrs[5]) & mask1;
q7_t src7 = (*src_ptrs[6]) & mask1;
q7_t src8 = (*src_ptrs[7]) & mask1;

uint32_t val1 = src1 + (src2 << 16);
uint32_t val2 = src3 + (src4 << 16);
uint32_t val3 = src5 + (src6 << 16);
uint32_t val4 = src7 + (src8 << 16);

result1 = __UQADD16(val1, summand);
result2 = __UQADD16(val2, summand);
result3 = __UQADD16(val3, summand);
result4 = __UQADD16(val4, summand);

*(dst_ptr++) = result1 & mask2;
*(dst_ptr++) = (result1 >> 16) & mask2;
*(dst_ptr++) = result2 & mask2;
*(dst_ptr++) = (result2 >> 16) & mask2;
*(dst_ptr++) = result3 & mask2;
*(dst_ptr++) = (result3 >> 16) & mask2;
*(dst_ptr++) = result4 & mask2;
*(dst_ptr++) = (result4 >> 16) & mask2;

// 남은 하나의 데이터 처리
*dst_ptr++ = (q15_t)(*src_ptrs[8]++) + input_offset;

two_column_buf += 9;

uint16_t num_col_a = input_ch * kernel_y * kernel_x; // 9
```

__UQADD16

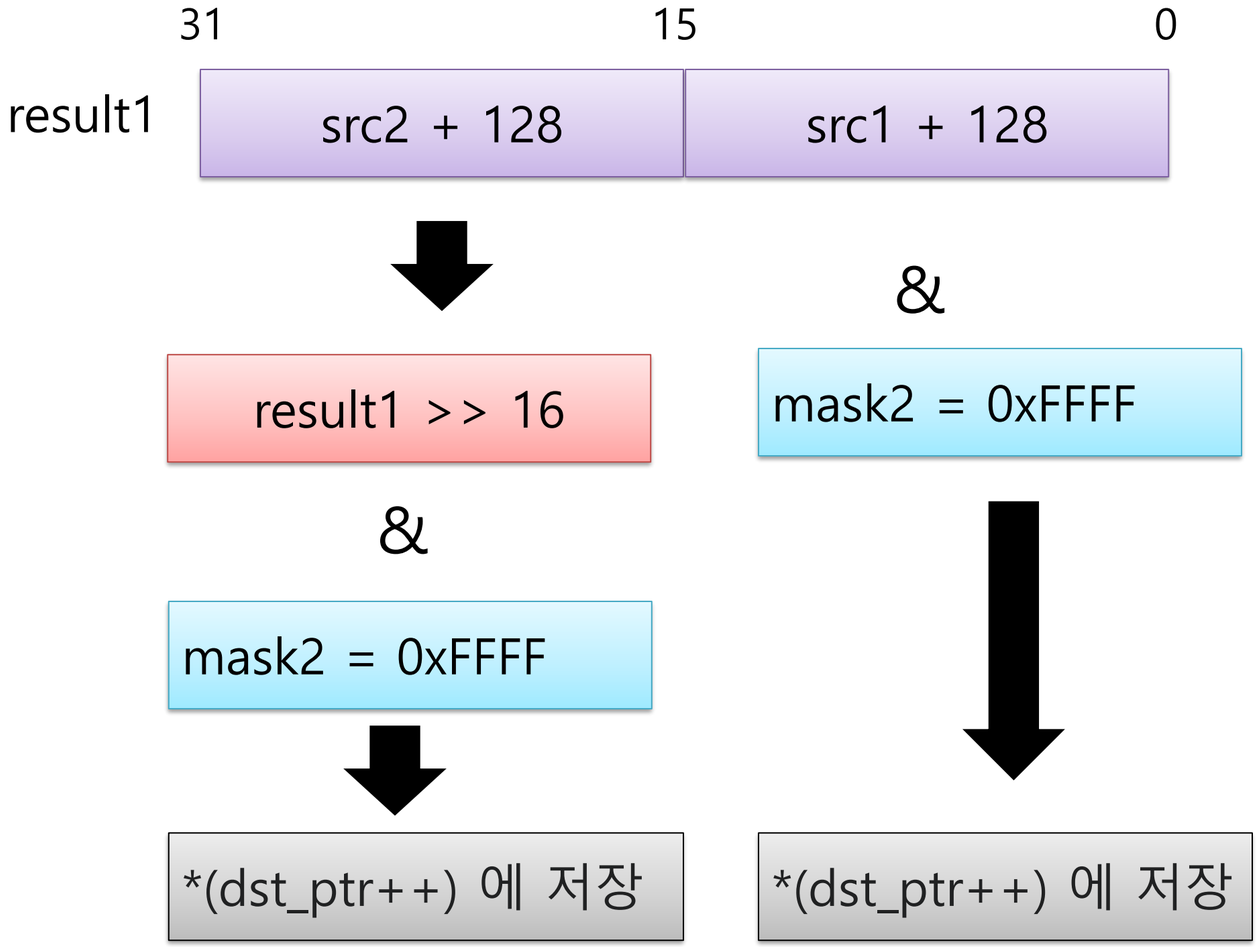
```
*dst++ = (q15_t)*src++ + offset;
```



convolution

arm_convolve_3x3_s8.c (상단 2)

```
q15_t *dst_ptr = two_column_buf;

q7_t src1 = (*src_ptrs[0]) & mask1;
q7_t src2 = (*src_ptrs[1]) & mask1;
q7_t src3 = (*src_ptrs[2]) & mask1;
q7_t src4 = (*src_ptrs[3]) & mask1;
q7_t src5 = (*src_ptrs[4]) & mask1;
q7_t src6 = (*src_ptrs[5]) & mask1;
q7_t src7 = (*src_ptrs[6]) & mask1;
q7_t src8 = (*src_ptrs[7]) & mask1;

uint32_t val1 = src1 + (src2 << 16);
uint32_t val2 = src3 + (src4 << 16);
uint32_t val3 = src5 + (src6 << 16);
uint32_t val4 = src7 + (src8 << 16);

result1 = __UQADD16(val1, summand);
result2 = __UQADD16(val2, summand);
result3 = __UQADD16(val3, summand);
result4 = __UQADD16(val4, summand);

*(dst_ptr++) = result1 & mask2;
*(dst_ptr++) = (result1 >> 16) & mask2;
*(dst_ptr++) = result2 & mask2;
*(dst_ptr++) = (result2 >> 16) & mask2;
*(dst_ptr++) = result3 & mask2;
*(dst_ptr++) = (result3 >> 16) & mask2;
*(dst_ptr++) = result4 & mask2;
*(dst_ptr++) = (result4 >> 16) & mask2;

// 남은 하나의 데이터 처리
*dst_ptr++ = (q15_t)(*src_ptrs[8]++) + input_offset;

two_column_buf += 9;

uint16_t num_col_a = input_ch * kernel_y * kernel_x; // 9
```



arm_convolve_s8.c (하단)

```
/* Computation is filed for every 2 columns */
if (two_column_buf == buffer_a + 2 * input_ch * kernel_y * kernel_x)
{
    out = arm_nn_mat_mult_kernel_s8_s16(filter_data,
                                        buffer_a,
                                        output_ch,
                                        output_shift,
                                        output_mult,
                                        out_offset,
                                        out_activation_min,
                                        out_activation_max,
                                        input_ch * kernel_y * kernel_x
                                        bias_data,
                                        out);

    /* counter reset */
    two_column_buf = buffer_a;
}
```

arm_nn_mult_kernel_s8_s16.c (계산)

```
uint16_t col_count = num_col_a;

while (col_count)
{
    q7_t a0 = *ip_a0++;
    q15_t b0 = *ip_b0++;
    q7_t a1 = *ip_a1++;
    q15_t b1 = *ip_b1++;

    ch_0_out_0 += a0 * b0;
    ch_0_out_1 += a0 * b1;
    ch_1_out_0 += a1 * b0;
    ch_1_out_1 += a1 * b1;
    col_count--;
} /* while over col_count */
```

원래 9번 반복

루프언롤링 : 8

arm_nn_mult_kernel_s8_s16.c

```
// 첫 번째 4개 데이터 처리
b0 = arm_nn_read_q15x2_ia(&ip_b0);
b1 = arm_nn_read_q15x2_ia(&ip_b1);

ip_a0 = read_and_pad(ip_a0, &a01, &a02);
ip_a1 = read_and_pad(ip_a1, &a11, &a12);

ch_0_out_0 = __SMLAD(a01, b0, ch_0_out_0);
ch_0_out_1 = __SMLAD(a01, b1, ch_0_out_1);
ch_1_out_0 = __SMLAD(a11, b0, ch_1_out_0);
ch_1_out_1 = __SMLAD(a11, b1, ch_1_out_1);

b0 = arm_nn_read_q15x2_ia(&ip_b0);
b1 = arm_nn_read_q15x2_ia(&ip_b1);

ch_0_out_0 = __SMLAD(a02, b0, ch_0_out_0);
ch_0_out_1 = __SMLAD(a02, b1, ch_0_out_1);
ch_1_out_0 = __SMLAD(a12, b0, ch_1_out_0);
ch_1_out_1 = __SMLAD(a12, b1, ch_1_out_1);

// 두 번째 4개 데이터 처리
ip_a0 = read_and_pad(ip_a0, &a03, &a04);
ip_a1 = read_and_pad(ip_a1, &a13, &a14);

b0 = arm_nn_read_q15x2_ia(&ip_b0);
b1 = arm_nn_read_q15x2_ia(&ip_b1);

ch_0_out_0 = __SMLAD(a03, b0, ch_0_out_0);
ch_0_out_1 = __SMLAD(a03, b1, ch_0_out_1);
ch_1_out_0 = __SMLAD(a13, b0, ch_1_out_0);
ch_1_out_1 = __SMLAD(a13, b1, ch_1_out_1);

b0 = arm_nn_read_q15x2_ia(&ip_b0);
b1 = arm_nn_read_q15x2_ia(&ip_b1);

ch_0_out_0 = __SMLAD(a04, b0, ch_0_out_0);
ch_0_out_1 = __SMLAD(a04, b1, ch_0_out_1);
ch_1_out_0 = __SMLAD(a14, b0, ch_1_out_0);
ch_1_out_1 = __SMLAD(a14, b1, ch_1_out_1);


while (col_count)              //25075/89016
{                               //2028번

    q7_t a0 = *ip_a0++;
    q15_t b0 = *ip_b0++;
    q7_t a1 = *ip_a1++;
    q15_t b1 = *ip_b1++;

    ch_0_out_0 += a0 * b0;
    ch_0_out_1 += a0 * b1;
    ch_1_out_0 += a1 * b0;
    ch_1_out_1 += a1 * b1;
    col_count--;
} // while over col_count
```

arm_nn_read_q15x2_ia

포인터 불러오고 + 2비트

read_and_pad

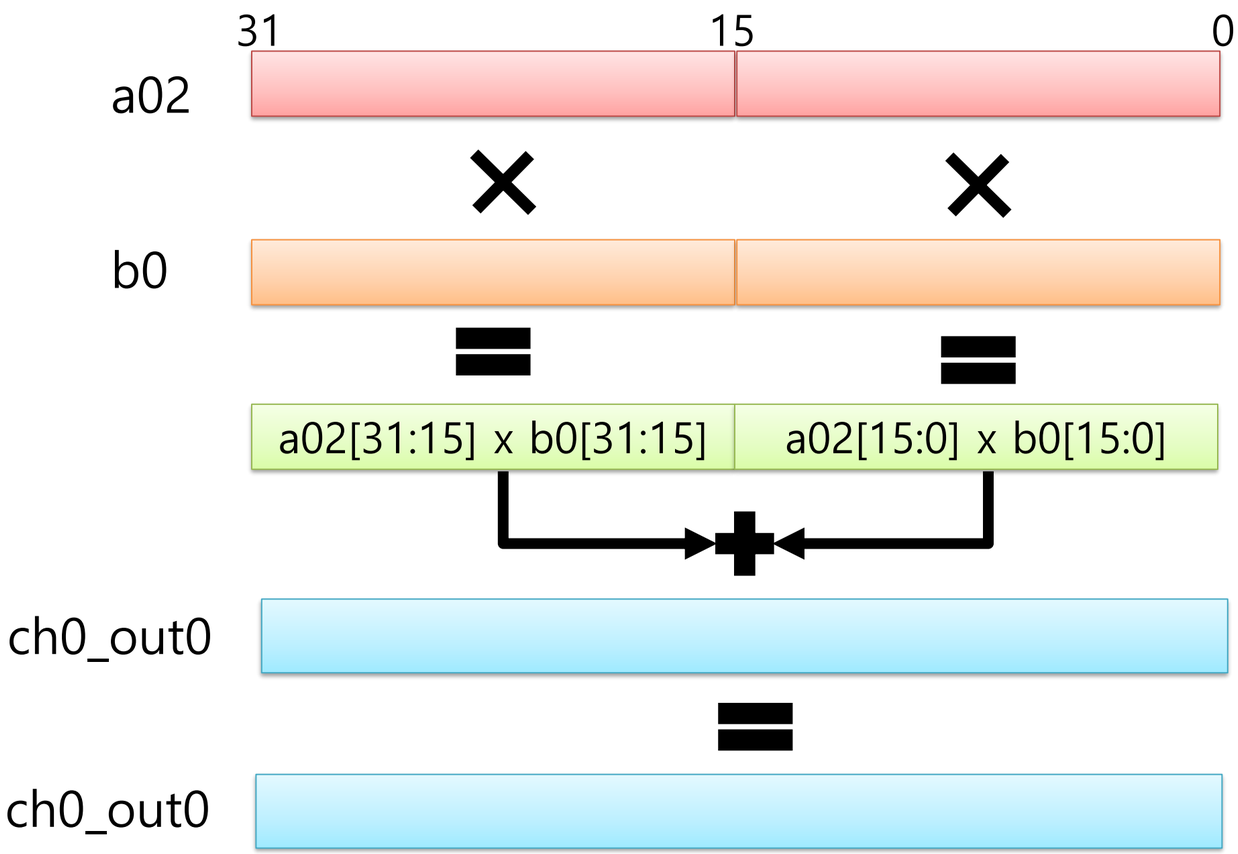
데이터 읽고 필요한 경우 패딩 추가

convolution

arm_nn_mult_kernel_s8_s16.c

```
ch_0_out_0 = __SMLAD(a02, b0, ch_0_out_0);
ch_0_out_1 = __SMLAD(a02, b1, ch_0_out_1);
ch_1_out_0 = __SMLAD(a12, b0, ch_1_out_0);
ch_1_out_1 = __SMLAD(a12, b1, ch_1_out_1);
```

__SMLAD



convolution

arm_nn_mult_kernel_s8_s16.c (기존)

```
ch_0_out_0 = arm_nn_requantize(ch_0_out_0, *out_mult, *out_shift);
ch_0_out_0 += out_offset;
ch_0_out_0 = MAX(ch_0_out_0, activation_min);
ch_0_out_0 = MIN(ch_0_out_0, activation_max);
*out_0++ = (q7_t)ch_0_out_0;

ch_0_out_1 = arm_nn_requantize(ch_0_out_1, *out_mult, *out_shift);
ch_0_out_1 += out_offset;
ch_0_out_1 = MAX(ch_0_out_1, activation_min);
ch_0_out_1 = MIN(ch_0_out_1, activation_max);
*out_1++ = (q7_t)ch_0_out_1;
out_mult++;
out_shift++;

ch_1_out_0 = arm_nn_requantize(ch_1_out_0, *out_mult, *out_shift);
ch_1_out_0 += out_offset;
ch_1_out_0 = MAX(ch_1_out_0, activation_min);
ch_1_out_0 = MIN(ch_1_out_0, activation_max);
*out_0++ = (q7_t)ch_1_out_0;

ch_1_out_1 = arm_nn_requantize(ch_1_out_1, *out_mult, *out_shift);
ch_1_out_1 += out_offset;
ch_1_out_1 = MAX(ch_1_out_1, activation_min);
ch_1_out_1 = MIN(ch_1_out_1, activation_max);
*out_1++ = (q7_t)ch_1_out_1;
out_mult++;
out_shift++;
```

arm_nn_mult_kernel_s8_s16.c (수정)

```
#define REQUANTIZE_AND_CLAMP_AND_STORE(ch_out, out_mult_ptr, out_shift_ptr, out_offset, activation_min, activation_max, out_ptr)
        ch_out = arm_nn_requantize(ch_out, *out_mult_ptr, *out_shift_ptr); \
        ch_out += out_offset; \
        ch_out = MAX(ch_out, activation_min); \
        ch_out = MIN(ch_out, activation_max); \
        *(out_ptr)++ = (q7_t)ch_out;
```

```
REQUANTIZE_AND_CLAMP_AND_STORE(ch_0_out_0, out_mult, out_shift, out_offset, activation_min, activation_max, out_0);
REQUANTIZE_AND_CLAMP_AND_STORE(ch_0_out_1, out_mult, out_shift, out_offset, activation_min, activation_max, out_1);
out_mult++;
out_shift++;

REQUANTIZE_AND_CLAMP_AND_STORE(ch_1_out_0, out_mult, out_shift, out_offset, activation_min, activation_max, out_0);
REQUANTIZE_AND_CLAMP_AND_STORE(ch_1_out_1, out_mult, out_shift, out_offset, activation_min, activation_max, out_1);
out_mult++;
out_shift++;
```

매크로 함수로 !

convolution

수정 전

```
CONV_2D took 1u ticks (25165 ms).
```

25165 micros

arm_q7_to_15_with_offset 함수 제거

```
CONV_2D took 1u ticks (19133 ms).
```

19133 micros

필터(ker_x, y) 루프 제거 + __UQADD simd 적용

```
CONV_2D took 1u ticks (14490 ms).
```

14490 micros

```
Class 0: 15
Class 1: 24
Class 2: 36
Class 3: 56
Class 4: 6
Class 5: 7
Class 6: -50
Class 7: 112
Class 8: 33
Class 9: 37
```

정확도 낮아짐

```
Output probabilities (int8):
Class 0: 12
Class 1: 23
Class 2: 37
Class 3: 52
Class 4: 5
Class 5: 1
Class 6: -51
Class 7: 114
Class 8: 32
Class 9: 31
```

arm_nn_mult_kernel_s8_s16 함수 최적화

```
CONV_2D took 1u ticks (11118 ms).
```

11118 micros

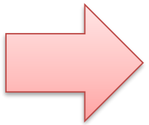
convolution

: SIMD 변경

__UQADD16

```
result1 = __UQADD16(val1, summand);
result2 = __UQADD16(val2, summand);
result3 = __UQADD16(val3, summand);
result4 = __UQADD16(val4, summand);
```

```
Output probabilities (int8):
Class 0: 12
Class 1: 23
Class 2: 37
Class 3: 52
Class 4: 5
Class 5: 1
Class 6: -51
Class 7: 114
Class 8: 32
Class 9: 31
```



__SADD16

```
//UQADD16 -> SADD16
result1 = __SADD16(val1, summand);
result2 = __SADD16(val2, summand);
result3 = __SADD16(val3, summand);
result4 = __SADD16(val4, summand);
```

```
Class 0: 15
Class 1: 24
Class 2: 36
Class 3: 56
Class 4: 6
Class 5: 7
Class 6: -50
Class 7: 112
Class 8: 33
Class 9: 37
```

사용하던 simd UQADD16 -> SADD16으로 바꿈

정확도 정상으로 복구됨

SADD16 : 상하위 16비트의 덧셈은 동일하나, 계산 결과에 따라
APSR(응용프로그램 상태 레지스터).GE플래그 설정

res[15:0] >= 0일 때 APSR.GE[1:0] = 11
res[31:16] >= 0일 때 APSR.GE[3:2] = 11

convolution : SIMD 제거

_SADD16 simd 사용

```
//UQADD16 -> SADD16
result1 = __SADD16(val1, summand);
result2 = __SADD16(val2, summand);
result3 = __SADD16(val3, summand);
result4 = __SADD16(val4, summand);
```

simd 사용 x

```
*(dst_ptr++) = src1+128;
*(dst_ptr++) = src2+128;
*(dst_ptr++) = src3+128;
*(dst_ptr++) = src4+128;
*(dst_ptr++) = src5+128;
*(dst_ptr++) = src6+128;
*(dst_ptr++) = src7+128;
*(dst_ptr++) = src8+128;
```

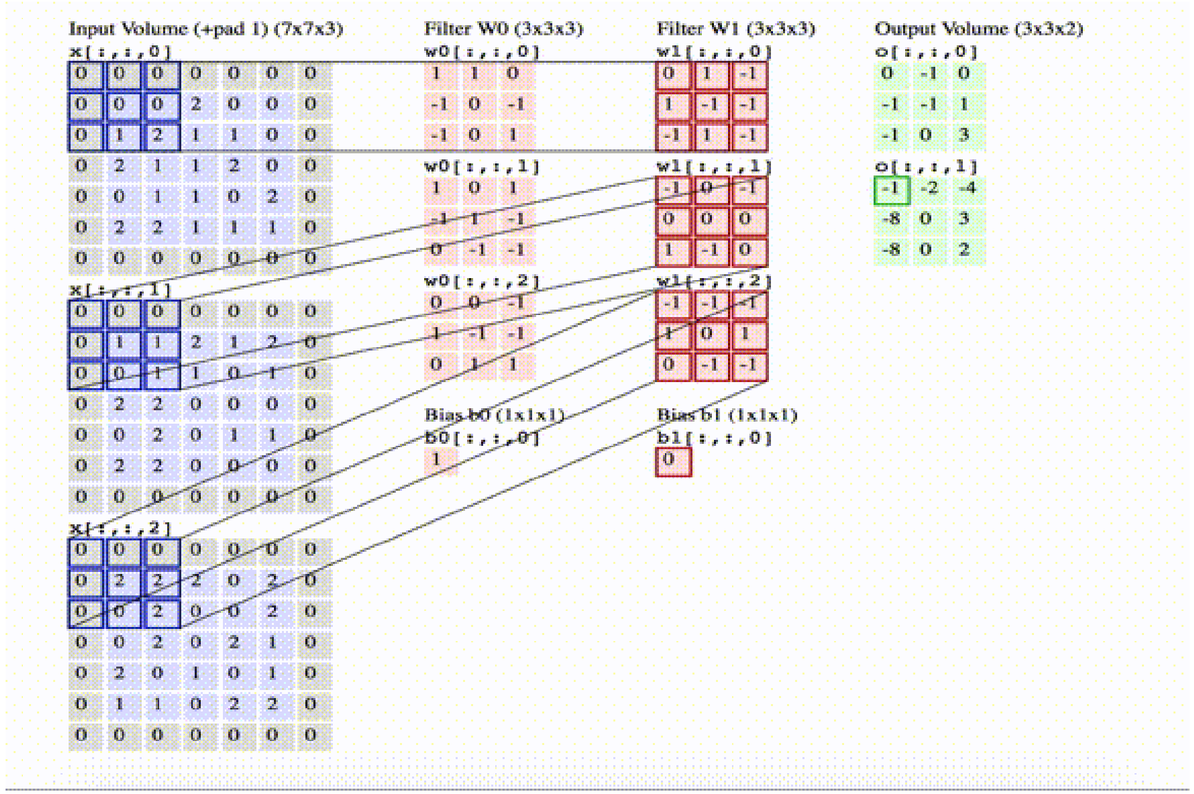
simd 사용 대신 일일이 더하는 것으로 바꿨을 때,
CONV_2D 시간 :
11151micros -> 11024micros

큰 차이가 나지는 않으나, 약간 감소

```
Invoke took lu ticks (14792 ms).
SHAPE took lu ticks (94 ms).
STRIDED_SLICE took lu ticks (98 ms).
PACK took lu ticks (83 ms).
RESHAPE took lu ticks (85 ms).
CONV_2D took lu ticks (11151 ms).
MAX_POOL_2D took lu ticks (1498 ms).
```

```
Invoke took lu ticks (14614 ms).
SHAPE took lu ticks (76 ms).
STRIDED_SLICE took lu ticks (98 ms).
PACK took lu ticks (83 ms).
RESHAPE took lu ticks (84 ms).
CONV_2D took lu ticks (11024 ms).
```

convolution : 검은 부분 연산 최소화

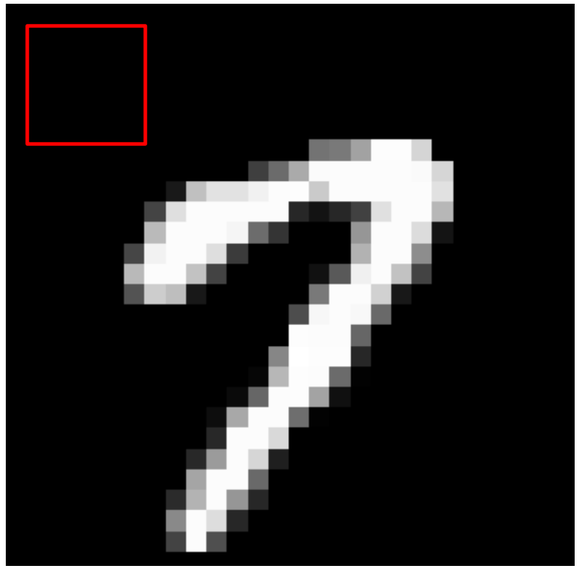


출처 : https://velog.io/@byu0hyun/%EB%94%A5%EB%9F%AC%EB%8B%9D-CNN-Conv2D-Layer

convolution연산의 과정을 보고

입력 데이터의 검은 부분에 해당하는 공통적이고 반복적인 값을 convolution연산 없이 바로 입력하여

불필요한 convolution 연산을 줄이고자 함



convolution : 검은 부분 연산 최소화

input_data 의 검은 부분

```
-128
-128
-128
-128
-128
-128
-128
-128
-128
-128
-128
-128
-128
-128
-128
```

arm_convolve_s8.c (기존)

```
/* Computation is filed for every 2 columns */
if (two_column_buf == buffer_a + 2 * input_ch * kernel_y * kernel_x)
{
    out = arm_nn_mat_mult_kernel_s8_s16(filter_data,
                                        buffer_a,
                                        output_ch,
                                        output_shift,
                                        output_mult,
                                        out_offset,
                                        out_activation_min,
                                        out_activation_max,
                                        input_ch * kernel_y * kernel_x,
                                        bias_data,
                                        out);

    /* counter reset */
    two_column_buf = buffer_a;
}
```

arm_covolve_3x3_s8.c

```
uint16_t num_col_a = input_ch * kernel_y * kernel_x; // 9
// 2열 버퍼 채워지면 함수 호출
//  Computation is filed for every 2 columns
if (two_column_buf == buffer_a + 2 * num_col_a) // 추가로18이 되면 인듯
{

    n[9] = output_ch;


    bool all_zero = true;
    q15_t *tmp_ptr = buffer_a; //two_column_buf - 18
    // 첫 18개의 값이 모두 0인지 확인
    for (int i = 0; i < 18; i++) {  //4438
       // n[3]++;
        if ((*(tmp_ptr++)) != 0) {    //100
            //n[2]++;
            all_zero = false;
            break;
        }
    }

    // 모두 0일 경우 out 배열의 첫 24개의 값을 out_activation_min로 설정
    if (all_zero) {
        for (int i = 1; i < 25; i++) {

            *(out++) = out_activation_min;

        }
    }


else{
                        //338번 반복 : 338 = 13 * 26
    out = arm_nn_mat_mult_kernel_s8_s16(filter_data,
                                        buffer_a,
                                        output_ch,
```

im2col에서 채우는 2열 버퍼의 값이 0인지 확인한 후,

모두 0일 경우, arm_nn_mat_mult_kernel_s8_s16함수에서 최종적으로 out_activation_min을 out주소에 저장하는 것을 확인하여

같은 기능 만듦.

convolution : 검은 부분 연산 최소화

기존

```
Output probabilities (int8):
Class 0: 15
Class 1: 24
Class 2: 36
Class 3: 56
Class 4: 6
Class 5: 7
Class 6: -50
Class 7: 112
Class 8: 33
Class 9: 37
```

```
Invoke took lu ticks (14637 ms).
SHAPE took lu ticks (104 ms).
STRIDED_SLICE took lu ticks (97 ms).
PACK took lu ticks (83 ms).
RESHAPE took lu ticks (83 ms).
CONV_2D took lu ticks (11053 ms).
MAX_POOL_2D took lu ticks (1441 ms).
RESHAPE took lu ticks (123 ms).
FULLY_CONNECTED took lu ticks (1141 ms).
```

검은 부분 연산없이 입력

```
Output probabilities (int8):
Class 0: 19
Class 1: -5
Class 2: 34
Class 3: 59
Class 4: 13
Class 5: 1
Class 6: -44
Class 7: 106
Class 8: 63
Class 9: 43
```

```
Invoke took lu ticks (8792 ms).
SHAPE took lu ticks (93 ms).
STRIDED_SLICE took lu ticks (96 ms).
PACK took lu ticks (81 ms).
RESHAPE took lu ticks (83 ms).
CONV_2D took lu ticks (5191 ms).
MAX_POOL_2D took lu ticks (1472 ms).
RESHAPE took lu ticks (94 ms).
FULLY_CONNECTED took lu ticks (1142 ms).
```

CONV_2D의 시간 :
11053micros -> 5191 micros
(약 47% 감소)

예측 또한 맞지만, 정확도의 변화

convolution : 검은 부분 연산 최소화

테스트 이미지 '5'로 바꿔 테스트

기존

```
Output probabilities (int8):
Class 0: -28
Class 1: -13
Class 2: -2
Class 3: 78
Class 4: -36
Class 5: 90
Class 6: -38
Class 7: -3
Class 8: 20
Class 9: 9

********************************
predicated_class : 5
total time : 14549
```

검은 부분 연산없이 입력

```
Output probabilities (int8):
Class 0: -27
Class 1: -38
Class 2: -6
Class 3: 84
Class 4: -30
Class 5: 84
Class 6: -35
Class 7: -9
Class 8: 46
Class 9: 13

********************************
predicated_class : 3
total time : 9837
```

테스트 이미지 7에서와는 달리

테스트 이미지 5로 바꾸었을 때,

추론 시간은 14549 micros -> 9837 micros로 4712micros감소하였으나,

검은 부분의 반복 연산을 없앤 코드에서 정확한 추론을 하지 못했음.

-> 예측으로 나온 3과 5의 확률은 동일하나, 기존 코드에 비해 정확도가 많이 떨어진 것으로 보임.

다시 기존 코드로 진행

convolution : 검은 부분 연산 최소화

정확도가 달라지게 된 원인?

arm_nn_mat_mult_kernel_s8_s16.c

```
#define REQUANTIZE_AND_CLAMP_AND_STORE(ch_out, out_mult_ptr, out_shift_ptr, out_offset, activation_min, activation_ma
        test[n[2]++] = ch_out;\
        ch_out = arm_nn_requantize(ch_out, *out_mult_ptr, *out_shift_ptr); \
        ch_out += out_offset; \
        ch_out = MAX(ch_out, activation_min); \
        ch_out = MIN(ch_out, activation_max); \
        *(out_ptr)++ = (q7_t)ch_out;
```

```
-10203
-10203
-28256
-28256
-18826
-18826
-2030
-2030
-79
-79
5137
5137
-122
-122
-3152
-3152
2876
2876
-27467
-27467
-18212
-18212
-184
-184
-10203
-10203
-28256
-28256
-18826
-18826
-2030
-2030
-79
```

같은 검은 부분임에도 위치마다 convolution 연산 결과가 다르며,

```
const uint16_t output_ch,   //12 개의 출력 채널(필터 수)
```

```
/* set up the second output pointers */
q7_t *out_1 = out_0 + output_ch;
```

```
REQUANTIZE_AND_CLAMP_AND_STORE(ch_0_out_0, out_mult, out_shift, out_offset, activation_min, activation_max, out_0);
REQUANTIZE_AND_CLAMP_AND_STORE(ch_0_out_1, out_mult, out_shift, out_offset, activation_min, activation_max, out_1);
```

총 12개의 출력 채널을 갖는데, 이중, 두 채널씩 같은 값을 갖음

fully_connected

arm_full_connected_s8.c

```c
arm_cmsis_nn_status arm_fully_connected_s8(const cmsis_nn_context *ctx,
                                           const cmsis_nn_fc_params *fc_params,
                                           const cmsis_nn_per_tensor_quant_params
                                           const cmsis_nn_dims *input_dims,
                                           const q7_t *input,
                                           const cmsis_nn_dims *filter_dims,
                                           const q7_t *kernel,
                                           const cmsis_nn_dims *bias_dims,
                                           const int32_t *bias,
                                           const cmsis_nn_dims *output_dims,
                                           q7_t *output)
{
    //s[2] = micros();
    (void)bias_dims;
    (void)ctx;
    (void)fc_params->filter_offset;

    int32_t batch_cnt = input_dims->n;

    while (batch_cnt)
    {
        arm_nn_vec_mat_mult_t_s8(input,
                                 kernel,
                                 bias,
                                 output,
                                 fc_params->input_offset,
                                 0,
                                 fc_params->output_offset,
                                 quant_params->multiplier,
                                 quant_params->shift,
                                 filter_dims->n, /* col_dim or accum_depth */
                                 output_dims->c, /* row_dim or output_depth */
                                 fc_params->activation.min,
                                 fc_params->activation.max,
                                 1L);
        input += filter_dims->n;
        output += output_dims->c;
        batch_cnt--;
    }
    //n[2]++;
    //t[2] += micros() -s[2];
    return (ARM_CMSIS_NN_SUCCESS);
}
```

arm_nn_vec_mult_t_s8.c (기존)

```c
#elif defined(ARM_MATH_DSP)
    const int32_t row_loop_cnt = rhs_rows / 2;
    const int16_t lhs_offset_s16 = (int16_t)lhs_offset;
    const uint32_t lhs_offset_s16x2 = __PKHBT(lhs_offset_s16, lh

    for (int32_t i = 0; i < row_loop_cnt; i++)
    {
        int32_t acc_0 = 0;
        int32_t acc_1 = 0;
        if (bias)
        {
            acc_0 = *bias++;
            acc_1 = *bias++;
        }

        const int32_t col_loop_cnt = rhs_cols / 4;

        const int8_t *lhs_vec = lhs;
        const int8_t *rhs_0 = rhs;
        const int8_t *rhs_1 = rhs + rhs_cols;
        rhs += 2 * rhs_cols;

        for (int j = col_loop_cnt; j != 0; j--)
        {
            int32_t vec_0 = arm_nn_read_q7x4_ia(&lhs_vec);
            int32_t vec_1 = __SXTAB16_RORn(lhs_offset_s16x2, (ui

            vec_0 = __SXTAB16(lhs_offset_s16x2, vec_0);

            int32_t ker_0 = arm_nn_read_q7x4_ia(&rhs_0);
            int32_t ker_1 = __SXTB16_RORn((uint32_t)ker_0, 8);
            ker_0 = __SXTB16(ker_0);

            acc_0 = __SMLAD(ker_1, vec_1, acc_0);
            acc_0 = __SMLAD(ker_0, vec_0, acc_0);

            ker_0 = arm_nn_read_q7x4_ia(&rhs_1);
            ker_1 = __SXTB16_RORn((uint32_t)ker_0, 8);
            ker_0 = __SXTB16(ker_0);

            acc_1 = __SMLAD(ker_1, vec_1, acc_1);
            acc_1 = __SMLAD(ker_0, vec_0, acc_1);
        }

        for (int k = col_loop_cnt * 4; k < rhs_cols; k++)
```

4개의 요소를
한 루프에 처리

fully_connected

루프언롤링 : 8

arm_nn_vec_mult_t_s8.c (수정)

```
const int32_t col_loop_cnt = rhs_cols / 32;  // 동시처리 4 , 루프언롤링 8

const int8_t *lhs_vec = lhs;
const int8_t *rhs_0 = rhs;
const int8_t *rhs_1 = rhs + rhs_cols;
rhs += 2 * rhs_cols;

for (int j = col_loop_cnt; j != 0; j--)  // 루프 언롤링 적용
{
    int32_t vec_0, vec_1, ker_0, ker_1;

    // 첫 번째 그룹
    vec_0 = arm_nn_read_q7x4_ia(&lhs_vec);
    vec_1 = __SXTAB16_RORn(lhs_offset_s16x2, (uint32_t)vec_0, 8);
    vec_0 = __SXTAB16(lhs_offset_s16x2, vec_0);
    ker_0 = arm_nn_read_q7x4_ia(&rhs_0);
    ker_1 = __SXTB16_RORn((uint32_t)ker_0, 8);
    ker_0 = __SXTB16(ker_0);
    acc_0 = __SMLAD(ker_1, vec_1, acc_0);
    acc_0 = __SMLAD(ker_0, vec_0, acc_0);
    ker_0 = arm_nn_read_q7x4_ia(&rhs_1);
    ker_1 = __SXTB16_RORn((uint32_t)ker_0, 8);
    ker_0 = __SXTB16(ker_0);
    acc_1 = __SMLAD(ker_1, vec_1, acc_1);
    acc_1 = __SMLAD(ker_0, vec_0, acc_1);

    // 두 번째 그룹
    vec_0 = arm_nn_read_q7x4_ia(&lhs_vec);
    vec_1 = __SXTAB16_RORn(lhs_offset_s16x2, (uint32_t)vec_0, 8);
    vec_0 = __SXTAB16(lhs_offset_s16x2, vec_0);
    ker_0 = arm_nn_read_q7x4_ia(&rhs_0);
    ker_1 = __SXTB16_RORn((uint32_t)ker_0, 8);
    ker_0 = __SXTB16(ker_0);
    acc_0 = __SMLAD(ker_1, vec_1, acc_0);
    acc_0 = __SMLAD(ker_0, vec_0, acc_0);
    ker_0 = arm_nn_read_q7x4_ia(&rhs_1);
    ker_1 = __SXTB16_RORn((uint32_t)ker_0, 8);
    ker_0 = __SXTB16(ker_0);
    acc_1 = __SMLAD(ker_1, vec_1, acc_1);
    acc_1 = __SMLAD(ker_0, vec_0, acc_1);
```

```
// 세 번째 그룹
vec_0 = arm_nn_read_q7x4_ia(&lhs_vec);
vec_1 = __SXTAB16_RORn(lhs_offset_s16x2, (uint32_t)vec_0, 8);
vec_0 = __SXTAB16(lhs_offset_s16x2, vec_0);
ker_0 = arm_nn_read_q7x4_ia(&rhs_0);
ker_1 = __SXTB16_RORn((uint32_t)ker_0, 8);
ker_0 = __SXTB16(ker_0);
acc_0 = __SMLAD(ker_1, vec_1, acc_0);
acc_0 = __SMLAD(ker_0, vec_0, acc_0);
ker_0 = arm_nn_read_q7x4_ia(&rhs_1);
ker_1 = __SXTB16_RORn((uint32_t)ker_0, 8);
ker_0 = __SXTB16(ker_0);
acc_1 = __SMLAD(ker_1, vec_1, acc_1);
acc_1 = __SMLAD(ker_0, vec_0, acc_1);

// 네 번째 그룹
vec_0 = arm_nn_read_q7x4_ia(&lhs_vec);
vec_1 = __SXTAB16_RORn(lhs_offset_s16x2, (uint32_t)vec_0, 8);
vec_0 = __SXTAB16(lhs_offset_s16x2, vec_0);
ker_0 = arm_nn_read_q7x4_ia(&rhs_0);
ker_1 = __SXTB16_RORn((uint32_t)ker_0, 8);
ker_0 = __SXTB16(ker_0);
acc_0 = __SMLAD(ker_1, vec_1, acc_0);
acc_0 = __SMLAD(ker_0, vec_0, acc_0);
ker_0 = arm_nn_read_q7x4_ia(&rhs_1);
ker_1 = __SXTB16_RORn((uint32_t)ker_0, 8);
ker_0 = __SXTB16(ker_0);
acc_1 = __SMLAD(ker_1, vec_1, acc_1);
acc_1 = __SMLAD(ker_0, vec_0, acc_1);

// 다섯 번째 그룹
vec_0 = arm_nn_read_q7x4_ia(&lhs_vec);
vec_1 = __SXTAB16_RORn(lhs_offset_s16x2, (uint32_t)vec_0, 8);
vec_0 = __SXTAB16(lhs_offset_s16x2, vec_0);
ker_0 = arm_nn_read_q7x4_ia(&rhs_0);
ker_1 = __SXTB16_RORn((uint32_t)ker_0, 8);
ker_0 = __SXTB16(ker_0);
acc_0 = __SMLAD(ker_1, vec_1, acc_0);
acc_0 = __SMLAD(ker_0, vec_0, acc_0);
ker_0 = arm_nn_read_q7x4_ia(&rhs_1);
ker_1 = __SXTB16_RORn((uint32_t)ker_0, 8);
ker_0 = __SXTB16(ker_0);
acc_1 = __SMLAD(ker_1, vec_1, acc_1);
acc_1 = __SMLAD(ker_0, vec_0, acc_1);
```

```
// 여섯 번째 그룹
vec_0 = arm_nn_read_q7x4_ia(&lhs_vec);
vec_1 = __SXTAB16_RORn(lhs_offset_s16x2, (uint32_t)vec_0, 8);
vec_0 = __SXTAB16(lhs_offset_s16x2, vec_0);
ker_0 = arm_nn_read_q7x4_ia(&rhs_0);
ker_1 = __SXTB16_RORn((uint32_t)ker_0, 8);
ker_0 = __SXTB16(ker_0);
acc_0 = __SMLAD(ker_1, vec_1, acc_0);
acc_0 = __SMLAD(ker_0, vec_0, acc_0);
ker_0 = arm_nn_read_q7x4_ia(&rhs_1);
ker_1 = __SXTB16_RORn((uint32_t)ker_0, 8);
ker_0 = __SXTB16(ker_0);
acc_1 = __SMLAD(ker_1, vec_1, acc_1);
acc_1 = __SMLAD(ker_0, vec_0, acc_1);

// 일곱 번째 그룹
vec_0 = arm_nn_read_q7x4_ia(&lhs_vec);
vec_1 = __SXTAB16_RORn(lhs_offset_s16x2, (uint32_t)vec_0, 8);
vec_0 = __SXTAB16(lhs_offset_s16x2, vec_0);
ker_0 = arm_nn_read_q7x4_ia(&rhs_0);
ker_1 = __SXTB16_RORn((uint32_t)ker_0, 8);
ker_0 = __SXTB16(ker_0);
acc_0 = __SMLAD(ker_1, vec_1, acc_0);
acc_0 = __SMLAD(ker_0, vec_0, acc_0);
ker_0 = arm_nn_read_q7x4_ia(&rhs_1);
ker_1 = __SXTB16_RORn((uint32_t)ker_0, 8);
ker_0 = __SXTB16(ker_0);
acc_1 = __SMLAD(ker_1, vec_1, acc_1);
acc_1 = __SMLAD(ker_0, vec_0, acc_1);

// 여덟 번째 그룹
vec_0 = arm_nn_read_q7x4_ia(&lhs_vec);
vec_1 = __SXTAB16_RORn(lhs_offset_s16x2, (uint32_t)vec_0, 8);
vec_0 = __SXTAB16(lhs_offset_s16x2, vec_0);
ker_0 = arm_nn_read_q7x4_ia(&rhs_0);
ker_1 = __SXTB16_RORn((uint32_t)ker_0, 8);
ker_0 = __SXTB16(ker_0);
acc_0 = __SMLAD(ker_1, vec_1, acc_0);
acc_0 = __SMLAD(ker_0, vec_0, acc_0);
ker_0 = arm_nn_read_q7x4_ia(&rhs_1);
ker_1 = __SXTB16_RORn((uint32_t)ker_0, 8);
ker_0 = __SXTB16(ker_0);
acc_1 = __SMLAD(ker_1, vec_1, acc_1);
acc_1 = __SMLAD(ker_0, vec_0, acc_1);
```

fully_connected

arm_nn_vec_mult_t_s8.c (수정) (하단)

```
// 나머지 데이터 처리
for (int k = col_loop_cnt * 32; k < rhs_cols; k++)
{
    const int32_t lhs_temp = (*lhs_vec + lhs_offset);
    lhs_vec++;
    acc_0 += lhs_temp * (*rhs_0);
    rhs_0++;
    acc_1 += lhs_temp * (*rhs_1);
    rhs_1++;
}

acc_0 = arm_nn_requantize(acc_0, dst_multiplier, dst_shift);
acc_1 = arm_nn_requantize(acc_1, dst_multiplier, dst_shift);
```

arm_nnsupportfunctions.h

```
__STATIC_FORCEINLINE q31_t arm_nn_requantize(const q31_t val, const q31_t multiplier, const q31_t shift)
{
#ifdef CMSIS_NN_USE_SINGLE_ROUNDING //단일 반올림
    const int64_t total_shift = 31 - shift;
    const int64_t new_val = val * (int64_t)multiplier;

    int32_t result = new_val >> (total_shift - 1);
    result = (result + 1) >> 1;

    return result;
#else   //이중 반올림
    return arm_nn_divide_by_power_of_two(arm_nn_doubling_high_mult_no_sat(val * (1 << LEFT_SHIFT(shift)),
                                                                          RIGHT_SHIFT(shift));
#endif
}
```

기존 이중 반올림 -> 단일 반올림

단일 반올림 : 한번만 반올림

이중 반올림 : 중간에 한번, 최종으로 한 번 반올림
-> 정확도 높이고, 오차 감소 , 계산 비용 증가

단일 반올림으로 전환 후,
시간 감소 하였고,
정확도 변화 없음

fully_connected

수정 전

```
FULLY_CONNECTED took lu ticks (1661 ms).
```

1661 micros

루프언롤링 : 2

```
FULLY_CONNECTED took lu ticks (1403 ms).
```

1403 micros

루프언롤링 : 4

```
FULLY_CONNECTED took lu ticks (1301 ms).
```

1301 micros

루프언롤링 : 8

```
FULLY_CONNECTED took lu ticks (1195 ms).
```

1195 micros

루프언롤링 : 16

```
FULLY_CONNECTED took lu ticks (1196 ms)
```

1196 micros

max_pool

arm_max_pool_s8.c

```
for (int i_y = 0, base_idx_y = -pad_y; i_y < output_y; base_idx_y += stride_y, i_y++)
{
    for (int i_x = 0, base_idx_x = -pad_x; i_x < output_x; base_idx_x += stride_x, i_x++)
    {
        /* Condition for kernel start dimension: (base_idx_<x,y> + kernel_<x,y>_start) >= 0 */
        const int32_t ker_y_start = MAX(0, -base_idx_y);
        const int32_t ker_x_start = MAX(0, -base_idx_x);

        /* Condition for kernel end dimension: (base_idx_<x,y> + kernel_<x,y>_end) < dim_src_<width,height> */
        const int32_t kernel_y_end = MIN(kernel_y, input_y - base_idx_y);
        const int32_t kernel_x_end = MIN(kernel_x, input_x - base_idx_x);

        int count = 0;

        for (int k_y = ker_y_start; k_y < kernel_y_end; k_y++)
        {
            for (int k_x = ker_x_start; k_x < kernel_x_end; k_x++)
            {
                const q7_t *start = src + channel_in * (k_x + base_idx_x + (k_y + base_idx_y) * input_x);

                if (count == 0)
                {
                    arm_memcpy_q7(dst, start, channel_in);
                    count++;
                }
                else
                {
                    compare_and_replace_if_larger_q7(dst, start, channel_in);
                }
            }
        }
        /* 'count' is expected to be non-zero here. */
        dst += channel_in;
    }
}
```

- arm_max_pool_s8 함수

- input 영역에서 filter 사이즈 만큼 차례대로 순회

- filter에 영역에서 차례대로 순회하며 이전 값과 비교(compare_and_replace_if_larger_q7)

- filter size – 1 만큼 함수를 호출하고 비교 연산을 수행

max_pool

arm_nnfunctions.h

```
arm_cmsis_nn_status arm_max_pool_s8(const cmsis_nn_context *ctx,
                                    const cmsis_nn_pool_params *pool_params,
                                    const cmsis_nn_dims *input_dims,
                                    const q7_t *input_data,
                                    const cmsis_nn_dims *filter_dims,
                                    const cmsis_nn_dims *output_dims,
                                    q7_t *output_data);
//
arm_cmsis_nn_status arm_max_pool_2x2_s8(const cmsis_nn_context *ctx,
                                    const cmsis_nn_pool_params *pool_params,
                                    const cmsis_nn_dims *input_dims,
                                    const q7_t *input_data,
                                    const cmsis_nn_dims *filter_dims,
                                    const cmsis_nn_dims *output_dims,
                                    q7_t *output_data);
```

pooling.cpp

```
if (input->type == kTfLiteInt8) {
    if (filter_dims.h == 2 && filter_dims.w == 2) {
        TFLITE_DCHECK_EQ(
          arm_max_pool_2x2_s8(&ctx, &pool_params, &input_dims,
                              micro::GetTensorData<int8_t>(input), &filter_dims,
                              &output_dims, micro::GetTensorData<int8_t>(output)),
            ARM_CMSIS_NN_SUCCESS);
    } else {
        TFLITE_DCHECK_EQ(
            arm_max_pool_s8(&ctx, &pool_params, &input_dims,
                              micro::GetTensorData<int8_t>(input), &filter_dims,
                              &output_dims, micro::GetTensorData<int8_t>(output)),
            ARM_CMSIS_NN_SUCCESS);
    }
```

- 기존 함수를 유지하고 새로운 함수 선언
- 2x2 max_pooling 연산에 특화된 연산 함수 사용
- 추가한 함수 arm_nnfunctions.h에서 선언
- pooling.cpp에서 해당 함수 호출 조건 추가
  (filter_dims.h == 2 && filter_dims.w == 2)

max_pool

arm_max_pool_2x2_s8.c

```c
static void compare_2x2(const q7_t *src1, const q7_t *src2, const q7_t *src3, const q7_t *src4, q7_t *dst, int32_t length)
{
    for (int i = 0; i < length; i++)
    {
        q7_t max_val = MAX(MAX(src1[i], src2[i]), MAX(src3[i], src4[i]));
        dst[i] = max_val;
    }
}
```

```c
for (int i_y = 0, base_idx_y = -pad_y; i_y < output_y; base_idx_y += stride_y, i_y++)
{
    for (int i_x = 0, base_idx_x = -pad_x; i_x < output_x; base_idx_x += 2 * stride_x, i_x += 2)
    {
        /* Condition for kernel start dimension: (base_idx_<x,y> + kernel_<x,y>_start) >= 0 */
        const int32_t ker_y_start = MAX(0, -base_idx_y);
        const int32_t ker_x_start = MAX(0, -base_idx_x);

        /* Condition for kernel end dimension: (base_idx_<x,y> + kernel_<x,y>_end) < dim_src_<width,height> */
        const int32_t kernel_y_end = MIN(2, input_y - base_idx_y);
        const int32_t kernel_x_end = MIN(2, input_x - base_idx_x);

        const q7_t *start1 = src + channel_in * (ker_x_start + base_idx_x + (ker_y_start + base_idx_y) * input_x);
        const q7_t *start2 = (ker_y_start + 1 < kernel_y_end) ? start1 + channel_in * input_x : start1;
        const q7_t *start3 = (ker_x_start + 1 < kernel_x_end) ? start1 + channel_in : start1;
        const q7_t *start4 = (ker_y_start + 1 < kernel_y_end && ker_x_start + 1 < kernel_x_end) ? start1 + channel_in * input_x + channel_in : start1;

        const q7_t *start5 = src + channel_in * (ker_x_start + base_idx_x + stride_x + (ker_y_start + base_idx_y) * input_x);
        const q7_t *start6 = (ker_y_start + 1 < kernel_y_end) ? start5 + channel_in * input_x : start5;
        const q7_t *start7 = (ker_x_start + 1 < kernel_x_end) ? start5 + channel_in : start5;
        const q7_t *start8 = (ker_y_start + 1 < kernel_y_end && ker_x_start + 1 < kernel_x_end) ? start5 + channel_in * input_x + channel_in : start5;

        compare_and_replace_if_larger_q7_2x2(start1, start2, start3, start4, dst, channel_in);
        dst += channel_in;

        if (i_x + 1 < output_x)
        {
            compare_and_replace_if_larger_q7_2x2(start5, start6, start7, start8, dst, channel_in);
            dst += channel_in;
        }
    }
}
```

- arm_max_pool_2x2_s8 함수
- input 영역에서 x 축으로 2개의 filter 영역을 한번에 처리
- 하나의 filter 영역을 처리할 때 한번의 호출만 있도록 수정
- compare_2x2 함수를 통해 필터 내 4개의 값을 비교

```
MAX_POOL_2D took lu ticks (1832 ms).
```

```
MAX_POOL_2D took lu ticks (1441 ms).
```

mnist.ino

사용하지 않은 Op resolver 제거

```
static tflite::MicroMutableOpResolver<10> micro_op_resolver;
micro_op_resolver.AddShape();
micro_op_resolver.AddStridedSlice();  //use
micro_op_resolver.AddPack();          //use
micro_op_resolver.AddMaxPool2D();
micro_op_resolver.AddFullyConnected();
//micro_op_resolver.AddAveragePool2D();   //no use
micro_op_resolver.AddConv2D();
//micro_op_resolver.AddDepthwiseConv2D(); //no use
micro_op_resolver.AddReshape();
//micro_op_resolver.AddSoftmax();         //no use
```

```
predicated_class : 7
total time : 14596
```

```
> predicated_class : 7
> total time : 14507
```

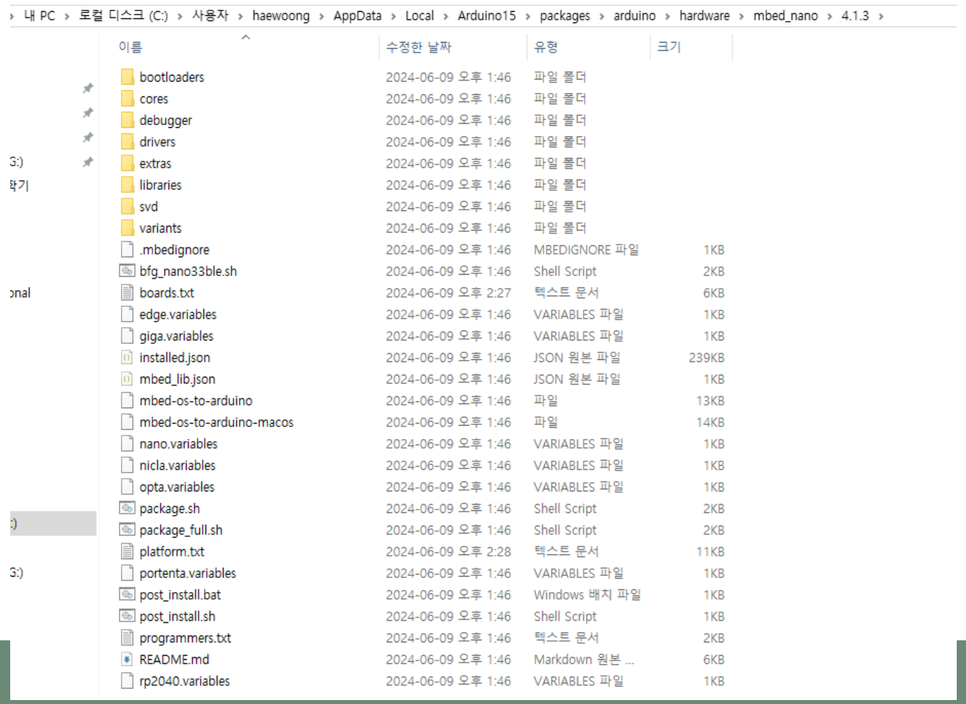
전체 시간 :

14596 micros -> 14507 micros
약 89 micros 감소

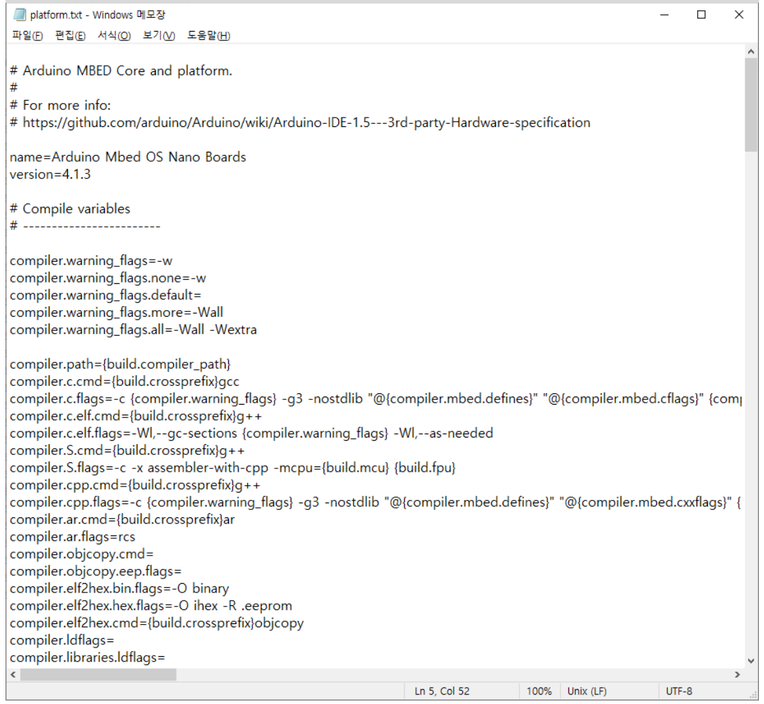
Arduino ide : arm-none-eabi-gcc 옵션

Arduino ide에서
각 보드의 아키텍쳐에 맞는 gcc는
보드매니저를 통해 다운로드 된다.

보드매니저에서 다운로드한 폴더



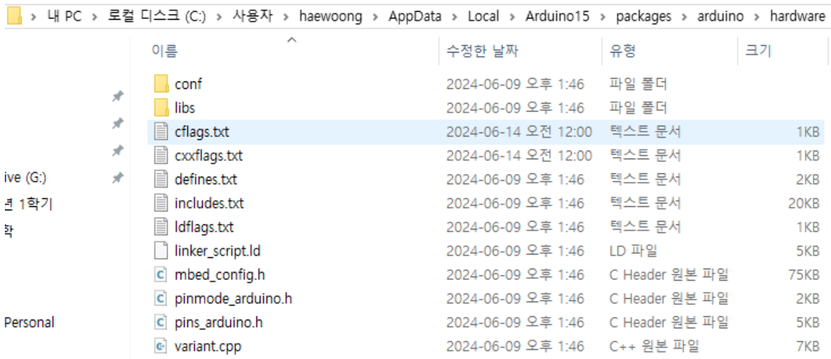
platform.txt



platform.txt에서 다양한 gcc 옵션을 넣을 수 있다.

Arduino ide ：arm-none-eabi-gcc 옵션

C:\Users\haewoong\AppData\Local\Arduino15\packages\arduino\hardware\mbed_nano\4.1.3\variants\ARDUINO_NANO33BLE



실행 시간에 영향을 주는 옵션:

-O2 :

최적화 수준에 따른 최적화 옵션

-fomit-frame-pointer :

함수 호출에서 사용하는 프레임 포인터를 생략

프레임 포인터로 사용될 레지스터를 다른 연산에서 사용가능

cflags.txt

*cflags.txt - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

```
-c
-std=gnu11
-DAPPLICATION_ADDR=0x10000
-DAPPLICATION_SIZE=0xf0000
-DMBED_RAM_SIZE=0x40000
-DMBED_RAM_START=0x20000000
-DMBED_ROM_SIZE=0x100000
-DMBED_ROM_START=0x0
-DMBED_TRAP_ERRORS_ENABLED=1
-O2
-Wall
-Wextra
-Wno-missing-field-initializers
-Wno-unused-parameter
-fdata-sections
-ffunction-sections
-fmessage-length=0
-fno-exceptions
-fomit-frame-pointer
-funsigned-char
-mcpu=cortex-m4
-mfloat-abi=softfp
-mfpu=fpv4-sp-d16
-mthumb
```

cxxflags.txt

*cxxflags.txt - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

```
-Wvla
-c
-fno-rtti
-std=gnu++14
-DAPPLICATION_ADDR=0x10000
-DAPPLICATION_SIZE=0xf0000
-DMBED_RAM_SIZE=0x40000
-DMBED_RAM_START=0x20000000
-DMBED_ROM_SIZE=0x100000
-DMBED_ROM_START=0x0
-DMBED_TRAP_ERRORS_ENABLED=1
-O2
-Wall
-Wextra
-Wno-missing-field-initializers
-Wno-unused-parameter
-fdata-sections
-ffunction-sections
-fmessage-length=0
-fno-exceptions
-fomit-frame-pointer
-funsigned-char
-mcpu=cortex-m4
-mfloat-abi=softfp
-mfpu=fpv4-sp-d16
-mthumb
```

최적화 최종 결과

```
Invoke took 1u ticks (89293 ms).
SHAPE took 1u ticks (15 ms).
STRIDED_SLICE took 1u ticks (35 ms).
PACK took 1u ticks (20 ms).
RESHAPE took 1u ticks (42 ms).
CONV_2D took 1u ticks (78929 ms).
MAX_POOL_2D took 1u ticks (6228 ms).
RESHAPE took 1u ticks (113 ms).
FULLY_CONNECTED took 1u ticks (3779 ms).
```

Int 8 적용

```
Invoke took 1u ticks (29690 ms).
SHAPE took 1u ticks (77 ms).
STRIDED_SLICE took 1u ticks (98 ms).
PACK took 1u ticks (83 ms).
RESHAPE took 1u ticks (85 ms).
CONV_2D took 1u ticks (25165 ms).
MAX_POOL_2D took 1u ticks (1832 ms).
RESHAPE took 1u ticks (96 ms).
FULLY_CONNECTED took 1u ticks (1661 ms).
```

최적화 적용

```
Invoke took 1u ticks (14594 ms).
SHAPE took 1u ticks (76 ms).
STRIDED_SLICE took 1u ticks (97 ms).
PACK took 1u ticks (83 ms).
RESHAPE took 1u ticks (102 ms).
CONV_2D took 1u ticks (10948 ms).
MAX_POOL_2D took 1u ticks (1477 ms).
RESHAPE took 1u ticks (95 ms).
FULLY_CONNECTED took 1u ticks (1169 ms).
```

O2 옵션 적용

```
Invoke took 1u ticks (13294 ms).
SHAPE took 1u ticks (74 ms).
STRIDED_SLICE took 1u ticks (92 ms).
PACK took 1u ticks (98 ms).
RESHAPE took 1u ticks (83 ms).
CONV_2D took 1u ticks (10053 ms).
MAX_POOL_2D took 1u ticks (1148 ms).
RESHAPE took 1u ticks (122 ms).
FULLY_CONNECTED took 1u ticks (1112 ms).
```

O3 옵션 적용

```
Invoke took 1u ticks (13560 ms).
SHAPE took 1u ticks (74 ms).
STRIDED_SLICE took 1u ticks (94 ms).
PACK took 1u ticks (98 ms).
RESHAPE took 1u ticks (83 ms).
CONV_2D took 1u ticks (10337 ms).
MAX_POOL_2D took 1u ticks (1161 ms).
RESHAPE took 1u ticks (94 ms).
FULLY_CONNECTED took 1u ticks (1107 ms).
```

Tflite file 확인

```
import flatbuffers
import tflite
```

```
# 모델 파일 경로
file_path = 'model.tflite'

# 모델 로드
model_buffer = load_tflite_model(file_path)

# FlatBuffers를 사용하여 모델 파싱
model = tflite.Model.GetRootAsModel(model_buffer, 0)

# 모델 정보 출력
print_model_info(model)

# OperatorCode 정보 출력
for i in range(model.OperatorCodesLength()):
    print_operator_code_info(model.OperatorCodes(i))

# 서브그래프 정보 출력
for i in range(model.SubgraphsLength()):
    subgraph = model.Subgraphs(i)
    print_subgraph_info(subgraph, i)

    # 텐서 정보 출력
    for j in range(subgraph.TensorsLength()):
        print_tensor_info(subgraph.Tensors(j), j)

    # 연산자 정보 출력
    for k in range(subgraph.OperatorsLength()):
        print_operator_info(subgraph.Operators(k), model)

# 버퍼 정보 출력
print("Number of buffers:", model.BuffersLength())
buffer_data = []
for i in range(model.BuffersLength()):
    buffer = model.Buffers(i)
    buffer_data.append(buffer.DataAsNumpy())
    if buffer.DataLength() > 0:
        print(f"Buffer {i} has data length:", buffer.DataLength())
    else:
        print(f"Buffer {i} is empty.")
    print()

# 메타데이터 정보 출력
for i in range(model.MetadataLength()):
    metadata = model.Metadata(i)
    print_metadata_info(metadata, buffer_data)
```

tflite 파일 확인

```
Model version: 3
Number of operator codes: 7
Number of subgraphs: 1
Number of buffers: 21
Number of metadata: 2

Operator code:
  Builtin code: SHAPE
  Version: 1

Operator code:
  Builtin code: STRIDED_SLICE
  Version: 1

Operator code:
  Builtin code: PACK
  Version: 1

Operator code:
  Builtin code: RESHAPE
  Version: 1

Operator code:
  Builtin code: CONV_2D
  Version: 3

Operator code:
  Builtin code: MAX_POOL_2D
  Version: 2

Operator code:
  Builtin code: FULLY_CONNECTED
  Version: 4

Subgraph 0:
  Number of tensors: 18
  Number of inputs: 1
  Number of outputs: 1
  Number of operators: 8

  Tensor 0:
    Name: b'serving_default_input_6:0'
    Type: 9
    Shape: [ 1 28 28]
    Quantization scale: [0.00392157]
    Quantization zero_point: [-128]
```

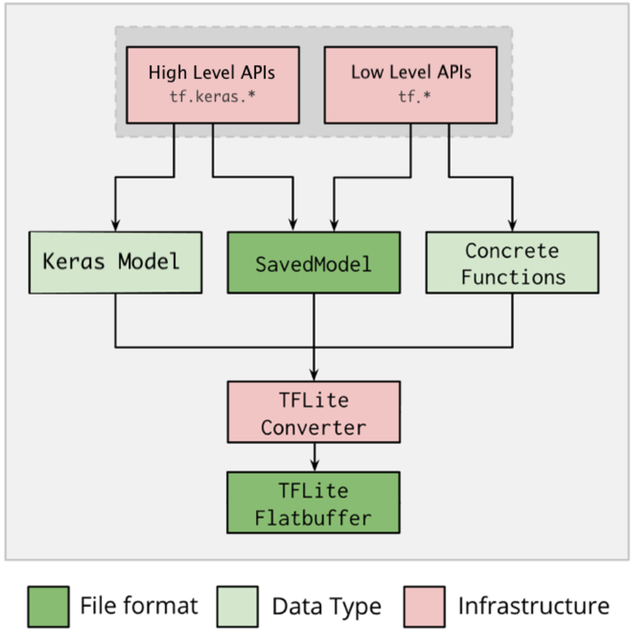
```
Tensor 1:
  Name: b'sequential_6/reshape_3/strided_slice/stack'
  Type: 2
  Shape: [1]
  Quantization scale: 0
  Quantization zero_point: 0

Tensor 2:
  Name: b'sequential_6/reshape_3/strided_slice/stack_1'
  Type: 2
  Shape: [1]
  Quantization scale: 0
  Quantization zero_point: 0

Tensor 3:
  Name: b'sequential_6/reshape_3/Reshape/shape/1'
  Type: 2
  Shape: []
  Quantization scale: 0
  Quantization zero_point: 0

Tensor 4:
  Name: b'sequential_6/reshape_3/Reshape/shape/3'
  Type: 2
  Shape: []
  Quantization scale: 0
  Quantization zero_point: 0

Tensor 5:
  Name: b'sequential_6/flatten_8/Const'
  Type: 2
  Shape: [2]
  Quantization scale: 0
  Quantization zero_point: 0

Tensor 6:
  Name: b'sequential_6/dense_6/BiasAdd/ReadVariableOp'
  Type: 2
  Shape: [10]
  Quantization scale: [7.3220326e-05]
  Quantization zero_point: [0]
```

Tflite file 확인

TensorFlow

(`.tflite` 파일 확장자로 식별되는 최적화된 FlatBuffer 형식).

tensorflow / tensorflow / lite / schema / **schema_v3.fbs**



https://www.tensorflow.org/lite/models/convert/convert_models?hl=ko

schema_v3.fbs - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

```
// limitations under the License.

// Revision History
// Version 0: Initial version.
// Version 1: Add subgraphs to schema.
// Version 2: Rename operators to conform to NN API.
// Version 3: Move buffer data from Model.Subgraph.Tensors to Model.Buffers.

namespace tflite;

// This corresponds to the version (4).
file_identifier "TFL3";
// File extension of any written files.
file_extension "tflite";

// The type of data stored in a tensor.
enum TensorType : byte {
  FLOAT32 = 0,
  FLOAT16 = 1,
  INT32 = 2,
  UINT8 = 3,
  INT64 = 4,
  STRING = 5,
}

// Parameters for converting a quantized tensor back to float. Given a
// quantized value q, the corresponding float value f should be:
//   f = scale * (q - zero_point)
table QuantizationParameters {
  min:[float];  // For importing back into tensorflow.
  max:[float];  // For importing back into tensorflow.
  scale:[float];
  zero_point:[long];
}

table Tensor {
  // The tensor shape. The meaning of each entry is operator-specific but
  // builtin ops use: [batch size, height, width, number of channels] (That's
  // Tensorflow's NHWC).
  shape:[int];
  type:TensorType;
  // An index that refers to the buffers table at the root of the model. Or,
  // if there is no data buffer associated (i.e. intermediate results), then
  // this is 0 (which refers to an always existent empty buffer).
```

Ln 35, Col 14 | 100% | Unix (LF) | UTF-8

Tflite file 확인

tflite file 버퍼 구조

```
Buffer 0 is empty.

Buffer 1 is empty.

Buffer 2 has data length: 4

Buffer 3 has data length: 4

Buffer 4 has data length: 4

Buffer 5 is empty.

Buffer 6 has data length: 8

Buffer 7 has data length: 40

Buffer 8 has data length: 20280

Buffer 9 has data length: 48

Buffer 10 has data length: 108

Buffer 11 is empty.

Buffer 12 is empty.

Buffer 13 is empty.

Buffer 14 is empty.

Buffer 15 is empty.

Buffer 16 is empty.

Buffer 17 is empty.

Buffer 18 is empty.

Buffer 19 has data length: 16

Buffer 20 has data length: 88
```

shema_generated.h

```
inline ModelT *Model::UnPack(const flatbuffers::resolver_function_t *_resolver) const {
  auto _o = std::unique_ptr<ModelT>(new ModelT());
  UnPackTo(_o.get(), _resolver);
  return _o.release();
}

inline void Model::UnPackTo(ModelT *_o, const flatbuffers::resolver_function_t *_resolver) const {
  (void)_o;
  (void)_resolver;
  { auto _e = version(); _o->version = _e; }
  { auto _e = operator_codes(); if (_e) { _o->operator_codes.resize(_e->size()); for (flatbuffers::uoffset_t _i = 0; _i < _e->si
  { auto _e = subgraphs(); if (_e) { _o->subgraphs.resize(_e->size()); for (flatbuffers::uoffset_t _i = 0; _i < _e->size(); _i++
  { auto _e = description(); if (_e) _o->description = _e->str(); }
  { auto _e = buffers(); if (_e) { _o->buffers.resize(_e->size()); for (flatbuffers::uoffset_t _i = 0; _i < _e->size(); _i++) {
  { auto _e = metadata_buffer(); if (_e) { _o->metadata_buffer.resize(_e->size()); for (flatbuffers::uoffset_t _i = 0; _i < _e->
  { auto _e = metadata(); if (_e) { _o->metadata.resize(_e->size()); for (flatbuffers::uoffset_t _i = 0; _i < _e->size(); _i++)
  { auto _e = signature_defs(); if (_e) { _o->signature_defs.resize(_e->size()); for (flatbuffers::uoffset_t _i = 0; _i < _e->si
}

inline flatbuffers::Offset<Model> Model::Pack(flatbuffers::FlatBufferBuilder &_fbb, const ModelT* _o, const flatbuffers::rehashe
  return CreateModel(_fbb, _o, _rehasher);
}
```

- tflite 파일의 정보를 확인하며 비어있는 버퍼를 확인
- 비어있는 버퍼의 용도를 확인하고자 tflite 읽어오는 부분을 확인
- 만약 비어있는 버퍼를 사용하지 않는다면 사용하지 않음으로써 모델의 사이즈를 줄일 수 있을 것임

Tflite file 확인

shema_generated.h

```
struct BufferBuilder {
  typedef Buffer Table;
  flatbuffers::FlatBufferBuilder &fbb_;
  flatbuffers::uoffset_t start_;
  void add_data(flatbuffers::Offset<flatbuffers::Vector<uint8_t>> data) {
    fbb_.AddOffset(Buffer::VT_DATA, data);
  }
  explicit BufferBuilder(flatbuffers::FlatBufferBuilder &_fbb)
        : fbb_(_fbb) {
    start_ = fbb_.StartTable();
  }
  flatbuffers::Offset<Buffer> Finish() {
    const auto end = fbb_.EndTable(start_);
    auto o = flatbuffers::Offset<Buffer>(end);
    return o;
  }
};

inline flatbuffers::Offset<Buffer> CreateBuffer(
    flatbuffers::FlatBufferBuilder &_fbb,
    flatbuffers::Offset<flatbuffers::Vector<uint8_t>> data = 0) {
  BufferBuilder builder_(_fbb);
  builder_.add_data(data);
  return builder_.Finish();
}

inline flatbuffers::Offset<Buffer> CreateBufferDirect(
    flatbuffers::FlatBufferBuilder &_fbb,
    const std::vector<uint8_t> *data = nullptr) {
  if (data) { _fbb.ForceVectorAlignment(data->size(), sizeof(uint8_t), 16); }
  auto data__ = data ? _fbb.CreateVector<uint8_t>(*data) : 0;
  return tflite::CreateBuffer(
      _fbb,
      data__);
}

flatbuffers::Offset<Buffer> CreateBuffer(flatbuffers::FlatBufferBuilder &_fbb, const BufferT *_o, const flatbuffers::rehasher_function_t *_

struct MetadataT : public flatbuffers::NativeTable {
  typedef Metadata TableType;
  std::string name{};
  uint32_t buffer = 0;
};
```

- shema_generated.h 파일에서 찾을 수 있는 Bufferbuilder 구조체와 CreateBuffer 함수, CreateBufferDirect 함수는 tflite 파일의 내용을 바탕으로 버퍼를 생성하는 것으로 예상됨
- 구체적인 추적을 통해 비어있는 버퍼의 사용 유무를 확인하고 사용하지 않는다면 tflite 파일에서 제거하여 생성하도록 시도할 수 있음
- 하지만, 시간 부족으로 인해 버퍼의 실제 사용을 확인하지 못하였음

감사합니다