

On-device AI 용 Neural Network Trainer 개발

- Pave the way to On-device Multi-modal LLMs -

창의적통합설계1 최종 발표

2024.12.20

Team A

송우경

장민혁

이현우

Contents

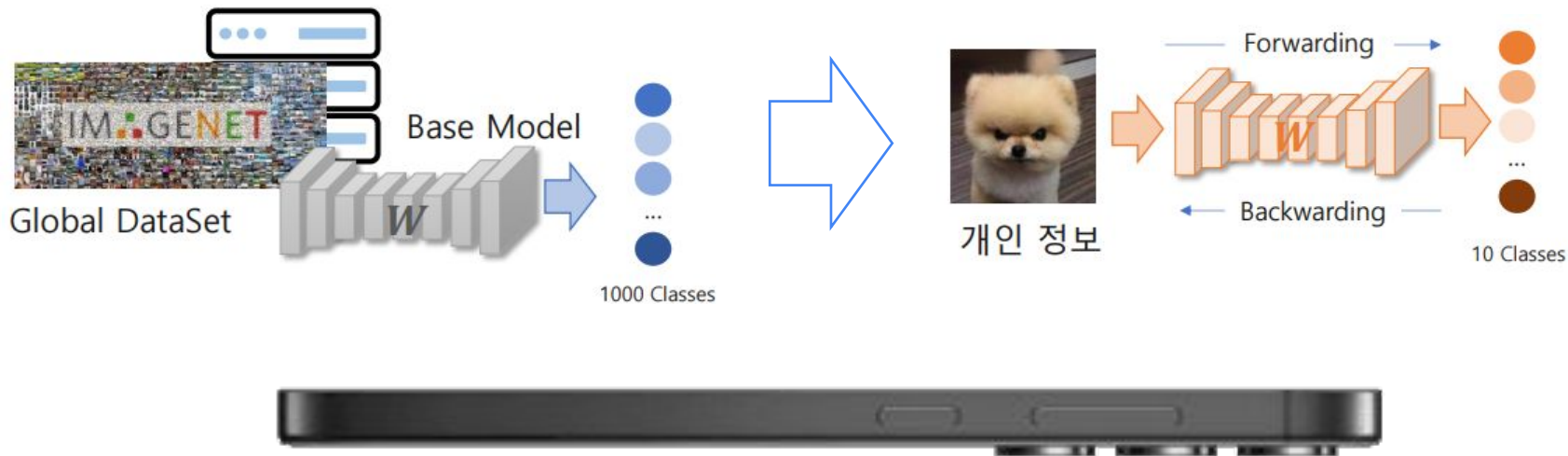
- Overview
- Goal/Problem & Requirement & Approach
- Development Environment
- Architecture
- Implementation Spec
- Results
- Demo
- Division and Assignment of Work

Overview

“Neural Network Trainer”

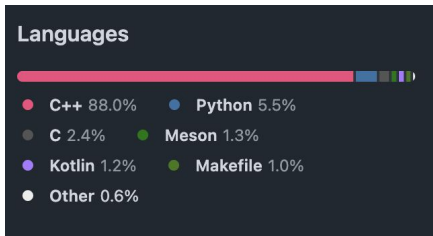


NNTrainer: Lightweight on-device AI framework



What is NNTrainer?

- “A Software Framework for training Neural Network models on devices.”
 - Open source, maintained by Samsung Electronics
 - Mostly C++ code with Doxygen
 - Works nicely with NNStreamer
 - **Addresses the system software aspect (i.e., how memory and computation are allocated and scheduled) to optimize**
 - **proactive swap, reordering procedures to minimize peak memory consumption**
 - **Any optimized algorithms can be implemented over NNTrainer**
- Focuses on fine-tuning over full-training in on-device environment
- Supports fair amount of layers/optimizers/loss functions/... (We call them by “operations” or “ops”)
 - But still many are not implemented yet



Contribute to NNTrainer By implementing 5 ops

Requirement

Some vital ops are missing:

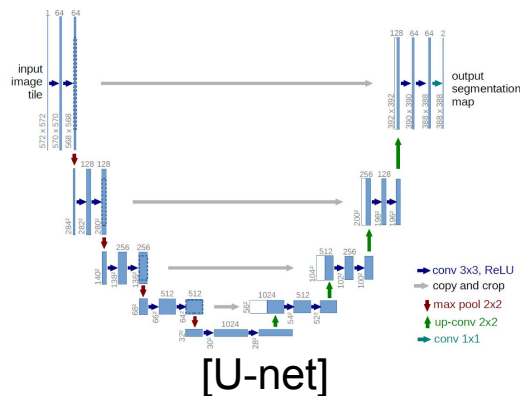
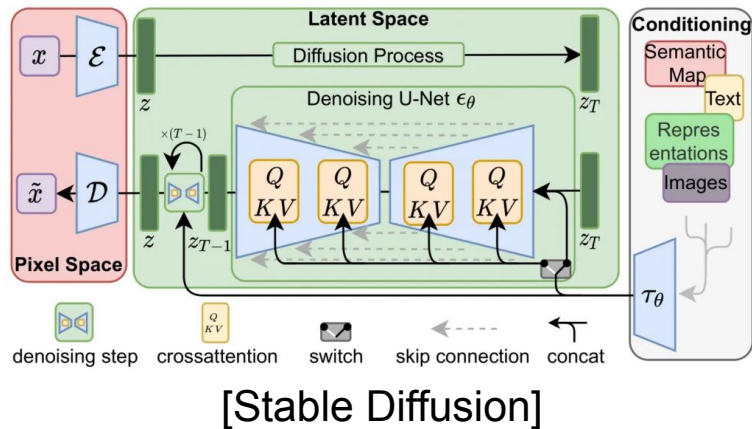
- **ConvTranspose2d**

- Diffusion model 의 기반이 되는 U-net 의 “up-conv” 단계에 사용됨

=> Enables on-device diffusion models
and thus **text-to-image multimodal LLMs**
(Dall-E, Stable Diffusion)



- Ops used in original diffusion model but not implemented in NNTrainer: **LinearLR**, **RMSProp**
- Other unimplemented but popular ops: **CosineAnnealingLR**, **AdamW**



Requirement & Approach

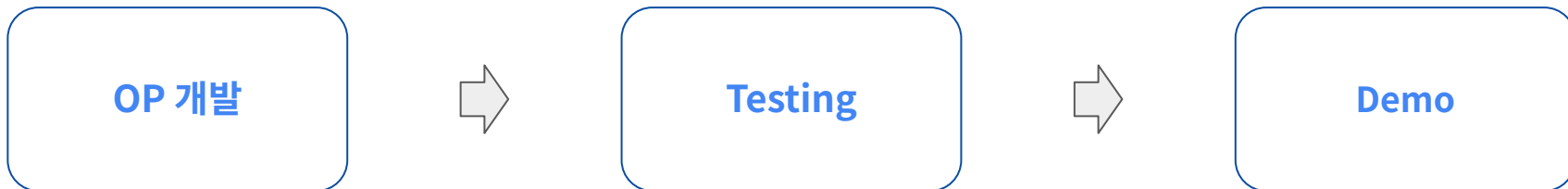
1. U-net 구현을 목표로!

- a. Implement [ConvTranspose2d](#) (and [LinearLR](#), [RMSProp](#))
- b. With performance and memory efficiency in mind! (On-device 환경을 생각한다.)

2. 그 전에, 연습삼아!

- a. Implement [CosineAnnealingLR](#), [AdamW](#)

개발 사이클



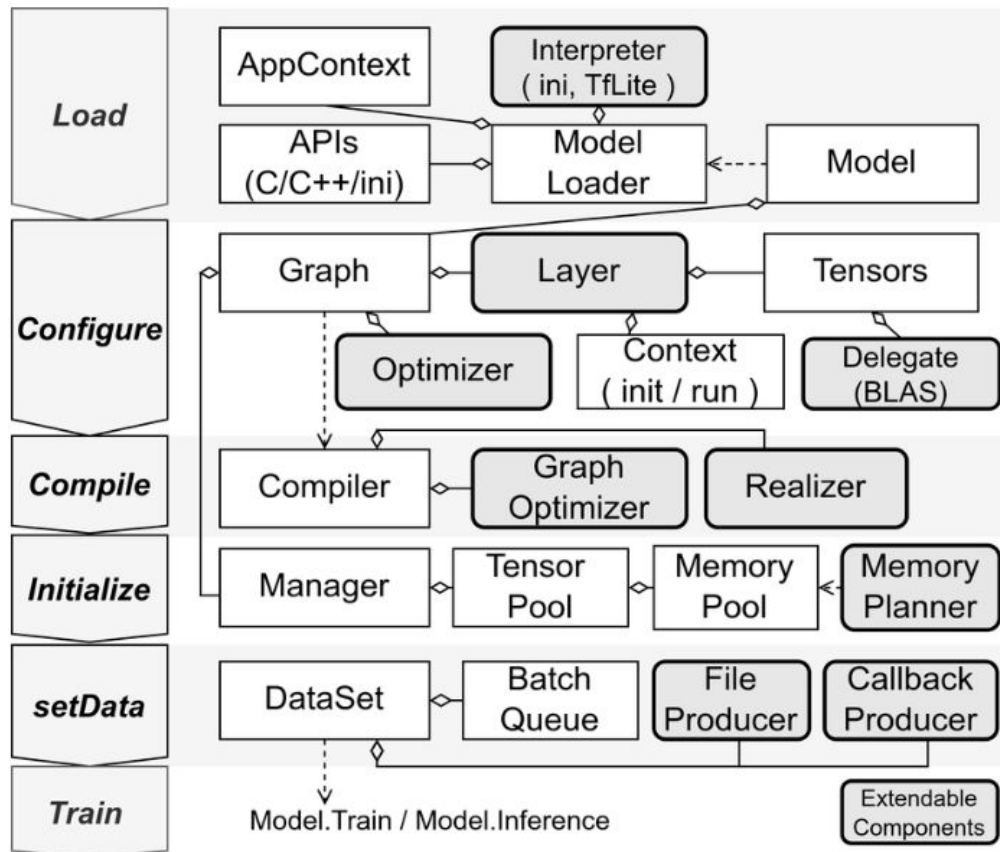
Development Environment

- Local machine with Ubuntu 22.04+
- C++
- pbuilder for packaging
- NNTrainer repo: [nntrainer/docs/getting-started.md](https://nntrainer.org/docs/getting-started.md) at main

Dependency list

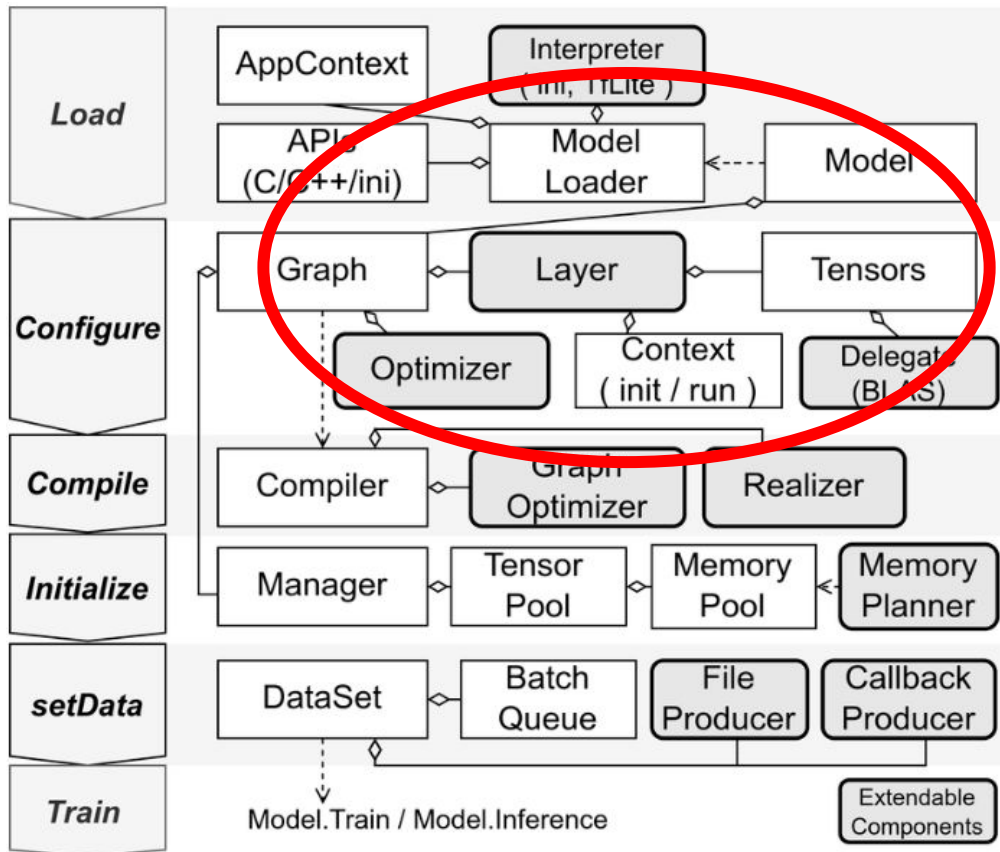
: gcc/g++, meson, libopenblas-dev, tensorflow-lite, libinparser,
libjsonparser, libjsoncpp, libcurl3, libgtest

Architecture



Abstract architecture of NNTrainer

Architecture



주로 contribute 하게 될 부분

- Model Loader, Model
- Layer
- Optimizer
- ...

Architecture : NNTrainer C++ API를 중심으로

```
int main(int argc, char *argv[]) {
    auto model = create_model();

    model->setProperty({"batch_size=" + std::to_string(batch_size),
                      "epochs=" + std::to_string(epochs),
                      "save_path=" + save_path});

    auto optimizer = ml::train::createOptimizer("SGD", {"learning_rate=" + std::to_string(learning_rate)});
    model->setOptimizer(std::move(optimizer));

    int status = model->compile();
    status = model->initialize();

    auto random_generator = getRandomDataGenerator();
    auto train_dataset = ml::train::createDataset(
        ml::train::DatasetType::GENERATOR, dataset_cb, random_generator.get());

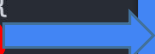
    model->setDataset(ml::train::DatasetModeType::MODE_TRAIN, std::move(train_dataset));

    model->train();

    return status;
}
```

Architecture : NNTrainer C++ API를 중심으로

```
int main(int argc, char *argv[]) {  
    auto model = create_model();  
  
    model->setProperty({"batch_size=" + std::to_string(10),  
                      "epochs=" + std::to_string(10),  
                      "save_path=" + std::string("my_model.h5")});  
  
    auto optimizer = ml::train::createOptimizer("adam");  
    model->setOptimizer(std::move(optimizer));  
  
    int status = model->compile();  
    status = model->initialize();  
  
    auto random_generator = getRandomDataGenerator();  
    auto train_dataset = ml::train::createDataset(  
        ml::train::DatasetType::GENERATOR, dataset_cb, random_generator.get());  
  
    model->setDataset(ml::train::DatasetModeType::MODE_TRAIN, std::move(train_dataset));  
  
    model->train();  
  
    return status;  
}
```



```
std::unique_ptr<ml::train::Model> create_model() {  
    std::unique_ptr<ml::train::Model> model =  
        ml::train::createModel(ml::train::ModelType::NEURAL_NET, {"loss=mse"});  
  
    model->addLayer(  
        ml::train::createLayer(  
            "input", {"input_shape=1:1:10"}  
        );  
    );  
  
    model->addLayer(  
        ml::train::createLayer("fully_connected", {"unit=5", "activation=softmax"}));  
  
    return model;  
}
```

실제 Layer example (Conv2dLayer)

```
class Conv2DLayer : public LayerImpl {  
    void forwarding(RunLayerContext &context, bool training) override;  
    void calcDerivative(RunLayerContext &context) override;  
    void calcGradient(RunLayerContext &context) override;  
    bool supportBackwarding() const override { return true; }  
}
```

Layer에 대한 Forward, Backward 등을 구현해야 함!

Op 들을 구현하면서 지켜야 할 contribution guide

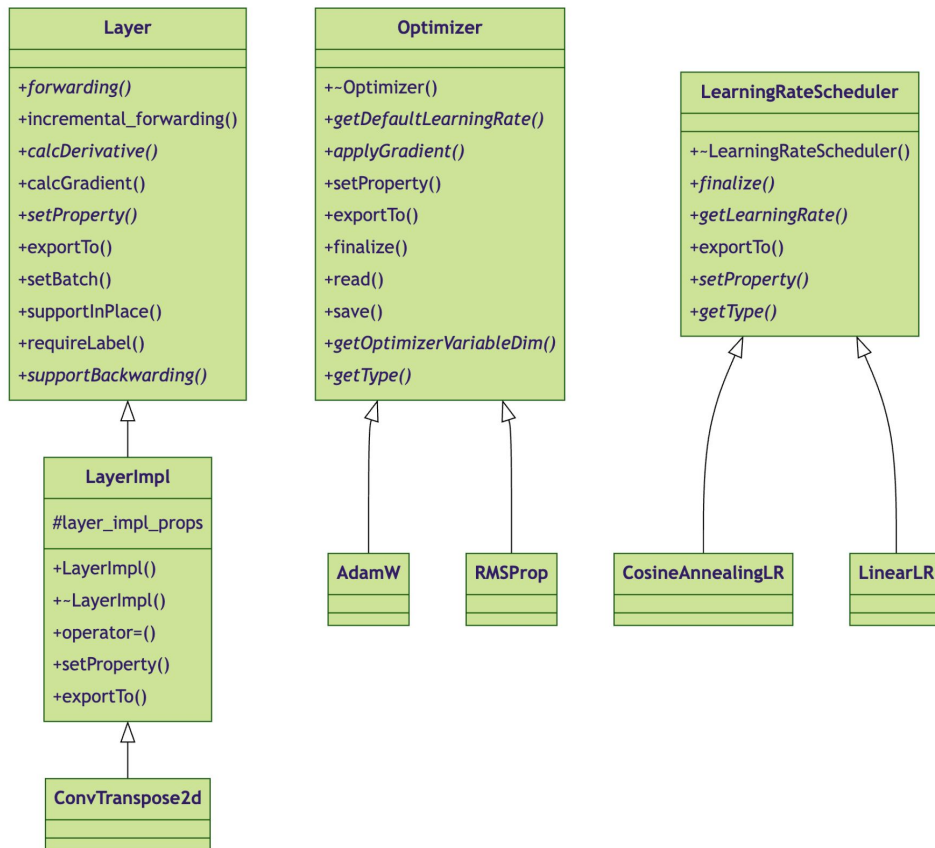
1. Optimizer, scheduler, layer 등 구현
2. 각 feature에 대한 unit test case (GoogleTest or SSAT) 구현
3. 구현한 feature 를 활용하는 simple example (optional) 구현
4. Doxygen을 통한 문서 작성

* Small commits and multiple PRs (회사 담당자님께서 부탁하신 점)

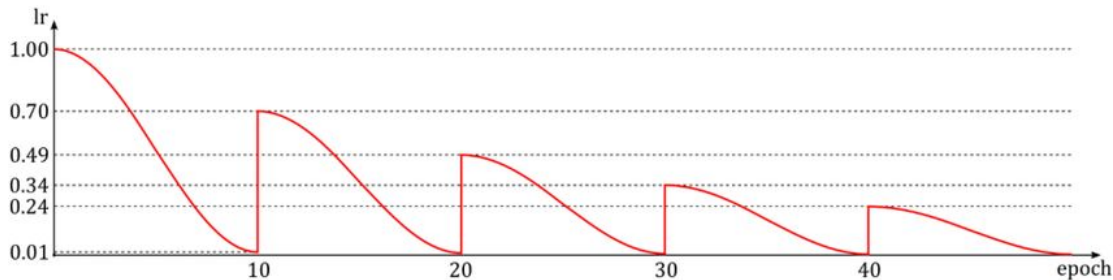
이번 프로젝트 동안 구현할 Op

1. **CosineAnnealingLR** scheduler
2. **AdamW** Optimizer
3. **ConvTranspose2d** Layer
4. **LinearLR** scheduler
5. **RMSProp** optimizer

Implementation Spec : Class Diagram



CosineAnnealingLR Scheduler



Cosine 함수 기반으로 학습률을 조정하는 **러닝 레이트 스케줄러**

해당 스케줄러를 `nntainer`의 `nntainer/optimizers` 에 구현!

AdamW Optimizer

```
for  $t = 1$  to ... do
  if maximize :
     $g_t \leftarrow -\nabla_{\theta} f_t(\theta_{t-1})$ 
  else
     $g_t \leftarrow \nabla_{\theta} f_t(\theta_{t-1})$ 
   $\theta_t \leftarrow \theta_{t-1} - \gamma \lambda \theta_{t-1}$ 
   $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$ 
   $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$ 
   $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ 
   $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ 
  if amsgrad
     $\widehat{v}_t^{max} \leftarrow \max(\widehat{v}_t^{max}, \widehat{v}_t)$ 
     $\theta_t \leftarrow \theta_t - \gamma \widehat{m}_t / (\sqrt{\widehat{v}_t^{max}} + \epsilon)$ 
  else
     $\theta_t \leftarrow \theta_t - \gamma \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ 
```

Weight decay를 가중치에 직접 적용하는 adam 변형 버전 AdamW optimizer

해당 optimizer를 [nntrainer](#)의 [nntrainer/optimizers](#) 에 구현!

ConvTranspose2d Layer

Diagram illustrating the ConvTranspose2d layer operation. The diagram shows four examples of 2D matrix multiplication, where a 2x2 input matrix is multiplied by a 3x3 kernel matrix to produce a 4x4 output matrix.

Example 1:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \times \begin{bmatrix} 1 & 2 & 3 \\ 2 & 2 & 1 \\ 3 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 & 0 \\ 2 & 2 & 1 & 0 \\ 3 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Example 2:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \times \begin{bmatrix} 1 & 2 & 3 \\ 2 & 2 & 1 \\ 3 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 2+2 & 3+4 & 6 \\ 2 & 2+4 & 1+4 & 2 \\ 3 & 2+6 & 1+4 & 2 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Example 3:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \times \begin{bmatrix} 1 & 2 & 3 \\ 2 & 2 & 1 \\ 3 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 4 & 7 & 6 \\ 2+3 & 6+6 & 5+9 & 2 \\ 3+6 & 8+6 & 5+2 & 2 \\ 9 & 6 & 3 & 0 \end{bmatrix}$$

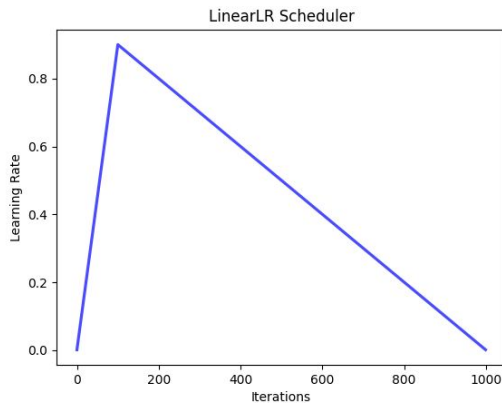
Example 4:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \times \begin{bmatrix} 1 & 2 & 3 \\ 2 & 2 & 1 \\ 3 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 4 & 7 & 6 \\ 5 & 12+4 & 14+8 & 2+12 \\ 9 & 14+8 & 7+8 & 2+4 \\ 9 & 6+12 & 3+8 & 4 \end{bmatrix}$$

입력 이미지를 업샘플링하는 레이어인 ConvTranspose2d layer

해당 layer를 `nntrainer`의 `nntrainer/layers` 에 구현!

LinearLR Scheduler



학습률을 일정한 비율로 선형적으로 감소시키는 스케줄러

해당 scheduler를 `nntrainer`의 `nntrainer/optimizers` 에 구현!

RMSProp Optimizer

For each Parameter w^j

(j subscript dropped for clarity)

$$\nu_t = \rho \nu_{t-1} + (1 - \rho) * g_t^2$$

$$\Delta \omega_t = -\frac{\eta}{\sqrt{\nu_t + \epsilon}} * g_t$$

$$\omega_{t+1} = \omega_t + \Delta \omega_t$$

η : Initial Learning rate

ν_t : Exponential Average of squares of gradients

g_t : Gradient at time t along ω^j


Gradient에 따라 학습률을 조정하는 적응형 학습률 기반의 optimizer

해당 optimizer를 `nntrainer`의 `nntrainer/optimizers` 에 구현!

Current Status

1. CosineAnnealingLR

- <https://github.com/nnstreamer/nntainer/pull/2754>

 Merged

2. AdamW

- <https://github.com/nnstreamer/nntainer/pull/2780>

 Merged

3. Conv2dTranspose

- <https://github.com/nnstreamer/nntainer/pull/2781>

 Merged

4. LinearLR

- <https://github.com/nnstreamer/nntainer/pull/2793>

 Merged

5. RMSProp

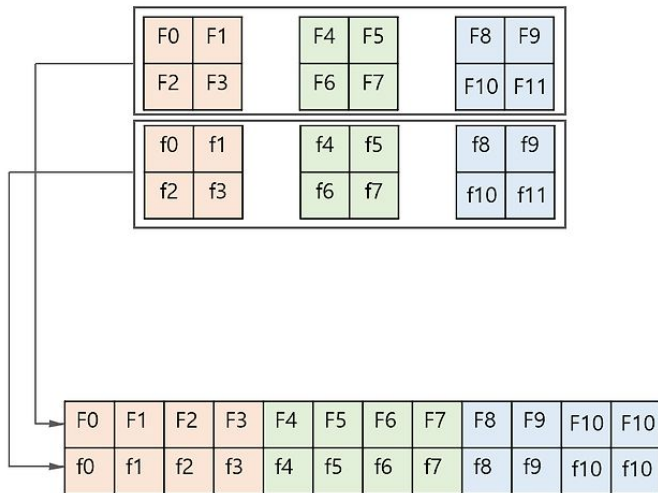
- <https://github.com/nnstreamer/nntainer/pull/2587>

Conv2dTranspose Optimization

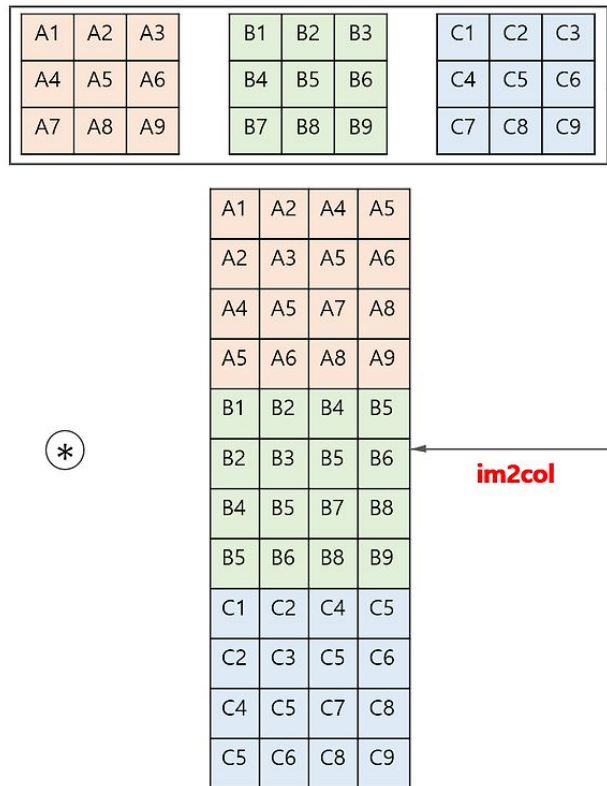
- naive convolution은 for문 돌면서 수많은 branch 만남.
pipelined CPU 성능 악화 -> **im2col** 적용
 - input 각 value를 곱해지는 filter 원소 위치에 맞게 행렬에 재배치
 - **matmul**로 환원된다.
 - 실제 구현은: im2col, col2im, reshape 필요
 - filter dim: (K,C,R,S) -> (K, CRS)
 - input dim: (N,C,H,W) -> (1,C,H,W) -> (CRS, OH*OW)
 - output: (K,OH*OW)
 - backward도 어차피 matmul이므로 최종 input derivative에 **col2im**

Current Status : How we optimized Conv2dTranspose

필터 (채널이 3인 필터 2개)

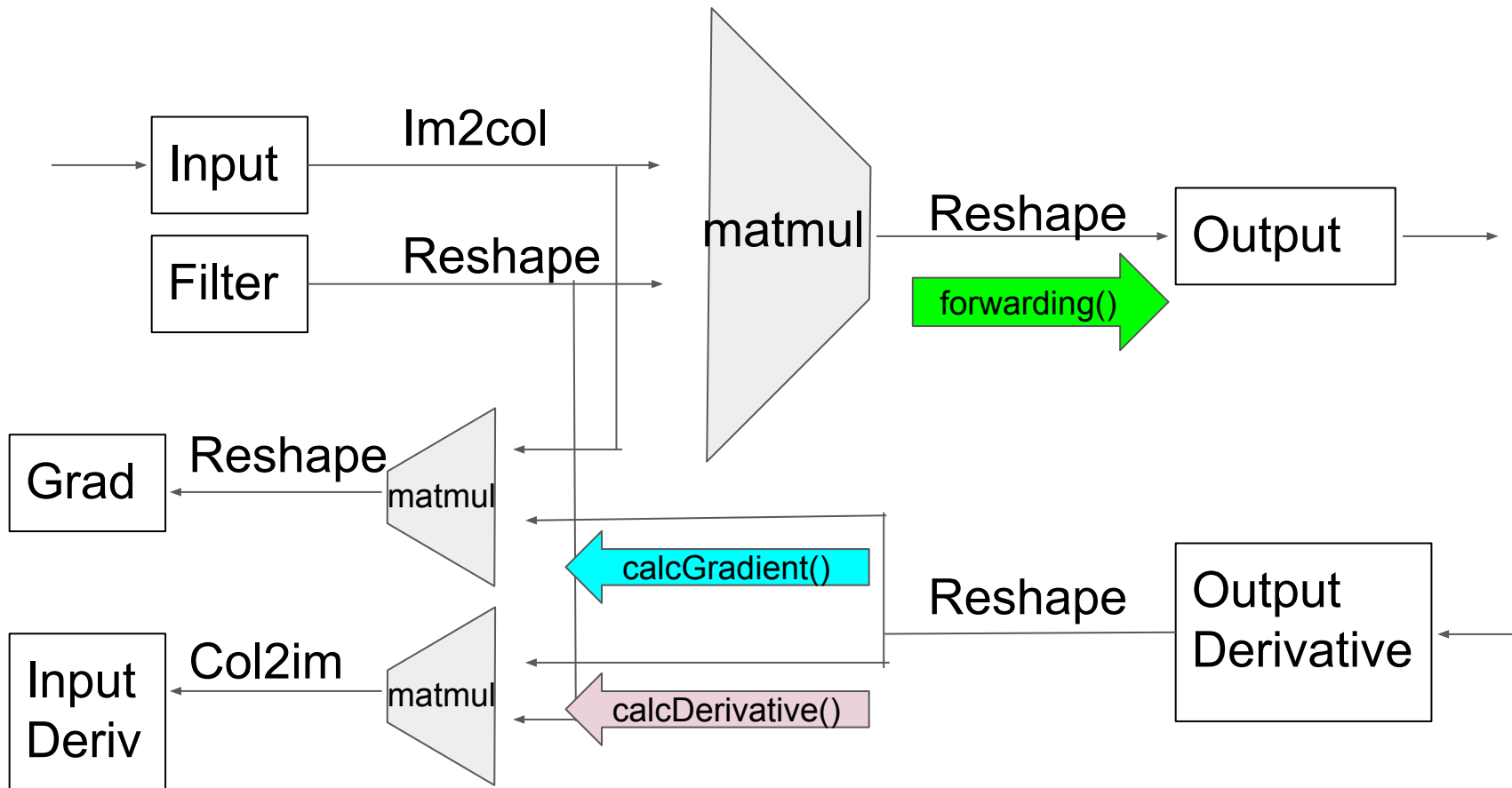


입력 데이터 (채널이 3인 데이터 1개)



*

Current Status : How we optimized Conv2dTranspose



Results : Layer 사용

```
model->addLayer(ml :: train :: createLayer("batch_normalization",
{
    withKey("activation", "relu"),
}));

model->addLayer(ml :: train :: createLayer("conv2dtranspose",
{
    withKey("filters", image_channels),
    withKey("kernel_size", {4, 4}),
    withKey("stride", {2, 2}),
    withKey("padding", 1),
}));

model->addLayer(ml :: train :: createLayer("batch_normalization",
{
    withKey("name", "recon_x"),
    withKey("activation", "sigmoid"),
}));
```

구현한 Layer는 위와 같이 model->addLayer 를 통해 추가함으로써, cpp로 구현된 모델 구성 가능

Results : Layer 사용

`model→train();`



```
void forwarding(RunLayerContext &context, bool training) override;
```

```
void calcDerivative(RunLayerContext &context) override;
```

```
void calcGradient(RunLayerContext &context) override;
```

`model→inference`

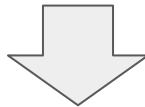


```
void forwarding(RunLayerContext &context, bool training) override;
```

적절한 함수 호출을 통해 train과 inference 과정 수행

Results : Scheduler 사용

```
auto lr_scheduler = ml::train::createLearningRateScheduler("Cosine");  
lr_scheduler->setProperty({"MaxLearningRate=0.0003"});  
lr_scheduler->setProperty({"MinLearningRate=0.00001"});  
lr_scheduler->setProperty({"DecaySteps=15"});  
optimizer->setLearningRateScheduler(std::move(lr_scheduler));
```



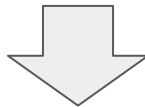
```
virtual double getLearningRate(size_t iteration) override;
```

Scheduler 또한, scheduler 객체를 생성하고 optimizers->setLearningRateScheduler를 통해 설정가능

optimizer에서 적절히 scheduler의 함수 호출

Results : Optimizer 사용

```
auto optimizer = ml::train::createOptimizer("adamW", {"learning_rate=0.001"});  
model->setOptimizer(std::move(optimizer));
```



```
double getDefaultLearningRate() const override { return 0.001; }  
  
void applyGradient(RunOptimizerContext &context) override;
```

Optimizer 또한, optimizer 객체를 생성하고 model->setOptimizer를 통해 설정 가능
model 실행 시, optimizer의 함수를 적절히 호출

Results : 실제 모델 학습 모습

Layer name	Layer type	Output dimension	Input layer
input0	input	1:1:28:28	
conv2d1	conv2d	1:4:14:14	input0
batch_normalization	batch_normalization	1:4:14:14	conv2d1
batch_normalization	activation	1:4:14:14	batch_normalization
conv2d3	conv2d	1:8:7:7	batch_normalization
batch_normalization	batch_normalization	1:8:7:7	conv2d3
batch_normalization	activation	1:8:7:7	batch_normalization
h	flatten	1:1:1:392	batch_normalization
h/generated_out_0	multiout	1:1:1:392	h
fc2	fully_connected	1:1:1:16	h/generated_out_0
fc2/generated_out_0	multiout	1:1:1:16	fc2
fc1	fully_connected	1:1:1:16	h/generated_out_0
fc1/generated_out_0	multiout	1:1:1:16	fc1
reparametrize	bottleneck	1:1:1:16	fc1/generated_out_0 fc2/generated_out_0
fully_connected5	fully_connected	1:1:1:392	reparametrize
fully_connected5/ac	activation	1:1:1:392	fully_connected5
reshape6	reshape	1:8:7:7	fully_connected5/ac
conv2dtranspose7	conv2dtranspose	1:4:14:14	reshape6
batch_normalization	batch_normalization	1:4:14:14	conv2dtranspose7
batch_normalization	activation	1:4:14:14	batch_normalization
conv2dtranspose9	conv2dtranspose	1:1:28:28	batch_normalization
recon_x	activation	1:1:28:28	conv2dtranspose9
vae_loss10	vae_loss	1:1:28:28	recon_x fc1/generated_out_0 fc2/generated_out_0

이렇게 실제 Layer, Optimizer, scheduler를
직접 사용하여 모델을 빌드할 수 있음

<- VAE 모델 레이어 구조

Results : 실제 모델 학습 모습

```
#1/15 - Training Loss: 3753.66 >> [ Accuracy: 0% - Validation Loss : 197177 ]
#2/15 - Training Loss: 2751.18 >> [ Accuracy: 0% - Validation Loss : 21264.3 ]
#3/15 - Training Loss: 2115.3 >> [ Accuracy: 0% - Validation Loss : 9363.32 ]
#4/15 - Training Loss: 1706.62 >> [ Accuracy: 0% - Validation Loss : 5956.27 ]
#5/15 - Training Loss: 1432.92 >> [ Accuracy: 0% - Validation Loss : 4417.4 ]
#6/15 - Training Loss: 1238.51 >> [ Accuracy: 0% - Validation Loss : 3584.28 ]
#7/15 - Training Loss: 1093.09 >> [ Accuracy: 0% - Validation Loss : 3039.6 ]
#8/15 - Training Loss: 977.455 >> [ Accuracy: 0% - Validation Loss : 2634.01 ]
#9/15 - Training Loss: 885.587 >> [ Accuracy: 0% - Validation Loss : 2320.18 ]
#10/15 - Training Loss: 808.74 >> [ Accuracy: 0% - Validation Loss : 2075.16 ]
#11/15 - Training Loss: 745.437 >> [ Accuracy: 0% - Validation Loss : 1860.18 ]
#12/15 - Training Loss: 689.879 >> [ Accuracy: 0% - Validation Loss : 1672.6 ]
#13/15 - Training Loss: 642.173 >> [ Accuracy: 0% - Validation Loss : 1506.49 ]
#14/15 - Training Loss: 600.107 >> [ Accuracy: 0% - Validation Loss : 1359.64 ]
#15/15 - Training Loss: 562.951 >> [ Accuracy: 0% - Validation Loss : 1228.59 ]
```

이렇게 실제 Layer, Optimizer, scheduler를
직접 사용하여 모델을 빌드할 수 있음

<- VAE 모델 학습되는 모습

Results : NNTrainer 사용

- 프로젝트 기간 동안 구현한 OP들 이외에도, 이미 구현되어 있는 OP들이 다수 존재
- Resnet, PicoGPT, LLaMA 등의 예시 model들이 구현되어 있음
- 누구나 자유롭게 NNTrainer를 이용해서 원하는 model 빌드 가능

Installation 방법 :

- <https://github.com/nstreamer/nntainer/blob/main/docs/getting-started.md>

Model 생성 튜토리얼 :

- <https://github.com/nstreamer/nntainer/blob/main/docs/how-to-create-model.md>

Results : 회사평가

NNTrainer 오픈소스 프로젝트에 대한 이해	NNTrainer의 구조와 기능의 이해	NNTrainer의 구조와 계층, 메모리 관리에 대해 설명할 수 있다.
	NNTrainer에 필요한 개발 프로젝트 구현	NNTrainer에 필요한 기능을 이해하고 활용가능한 새 프로젝트를 기획했다.
	오픈소스 기여 절차에 대한 이해	기존 오픈소스 프로젝트의 요구사항을 파악하여 효과적으로 기여할 수 있다.

Results : 회사평가

Basic feature 구현	Scheduler, Optimizer 추가	2종류 이상의 scheduler, optimizer를 추가했다.
	Unit test 작성	구현한 활성 함수 기능 전반(80% 이상)에 대한 test를 작성했다.

Results : 회사평가

Advanced feature 구현	Layer 추가	Forward와 backward pass를 포함하여, Layer의 핵심 기능을 제대로 구현했다.
	Unit test 작성	구현한 Layer 기능 전반(80% 이상)에 대한 test를 작성했다.

Demo

- NNTrainer에 새롭게 개발한 Conv2dTranspose 레이어를 이용해 VAE 구현
- 파이토치에서도 VAE 구현
- NNTrainer 버전과 파이토치 버전을 상호 비교 진행

자세한 내용은 동영상에서..

Division and Assignment of Work

[구현할 것]

CosineAnnealingLR scheduler

AdamW Optimizer

ConvTranspose2D Layer

LinearLR scheduler

RMSProp optimizer

VAE model

[구현 중 해야 할 일]

OP 로직 구현

Unit test 구현

최적화 수행

디버깅 및 리팩토링

모델 학습

모델 테스트

[담당]

	CosAnneal LR	AdamW	ConvTrans	LinearLR	RMSProp	VAE
로직 구현	이현우, 장민혁	장민혁, 송우경	송우경, 장민혁	이현우, 장민혁	장민혁, 송우경	장민혁, 이현우
Unit test 구현	이현우	이현우, 장민혁	이현우, 송우경	이현우	이현우, 장민혁	x
최적화 수행	이현우, 송우경	장민혁, 송우경	송우경	이현우, 송우경	장민혁, 송우경	x
디버깅 및 리팩토링	이현우	장민혁	송우경	이현우	장민혁	x
모델 학습	x	x	x	x	x	장민혁, 이현우
모델 테스트	x	x	x	x	x	장민혁, 이현우

감사합니다

발표자

이현우 | dlgusdn0414@snu.ac.kr