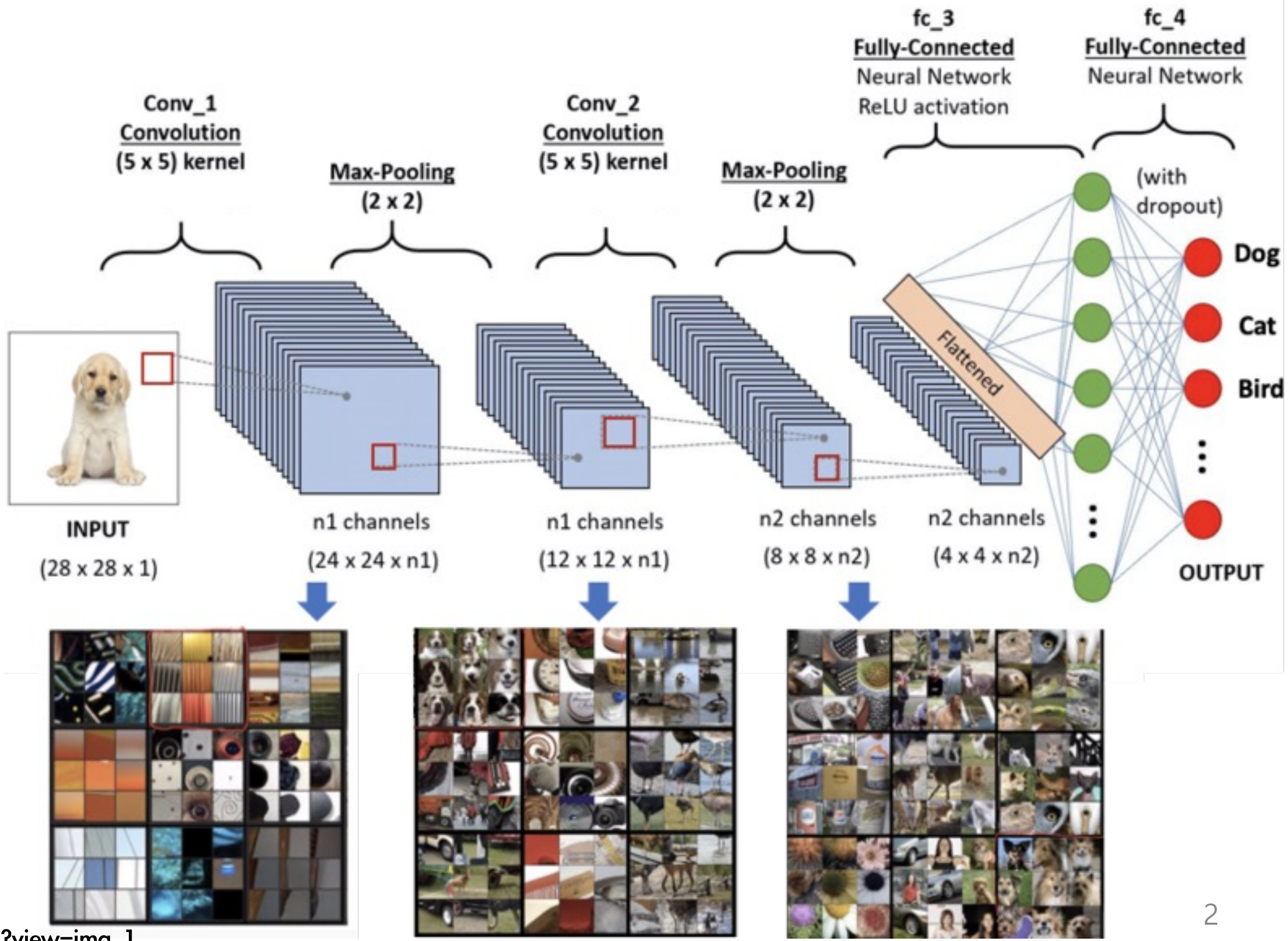
CNN Architecture & Its Examples

CNN Examples

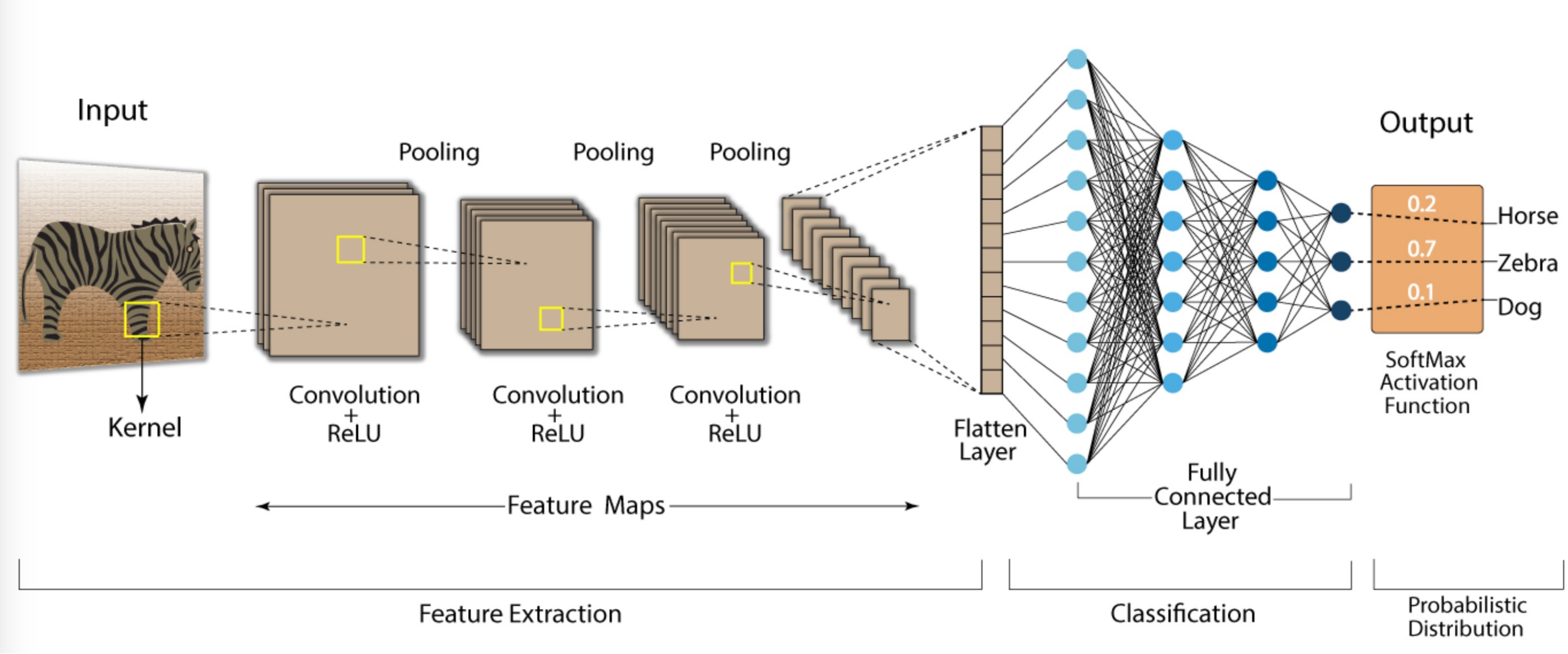
- ◈A CNN Example
  - – Conv2D Layer
  - – Max-Pooling Layer
  - – Conv2D Layer
  - – Max-Pooling Layer
  - – Flatten Layer
  - – Fully-Connected (Dense or Linear) Layer
  - – Fully-Connected (Dense or Linear) Layer



[Source] https://m.blog.naver.com/bananacco/221928562116?view=img_1

CNN Examples

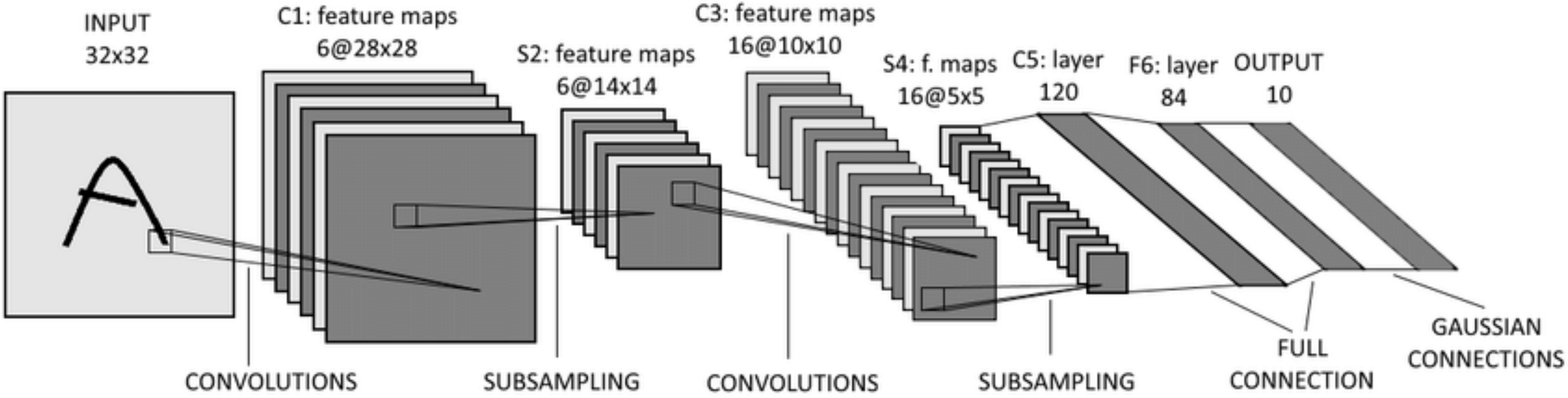
◈A CNN Example



[Source] https://developersbreach.com/convolution-neural-network-deep-learning/

CNN Examples - LeNet-5

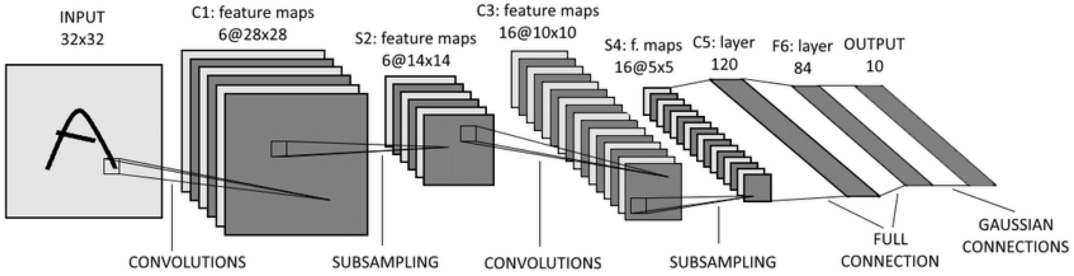
- LeNet-5 1998 (LeCun et al.) - 1/4
  - The whole architecture is [Input - CONV - POOL - CONV - POOL - CONV - FC1 - FC2 - Output]



CNN Examples - LeNet-5

◈LeNet-5 1998 (LeCun et al.) - 2/4

- C1 (Conv2D Layer): 6 filters of size 5x5
  - Padding: 0, Stride: 1
  - Input: 1x32x32 → output: 6x28x28
  - Number of parameters: 1x5x5x6+6=156



$$\boxed{(\boldsymbol{O_h}, \boldsymbol{O_w}) = \left(\frac{\boldsymbol{I_h} - \boldsymbol{F_h} + \boldsymbol{2P_h}}{\boldsymbol{S_h}} + \mathbf{1}, \frac{\boldsymbol{I_w} - \boldsymbol{F_w} + \boldsymbol{2P_w}}{\boldsymbol{S_w}} + \mathbf{1}\right) = \left(\frac{\mathbf{32} - \mathbf{5} + \mathbf{2}\times\mathbf{0}}{\mathbf{1}} + \mathbf{1}, \frac{\mathbf{32} - \mathbf{5} + \mathbf{2}\times\mathbf{0}}{\mathbf{1}} + \mathbf{1}\right) = (\mathbf{28}, \mathbf{28})}$$

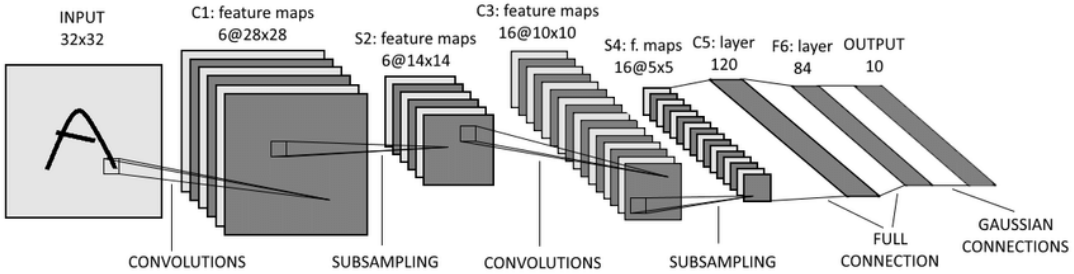
- S2 (Subsampling or Pooling Layer): 6 kernel of size 2x2
  - Stride: 2
  - Input: 6x28x28 → output: 6x14x14

$$\boxed{(\boldsymbol{O_h}, \boldsymbol{O_w}) = \left(\frac{\boldsymbol{I_h} - \boldsymbol{F_h}}{\boldsymbol{S_h}} + \mathbf{1}, \frac{\boldsymbol{I_w} - \boldsymbol{F_w}}{\boldsymbol{S_w}} + \mathbf{1}\right) = \left(\frac{\mathbf{28} - \mathbf{2}}{\mathbf{2}} + \mathbf{1}, \frac{\mathbf{28} - \mathbf{2}}{\mathbf{2}} + \mathbf{1}\right) = (\mathbf{14}, \mathbf{14})}$$

CNN Examples - LeNet-5

◈LeNet-5 1998 (LeCun et al.) - 3/4

- C3 (Conv2D Layer): 16 filters of size 5x5
  - Padding: 0, Stride: 1
  - Input: 6x14x14 → output: 16x10x10
  - Number of parameters: 6x5x5x16+16=2,416



$$\boxed{(\boldsymbol{O_h}, \boldsymbol{O_w}) = \left(\frac{\boldsymbol{I_h - F_h + 2P_h}}{\boldsymbol{S_h}} + \mathbf{1}, \frac{\boldsymbol{I_w - F_w + 2P_w}}{\boldsymbol{S_w}} + \mathbf{1}\right) = \left(\frac{\mathbf{14 - 5 + 2\times0}}{\mathbf{1}} + \mathbf{1}, \frac{\mathbf{14 - 5 + 2\times0}}{\mathbf{1}} + \mathbf{1}\right) = (\mathbf{10}, \mathbf{10})}$$

- S4 (Subsampling or Pooling Layer): 6 filters of size 2x2
  - Stride: 2
  - Input: 16x10x10 → output: 16x5x5

$$\boxed{(\boldsymbol{O_h}, \boldsymbol{O_w}) = \left(\frac{\boldsymbol{I_h - F_h}}{\boldsymbol{S_h}} + \mathbf{1}, \frac{\boldsymbol{I_w - F_w}}{\boldsymbol{S_w}} + \mathbf{1}\right) = \left(\frac{\mathbf{10 - 2}}{\mathbf{2}} + \mathbf{1}, \frac{\mathbf{10 - 2}}{\mathbf{2}} + \mathbf{1}\right) = (\mathbf{5}, \mathbf{5})}$$

CNN Examples - LeNet-5

◈LeNet-5 1998 (LeCun et al.) - 4/4

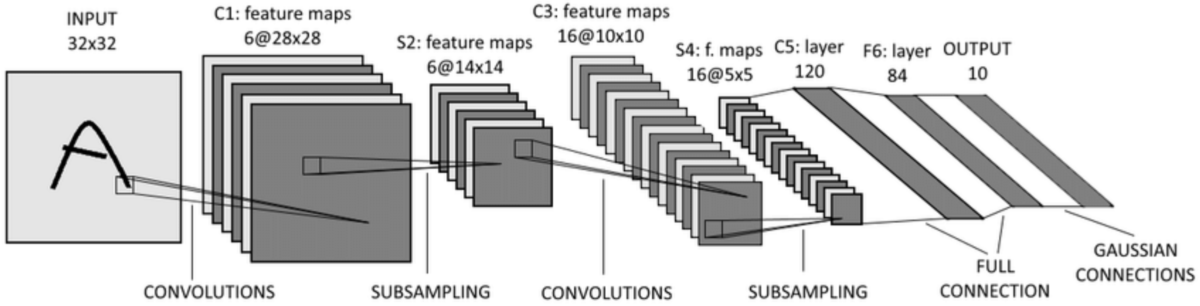
– C5 (Conv2D Layer): 120 filters of size 5x5 ← Flatten Layer

- Padding: 0, Stride: 1
- Input: 16x5x5 → output: 120x1x1
- Number of parameters: 16x5x5x120+120 = 48,120

$$(O_h, O_w) = \left(\frac{I_h - F_h + 2P_h}{S_h} + 1, \frac{I_w - F_w + 2P_w}{S_w} + 1\right) = \left(\frac{5 - 5 + 2\times 0}{1} + 1, \frac{5 - 5 + 2\times 0}{1} + 1\right) = (1, 1)$$

– F6 (FC Layer)

- Input: 120 → output: 84
- Number of parameters: 120x84+84 = 10,164

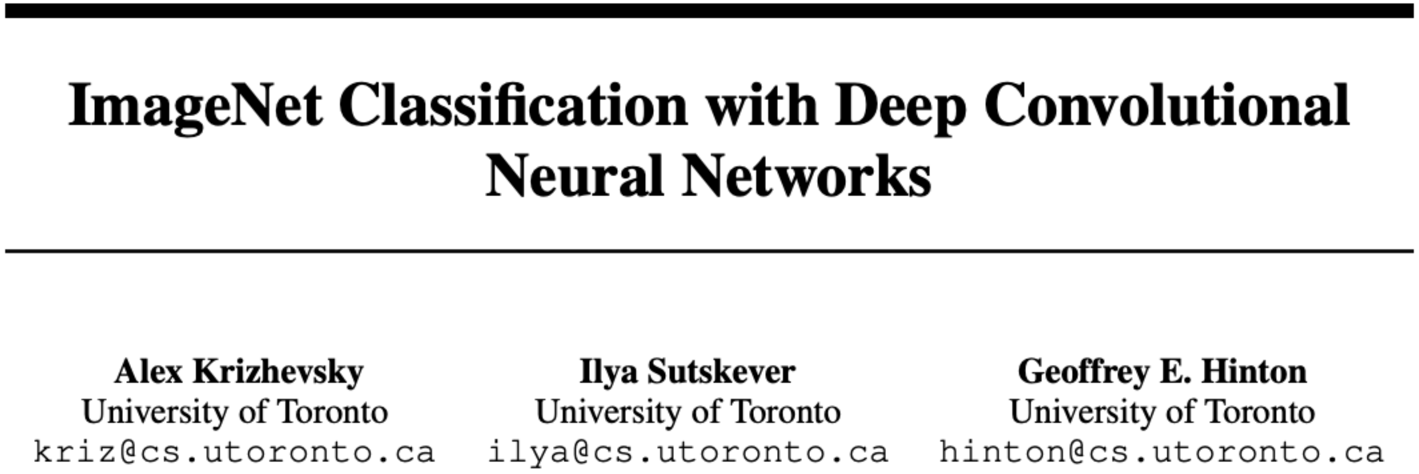


– OUTPUT (FC Layer)

- Input: 84 → output: 10
- Number of parameters: 84x10+10 = 850

CNN Examples - AlexNet

- AlexNet (Alex Krizhevsky et al.)
  - Paper title
    - ImageNet Classification with Deep Convolutional Neural Networks, 2012

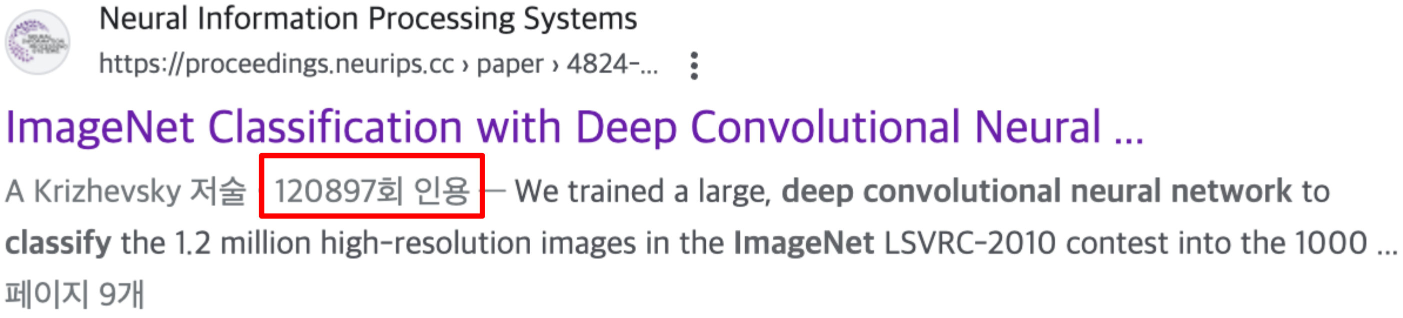


ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca



Neural Information Processing Systems
https://proceedings.neurips.cc › paper › 4824-...

ImageNet Classification with Deep Convolutional Neural ...

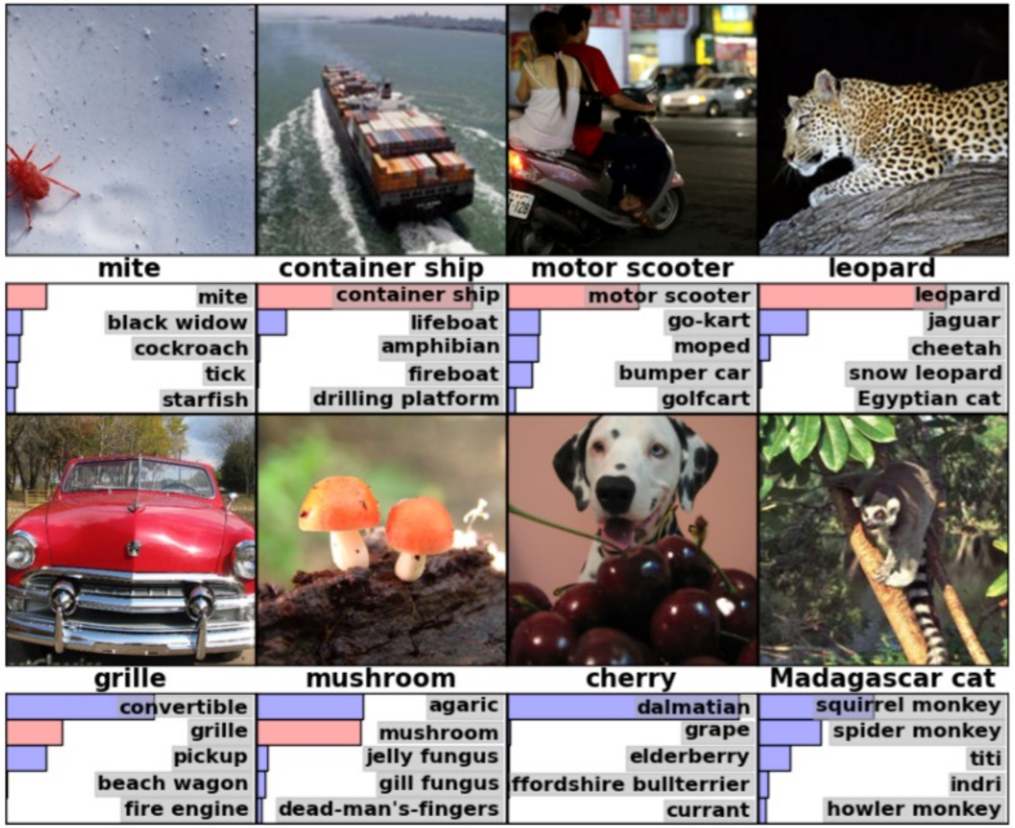
A Krizhevsky 저술 · 120897회 인용 — We trained a large, **deep convolutional neural network** to **classify** the 1.2 million high-resolution images in the **ImageNet** LSVRC-2010 contest into the 1000 ...
페이지 9개

CNN Examples - AlexNet

mite: 진드기
leopard: 표범
Grille: 자동차 엔진 방열판

◈AlexNet (Alex Krizhevsky et al.)

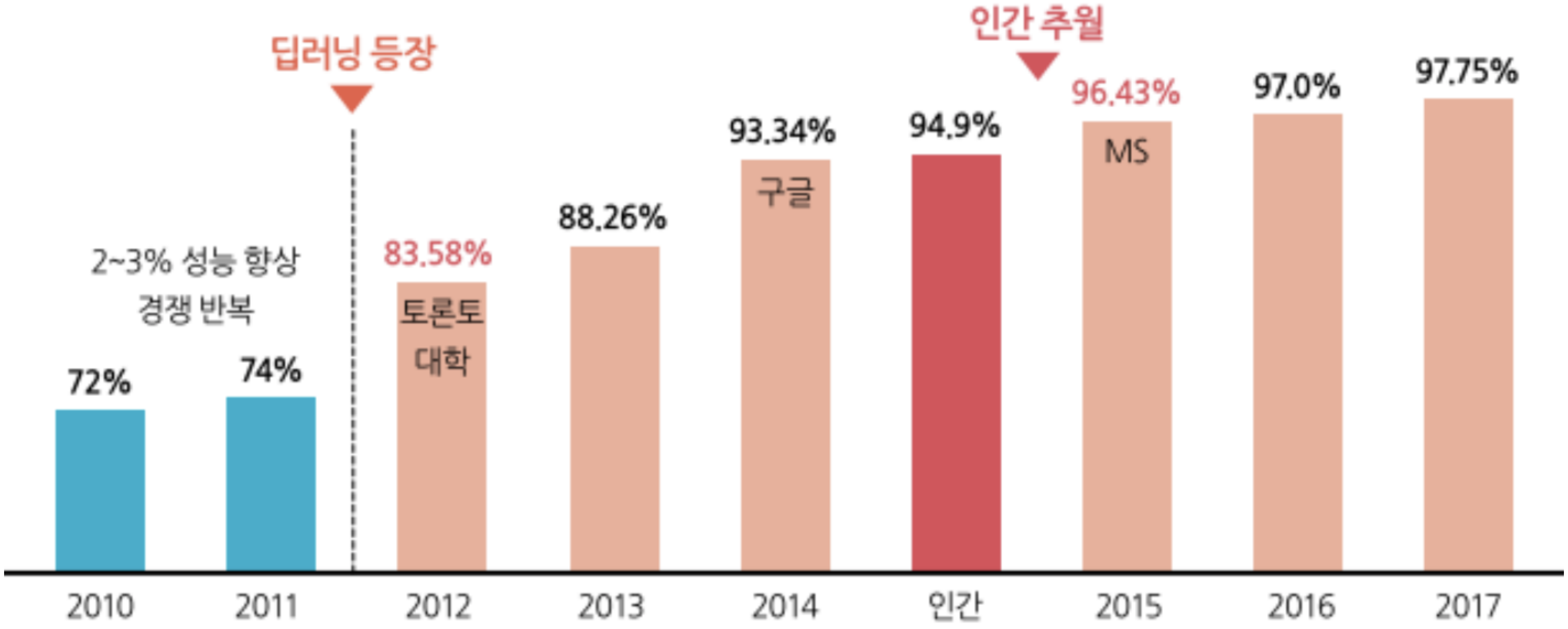
- "ImageNet" Data set
  - A large-scale dataset of images that is widely used in computer vision research & machine learning
  - Over 15M labeled high resolution images collected from web
  - Roughly 22,000 categories
- ILSVRC
  - The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) is a competition that used this dataset to push the development of image classification algorithms
  - 1.2M images in 1,000 categories
  - Classification: make 5 guesses about the image label
  - Many state-of-the-art deep neural networks, such as AlexNet, VGG, and ResNet, have been trained and evaluated on ImageNet data



CNN Examples - AlexNet

◈ AlexNet (Alex Krizhevsky et al.)

– ILSVRC History



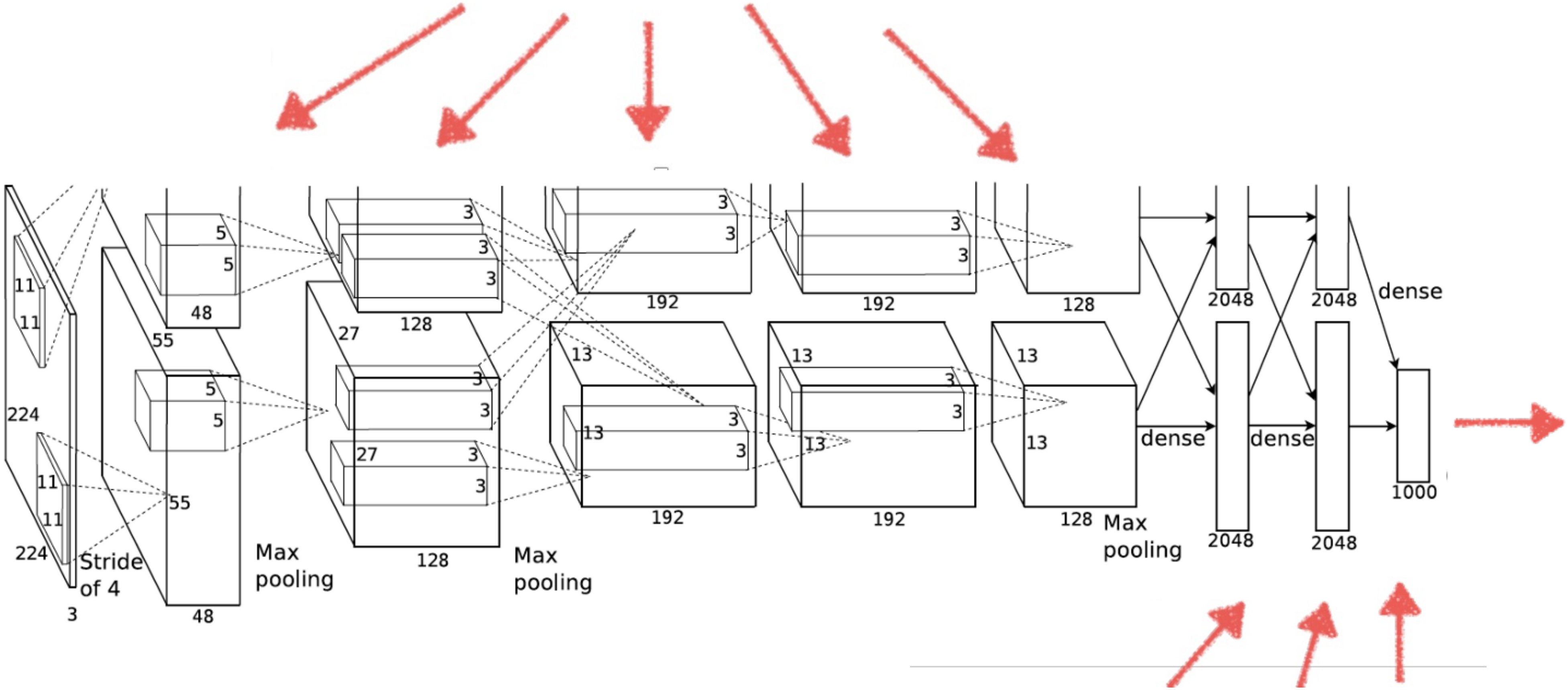
AlexNet ZFNet 1. VGGNet ResNet GoogleNet-v4 SENet

2. GoogleNet

CNN Examples - AlexNet

- AlexNet (Alex Krizhevsky et al.)

5 Convolutional Layers



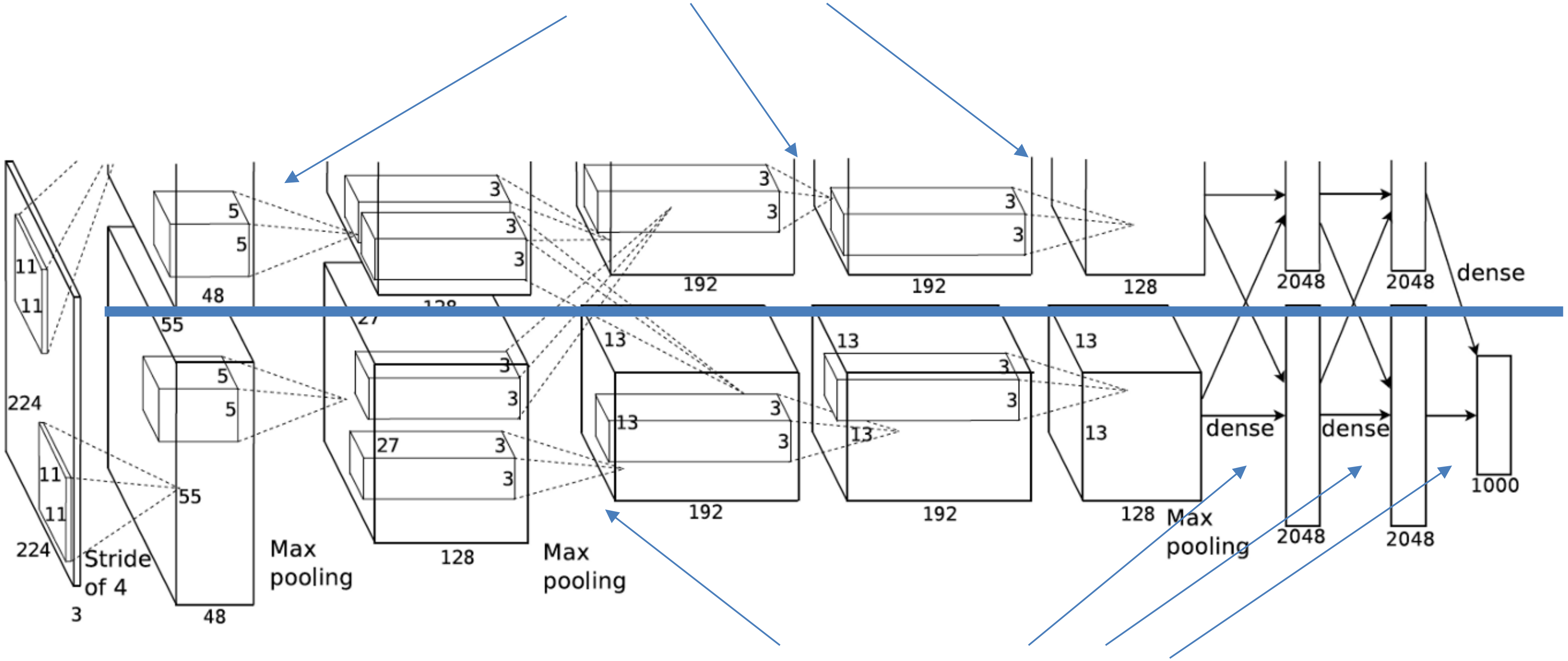
1000-way Softmax

3 Fully Connected Layers

CNN Examples

- AlexNet (Alex Krizhevsky et al.)

Intra-GPU connections

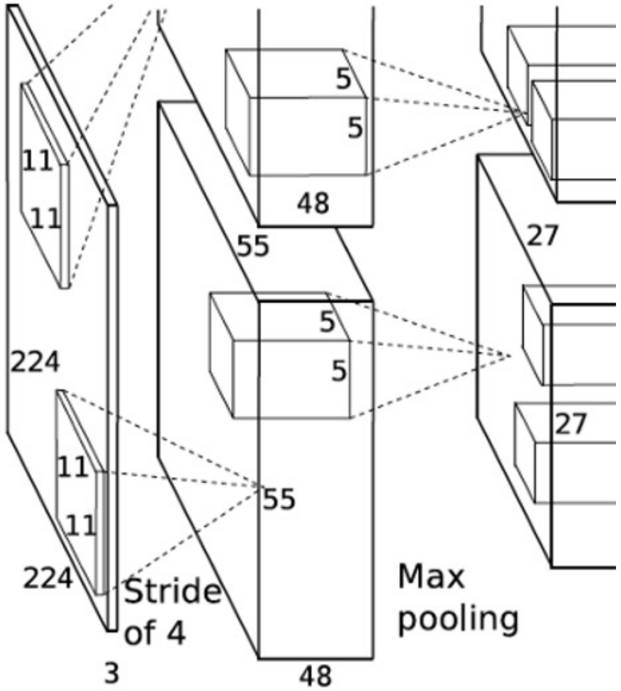


Inter-GPU connections

Top-1 and Top-5 error rates decreases by 1.7% & 1.2% respectively, comparing to the net trained with one GPU and half neurons!!

CNN Examples

◈Input Layer & Conv-1 Layer & MaxPool-1 Layer @ AlexNet



[Input Layer]

Images: 3 x 227 x 227 ← AlexNet image size should be **227×227×3**, instead of **224×224×3**

[Conv-1 Layer]

96 kernels of size 11 x 11

Stride=4, Padding=0

Input: 3 x 227 x 227 → Output: 96 x 55 x 55 (= 2 x 48 x 55 x 55)

Number of parameters: 3 x 11 x 11 x 96 + 96 = 34,944

$$(O_h, O_w) = \left(\frac{I_h - F_h + 2P_h}{S_h} + 1, \frac{I_w - F_w + 2P_w}{S_w} + 1\right) = \left(\frac{227 - 11 + 2\times 0}{4} + 1, \frac{227 - 11 + 2\times 0}{4} + 1\right) = (55, 55)$$

[MaxPool Layer]

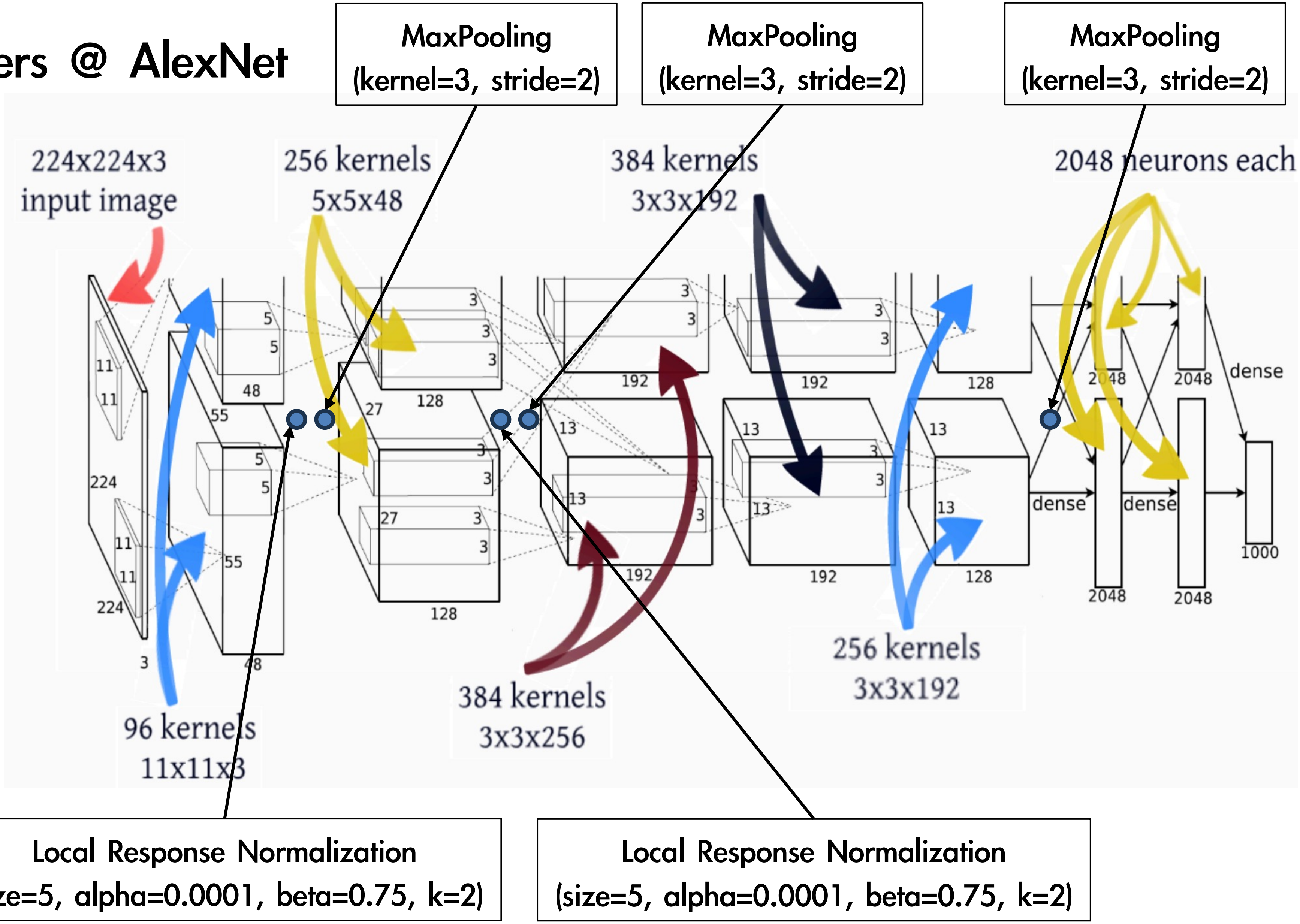
kernels of size 3 x 3, Stride=2

Input: 96 x 55 x 55 → Output: 96 x 27 x 27 (= 2 x 48 x 27 x 27)

$$(O_h, O_w) = \left(\frac{I_h - F_h}{S_h} + 1, \frac{I_w - F_w}{S_w} + 1\right) = \left(\frac{55 - 3}{2} + 1, \frac{55 - 3}{2} + 1\right) = (27, 27)$$

CNN Examples

- All Layers @ AlexNet



MaxPooling (kernel=3, stride=2)

MaxPooling (kernel=3, stride=2)

MaxPooling (kernel=3, stride=2)

Local Response Normalization (size=5, alpha=0.0001, beta=0.75, k=2)

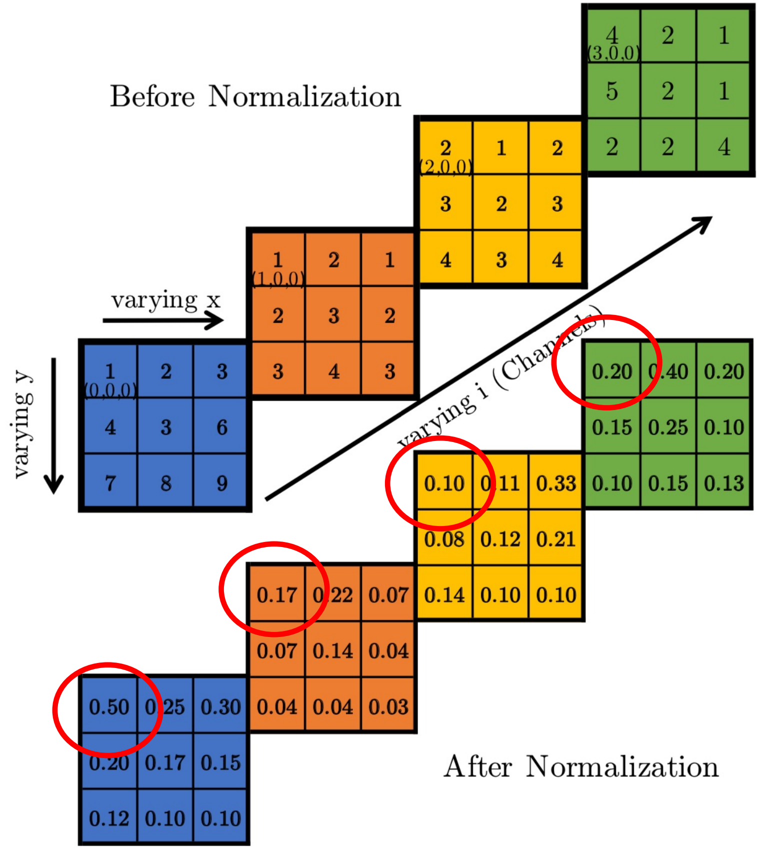
Local Response Normalization (size=5, alpha=0.0001, beta=0.75, k=2)

CNN Examples

◈ Local Response Normalization @ AlexNet

– Local Response Normalization

- Normalization is carried out in the channel dimension



Local Response Normalization
(size=3, alpha=1, beta=1, k=0)

$$b^i_{x,y} = a^i_{x,y} / \left(k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a^j_{x,y})^2 \right)^{\beta}$$

i: Channel index, N: Number of channel (=4)

$$0.20 = \frac{4}{2^2 + 4^2} = \frac{4}{20}$$

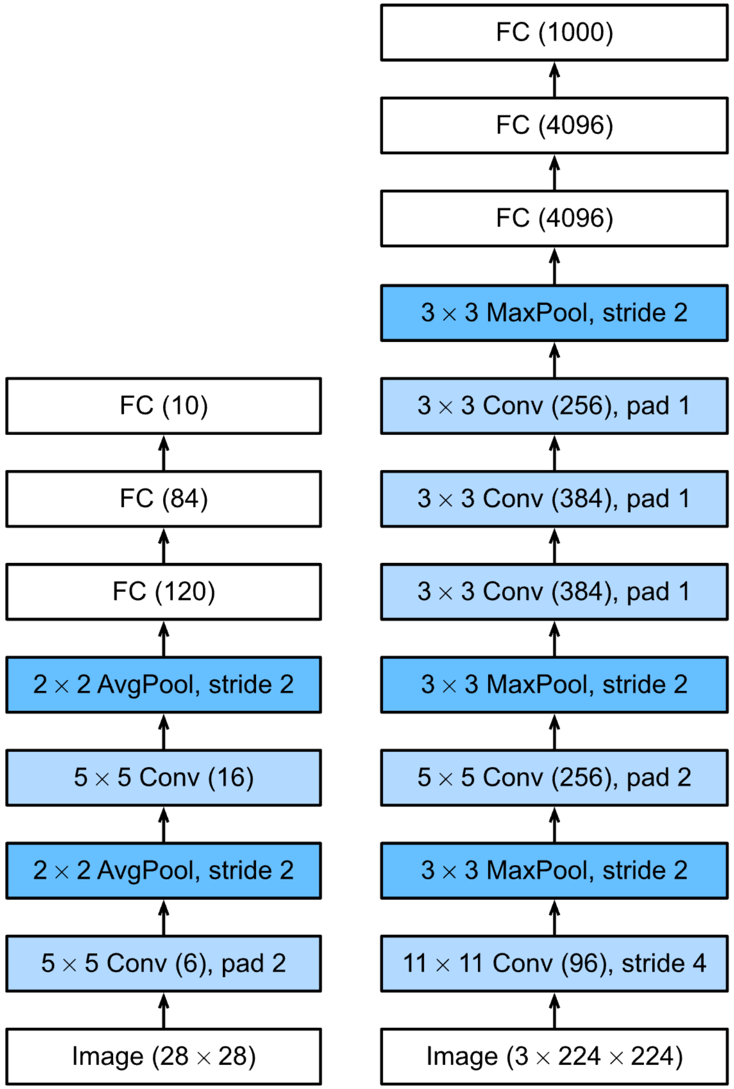
$$0.10 \approx \frac{2}{1^2 + 2^2 + 4^2} = \frac{2}{21}$$

$$0.17 \approx \frac{1}{1^2 + 1^2 + 2^2} = \frac{1}{6}$$

$$0.5 = \frac{1}{1^2 + 1^2}$$

CNN Examples

- LeNet (1998) vs. AlexNet (2012)



[Source] https://d2l.ai/chapter_convolutional-modern/alexnet.html

CNN Implementation
- MNIST CNN Train –
(Best Practice)

link_dl/_01_code/_07_cnn/a_mnist_train_cnn.py

MNIST CNN Train

◈MNIST CNN Train

```python
import torch
from torch import nn, optim
from datetime import datetime
import os
import wandb
from pathlib import Path
# BASE_PATH: /Users/yhhan/git/link_dl
BASE_PATH = str(Path(__file__).resolve().parent.parent.parent)
import sys
sys.path.append(BASE_PATH)
CURRENT_FILE_PATH = os.path.dirname(os.path.abspath(__file__))
CHECKPOINT_FILE_PATH = os.path.join(CURRENT_FILE_PATH, "checkpoints")
if not os.path.isdir(CHECKPOINT_FILE_PATH):
  os.makedirs(os.path.join(CURRENT_FILE_PATH, "checkpoints"))

import sys
sys.path.append(BASE_PATH)

from _01_code._06_fcn_best_practice.c_trainer import ClassificationTrainer
from _01_code._06_fcn_best_practice.f_mnist_train_fcn import get_data
from _01_code._06_fcn_best_practice.e_arg_parser import get_parser
```

link_dl/_01_code/_07_cnn/a_mnist_train_cnn.py

MNIST CNN Train

◈MNIST CNN Train

```
def get_cnn_model():
  class MyModel(nn.Module):
    def __init__(self, in_channels, n_output):
      super().__init__()

      self.model = nn.Sequential(
        # B x 1 x 28 x 28 --> B x 6 x (28 - 5 + 1) x (28 - 5 + 1) = B x 6 x 24 x 24
        nn.Conv2d(in_channels=in_channels, out_channels=6, kernel_size=(5, 5), stride=(1, 1)),
        # B x 6 x 24 x 24 --> B x 6 x 12 x 12
        nn.MaxPool2d(kernel_size=2, stride=2),
        nn.ReLU(),
        # B x 6 x 12 x 12 --> B x 16 x (12 - 5 + 1) x (12 - 5 + 1) = 16 x 8 x 8
        nn.Conv2d(in_channels=6, out_channels=16, kernel_size=(5, 5), stride=(1, 1)),
        # B x 16 x 8 x 8 --> B x 16 x 4 x 4
        nn.MaxPool2d(kernel_size=2, stride=2),
        nn.ReLU(),
        nn.Flatten(),
        nn.Linear(256, 128),
        nn.ReLU(),
        nn.Linear(128, n_output),
      )
```

link_dl/_01_code/_07_cnn/a_mnist_train_cnn.py

MNIST CNN Train

◈MNIST CNN Train

```
def get_cnn_model():
  class MyModel(nn.Module):
    def __init__(self, in_channels, n_output):
      ...

    def forward(self, x):
      x = self.model(x)
      return x

  # 1 * 28 * 28
  my_model = MyModel(in_channels=1, n_output=10)

  return my_model
```

link_dl/_01_code/_07_cnn/a_mnist_train_cnn.py

MNIST CNN Train

◈MNIST CNN Train

```python
def main(args):
  run_time_str = datetime.now().astimezone().strftime('%Y-%m-%d_%H-%M-%S')

  config = {
    'epochs': args.epochs,
    'batch_size': args.batch_size,
    'validation_intervals': args.validation_intervals,
    'learning_rate': args.learning_rate,
    'early_stop_patience': args.early_stop_patience
  }
  project_name = "cnn_mnist"
  wandb.init(
    mode="online" if args.wandb else "disabled",
    project=project_name,
    notes="mnist experiment with cnn",
    tags=["cnn", "mnist"],
    name=run_time_str,
    config=config
  )
  print(args)
  print(wandb.config)
```

link_dl/_01_code/_07_cnn/a_mnist_train_cnn.py

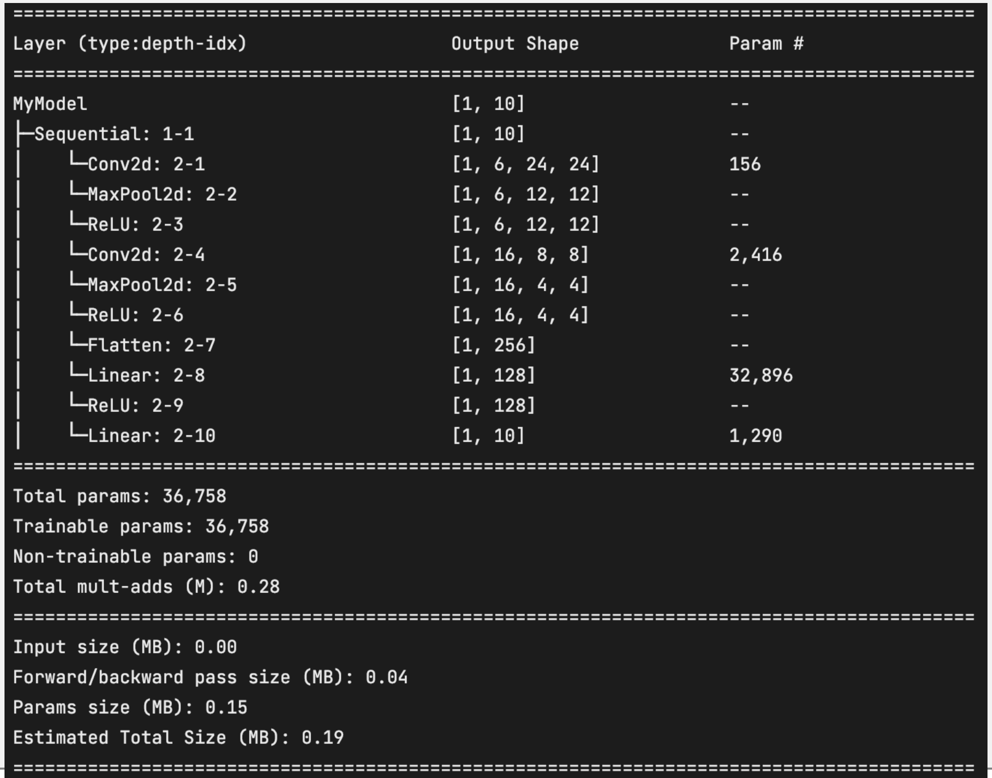
MNIST CNN Train

◈MNIST CNN Train

```
def main(args):
  ...
  device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
  print(f"Training on device {device}.")

  train_data_loader, validation_data_loader, mnist_transforms = get_data(flatten=False)
  model = get_cnn_model()
  model.to(device)
  wandb.watch(model)

  from torchinfo import summary
  summary(model=model, input_size=(1, 1, 28, 28))
```



```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
MyModel                                  [1, 10]                   --
├─Sequential: 1-1                        [1, 10]                   --
│    └─Conv2d: 2-1                       [1, 6, 24, 24]            156
│    └─MaxPool2d: 2-2                    [1, 6, 12, 12]            --
│    └─ReLU: 2-3                         [1, 6, 12, 12]            --
│    └─Conv2d: 2-4                       [1, 16, 8, 8]             2,416
│    └─MaxPool2d: 2-5                    [1, 16, 4, 4]             --
│    └─ReLU: 2-6                         [1, 16, 4, 4]             --
│    └─Flatten: 2-7                      [1, 256]                  --
│    └─Linear: 2-8                       [1, 128]                  32,896
│    └─ReLU: 2-9                         [1, 128]                  --
│    └─Linear: 2-10                      [1, 10]                   1,290
==========================================================================================
Total params: 36,758
Trainable params: 36,758
Non-trainable params: 0
Total mult-adds (M): 0.28
==========================================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 0.04
Params size (MB): 0.15
Estimated Total Size (MB): 0.19
==========================================================================================
```

link_dl/_01_code/_07_cnn/a_mnist_train_cnn.py

MNIST CNN Train

◈MNIST CNN Train

```python
def main(args):
  ...
  optimizer = optim.SGD(model.parameters(), lr=wandb.config.learning_rate)
  classification_trainer = ClassificationTrainer(
    project_name, model, optimizer, train_data_loader, validation_data_loader, mnist_transforms,
    run_time_str, wandb, device, CHECKPOINT_FILE_PATH
  )
  classification_trainer.train_loop()

  wandb.finish()


if __name__ == "__main__":
  parser = get_parser()
  args = parser.parse_args()
  main(args)
  # python _01_code/_07_cnn/a_mnist_train_cnn.py --wandb -b 2048 -r 1e-3 -v 10
  # python _01_code/_07_cnn/a_mnist_train_cnn.py --no-wandb -b 2048 -r 1e-3 -v 10
```

CNN Implementation
- MNIST CNN Test –
(Best Practice)

link_dl/_01_code/_07_cnn/b_mnist_test_cnn.py

MNIST CNN Train

◈MNIST CNN Test

```
import numpy as np
import torch
import os

from matplotlib import pyplot as plt
from pathlib import Path

BASE_PATH = str(Path(__file__).resolve().parent.parent.parent)  # BASE_PATH: /Users/yhhan/git/link_dl
CURRENT_FILE_PATH = os.path.dirname(os.path.abspath(__file__))
CHECKPOINT_FILE_PATH = os.path.join(CURRENT_FILE_PATH, "checkpoints")

import sys
sys.path.append(BASE_PATH)

from _01_code._07_cnn.a_mnist_train_cnn import get_cnn_model
from _01_code._06_fcn_best_practice.d_tester import ClassificationTester
from _01_code._06_fcn_best_practice.g_mnist_test_fcn import get_test_data
```

link_dl/_01_code/_07_cnn/b_mnist_test_cnn.py

MNIST CNN Train

◈MNIST CNN Test

```python
def main():
  mnist_test_images, test_data_loader, mnist_transforms = get_test_data(flatten=False)

  test_model = get_cnn_model()
  classification_tester = ClassificationTester(
    "mnist", test_model, test_data_loader, mnist_transforms, CHECKPOINT_FILE_PATH
  )
  classification_tester.test()

  img, label = mnist_test_images[0]
  print("     LABEL:", label)
  plt.imshow(img)
  plt.show()

  output = classification_tester.test_single(
    torch.tensor(np.array(mnist_test_images[0][0])).unsqueeze(dim=0).unsqueeze(dim=0)
  ) # (1, 1, 28, 28)
  print("PREDICTION:", output)

if __name__ == "__main__":
  main()
```

CNN Implementation
- CIFAR10 CNN Train –
(Best Practice)

link_dl/_01_code/_07_cnn/c_cifar10_train_cnn.py

CIFAR10 CNN Train

◈CIFAR10 CNN Train

```
import torch
from torch import nn, optim
from datetime import datetime
import os
import wandb
from pathlib import Path
# # BASE_PATH: /Users/yhhan/git/link_dl
BASE_PATH = str(Path(__file__).resolve().parent.parent.parent)
import sys
sys.path.append(BASE_PATH)
CURRENT_FILE_PATH = os.path.dirname(os.path.abspath(__file__))
CHECKPOINT_FILE_PATH = os.path.join(CURRENT_FILE_PATH, "checkpoints")
if not os.path.isdir(CHECKPOINT_FILE_PATH):
  os.makedirs(os.path.join(CURRENT_FILE_PATH, "checkpoints"))

import sys
sys.path.append(BASE_PATH)

from _01_code._06_fcn_best_practice.c_trainer import ClassificationTrainer
from _01_code._06_fcn_best_practice.h_cifar10_train_fcn import get_data
from _01_code._06_fcn_best_practice.e_arg_parser import get_parser
```

link_dl/_01_code/_07_cnn/c_cifar10_train_cnn.py

CIFAR10 CNN Train

◈CIFAR10 CNN Train

```python
def get_cnn_model():
  class MyModel(nn.Module):
    def __init__(self, in_channels, n_output):
      super().__init__()

      self.model = nn.Sequential(
        # B x 3 x 32 x 32 --> B x 6 x (32 - 5 + 1) x (32 - 5 + 1) = B x 6 x 28 x 28
        nn.Conv2d(in_channels=in_channels, out_channels=6, kernel_size=(5, 5), stride=(1, 1)),
        # B x 6 x 28 x 28 --> B x 6 x 14 x 14
        nn.MaxPool2d(kernel_size=2, stride=2),
        nn.ReLU(),
        # B x 6 x 14 x 14 --> B x 16 x (14 - 5 + 1) x (14 - 5 + 1) = B x 16 x 10 x 10
        nn.Conv2d(in_channels=6, out_channels=16, kernel_size=(5, 5), stride=(1, 1)),
        # B x 16 x 10 x 10 --> B x 16 x 5 x 5
        nn.MaxPool2d(kernel_size=2, stride=2),
        nn.ReLU(),
        nn.Flatten(),
        nn.Linear(400, 128),
        nn.ReLU(),
        nn.Linear(128, n_output),
      )
```

link_dl/_01_code/_07_cnn/c_cifar10_train_cnn.py

CIFAR10 CNN Train

◈CIFAR10 CNN Train

```
def get_cnn_model():
  class MyModel(nn.Module):
    def __init__(self, in_channels, n_output):
      ...

    def forward(self, x):
      x = self.model(x)
      # print(x.shape, "!!!")
      return x

  # 3 * 32 * 32
  my_model = MyModel(in_channels=3, n_output=10)

  return my_model
```

link_dl/_01_code/_07_cnn/c_cifar10_train_cnn.py

CIFAR10 CNN Train

◈CIFAR10 CNN Train

```python
def main(args):
  run_time_str = datetime.now().astimezone().strftime('%Y-%m-%d_%H-%M-%S')

  config = {
    'epochs': args.epochs,
    'batch_size': args.batch_size,
    'validation_intervals': args.validation_intervals,
    'learning_rate': args.learning_rate,
    'early_stop_patience': args.early_stop_patience
  }
  project_name = "cnn_cifar10"
  wandb.init(
    mode="online" if args.wandb else "disabled",
    project=project_name,
    notes="cifar10 experiment with cnn",
    tags=["cnn", "cifar10"],
    name=run_time_str,
    config=config
  )
  print(args)
  print(wandb.config)
```

link_dl/_01_code/_07_cnn/c_cifar10_train_cnn.py

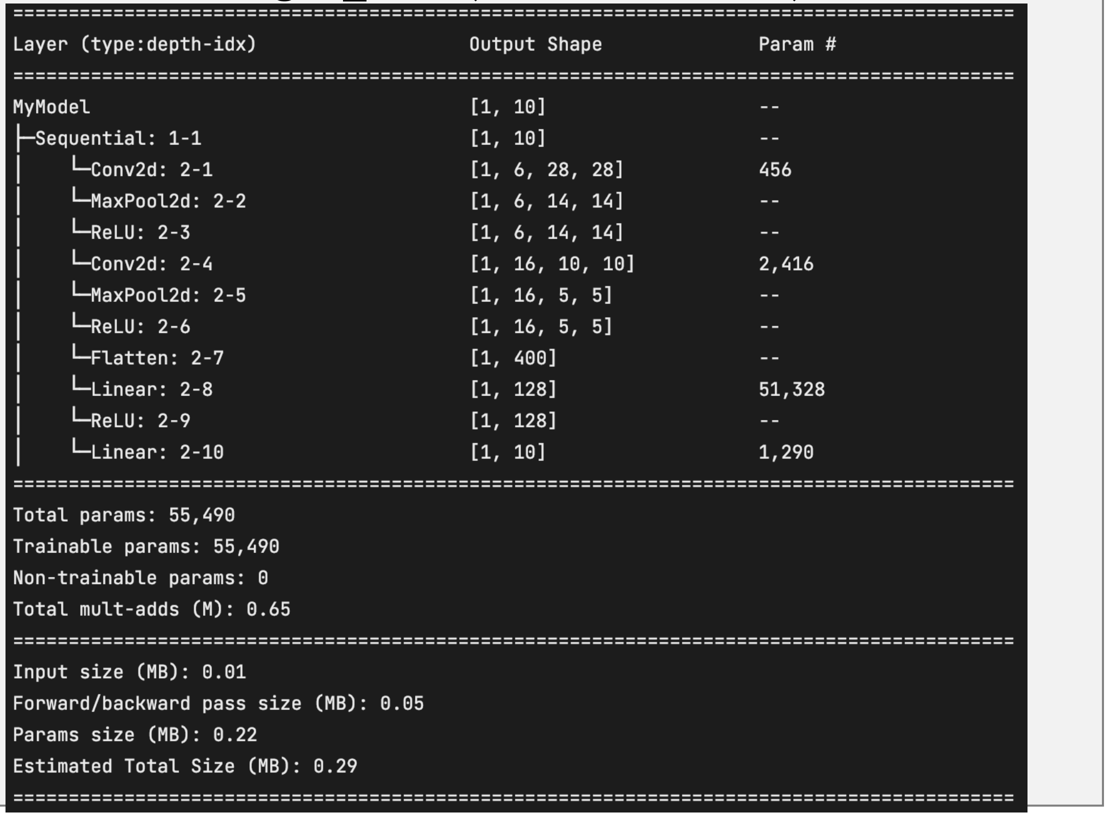
CIFAR10 CNN Train

◈CIFAR10 CNN Train

```python
def main(args):
  ...
  device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
  print(f"Training on device {device}.")

  train_data_loader, validation_data_loader, cifar10_transforms = get_data(flatten=False)
  model = get_cnn_model()
  model.to(device)
  wandb.watch(model)

  from torchinfo import summary
  summary(model=model, input_size=(1, 3, 32, 32))
```



```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
MyModel                                  [1, 10]                   --
├─Sequential: 1-1                        [1, 10]                   --
│    └─Conv2d: 2-1                       [1, 6, 28, 28]            456
│    └─MaxPool2d: 2-2                    [1, 6, 14, 14]            --
│    └─ReLU: 2-3                         [1, 6, 14, 14]            --
│    └─Conv2d: 2-4                       [1, 16, 10, 10]           2,416
│    └─MaxPool2d: 2-5                    [1, 16, 5, 5]             --
│    └─ReLU: 2-6                         [1, 16, 5, 5]             --
│    └─Flatten: 2-7                      [1, 400]                  --
│    └─Linear: 2-8                       [1, 128]                  51,328
│    └─ReLU: 2-9                         [1, 128]                  --
│    └─Linear: 2-10                      [1, 10]                   1,290
==========================================================================================
Total params: 55,490
Trainable params: 55,490
Non-trainable params: 0
Total mult-adds (M): 0.65
==========================================================================================
Input size (MB): 0.01
Forward/backward pass size (MB): 0.05
Params size (MB): 0.22
Estimated Total Size (MB): 0.29
==========================================================================================
```

link_dl/_01_code/_07_cnn/c_cifar10_train_cnn.py

CIFAR10 CNN Train

◈CIFAR10 CNN Train

```
def main(args):
  ...
  optimizer = optim.SGD(model.parameters(), lr=wandb.config.learning_rate)

  classification_trainer = ClassificationTrainer(
    project_name, model, optimizer, train_data_loader, validation_data_loader, cifar10_transforms,
    run_time_str, wandb, device, CHECKPOINT_FILE_PATH
  )
  classification_trainer.train_loop()

  wandb.finish()


if __name__ == "__main__":
  parser = get_parser()
  args = parser.parse_args()
  main(args)
  # python _01_code/_07_cnn/c_cifar10_train_cnn.py --wandb -b 2048 -r 1e-3 -v 10
  # python _01_code/_07_cnn/c_cifar10_train_cnn.py --no-wandb -b 2048 -r 1e-3 -v 10
```

CNN Implementation
- CIFAR10 CNN Test –
(Best Practice)

link_dl/_01_code/_07_cnn/d_cifar10_test_cnn.py

CIFAR10 CNN Test

◈CIFAR10 CNN Test

```
import numpy as np
import torch
import os

from matplotlib import pyplot as plt
from pathlib import Path

BASE_PATH = str(Path(__file__).resolve().parent.parent.parent)  # BASE_PATH: /Users/yhhan/git/link_dl
CURRENT_FILE_PATH = os.path.dirname(os.path.abspath(__file__))
CHECKPOINT_FILE_PATH = os.path.join(CURRENT_FILE_PATH, "checkpoints")

import sys
sys.path.append(BASE_PATH)

from _01_code._07_cnn.c_cifar10_train_cnn import get_cnn_model
from _01_code._06_fcn_best_practice.d_tester import ClassificationTester
from _01_code._06_fcn_best_practice.i_cifar10_test_fcn import get_test_data
```

link_dl/_01_code/_07_cnn/d_cifar10_test_cnn.py

CIFAR10 CNN Test

◈CIFAR10 CNN Test

```python
def main():
  cifar10_test_images, test_data_loader, cifar10_transforms = get_test_data(flatten=False)

  test_model = get_cnn_model()
  classification_tester = ClassificationTester(
    "cifar10", test_model, test_data_loader, cifar10_transforms, CHECKPOINT_FILE_PATH
  )
  classification_tester.test()

  img, label = cifar10_test_images[0]
  print("     LABEL:", label)
  plt.imshow(img)
  plt.show()

  output = classification_tester.test_single(
    torch.tensor(np.array(cifar10_test_images[0][0])).permute(2, 0, 1).unsqueeze(dim=0)
  ) # (1, 3, 32, 32)
  print("PREDICTION:", output)

if __name__ == "__main__":
  main()
```

CNN Implementation
- CIFAR10 AlexNet Train –
(Best Practice)

link_dl/_01_code/_07_cnn/e_cifar10_train_alexnet.py

CIFAR10 AlexNet Train

◈CIFAR10 AlexNet Train

```python
def get_alexnet_model():
  class AlexNet(nn.Module):
    def __init__(self, in_channels=3, n_output=10):
        super().__init__()
        # The image in the original paper states that width and height are 224 pixels, but
        # the correct input size should be : (B x 3 x 227 x 227)
        self.cnn = nn.Sequential(
            # B x 3 x 32 x 32 -> B x 64 x (32 - 3 + 1) x (32 - 3 + 1) = B x 64 x 30 x 30
            nn.Conv2d(in_channels=in_channels, out_channels=64, kernel_size=(3, 3), stride=(1, 1)),
            nn.ReLU(),
            nn.LocalResponseNorm(size=3, alpha=0.0001, beta=0.75, k=2),
            # B x 64 x 30 x 30 -> B x 64 x ((30 - 2) / 2 + 1) x ((30 - 2) / 2 + 1) = B x 64 x 15 x 15
            nn.MaxPool2d(kernel_size=2, stride=2),

            # B x 64 x 15 x 15 -> B x 64 x (15 - 3 + 2 + 1) x (15 - 3 + 2 + 1) = B x 192 x 15 x 15
            nn.Conv2d(64, 192, (3, 3), (1, 1), padding=1),
            nn.ReLU(),
            nn.LocalResponseNorm(size=3, alpha=0.0001, beta=0.75, k=2),
            # B x 192 x 15 x 15 -> B x 192 x ((15 - 3) / 2 + 1) x ((15 - 3) / 2 + 1) = B x 192 x 7 x 7
            nn.MaxPool2d(kernel_size=3, stride=2),
```

link_dl/_01_code/_07_cnn/e_cifar10_train_alexnet.py

CIFAR10 AlexNet Train

◈CIFAR10 AlexNet Train

```python
def get_alexnet_model():
  class AlexNet(nn.Module):
    def __init__(self, in_channels=3, n_output=10):
        super().__init__()
        ...
        self.cnn = nn.Sequential(
            ...
            # B x 192 x 7 x 7 -> B x 256 x ((7 - 3 + 2) / 1 + 1) x ((13 - 3 + 2) / 1 + 1) = B x 256 x 7 x 7
            nn.Conv2d(192, 256, (3, 3), (1, 1), padding=1),
            nn.ReLU(),

            # B x 256 x 7 x 7 -> B x 256 x ((7 - 3 + 2) / 1 + 1) x ((13 - 3 + 2) / 1 + 1) = B x 256 x 7 x 7
            nn.Conv2d(256, 256, (3, 3), (1, 1), padding=1),
            nn.ReLU(),

            # B x 256 x 7 x 7 -> B x 192 x ((7 - 2) / 1 + 1) x ((7 - 2) / 1 + 1) = B x 192 x 6 x 6
            nn.Conv2d(256, 192, (2, 2), (1, 1)),
            nn.ReLU(),
            # B x 192 x 6 x 6 -> B x 192 x ((6 - 2) / 2 + 1) x ((6 - 2) / 2 + 1) = B x 192 x 3 x 3
            nn.MaxPool2d(kernel_size=2, stride=2),
        )
```

link_dl/_01_code/_07_cnn/e_cifar10_train_alexnet.py

CIFAR10 AlexNet Train

◈CIFAR10 AlexNet Train

```
def get_alexnet_model():
  class AlexNet(nn.Module):
    def __init__(self, in_channels=3, n_output=10):
        super().__init__()
        ...

        # classifier is just a name for linear layers
        self.fcn = nn.Sequential(
            nn.Dropout(p=0.5),
            nn.Linear(in_features=192 * 3 * 3, out_features=512),
            nn.ReLU(),
            nn.Dropout(p=0.5),
            nn.Linear(in_features=512, out_features=512),
            nn.ReLU(),
            nn.Linear(in_features=512, out_features=n_output),
        )
```

link_dl/_01_code/_07_cnn/e_cifar10_train_alexnet.py

CIFAR10 AlexNet Train

◈CIFAR10 AlexNet Train

```
def get_alexnet_model():
  class AlexNet(nn.Module):


    ...

    def forward(self, x):
        """
        Pass the input through the net.
        """
        x = self.cnn(x)
        x = x.view(-1, 192 * 3 * 3)  # reduce the dimensions for linear layer input
        return self.fcn(x)

  my_model = AlexNet(in_channels=3, n_output=10)

  return my_model
```

link_dl/_01_code/_07_cnn/e_cifar10_train_alexnet.py

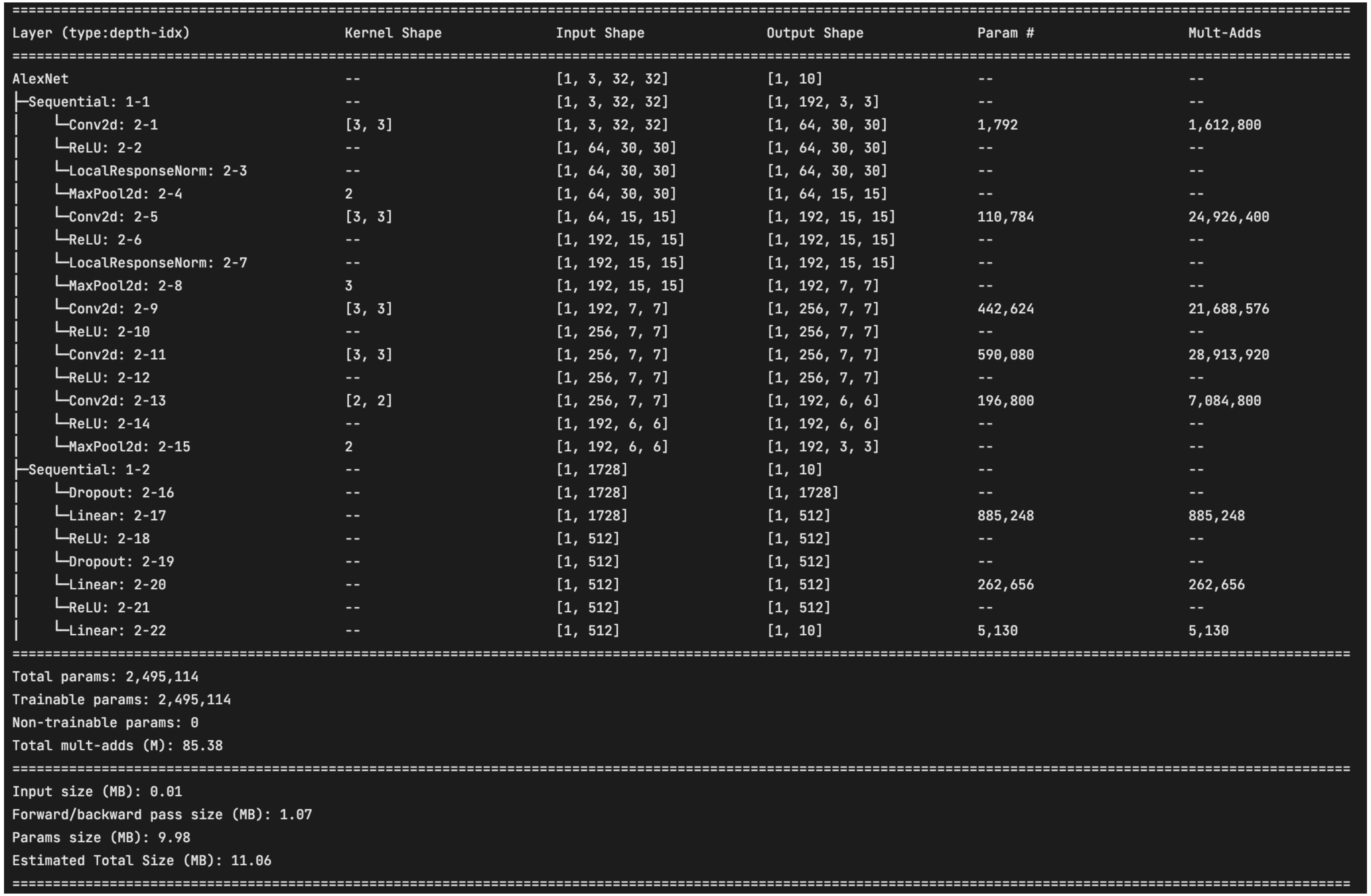
CIFAR10 AlexNet Train

◈CIFAR10 AlexNet Train

```
from torchinfo import summary

model = get_alexnet_model()

summary(
  model=model,
  input_size=(1, 3, 32, 32),
  col_names=[
    "kernel_size",
    "input_size",
    "output_size",
    "num_params",
    "mult_adds”
  ]
)
```



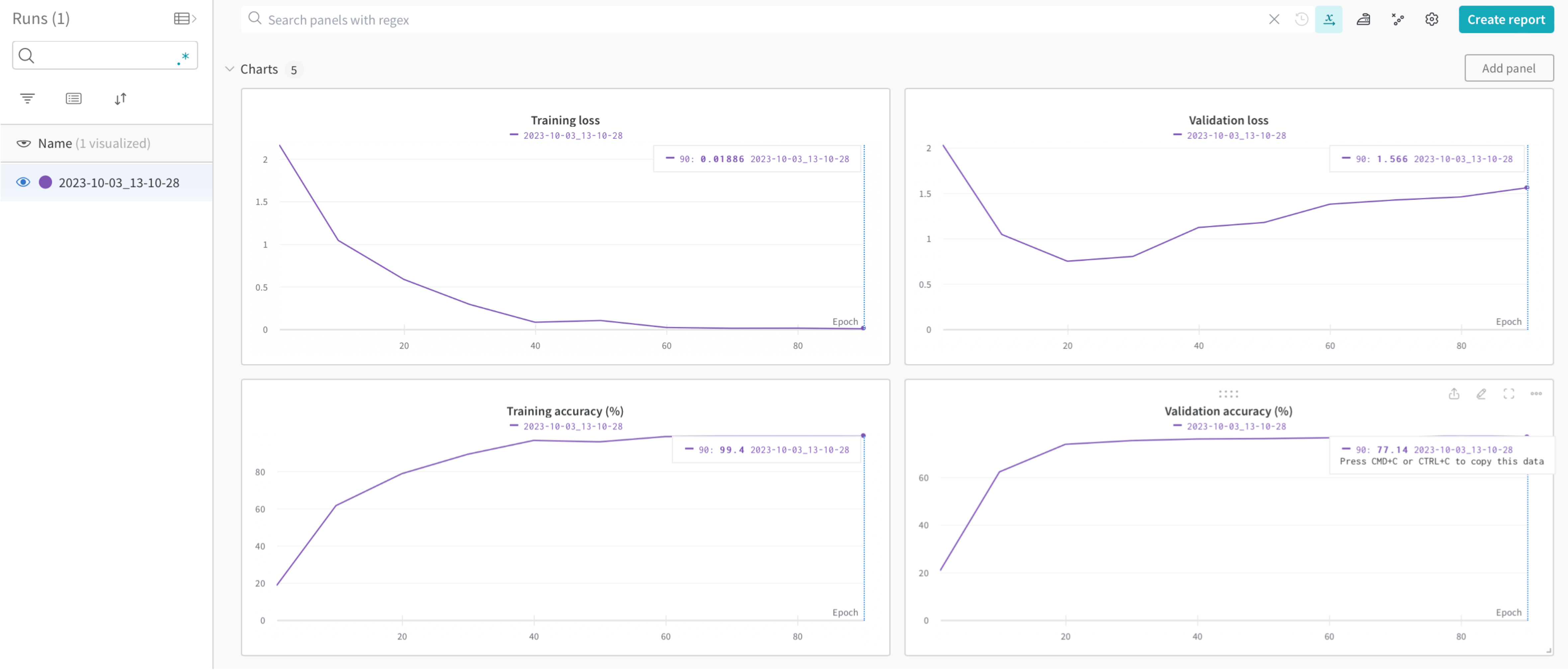
```
=====================================================================================================================================================================
Layer (type:depth-idx)                   Kernel Shape              Input Shape               Output Shape              Param #                   Mult-Adds
=====================================================================================================================================================================
AlexNet                                  --                        [1, 3, 32, 32]            [1, 10]                   --                        --
├─Sequential: 1-1                        --                        [1, 3, 32, 32]            [1, 192, 3, 3]            --                        --
│    └─Conv2d: 2-1                       [3, 3]                    [1, 3, 32, 32]            [1, 64, 30, 30]           1,792                     1,612,800
│    └─ReLU: 2-2                         --                        [1, 64, 30, 30]           [1, 64, 30, 30]           --                        --
│    └─LocalResponseNorm: 2-3            --                        [1, 64, 30, 30]           [1, 64, 30, 30]           --                        --
│    └─MaxPool2d: 2-4                    2                         [1, 64, 30, 30]           [1, 64, 15, 15]           --                        --
│    └─Conv2d: 2-5                       [3, 3]                    [1, 64, 15, 15]           [1, 192, 15, 15]          110,784                   24,926,400
│    └─ReLU: 2-6                         --                        [1, 192, 15, 15]          [1, 192, 15, 15]          --                        --
│    └─LocalResponseNorm: 2-7            --                        [1, 192, 15, 15]          [1, 192, 15, 15]          --                        --
│    └─MaxPool2d: 2-8                    3                         [1, 192, 15, 15]          [1, 192, 7, 7]            --                        --
│    └─Conv2d: 2-9                       [3, 3]                    [1, 192, 7, 7]            [1, 256, 7, 7]            442,624                   21,688,576
│    └─ReLU: 2-10                        --                        [1, 256, 7, 7]            [1, 256, 7, 7]            --                        --
│    └─Conv2d: 2-11                      [3, 3]                    [1, 256, 7, 7]            [1, 256, 7, 7]            590,080                   28,913,920
│    └─ReLU: 2-12                        --                        [1, 256, 7, 7]            [1, 256, 7, 7]            --                        --
│    └─Conv2d: 2-13                      [2, 2]                    [1, 256, 7, 7]            [1, 192, 6, 6]            196,800                   7,084,800
│    └─ReLU: 2-14                        --                        [1, 192, 6, 6]            [1, 192, 6, 6]            --                        --
│    └─MaxPool2d: 2-15                   2                         [1, 192, 6, 6]            [1, 192, 3, 3]            --                        --
├─Sequential: 1-2                        --                        [1, 1728]                 [1, 10]                   --                        --
│    └─Dropout: 2-16                     --                        [1, 1728]                 [1, 1728]                 --                        --
│    └─Linear: 2-17                      --                        [1, 1728]                 [1, 512]                  885,248                   885,248
│    └─ReLU: 2-18                        --                        [1, 512]                  [1, 512]                  --                        --
│    └─Dropout: 2-19                     --                        [1, 512]                  [1, 512]                  --                        --
│    └─Linear: 2-20                      --                        [1, 512]                  [1, 512]                  262,656                   262,656
│    └─ReLU: 2-21                        --                        [1, 512]                  [1, 512]                  --                        --
│    └─Linear: 2-22                      --                        [1, 512]                  [1, 10]                   5,130                     5,130
=====================================================================================================================================================================
Total params: 2,495,114
Trainable params: 2,495,114
Non-trainable params: 0
Total mult-adds (M): 85.38
=====================================================================================================================================================================
Input size (MB): 0.01
Forward/backward pass size (MB): 1.07
Params size (MB): 9.98
Estimated Total Size (MB): 11.06
=====================================================================================================================================================================
```

link_dl/_01_code/_07_cnn/e_cifar10_train_alexnet.py

CIFAR10 AlexNet Train

◈CIFAR10 AlexNet Train

– https://wandb.ai/link-koreatech/alexnet_cifar10/workspace?workspace=user-link-koreatech



CNN Implementation
- CIFAR10 AlexNet Test –
(Best Practice)

link_dl/_01_code/_07_cnn/f_cifar10_test_alexnet.py

CIFAR10 AlexNet Test

◈CIFAR10 AlexNet Test

```
import numpy as np
import torch
import os

from matplotlib import pyplot as plt
from pathlib import Path

BASE_PATH = str(Path(__file__).resolve().parent.parent.parent)  # BASE_PATH: /Users/yhhan/git/link_dl
CURRENT_FILE_PATH = os.path.dirname(os.path.abspath(__file__))
CHECKPOINT_FILE_PATH = os.path.join(CURRENT_FILE_PATH, "checkpoints")

import sys
sys.path.append(BASE_PATH)

from _01_code._06_fcn_best_practice.d_tester import ClassificationTester
from _01_code._06_fcn_best_practice.i_cifar10_test_fcn import get_cifar10_test_data
from _01_code._07_cnn.e_cifar10_train_alexnet import get_alexnet_model
```

link_dl/_01_code/_07_cnn/f_cifar10_test_alexnet.py

CIFAR10 AlexNet Test

◈CIFAR10 AlexNet Test

```
def main():
  cifar10_test_images, test_data_loader, cifar10_transforms = get_cifar10_test_data(flatten=False)

  test_model = get_alexnet_model()

  project_name = "alexnet_cifar10”
  classification_tester = ClassificationTester(
    project_name, test_model, test_data_loader, cifar10_transforms, CHECKPOINT_FILE_PATH
  )
  classification_tester.test()

  print()

  img, label = cifar10_test_images[0]
  print("     LABEL:", label)
  plt.imshow(img)
  plt.show()
```

link_dl/_01_code/_07_cnn/f_cifar10_test_alexnet.py

CIFAR10 AlexNet Test

◈CIFAR10 AlexNet Test

```
def main():
  ...

  # torch.tensor(np.array(cifar10_test_images[0][0])).permute(2, 0, 1).unsqueeze(dim=0).shape: (1, 3, 32, 32)
  output = classification_tester.test_single(
    torch.tensor(np.array(cifar10_test_images[0][0])).permute(2, 0, 1).unsqueeze(dim=0)
  )
  print("PREDICTION:", output)


if __name__ == "__main__":
  main()
```