January 7, 2024

Module

Knowledge Distillation

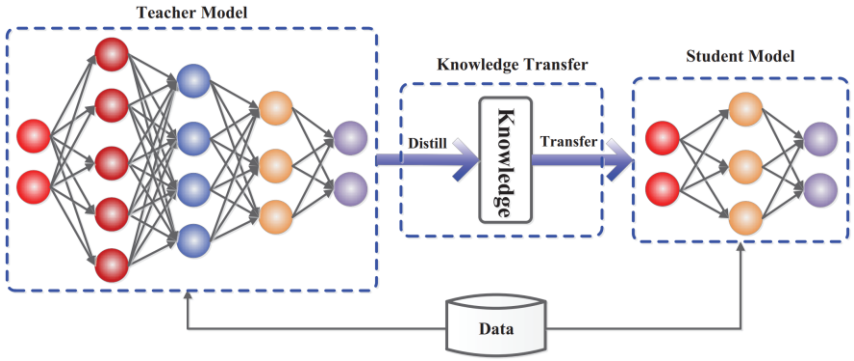
Ph.D. Ngo Ba Hung

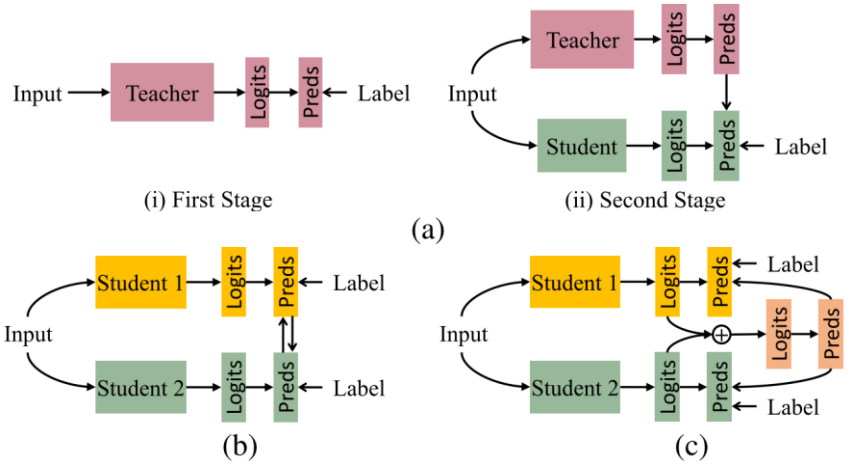
Email: ngohung@jnu.ac.kr
Facebook: https://www.facebook.com/hung.ngo.7121

Graduate School of Data Science
Chonnam National University

Overview

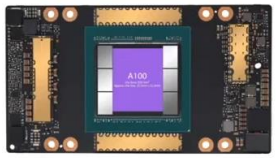
- What is knowledge distillation?
  - Knowledge distillation refers to the process of transferring the knowledge from a large model or set of models to a single smaller model that can be practically deployed under real-world constraints.

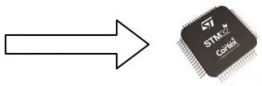




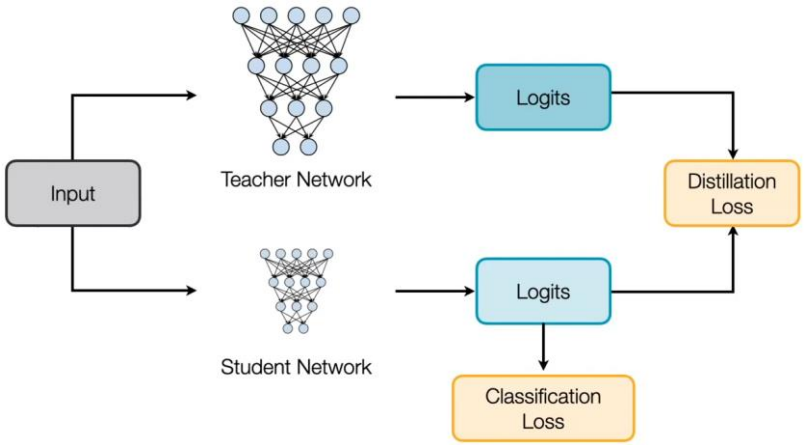
Overview

- What is knowledge distillation?
  - Knowledge distillation refers to the process of transferring the knowledge from a large model or set of models to a single smaller model that can be practically deployed under real-world constraints.





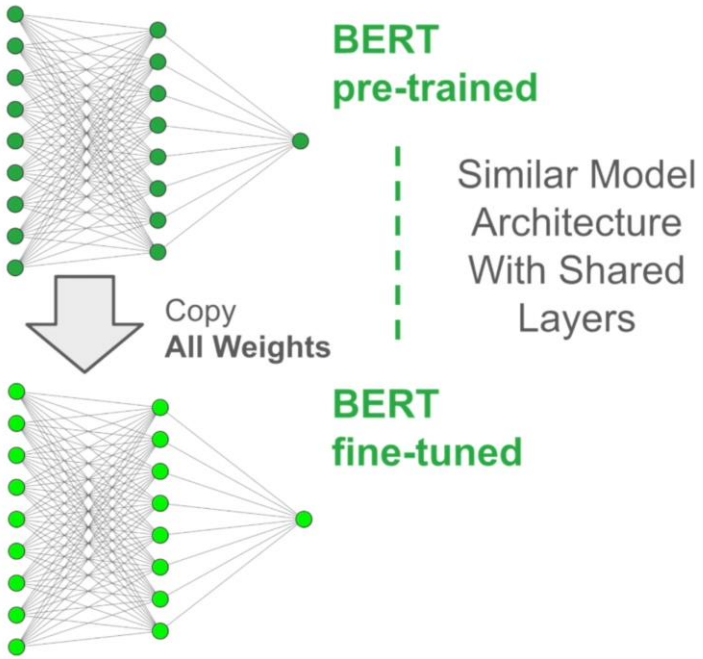
	Cloud AI	Tiny AI
Computation (fp32)	19.5 TFLOPS	MFLOPs
Memory	80GB	256kB
Neural Network	ResNet ViT-Large …	MCUNet MobileNetV2-Tiny …



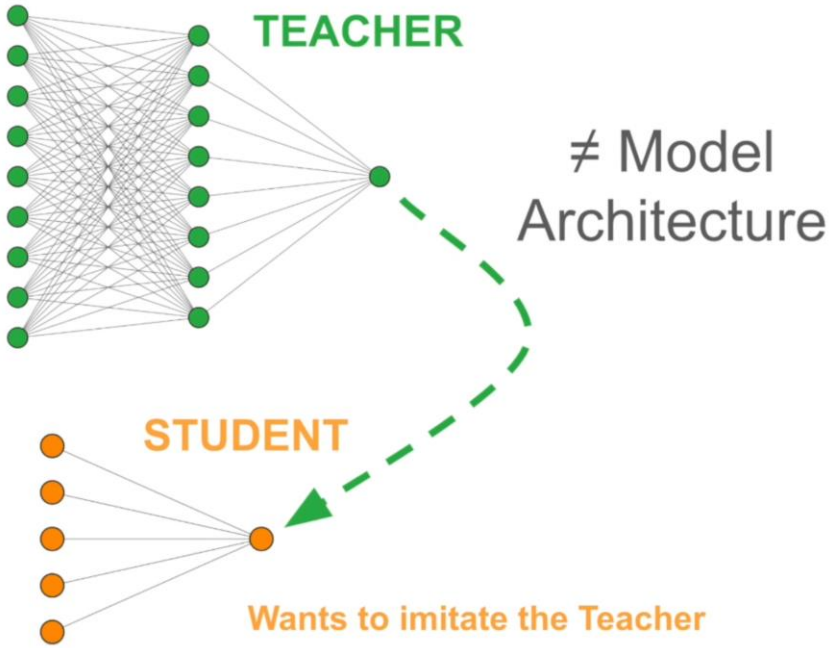
Overview

- What is the different between Transfer Learning and Knowledge Distillation?

Transfer Learning

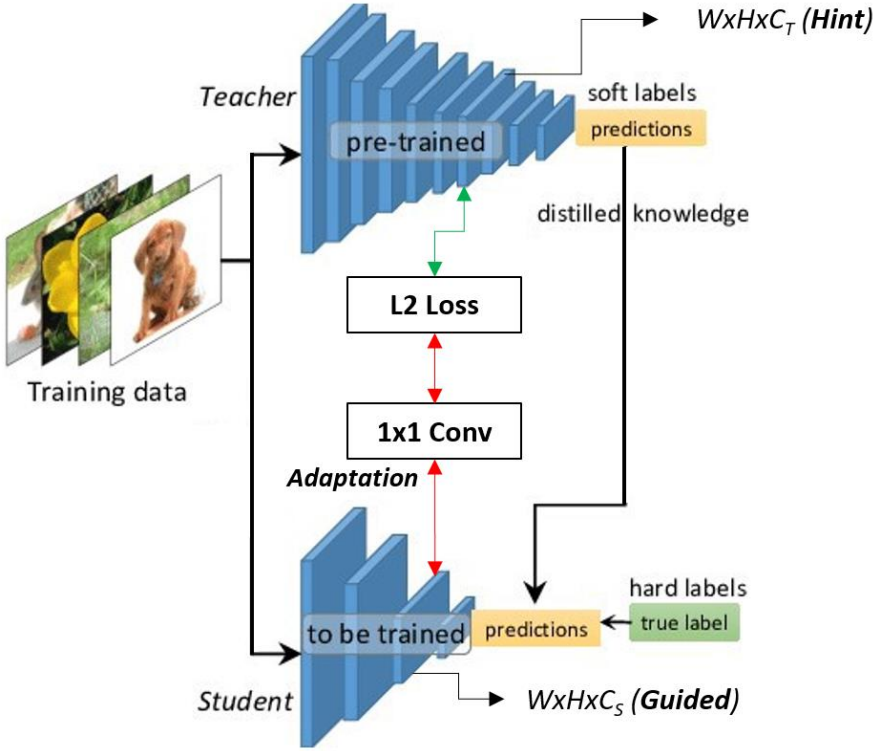


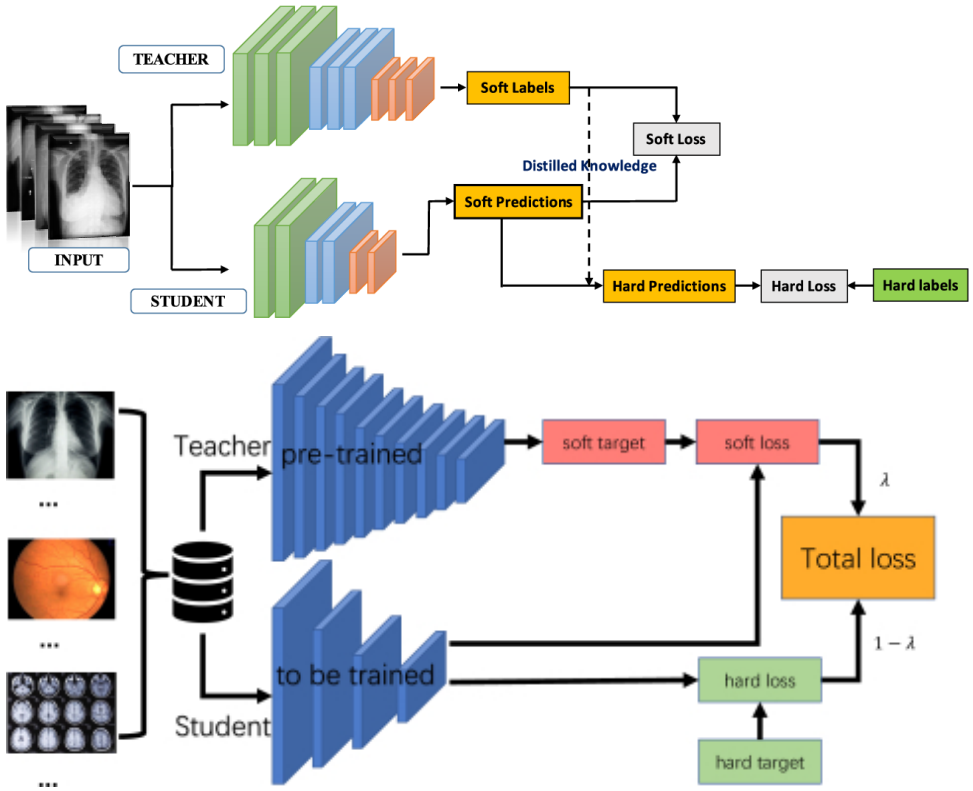
Knowledge Distillation



Applications

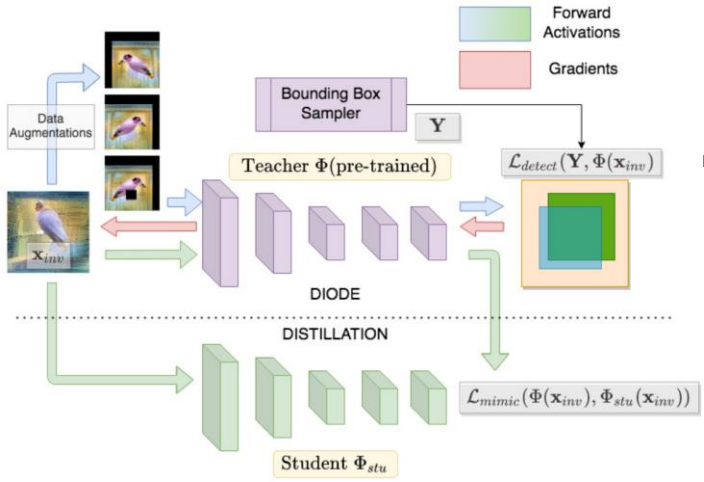
- Classification

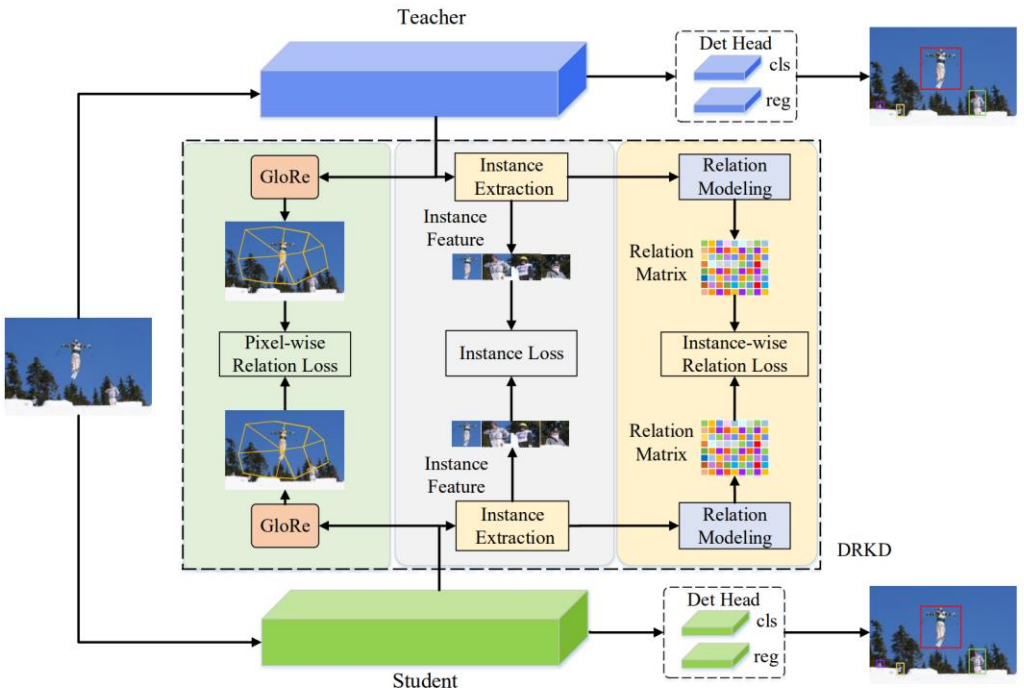




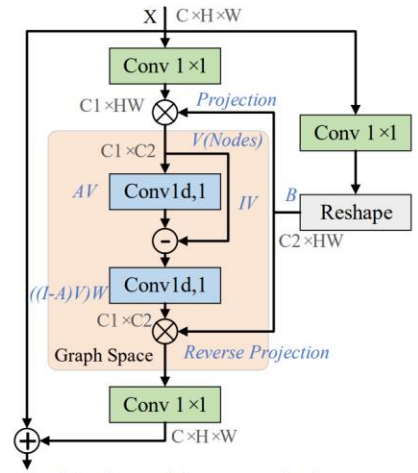
Applications

- Object Detection

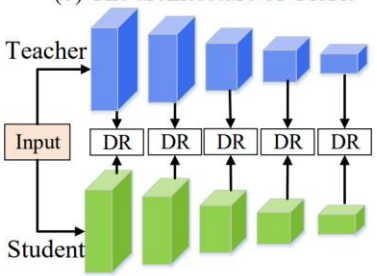




(a) Overview of the dual relation knowledge distillation



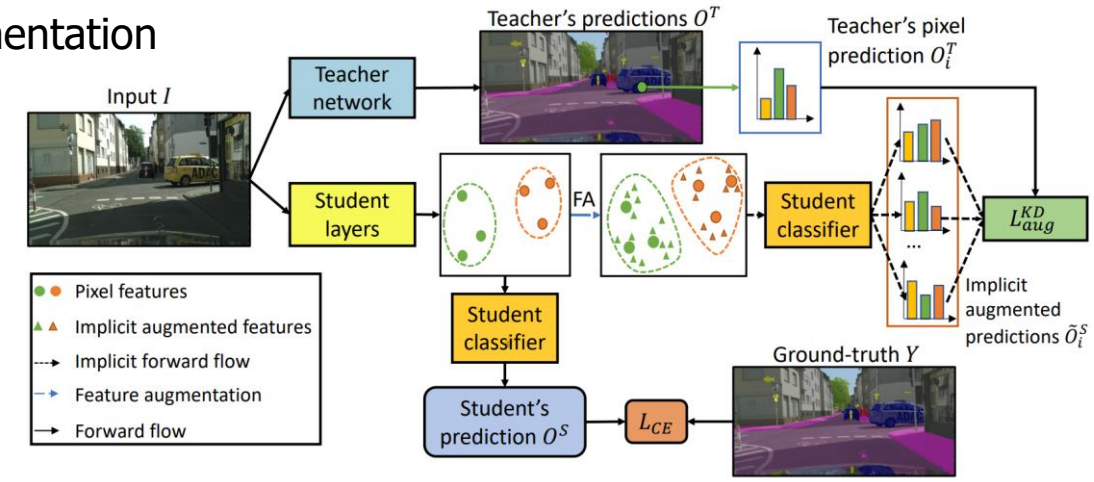
(b) The architecture of GloRe

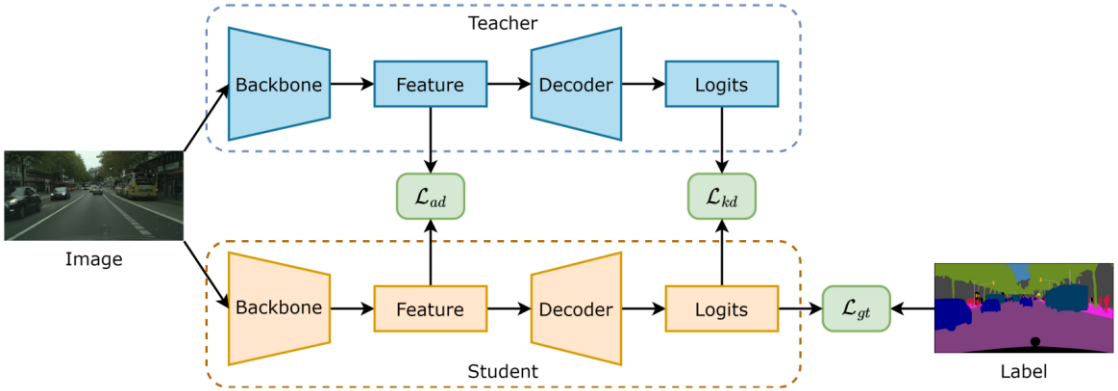


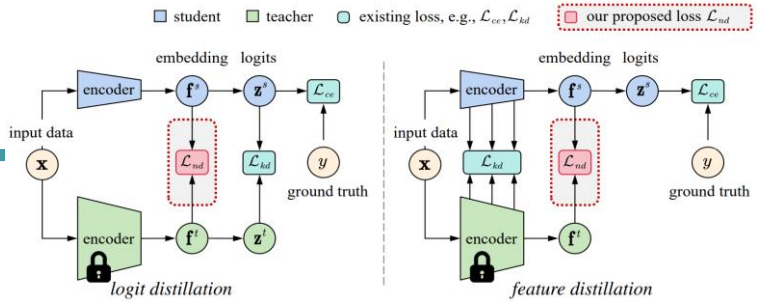
(c) Multi-scale DRKD

Applications

- Semantic Segmentation

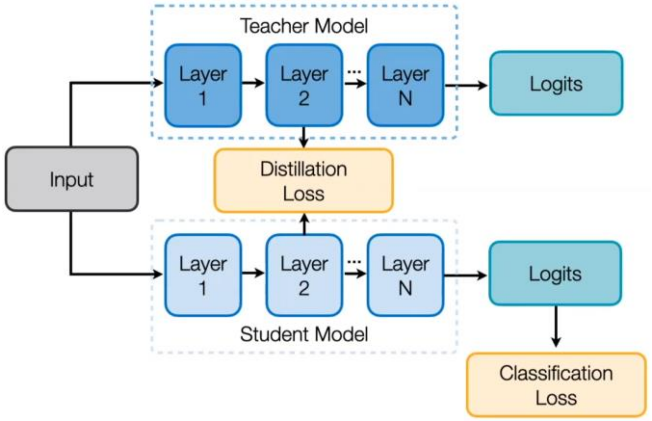




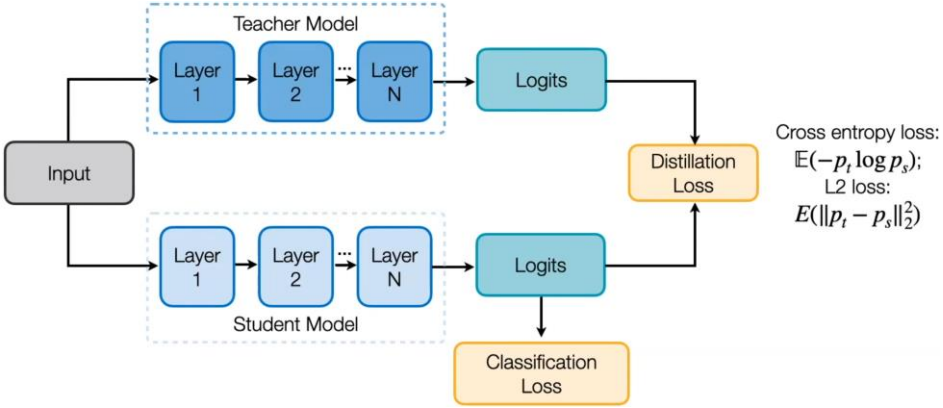


Knowledge Distillation Mechanism

Feature-based knowledge distillation

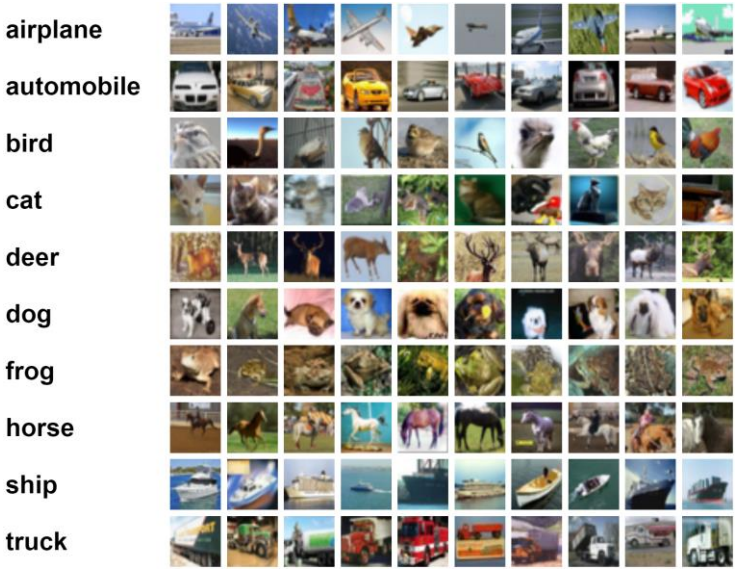


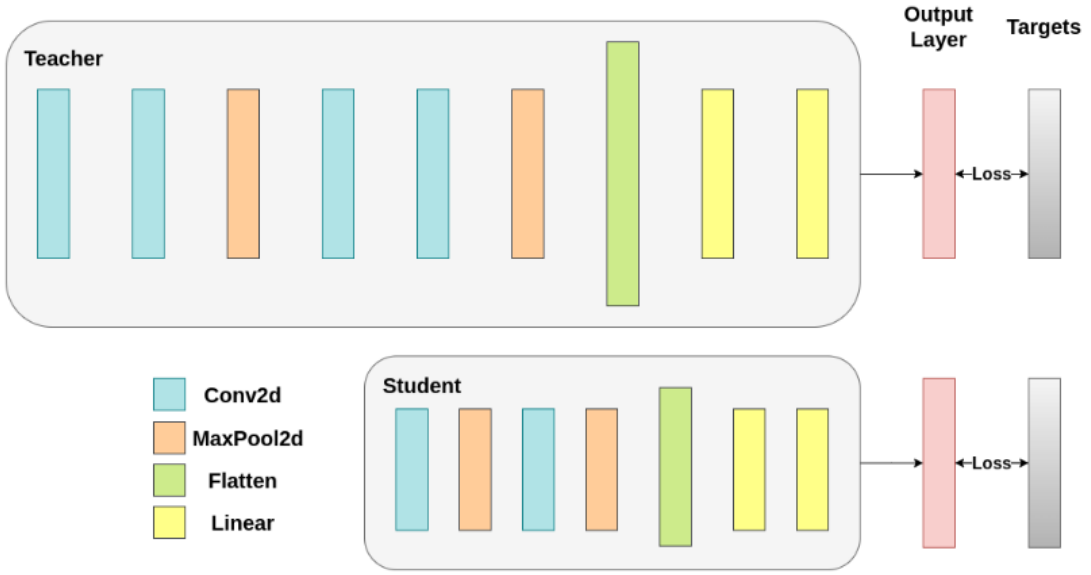
Logit-based knowledge distillation



Implementation

- Implementation
  - Dataset
    - CIFAR-10
      - 60,000 color images
      - 10 classes





Implementation

- Prerequisites
  - 1 GPU, 4GB of memory
  - PyTorch v2.0 or later
  - CIFAR-10 dataset

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision.transforms as transforms
import torchvision.datasets as datasets

# Check if GPU is available, and if not, use the CPU
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

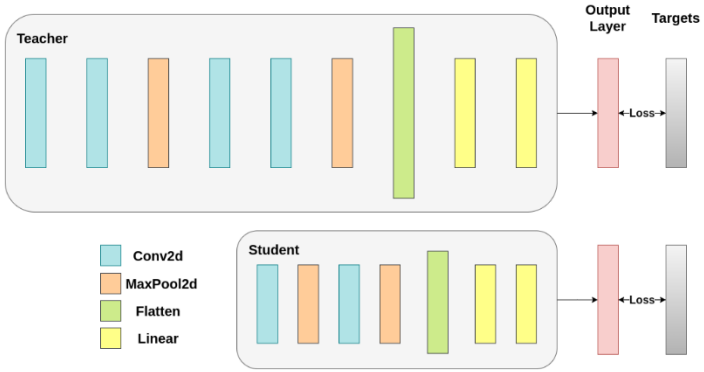
Implementation

- Preprocessing data

```
# Below we are preprocessing data for CIFAR-10. We use an arbitrary batch size of 128.
transforms_cifar = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])

# Loading the CIFAR-10 dataset:
train_dataset = datasets.CIFAR10(root='./data', train=True, download=True,
transform=transforms_cifar)
test_dataset = datasets.CIFAR10(root='./data', train=False, download=True,
transform=transforms_cifar)
#Dataloaders
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=128, shuffle=True,
num_workers=2)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=128, shuffle=False,
num_workers=2)
```

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data/cifar-10-python.tar.gz

  0%|          | 0/170498071 [00:00<?, ?it/s]
  0%|          | 622592/170498071 [00:00<00:27, 6210349.35it/s]
  5%|4         | 8486912/170498071 [00:00<00:03, 48771397.52it/s]
 12%|#1        | 20086784/170498071 [00:00<00:01, 79367591.17it/s]
 19%|#8        | 31752192/170498071 [00:00<00:01, 94044334.15it/s]
 25%|##5       | 43057152/170498071 [00:00<00:01, 100823260.25it/s]
 32%|###1      | 54132736/170498071 [00:00<00:01, 104195572.93it/s]
 38%|###8      | 65306624/170498071 [00:00<00:00, 106650125.66it/s]
 45%|####4     | 75988992/170498071 [00:00<00:00, 105374882.88it/s]
 51%|#####     | 86540288/170498071 [00:00<00:00, 105048267.06it/s]
 57%|#####6    | 97124352/170498071 [00:01<00:00, 105250257.59it/s]
 63%|######3   | 107675648/170498071 [00:01<00:00, 104361415.00it/s]
 69%|######9   | 118128640/170498071 [00:01<00:00, 102397650.84it/s]
 75%|#######5  | 128385024/170498071 [00:01<00:00, 102222393.58it/s]
 81%|########1 | 138641408/170498071 [00:01<00:00, 101991945.72it/s]
 87%|########7 | 148865024/170498071 [00:01<00:00, 101487635.30it/s]
 93%|#########3| 159023104/170498071 [00:01<00:00, 101154555.31it/s]
 99%|#########9| 169148416/170498071 [00:01<00:00, 100790133.02it/s]
100%|##########| 170498071/170498071 [00:01<00:00, 98113633.34it/s]
Extracting ./data/cifar-10-python.tar.gz to ./data
Files already downloaded and verified
```

Implementation



- Build a deeper neural network and a lightweight neural network

```python
# Deeper neural network class to be used as teacher:
class DeepNN(nn.Module):
    def __init__(self, num_classes=10):
        super(DeepNN, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 128, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.Conv2d(128, 64, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d(64, 64, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.Conv2d(64, 32, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),
        )
        self.classifier = nn.Sequential(
            nn.Linear(2048, 512),
            nn.ReLU(),
            nn.Dropout(0.1),
            nn.Linear(512, num_classes)
        )

    def forward(self, x):
        x = self.features(x)
        x = torch.flatten(x, 1)
        x = self.classifier(x)
        return x
```

```python
# Lightweight neural network class to be used as student:
class LightNN(nn.Module):
    def __init__(self, num_classes=10):
        super(LightNN, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 16, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d(16, 16, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),
        )
        self.classifier = nn.Sequential(
            nn.Linear(1024, 256),
            nn.ReLU(),
            nn.Dropout(0.1),
            nn.Linear(256, num_classes)
        )

    def forward(self, x):
        x = self.features(x)
        x = torch.flatten(x, 1)
        x = self.classifier(x)
        return x
```

Implementation

- Training

```
def train(model, train_loader, epochs, learning_rate, device):
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=learning_rate)

    model.train()

    for epoch in range(epochs):
        running_loss = 0.0
        for inputs, labels in train_loader:
            # inputs: A collection of batch_size images
            # labels: A vector of dimensionality batch_size with integers denoting class of
each image
            inputs, labels = inputs.to(device), labels.to(device)

            optimizer.zero_grad()
            outputs = model(inputs)

            # outputs: Output of the network for the collection of images. A tensor of
dimensionality batch_size x num_classes
            # labels: The actual labels of the images. Vector of dimensionality batch_size
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            running_loss += loss.item()

        print(f"Epoch {epoch+1}/{epochs}, Loss: {running_loss / len(train_loader)}")
```

Implementation

- Testing

```
def test(model, test_loader, device):
    model.to(device)
    model.eval()

    correct = 0
    total = 0

    with torch.no_grad():
        for inputs, labels in test_loader:
            inputs, labels = inputs.to(device), labels.to(device)

            outputs = model(inputs)
            _, predicted = torch.max(outputs.data, 1)

            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    accuracy = 100 * correct / total
    print(f"Test Accuracy: {accuracy:.2f}%")
    return accuracy
```

Implementation

❑ Testing

```
torch.manual_seed(42)
nn_deep = DeepNN(num_classes=10).to(device)
train(nn_deep, train_loader, epochs=10, learning_rate=0.001, device=device)
test_accuracy_deep = test(nn_deep, test_loader, device)

# Instantiate the lightweight network:
torch.manual_seed(42)
nn_light = LightNN(num_classes=10).to(device)
```

```
train(nn_light, train_loader, epochs=10, learning_rate=0.001, device=device)
test_accuracy_light_ce = test(nn_light, test_loader, device)
```

```
total_params_deep = "{:,}".format(sum(p.numel() for p in nn_deep.parameters()))
print(f"DeepNN parameters: {total_params_deep}")
total_params_light = "{:,}".format(sum(p.numel() for p in nn_light.parameters()))
print(f"LightNN parameters: {total_params_light}")
```

Implementation

```
DeepNN parameters: 1,186,986
LightNN parameters: 267,738
```

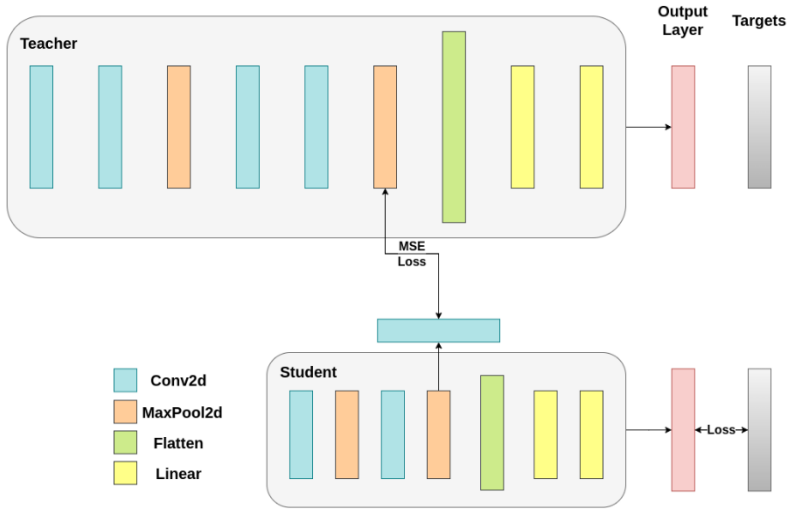
- Testing without knowledge distillation

```
Epoch 1/10, Loss: 1.326111341681322
Epoch 2/10, Loss: 0.8613990703812036
Epoch 3/10, Loss: 0.6715770787595178
Epoch 4/10, Loss: 0.5298002164839478
Epoch 5/10, Loss: 0.4070205725062534
Epoch 6/10, Loss: 0.3021288668865438
Epoch 7/10, Loss: 0.2165230909538696
Epoch 8/10, Loss: 0.17487331569347236
Epoch 9/10, Loss: 0.13452480149352947
Epoch 10/10, Loss: 0.11425360603748685
Test Accuracy: 74.74%
```

```
Epoch 1/10, Loss: 1.4659436395406114
Epoch 2/10, Loss: 1.15358717240336
Epoch 3/10, Loss: 1.0244640207961393
Epoch 4/10, Loss: 0.9230101669535917
Epoch 5/10, Loss: 0.8481090013938182
Epoch 6/10, Loss: 0.7807804391817059
Epoch 7/10, Loss: 0.716010201922463
Epoch 8/10, Loss: 0.6586926555084756
Epoch 9/10, Loss: 0.6042922610974373
Epoch 10/10, Loss: 0.5536436764023188
Test Accuracy: 70.82%
```

Implementation

- Testing with knowledge distillation

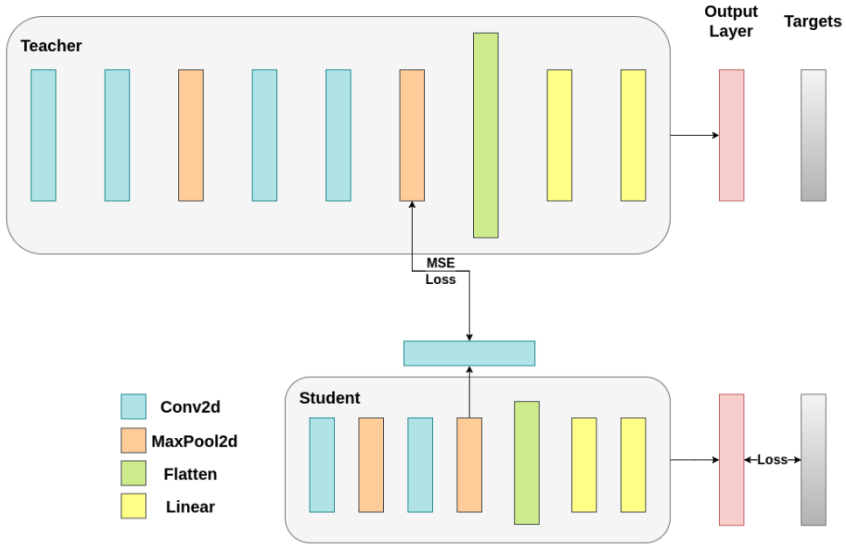



```
def train_mse_loss(teacher, student, train_loader, epochs, learning_rate, feature_map_weight,
ce_loss_weight, device):
    ce_loss = nn.CrossEntropyLoss()
    mse_loss = nn.MSELoss()
    optimizer = optim.Adam(student.parameters(), lr=learning_rate)

    teacher.to(device)
    student.to(device)
    teacher.eval()  # Teacher set to evaluation mode
    student.train() # Student to train mode

    for epoch in range(epochs):
        running_loss = 0.0
        for inputs, labels in train_loader:
            inputs, labels = inputs.to(device), labels.to(device)

            optimizer.zero_grad()

            # Again ignore teacher logits
            with torch.no_grad():
                _, teacher_feature_map = teacher(inputs)

            # Forward pass with the student model
            student_logits, regressor_feature_map = student(inputs)
```

```
            # Forward pass with the student model
            student_logits, regressor_feature_map = student(inputs)

            # Calculate the loss
            hidden_rep_loss = mse_loss(regressor_feature_map, teacher_feature_map)

            # Calculate the true label loss
            label_loss = ce_loss(student_logits, labels)

            # Weighted sum of the two losses
            loss = feature_map_weight * hidden_rep_loss + ce_loss_weight * label_loss

            loss.backward()
            optimizer.step()

            running_loss += loss.item()

        print(f"Epoch {epoch+1}/{epochs}, Loss: {running_loss / len(train_loader)}")
```

Implementation

- Testing with knowledge distillation

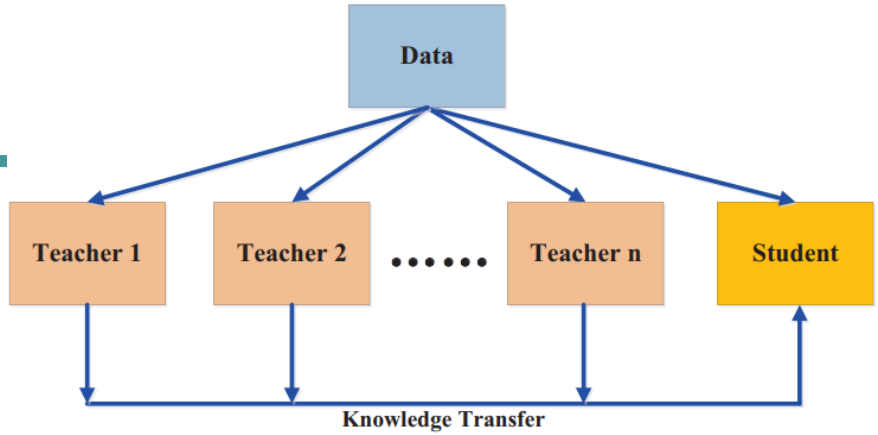



```
torch.manual_seed(42)
modified_nn_light_reg = ModifiedLightNNRegressor(num_classes=10).to(device)

# We do not have to train the modified deep network from scratch of course, we just load its
weights from the trained instance
modified_nn_deep_reg = ModifiedDeepNNRegressor(num_classes=10).to(device)
modified_nn_deep_reg.load_state_dict(nn_deep.state_dict())

# Train and test once again
train_mse_loss(teacher=modified_nn_deep_reg, student=modified_nn_light_reg,
train_loader=train_loader, epochs=10, learning_rate=0.001, feature_map_weight=0.25,
ce_loss_weight=0.75, device=device)
test_accuracy_light_ce_and_mse_loss = test_multiple_outputs(modified_nn_light_reg, test_loader,
device)
```

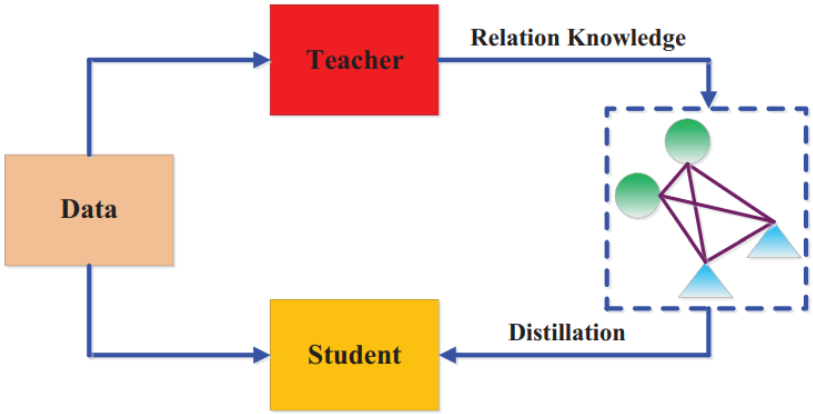
```
Epoch 1/10, Loss: 1.6942063466362332
Epoch 2/10, Loss: 1.3241598197566273
Epoch 3/10, Loss: 1.1833922349278578
Epoch 4/10, Loss: 1.0902259563241163
Epoch 5/10, Loss: 1.013230376231396
Epoch 6/10, Loss: 0.9511413255615917
Epoch 7/10, Loss: 0.8970147511538338
Epoch 8/10, Loss: 0.849607605610967
Epoch 9/10, Loss: 0.8063493360338918
Epoch 10/10, Loss: 0.7692910728552153
Test Accuracy: 71.06%
```

```
Teacher accuracy: 74.74%
Student accuracy without teacher: 70.82%
```

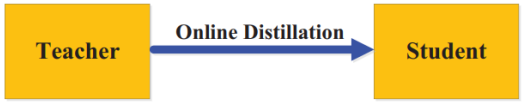
Training

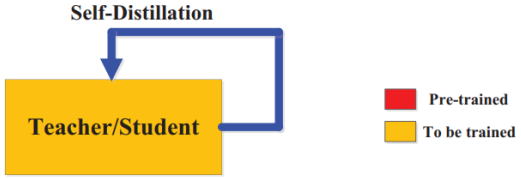


Offline Knowledge Distillation

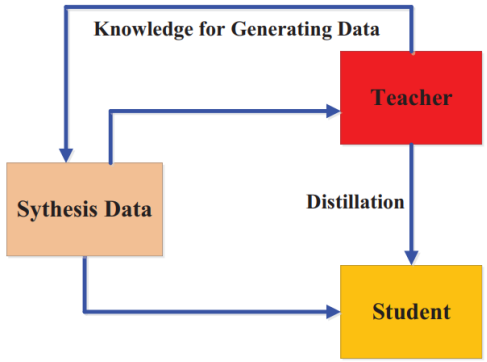


Online Knowledge Distillation





Offline Knowledge Distillation

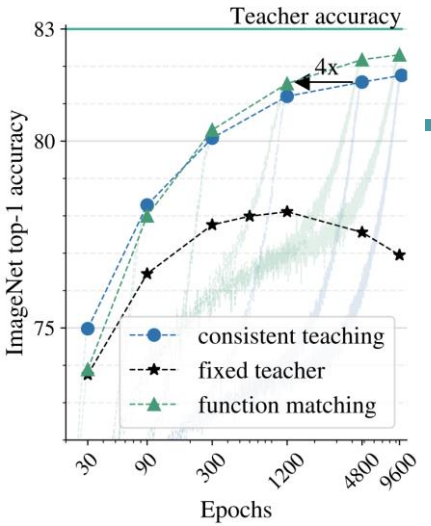


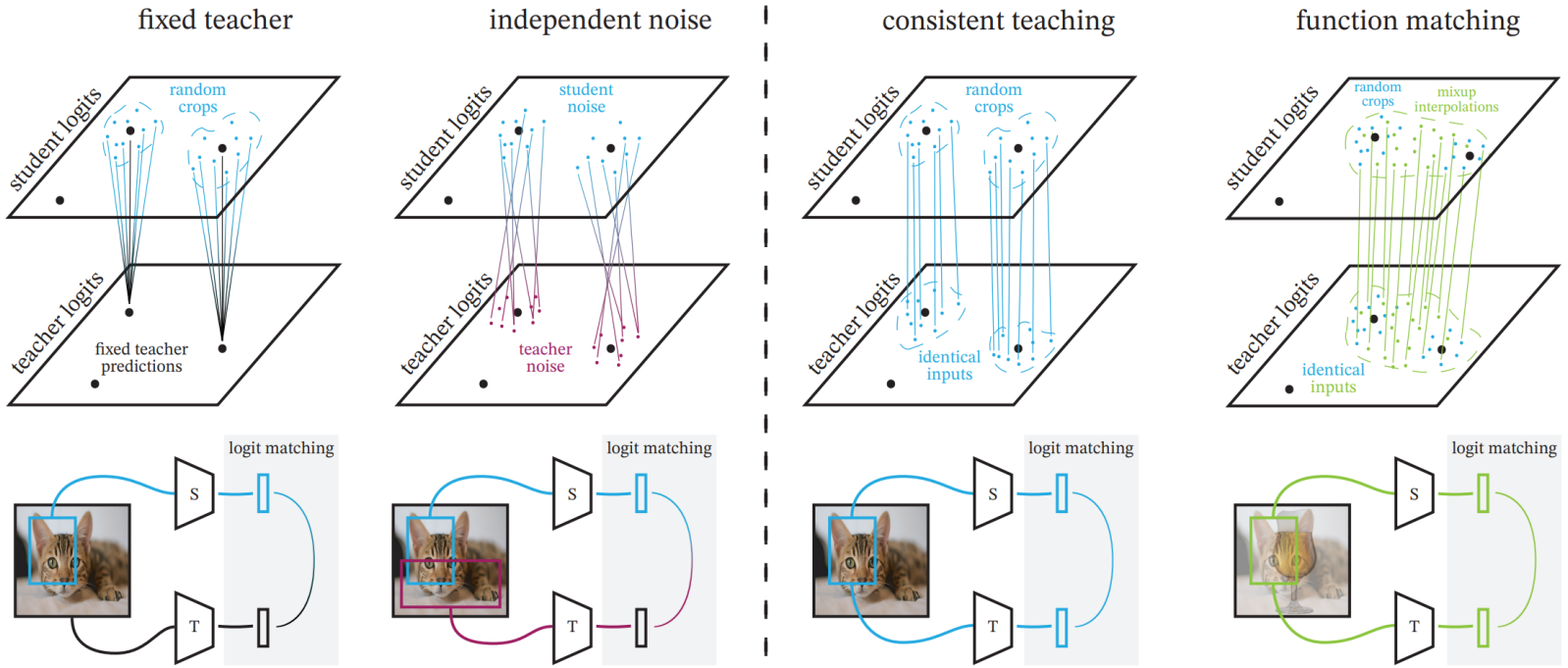
Topic 3: Knowledge Distillation

What is a good teacher?

Method

- A good teacher is patient and consistent



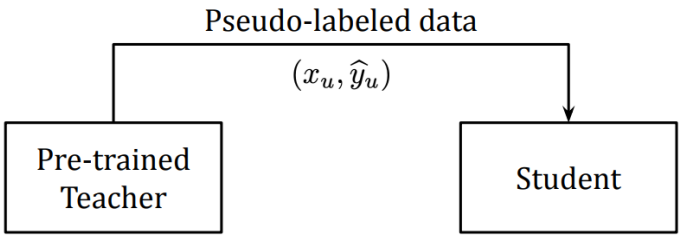


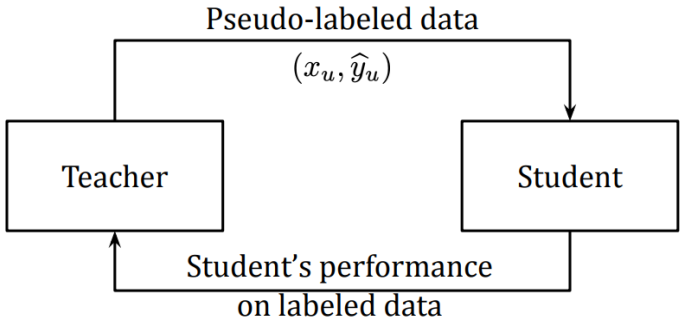
Topic 3: Knowledge Distillation

Method

- Meta Pseudo Labels

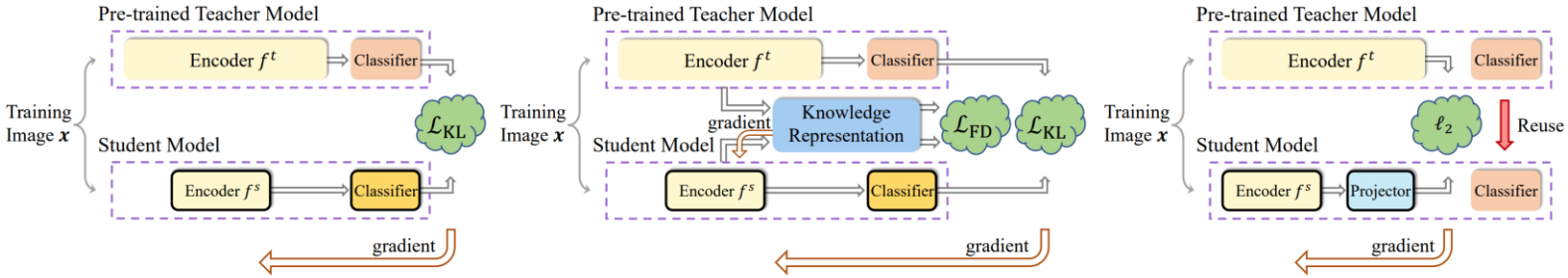
	Method	CIFAR-10-4K (mean ± std)	SVHN-1K (mean ± std)	ImageNet-10% Top-1	ImageNet-10% Top-5
Label Propagation Methods	Temporal Ensemble [35]	83.63 ± 0.63	92.81 ± 0.27	–	
	Mean Teacher [64]	84.13 ± 0.28	94.35 ± 0.47	–	
	VAT + EntMin [44]	86.87 ± 0.39	94.65 ± 0.19	–	83.39
	LGA + VAT [30]	87.94 ± 0.19	93.42 ± 0.36	–	
	ICT [71]	92.71 ± 0.02	96.11 ± 0.04	–	
	MixMatch [5]	93.76 ± 0.06	96.73 ± 0.31	–	
	ReMixMatch [4]	94.86 ± 0.04	97.17 ± 0.30	–	
	EnAET [72]	94.65	97.08	–	
	FixMatch [58]	95.74 ± 0.05	97.72 ± 0.38	71.5	89.1
	UDA* [76]	94.53 ± 0.18	97.11 ± 0.17	68.07	88.19
Self-Supervised Methods	SimCLR [8, 9]	–	–	71.7	90.4
	MOCOv2 [10]	–	–	71.1	–
	PCL [38]	–	–	–	85.6
	PIRL [43]	–	–	–	84.9
	BYOL [21]	–	–	68.8	89.0
	Meta Pseudo Labels	**96.11 ± 0.07**	**98.01 ± 0.07**	**73.89**	**91.38**





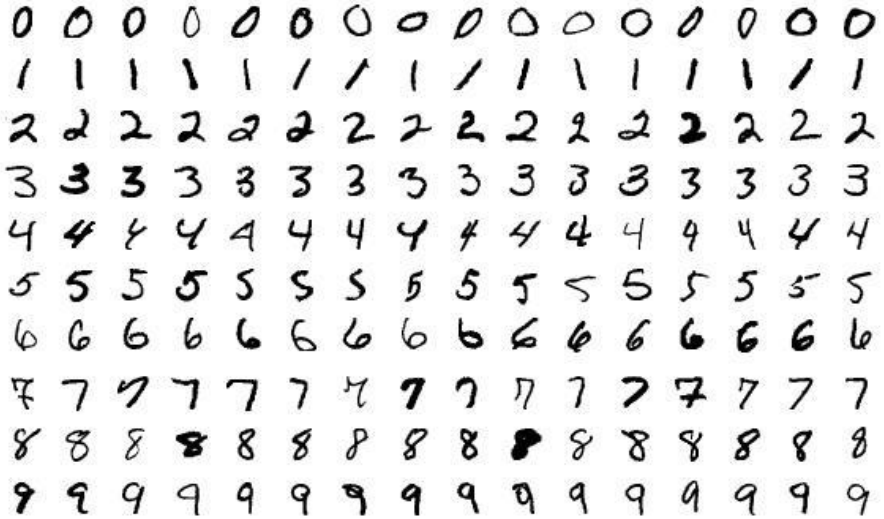
Method

❑ Knowledge Distillation With the Reused Teacher Classifier

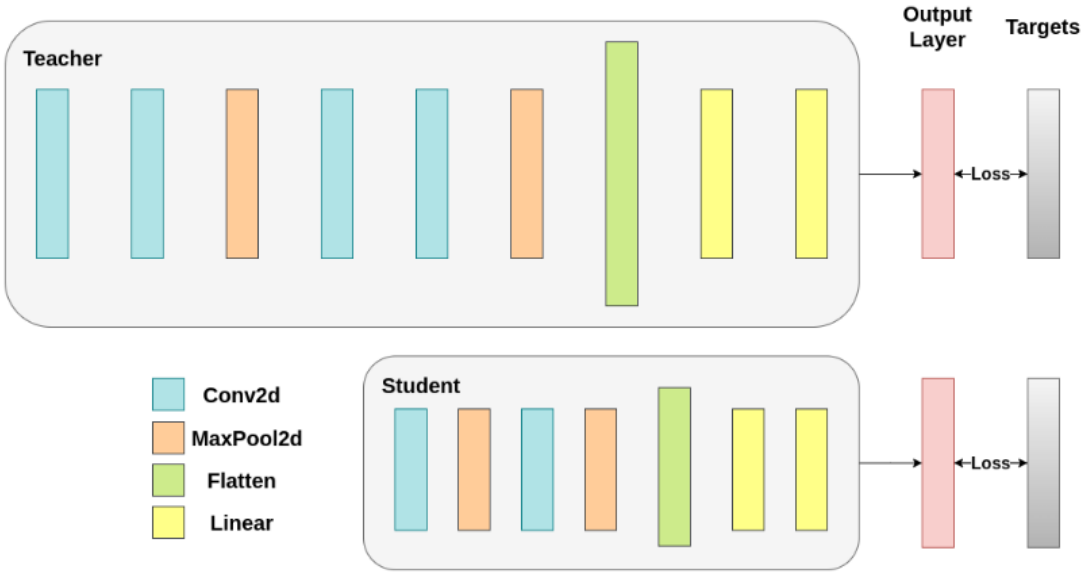


Student	WRN-40-1 71.92 ± 0.17	ResNet-8x4 73.09 ± 0.30	ResNet-110 74.37 ± 0.17	ResNet-116 74.46 ± 0.09	VGG-8 70.46 ± 0.29	ResNet-8x4 73.09 ± 0.30	ShuffleNetV2 72.60 ± 0.12
KD [24]	74.12 ± 0.29	74.42 ± 0.05	76.25 ± 0.34	76.14 ± 0.32	72.73 ± 0.15	75.28 ± 0.18	75.60 ± 0.21
FitNet [39]	74.17 ± 0.22	74.32 ± 0.08	76.08 ± 0.13	76.20 ± 0.17	72.91 ± 0.18	75.02 ± 0.31	75.82 ± 0.22
AT [53]	74.67 ± 0.18	75.07 ± 0.03	76.67 ± 0.28	76.84 ± 0.25	71.90 ± 0.13	75.74 ± 0.09	75.41 ± 0.10
SP [46]	73.90 ± 0.17	74.29 ± 0.07	76.43 ± 0.39	75.99 ± 0.26	73.12 ± 0.10	74.84 ± 0.08	75.77 ± 0.08
VID [1]	74.59 ± 0.17	74.55 ± 0.10	76.17 ± 0.22	76.53 ± 0.24	73.19 ± 0.23	75.56 ± 0.13	75.22 ± 0.07
CRD [44]	74.80 ± 0.33	75.59 ± 0.07	76.86 ± 0.09	76.83 ± 0.13	73.54 ± 0.19	75.78 ± 0.27	77.04 ± 0.61
SRRL [50]	74.64 ± 0.14	75.39 ± 0.34	76.75 ± 0.14	77.19 ± 0.09	73.23 ± 0.16	76.12 ± 0.18	76.19 ± 0.35
SemCKD [6]	74.41 ± 0.16	76.23 ± 0.04	76.62 ± 0.14	76.69 ± 0.48	75.27 ± 0.13	75.85 ± 0.16	77.62 ± 0.32
SimKD	**75.56 ± 0.27**	**78.08 ± 0.15**	**77.82 ± 0.15**	**77.90 ± 0.11**	**75.76 ± 0.12**	**76.75 ± 0.23**	**78.39 ± 0.27**
Teacher	WRN-40-2 76.31	ResNet-32x4 79.42	ResNet-110x2 78.18	ResNet-110x2 78.18	ResNet-32x4 79.42	WRN-40-2 76.31	ResNet-32x4 79.42

Implementation



- Implementation
  - Dataset
    - MNIST



Implementation

- Prerequisites
  - Keras
    - Setup
    - Construct Distiller() class
    - Create student and teacher models
    - Prepare the dataset
    - Train the teacher
    - Distill teacher to student

```
import os

import keras
from keras import layers
from keras import ops
import numpy as np
```

Implementation

- Construct Distiller() class

```python
class Distiller(keras.Model):
    def __init__(self, student, teacher):
        super().__init__()
        self.teacher = teacher
        self.student = student

    def compile(
        self,
        optimizer,
        metrics,
        student_loss_fn,
        distillation_loss_fn,
        alpha=0.1,
        temperature=3,
    ):
        super().compile(optimizer=optimizer, metrics=metrics)
        self.student_loss_fn = student_loss_fn
        self.distillation_loss_fn = distillation_loss_fn
        self.alpha = alpha
        self.temperature = temperature

    def compute_loss(
        self, x=None, y=None, y_pred=None, sample_weight=None, allow_empty=False
    ):
        teacher_pred = self.teacher(x, training=False)
        student_loss = self.student_loss_fn(y, y_pred)

        distillation_loss = self.distillation_loss_fn(
            ops.softmax(teacher_pred / self.temperature, axis=1),
            ops.softmax(y_pred / self.temperature, axis=1),
        ) * (self.temperature**2)

        loss = self.alpha * student_loss + (1 - self.alpha) * distillation_loss
        return loss

    def call(self, x):
        return self.student(x)
```

Implementation

- Create student and teacher models

```
# Create the teacher
teacher = keras.Sequential(
    [
        keras.Input(shape=(28, 28, 1)),
        layers.Conv2D(256, (3, 3), strides=(2, 2), padding="same"),
        layers.LeakyReLU(negative_slope=0.2),
        layers.MaxPooling2D(pool_size=(2, 2), strides=(1, 1), padding="same"),
        layers.Conv2D(512, (3, 3), strides=(2, 2), padding="same"),
        layers.Flatten(),
        layers.Dense(10),
    ],
    name="teacher",
)
```

```
# Create the student
student = keras.Sequential(
    [
        keras.Input(shape=(28, 28, 1)),
        layers.Conv2D(16, (3, 3), strides=(2, 2), padding="same"),
        layers.LeakyReLU(negative_slope=0.2),
        layers.MaxPooling2D(pool_size=(2, 2), strides=(1, 1), padding="same"),
        layers.Conv2D(32, (3, 3), strides=(2, 2), padding="same"),
        layers.Flatten(),
        layers.Dense(10),
    ],
    name="student",
)
```

Implementation

- Prepare the dataset

```
# Prepare the train and test dataset.
batch_size = 64
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()

# Normalize data
x_train = x_train.astype("float32") / 255.0
x_train = np.reshape(x_train, (-1, 28, 28, 1))

x_test = x_test.astype("float32") / 255.0
x_test = np.reshape(x_test, (-1, 28, 28, 1))
```

- Train the teacher

```
# Train teacher as usual
teacher.compile(
    optimizer=keras.optimizers.Adam(),
    loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=[keras.metrics.SparseCategoricalAccuracy()],
)

# Train and evaluate teacher on data.
teacher.fit(x_train, y_train, epochs=5)
teacher.evaluate(x_test, y_test)
```

Implementation

- Distill teacher to student

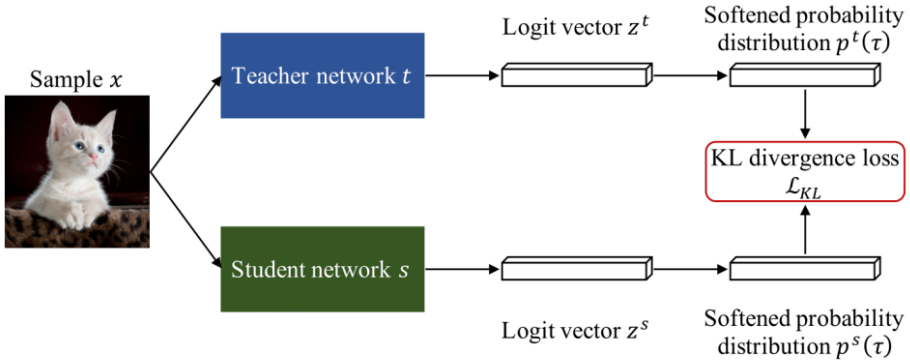
```
# Initialize and compile distiller
distiller = Distiller(student=student, teacher=teacher)
distiller.compile(
    optimizer=keras.optimizers.Adam(),
    metrics=[keras.metrics.SparseCategoricalAccuracy()],
    student_loss_fn=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    distillation_loss_fn=keras.losses.KLDivergence(),
    alpha=0.1,
    temperature=10,
)

# Distill teacher to student
distiller.fit(x_train, y_train, epochs=3)

# Evaluate student on test dataset
distiller.evaluate(x_test, y_test)
```

*Kim, T., et al. "Comparing kullback-leibler divergence and mean squared error loss in knowledge distillation. *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence (IJCAI-21).*

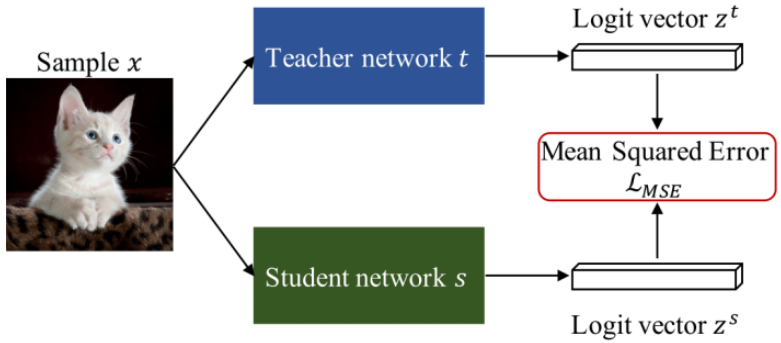
Comparing KLD vs MSE

- KLD vs MSE

WRN-28-4	WRN-16-4	WRN-16-2	Test accuracy
X	X	$\mathcal{L}_{CE}$	72.68 %
X	$\mathcal{L}_{CE}$ (77.28%)	$\mathcal{L}_{KL}(\tau = 3)$	74.84 %
		$\mathcal{L}_{KL}(\tau = 20)$	75.42 %
		$\mathcal{L}_{MSE}$	75.58 %
$\mathcal{L}_{CE}$ (78.88%)	X	$\mathcal{L}_{KL}(\tau = 3)$	74.24 %
		$\mathcal{L}_{KL}(\tau = 20)$	75.15 %
		$\mathcal{L}_{MSE}$	75.54 %
$\mathcal{L}_{CE}$ (78.88%)	$\mathcal{L}_{KL}(\tau = 3)$ (78.76%)	$\mathcal{L}_{KL}(\tau = 3)$	74.52 %
		$\mathcal{L}_{KL}(\tau = 20)$	75.47 %
		$\mathcal{L}_{MSE}$	**75.78 %**
	$\mathcal{L}_{MSE}$ (79.03%)	$\mathcal{L}_{KL}(\tau = 3)$	74.83 %
		$\mathcal{L}_{KL}(\tau = 20)$	75.47 %
		$\mathcal{L}_{MSE}$	75.48 %



(a) KD with $\mathcal{L}_{KL}$ between the softened probability distributions.



(b) KD with $\mathcal{L}_{MSE}$ between the logit vectors.

THANK YOU