Project B: Knowledge Distillation for Building Lightweight Deep Learning Models in Visual Classification Tasks

***Abstract*—This paper is a report for Project A of ECE1512 2022W, University of Toronto. In this paper, we introduce two tasks assigned to us in detail, including the implementation and evaluation of KD based on two couples of Teacher-Student Models. For the advanced knowledge distillation method, we choose the Early-Stopping Knowledge Distillation. We reviewed the paper 2 in the Lab Manual and implemented the corresponding methodology. Moreover, we fit the methods on different models and evaluated their performance accordingly. This project remote repository is attached on GitHub: https://github.com/Awesome-guys-in-ECE1747/ECE1512_2022W_ProjectRepo_J.Xu_and_W.Xu**

I. Task 1: Knowledge Distillation in MNIST dataset

In this task, we basically implement the load & preprocess of dataset, model construction, training and evaluation for teacher and student models using our own training functions. In addition, we implement the Early-Stopping Knowledge Distillation as the improving algorithm.

A. Question 1

In this section, we read the paper of Geoffrey Hinton [2], and answer the assigned questions.

1) SubQuestion a: The purpose of using Knowledge Distillation is to compress the knowledge in an ensemble to a single model which is much easier to deploy without losing much accuracy.

As is widely acknowledged, in machine learning, large models usually have higher knowledge capacity than small models, thus improving state of the art on more tasks. However, with the increase of parameters in models, it takes more time and money to deploy, train and evaluate larger models. In the resource-restricted systems such as mobile devices, it's hard for them to train the models. What's more, in real-time systems, although they have enough resources for training, the low efficiency means that it's inapplicable. In order to solve it, smaller models are proposed to mimic the prediction of large models, especially in edge device such as mobile device. In a word, the goal of knowledge distillation is transferring from the teacher model to the student model and making a lightweight model that is fast, memory-efficient, and energy-efficient.

2) SubQuestion b: The knowledge is a learned mapping from input vectors to output vectors.

In the previous studies, researchers tend to consider the parameters in the model as the knowledge, but it's inapplicable here. If we want to compress the model without losing too much accuracy, we have to update this concept. The basic nature of all the functions and models are mapping from the input to the output. If we want the student model which is the simple model to act like teacher while having less parameters, it means they should have similar output vectors. So both student and teacher models should have similar functions of mapping from the same input vectors to the similar output vectors.

3) SubQuestion c: The temperature hyperparameter T is a parameter used to adjust the corresponding loss values for different labels in the soft targets. T makes the labels which have low probabilities influence more in the cross-entropy cost function during the transfer stage.

In our normal training, we only pay attention to the difference between result with highest probability and ground truth. This kind of similarity is constructed based on large amount of samples. When the soft targets have high entropy, they'll provide much more information for the student model training than hard targets. However, in this kind of situation, the results which have a low probability contribute little to the loss functions, thus providing little information for the knowledge transferring stage. For example, the teacher model in task1 has a high accuracy on MNIST, much of the information about the learned model is contained in the options which have very small probabilities in the soft targets. To solve it, Hinton [2] put forward the temperature hyperparameter T to magnify the corresponding loss values of low probability options.

So the effect of the temperature hyperparameter is decreasing the influence of highest probability prediction and increase the influence of other labels in the knowledge transferring stage.

4) SubQuestion d: This is the graph of Knowledge Distillation. For the teacher model, assuming that the output vector of the final fully connection layer is z, then z_i will be the logits of label i. The predicted probability for the input vector to belong to label i is calculated with softmax function:

$$p_i = \frac{exp(z_i)}{\sum_j exp(z_j)}$$

where p_i is the probability of being label i, z_i is the logits of label i, j is the list of all the labels. According to it, we will calculate the loss value of teacher model like the normal

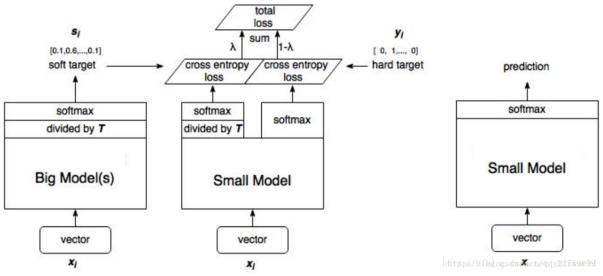

Fig. 1. Knowledge Distillation

neural networks. The formula is:

$$L_{teacher} = -\sum_{i}^{N} y_i log(p_i)$$

where N is the number of classes in total, y_i is a Boolean value, if the sample in input vector belongs to label i, it'll be 1, else it'll be 0. p_i is the predicted probability for the input vector to belong to label i.

For the student model, we will first talk about "how does the task balance parameter affect student learning?". The temperature hyperparameter T controls the importance and influence of every soft targets by revising the predicted probability for the input vector to belong to label i. The revised version of probability for label i under temperature T, named q_i^T, is:

$$q_i^T = \frac{exp(z_i/T)}{\sum_j exp(z_j/T)}$$

where q_i^T is the probability of being label i, z_i is the logits of label i in student model, j is the list of all the labels. Higher temperature will make the probability distribution more balanced. To be specific, suppose:

$$T \to \infty$$

, then every labels will have the same probabilities. On the contrary, if

$$T \to 0$$

soft targets will become one-hot tensor, which is named "hard targets".

Let's continue to talk about the loss value of student model. The traditional knowledge distillation is the combination of both the student loss and distillation loss. The formula for the combination is:

$$L = \alpha L_{soft} + \beta L_{hard}$$

where L_{soft} is the distill loss (corresponding to soft targets) and student loss (corresponding to hard targets). α is the weight of L_{soft} which is defined manually, and β is the weight of L_{hard} which equals $(1-\alpha)$ here. For the student loss, it should be similar to teacher model. This is named L_{hard} for it's the hard targets loss of the student model. The formula is:

$$L_{hard} = -\sum_{i}^{N} y_i log(q_i^1)$$

where N is the number of classes in total, y_i is a Boolean value, if the sample in input vector belongs to label i, it'll be 1, else it'll be 0. q_i^1 is the predicted probability for the input vector to belong to label i in temperature 1. For the soft targets loss, the formula is:

$$L_{soft} = -\sum_{i}^{N} p_i^T log(q_i^T)$$

where N is the number of classes in total, p_i^T is the predicted probability for the input vector in teacher model to belong to label i in temperature T, q_i^T is the predicted probability for the input vector in student model to belong to label i in temperature T. To explain p_i^T in detail, we listed the formula too:

$$p_i^T = \frac{exp(v_i/T)}{\sum_j exp(v_j/T)}$$

where p_i^T is the probability of being label i, v_i is the logits of label i in teacher model, j is the list of all the labels.

5) SubQuestion e: From our perspective, the knowledge distillation can be considered as a regularization technique. As is illustrated in the paper [2], we can use soft targets to prevent overfitting. It's the same effect as regularization. In an unbalanced dataset, assume one of the classes named 'A' has many more samples than other classes, it's highly likely that the model with a full softmax layer over all classes will be overfitting on this special class A. The reason is that: if there are more samples of class A in the input, more information will be applied into the model, thus making the information of other classes which have low predicted possibilities become less important. However, the information in these low possibilities in the soft labels is very important. They contains the information which traditional one-hot labels don't have.

In contrast, if we use soft targets of knowledge distillation as the training targets, we can prevent class A from passing too much information into the model, thus making the information of other labels which have low possibilities more important. If we adjust the hyperparameters carefully, we can get balanced information from both the large-sized labels and small-sized labels, which will lead to the good performance of the model. It's the same as regularization technique.

B. Question 2

In this section, we build a cumbersome teacher and a lightweight student model training on MNIST dataset. The network structure is plotted by $model : summary()$. The teacher model contains two convolutional layers with relu function, two max-pooling layers, a flatten layer and two dense layers. It's not a complex network, the implementation code is as followed.

```
from keras.models import Sequential
from keras.layers import Input, Conv2D,
    MaxPooling2D, Dense, Flatten, Dropout

# print(mnist_test)
```

```
# Build CNN teacher.
def build_cnn():
  cnn_model = Sequential()
  cnn_model.add(Input((28,28,1)))
  cnn_model.add(Conv2D(filters=32,
      kernel_size=(3,3), strides= (1,1),
      activation='relu'))
  cnn_model.add(MaxPooling2D(pool_size=(2,2)
      , strides= (1,1)))
  cnn_model.add(Conv2D(filters=64,
      kernel_size=(3,3), strides= (1,1),
      activation='relu'))
  cnn_model.add(MaxPooling2D(pool_size=(2,2)
      , strides= (2,2)))
  cnn_model.add(Flatten())
  cnn_model.add(Dropout(rate=0.5))
  cnn_model.add(Dense(units=128, activation=
      'relu'))
  cnn_model.add(Dropout(rate=0.5))
  cnn_model.add(Dense(NUM_CLASSES,
      activation='softmax'))
  return cnn_model

cnn_model=build_cnn()
print("\n\n\n============ Teacher Model
    ============\n")
cnn_model.summary()
```

The screenshot of teacher model is shown below: The student model contains a flatten layer and three dense layers.The code is listed as below:

```
========== Teacher Model ==========

Model: "sequential"

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320

max_pooling2d (MaxPooling2D  (None, 25, 25, 32)        0
)

conv2d_1 (Conv2D)            (None, 23, 23, 64)        18496

max_pooling2d_1 (MaxPooling  (None, 11, 11, 64)        0
2D)

flatten (Flatten)            (None, 7744)              0

dropout (Dropout)            (None, 7744)              0

dense (Dense)                (None, 128)               991360

dropout_1 (Dropout)          (None, 128)               0

dense_1 (Dense)              (None, 10)                1290

=================================================================
Total params: 1,011,466
Trainable params: 1,011,466
Non-trainable params: 0
_________________________________________________________________
```

Fig. 2. Teacher Model

```
# Build fully connected student.
def build_fc():
  fc_model = Sequential()
  fc_model.add(Input((28,28,1)))
  fc_model.add(Flatten())
  fc_model.add(Dense(units=784,activation='relu
      '))
  fc_model.add(Dense(units=784,activation='relu
      '))
  fc_model.add(Dense(NUM_CLASSES, activation='
      softmax'))
  return fc_model
fc_model=build_fc()
print("\n\n\n============ Student Model
    ============\n")
fc_model.summary()
```

The screenshot of student model summary is shown below:

```
|
========== Student Model ==========

Model: "sequential_1"

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_1 (Flatten)          (None, 784)               0

dense_2 (Dense)              (None, 784)               615440

dense_3 (Dense)              (None, 784)               615440

dense_4 (Dense)              (None, 10)                7850

=================================================================
Total params: 1,238,730
Trainable params: 1,238,730
Non-trainable params: 0
_________________________________________________________________
```

Fig. 3. Student Model

C. Question 3

In this section, we will talk about the implementation of loss functions in teacher and student models, consisting of three parts.

The first part is the loss function for teacher model. The code is shown below.

We first calculated the logits of teacher model using the input images. Then we call the function in tensorflow $tf.nn.sparse_softmax_cross_entropy_with_logits$ to calculate the cross-entropy loss with logits. The logits parameter is the $subclass_logits$ which is calculated above, and the label is $tf.argmax(labels, 1)$, since the $labels$ is a 1D vector which has 10 elements. In the vector, only the true label is set to 1 while others are set to 0. As a result, $tf.argmax(labels, 1)$ is the true label.

```
@tf.function
def compute_teacher_loss(images, labels):
  """Compute subclass knowledge distillation
      teacher loss for given images
    and labels.

  Args:
    images: Tensor representing a batch of
        images.
    labels: Tensor representing a batch of
        labels.
```

```
  Returns:
    Scalar loss Tensor.
  """
  subclass_logits = cnn_model(images, training=
      True)
  # print(subclass_logits.shape)
  # print(tf.argmax(labels, 1).shape)
  # Compute cross-entropy loss for subclasses.

  # your code start from here for step 3
  cross_entropy_loss_value = tf.nn.
      sparse_softmax_cross_entropy_with_logits(
      logits=subclass_logits, labels=tf.argmax(
      labels, 1))


  return cross_entropy_loss_value
```

The second part is the $distillation_loss$ function. In this function, we first define the α and the temperature T according to the requirement in the lab manual. Then we calculated the soft targets according to the formula above and the softmax function. After that, we will calculate the cross-entropy loss like the teacher model loss. At last, we calculated the average number of cross-entropy multiply T_2 as the soft targets loss value, named L_{soft}.

The code is shown below.

```
# Hyperparameters for distillation (need to be
    tuned).
ALPHA = 0.5 # task balance between cross-
    entropy and distillation loss
DISTILLATION_TEMPERATURE = 4. # temperature
    hyperparameter
import numpy as np
import torch
import torch.nn as nn
def distillation_loss(teacher_logits: tf.
    Tensor, student_logits: tf.Tensor,
                  temperature: Union[float, tf.
                      Tensor]):
  """Compute distillation loss.

  This function computes cross entropy between
      softened logits and softened
  targets. The resulting loss is scaled by the
      squared temperature so that
  the gradient magnitude remains approximately
      constant as the temperature is
  changed. For reference, see Hinton et al.,
      2014, "Distilling the knowledge in
  a neural network."

  Args:
    teacher_logits: A Tensor of logits provided
        by the teacher.
    student_logits: A Tensor of logits provided
        by the student, of the same
      shape as `teacher_logits`.
    temperature: Temperature to use for
        distillation.

  Returns:
    A scalar Tensor containing the distillation
        loss.
  """
  # print(tf.reduce_sum(tf.exp(teacher_logits/
      temperature),axis=1,keepdims=True).shape)
  soft_targets = tf.exp((teacher_logits-np.max(
      teacher_logits,axis=-1,keepdims=True))/
      temperature)/tf.reduce_sum(tf.exp(np.max(
      teacher_logits,axis=-1,keepdims=True)/
      temperature),axis=1,keepdims=True)

  return tf.reduce_mean(tf.nn.
      softmax_cross_entropy_with_logits(
      soft_targets, student_logits /
      temperature)) * temperature ** 2
```

The thrid part is the student loss which combines both L_{hard} and L_{soft}.First, we calculate the logits for student model as $student_subclass_logits$ and logits for taecher model as $teacher_subclass_logits$. Then we call the $distillation_loss$ function defined above to calculate the soft targets loss value L_{soft}. After that, we calculate the L_{hard} like the teacher loss. We use the $tf.nn.sparse_softmax_cross_entropy_with_logits$ function to compute the student hard loss. At last, for the overall loss value, we calculate the combination of L_{hard} and L_{soft} with their weights accordingly.

The code is shown below:

```
def compute_student_loss(images, labels):
  """Compute subclass knowledge distillation
      student loss for given images
    and labels.

  Args:
    images: Tensor representing a batch of
        images.
    labels: Tensor representing a batch of
        labels.

  Returns:
    Scalar loss Tensor.
  """
  student_subclass_logits = fc_model(images,
      training=True)

  # Compute subclass distillation loss between
      student subclass logits and
  # softened teacher subclass targets
      probabilities.

  teacher_subclass_logits = cnn_model(images,
      training=False)
  distillation_loss_value = distillation_loss(
      teacher_subclass_logits,
      student_subclass_logits,
      DISTILLATION_TEMPERATURE)
  L_hard=tf.nn.
      sparse_softmax_cross_entropy_with_logits(
      logits=student_subclass_logits, labels=tf
      .argmax(labels, 1))
  # print(labels)
  # Compute cross-entropy loss with hard
      targets.

  cross_entropy_loss_value = ALPHA*
```

```
    distillation_loss_value+(1-ALPHA)*L_hard

  return cross_entropy_loss_value
```

D. Question 4

In this section, we complete the $train_and_evaluate$ function. First, we define the optimizer with $Adam$ and set learning rate as 0.001. Then we use a for loop to simulate the training process in tensorflow library. In the each epoch, we calculate the loss value by calling the $compute_loss_fn$ function and pass in the images and corresponding labels in the training batch. The $compute_loss_fn$ function is the loss function name passed in. Then we will apply the loss values to compute the gradients with function $tape.gradient$. After that, we will apply the gradients to modify the model's variables. At the end of the epoch, we call the $compute_num_correct$ function to evaluate the performance of this training epoch.

The code is listed below.

```
def train_and_evaluate(model,
    compute_loss_fn):
"""Perform training and evaluation for a
    given model.

Args:
  model: Instance of tf.keras.Model.
  compute_loss_fn: A function that computes
      the training loss given the
    images, and labels.
"""
optimizer = tf.keras.optimizers.Adam(
    learning_rate=0.001)
res=0.0
for epoch in range(1, NUM_EPOCHS + 1):
  # Run training.
  print('Epoch {}: '.format(epoch), end='')
  for images, labels in mnist_train:
    with tf.GradientTape() as tape:
      loss_value = compute_loss_fn(images,
          labels)

    grads = tape.gradient(loss_value,model.
        variables)
    optimizer.apply_gradients(zip(grads,
        model.variables))

  # Run evaluation.
  num_correct = 0
  num_total = builder.info.splits['test'].
      num_examples
  for images, labels in mnist_test:
    num_correct += tf.cast(
        compute_num_correct(model,images,
        labels)[0],tf.int32)
  print("Class_accuracy: " + '{:.2f}%'.
      format(
      num_correct / num_total * 100))
  res=num_correct / num_total * 100
return res
```

E. Question 5

For the first part, we will talk about the training and test accuracy of the teacher amd student model. First is the teacher model, we just call the $train_and_evaluate$ function. The code is listed below.

```
# your code start from here for step 5
train_and_evaluate(cnn_model,
    compute_teacher_loss)
```

The screenshot of the accuracy is shown below.

```
Epoch 1: Class_accuracy: 96.96%
Epoch 2: Class_accuracy: 97.65%
Epoch 3: Class_accuracy: 98.11%
Epoch 4: Class_accuracy: 98.38%
Epoch 5: Class_accuracy: 98.27%
Epoch 6: Class_accuracy: 98.48%
Epoch 7: Class_accuracy: 98.64%
Epoch 8: Class_accuracy: 98.56%
Epoch 9: Class_accuracy: 98.66%
Epoch 10: Class_accuracy: 98.74%
Epoch 11: Class_accuracy: 98.89%
Epoch 12: Class_accuracy: 98.78%
<tf.Tensor: shape=(), dtype=float64, numpy=98.78>
```

Fig. 4. Teacher Model Testing Accuracy

Second is the student model, since we need to tune the distillation hyperparameters, we used a loop to do it. We set the parameter temperature to $1, 2, 4, 16, 32, 64$ and set alpha to $0.1, 0.3, 0.5, 0.7, 0.9$. In total, we trained 30 models. The code is shown below.

```
T=[1,2,4,16,32,64]
A=[0.1,0.3,0.5,0.7,0.9]
y=[]
for i in T:
  for j in A:
    DISTILLATION_TEMPERATURE = i # Temperature
        hyperparameter
    ALPHA=j
    fc_model=build_fc()
    print('temperature:'+str(i)+' Alpha:'+str(j
        ))
    acu=train_and_evaluate(fc_model,
        compute_student_loss)
    fc_model.save('student_MNIST_t'+str(i)+'a'+
        str(j)+'.h5')
    y.append(acu)
# train_and_evaluate(fc_model,
    compute_student_loss)
```

Then we print out the training accuracy stored with the code below.

```
# fc_model.save('student_MNIST_4.h5')
for i in range(6):
  for j in range(5):
    print('Temperature:'+str(i)+' Alpha:'+str(j
        )+' Accuracy:'+str(y[i*5+j]))
```

Here is the screenshot of testing results. And here is the

```
Temperature:1 Alpha:0.1 Accuracy:tf.Tensor(97.78999999999999, shape=(), dtype=float64)
Temperature:1 Alpha:0.3 Accuracy:tf.Tensor(97.66, shape=(), dtype=float64)
Temperature:1 Alpha:0.5 Accuracy:tf.Tensor(97.91, shape=(), dtype=float64)
Temperature:1 Alpha:0.7 Accuracy:tf.Tensor(97.94, shape=(), dtype=float64)
Temperature:1 Alpha:0.9 Accuracy:tf.Tensor(97.55, shape=(), dtype=float64)
Temperature:2 Alpha:0.1 Accuracy:tf.Tensor(97.74000000000001, shape=(), dtype=float64)
Temperature:2 Alpha:0.3 Accuracy:tf.Tensor(97.89, shape=(), dtype=float64)
Temperature:2 Alpha:0.5 Accuracy:tf.Tensor(97.86, shape=(), dtype=float64)
Temperature:2 Alpha:0.7 Accuracy:tf.Tensor(97.74000000000001, shape=(), dtype=float64)
Temperature:2 Alpha:0.9 Accuracy:tf.Tensor(97.59, shape=(), dtype=float64)
Temperature:4 Alpha:0.1 Accuracy:tf.Tensor(97.95, shape=(), dtype=float64)
Temperature:4 Alpha:0.3 Accuracy:tf.Tensor(97.5, shape=(), dtype=float64)
Temperature:4 Alpha:0.5 Accuracy:tf.Tensor(97.48, shape=(), dtype=float64)
Temperature:4 Alpha:0.7 Accuracy:tf.Tensor(97.83, shape=(), dtype=float64)
Temperature:4 Alpha:0.9 Accuracy:tf.Tensor(97.37, shape=(), dtype=float64)
Temperature:16 Alpha:0.1 Accuracy:tf.Tensor(97.92999999999999, shape=(), dtype=float64)
Temperature:16 Alpha:0.3 Accuracy:tf.Tensor(97.82, shape=(), dtype=float64)
Temperature:16 Alpha:0.5 Accuracy:tf.Tensor(97.72, shape=(), dtype=float64)
Temperature:16 Alpha:0.7 Accuracy:tf.Tensor(97.55, shape=(), dtype=float64)
Temperature:16 Alpha:0.9 Accuracy:tf.Tensor(98.08, shape=(), dtype=float64)
Temperature:32 Alpha:0.1 Accuracy:tf.Tensor(98.03, shape=(), dtype=float64)
Temperature:32 Alpha:0.3 Accuracy:tf.Tensor(97.92, shape=(), dtype=float64)
Temperature:32 Alpha:0.5 Accuracy:tf.Tensor(97.71, shape=(), dtype=float64)
Temperature:32 Alpha:0.7 Accuracy:tf.Tensor(97.84, shape=(), dtype=float64)
Temperature:32 Alpha:0.9 Accuracy:tf.Tensor(97.44, shape=(), dtype=float64)
Temperature:64 Alpha:0.1 Accuracy:tf.Tensor(97.85000000000001, shape=(), dtype=float64)
Temperature:64 Alpha:0.3 Accuracy:tf.Tensor(97.72, shape=(), dtype=float64)
Temperature:64 Alpha:0.5 Accuracy:tf.Tensor(97.55, shape=(), dtype=float64)
Temperature:64 Alpha:0.7 Accuracy:tf.Tensor(98.04, shape=(), dtype=float64)
Temperature:64 Alpha:0.9 Accuracy:tf.Tensor(97.82, shape=(), dtype=float64)
```

Fig. 5. Teacher Model Testing Accuracy

Temperature	Alpha	Test Accuracy
1	0.1	97.78999999999999
1	0.3	97.66
1	0.5	97.91
1	0.7	97.94
1	0.9	97.55
2	0.1	97.74000000000001
2	0.3	97.89
2	0.5	97.86
2	0.7	97.74000000000001
2	0.9	97.59
4	0.1	97.95
4	0.3	97.5
4	0.5	97.48
4	0.7	97.83
4	0.9	97.37
16	0.1	97.92999999999999
16	0.3	97.82
16	0.5	97.72
16	0.7	97.55
16	0.9	98.08
32	0.1	98.03
32	0.3	97.92
32	0.5	97.71
32	0.7	97.84
32	0.9	97.44
64	0.1	97.85000000000001
64	0.3	97.72
64	0.5	97.55
64	0.7	98.04
64	0.9	97.82

test accuracy table for the hyperparameter tuning procedure. As is shown in the table, we think that the best alpha should be 16, and the best temperature is 0.9. However, as we can see here in the table, the hyperparameters do not affect the test accuracy too much. The variance is smaller than 1%. So we can conclude that the adjustment of hyperparameter does not help a lot for the student models to improve their accuracy and mimic the predictions of the teacher model. It's highly possible that the difference is jitters of in the training.

F. Question 6

In this section, we will talk about the plot of a curve which is student test accuracy versus temperature hyperparameters. According to the requirement in the Lab Manual, we set the task balance parameter α as 0.5, and use a loop to change the temperature in the list. In every loop, we change the temperature, reset the model and retrain it. The code is shown below.

```
T=[1,2,4,16,32,64]
y=[]
for i in T:
  DISTILLATION_TEMPERATURE = i # Temperature
      hyperparameter
  fc_model=build_fc()
  print('temperature:'+str(i))
  acu=train_and_evaluate(fc_model,
      compute_student_loss)
  fc_model.save('student_MNIST_'+str(i)+'.h5')
  y.append(acu)
```

The screenshot of the training and testing is shown here. Then

```
temperature:1
Epoch 1: Class_accuracy: 95.70%
Epoch 2: Class_accuracy: 96.52%
Epoch 3: Class_accuracy: 97.07%
Epoch 4: Class_accuracy: 97.07%
Epoch 5: Class_accuracy: 97.42%
Epoch 6: Class_accuracy: 97.90%
Epoch 7: Class_accuracy: 97.66%
Epoch 8: Class_accuracy: 97.41%
Epoch 9: Class_accuracy: 97.38%
Epoch 10: Class_accuracy: 98.02%
Epoch 11: Class_accuracy: 97.48%
Epoch 12: Class_accuracy: 97.93%
WARNING:tensorflow:Compiled the loaded model, but the comp
WARNING:tensorflow:Compiled the loaded model, but the comp
temperature:2
Epoch 1: Class_accuracy: 95.46%
Epoch 2: Class_accuracy: 96.50%
Epoch 3: Class_accuracy: 96.88%
```

Fig. 6. Training and testing procedure

we use *matplotlib.pyplot* library to plot the curve. The code is like below.

```
import matplotlib.pyplot as plt
x=['1','2','4','16','32','64']
l=plt.plot(x,y)
plt.title('Test accuracy vs. tempreture curve'
    )
plt.xlabel('temperature')
plt.ylabel('test_accuracy')
plt.legend()
plt.show()
```

The curve is shown here. As is clearly shown in the picture, the accuracy decreases in general, but it does not decrease a lot. The reason for decreasing is that: As we have stated above, with a high temperature,

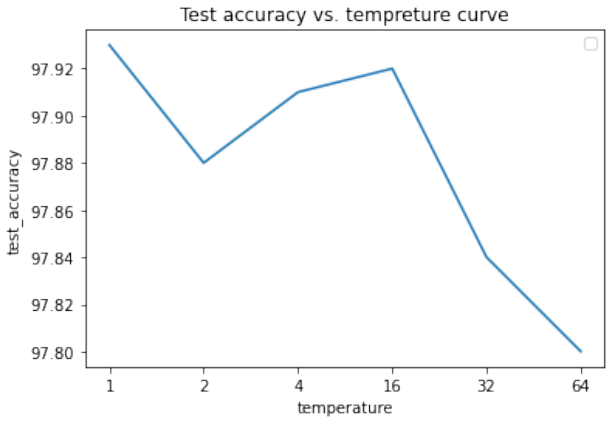

Fig. 7. Student Accuracy VS. Temperature

G. Question 7

In this section, we will train the student model from scratch without using Knowledge Distillation. We will firstly create a new model. In the loss function for the student, we will calculate the logits, and pass it into the $tf.nn.sparse_softmax_cross_entropy_with_logits$ function like we do in teacher model. Then we will pass this function into $train_and_evaluate$ function to do the training.

The code is shown here.

```
# Build fully connected student.
# fc_model_no_distillation = tf.keras.
    Sequential()
fc_model_no_distillation = build_fc()

def compute_plain_cross_entropy_loss(images,
    labels):
  """Compute plain loss for given images and
      labels.

  For fair comparison and convenience, this
      function also performs a
  LogSumExp over subclasses, but does not
      perform subclass distillation.

  Args:
    images: Tensor representing a batch of
        images.
    labels: Tensor representing a batch of
        labels.

  Returns:
    Scalar loss Tensor.
  """
  # your code start from here for step 7
  student_subclass_logits =
      fc_model_no_distillation(images, training
      =True)
  cross_entropy_loss = tf.nn.
      sparse_softmax_cross_entropy_with_logits(
      logits=student_subclass_logits, labels=tf
      .argmax(labels, 1))
  return cross_entropy_loss


train_and_evaluate(fc_model_no_distillation,
    compute_plain_cross_entropy_loss)
```

The screenshot of the test accuracy is shown below.

```
Epoch 1: Class_accuracy: 94.55%
Epoch 2: Class_accuracy: 96.21%
Epoch 3: Class_accuracy: 97.02%
Epoch 4: Class_accuracy: 97.14%
Epoch 5: Class_accuracy: 97.19%
Epoch 6: Class_accuracy: 97.36%
Epoch 7: Class_accuracy: 97.49%
Epoch 8: Class_accuracy: 97.83%
Epoch 9: Class_accuracy: 97.67%
Epoch 10: Class_accuracy: 97.33%
Epoch 11: Class_accuracy: 97.79%
Epoch 12: Class_accuracy: 97.20%
<tf.Tensor: shape=(), dtype=float64, numpy=97.2>
```

Fig. 8. Test Accuracy of the Student Model without KD

As we can see here, the test accuracy of students model with knowledge distillation is, while student model without knowledge distillation achieves 97.20% accuracy in the final epoch. So the test accuracy of student models with knowledge distillation is higher than that of student model without knowledge distillation. (For fair comparison, we set the epoch to 12.)

H. Question 8

In this part, we compute the number of parameters and FLOPs(floating-point operations per second) in teacher model and student model We use an open-source library called $keras_flops$ to do that. We first install this library with a command line.

```
!pip install keras_flops
```

Then we call the functions in the library to do the calculation. For the number of parameters, we use compute it by adding up the number in every variables. For the FLOPs, we call the library function which is get_flops like the examples given in the document. The code is shown below.

```
from keras_flops import get_flops
# flops = tf.profiler.profile(fc_model,
    options=tf.profiler.
    ProfileOptionBuilder.float_operation())
print('teacher model:')
print('Parameter number:'+str(np.sum([np.
    prod(vp.get_shape().as_list()) for vp
    in cnn_model.variables])))
print('FlOPs:'+str(get_flops(cnn_model,
    batch_size=BATCH_SIZE)/1000000000.0))
print('student model:')
print('Parameter number:'+str(np.sum([np.
    prod(vp.get_shape().as_list()) for vp
    in fc_model.variables])))
```

```
print('FlOPs:'+str(get_flops(fc_model,
    batch_size=BATCH_SIZE)/1000000000.0))
print('student model:')
print('Parameter number:'+str(np.sum([np.
    prod(vp.get_shape().as_list()) for vp
    in fc_model_no_distillation.variables])
    ))
print('FlOPs:'+str(get_flops(
    fc_model_no_distillation,batch_size=
    BATCH_SIZE)/1000000000.0))
# flops = tf.profiler.experimental.Profile(
    cnn_model, options=tf.profiler.
    ProfileOptionBuilder.float_operation())
# params = tf.profiler.profile(cnn_model,
    options=tf.profiler.
    ProfileOptionBuilder.
    trainable_variables_parameter())
# print('student model:')
# flops = tf.profiler.profile(fc_model,
    options=tf.profiler.
    ProfileOptionBuilder.float_operation())
# params = tf.profiler.profile(fc_model,
    options=tf.profiler.
    ProfileOptionBuilder.
    trainable_variables_parameter())
```

The result screenshot is shown here.

```
teacher model:
Parameter number:1011466
FlOPs:5.642779648
student model with KD:
Parameter number:1238730
FlOPs:0.633838592
student model without KD:
Parameter number:1238730
FlOPs:0.633838592
```

Fig. 9. Parameter Number and FLOPs

From the screenshot, we can see that the teacher model has parameters and FLOPs is times/second. The student model with KD has parameters, and FLOPs is times/second. The student model without KD has parameters, and FLOPs is times/second.

I. Question 9

In this part, we will talk about our XAI methodology used to build the explanation map for the models.

J. Question 10

In this part, we read Jang Hyun Cho and Bharath Hariharan's [1] paper. The novel idea is.

K. Question 11

How does the proposed method improve the student performance in comparison to the conventional KD?

L. Question 12

What are some limitations of the proposed method? How can they be addressed?

M. Question 13

In this part, we implement the Early-Stopping Knowledge Distillation.

```
def train_and_evaluate_ESKD(model,
    compute_loss_fn,epoch_num=12):
  """Perform training and evaluation for a
      given model.

  Args:
    model: Instance of tf.keras.Model.
    compute_loss_fn: A function that computes
        the training loss given the
      images, and labels.
  """

  # your code start from here for step 4
  optimizer = tf.keras.optimizers.Adam(
      learning_rate=0.001)
  res=0.0
  for epoch in range(1, epoch_num + 1):
    # Run training.
    print('Epoch {}: '.format(epoch), end='')
    loss_total=0
    count_total=0
    for images, labels in mnist_train:
      with tf.GradientTape() as tape:
        loss_value = compute_loss_fn(images,
            labels)
      grads = tape.gradient(loss_value,model.
          variables)
      optimizer.apply_gradients(zip(grads, model
          .variables))
    # Run evaluation.
    num_correct = 0
    num_total = builder.info.splits['test'].
        num_examples
    for images, labels in mnist_test:
      num_correct += tf.cast(compute_num_correct
          (model,images,labels)[0],tf.int32)
    print("Class_accuracy: " + '{:.2f}%'.format
        (
       num_correct / num_total * 100))
    res=num_correct / num_total * 100
  return res
```

```
T=[1,2,3,4,5,6,7,8,9,10,11,12]
y=[]
for i in T:
  # NUM_EPOCHS = 3 #temperature hyperparameter
  cnn_model=build_cnn()
  fc_model=build_fc()
  print('EpochNumber: '+str(i))
  print('Training teacher')
  @tf.function
  def compute_teacher_loss(images, labels):
    """Compute subclass knowledge distillation
        teacher loss for given images
      and labels.
```

```
    Args:
      images: Tensor representing a batch of
          images.
      labels: Tensor representing a batch of
          labels.

    Returns:
      Scalar loss Tensor.
    """
    subclass_logits = cnn_model(images,
        training=True)
    # print(subclass_logits.shape)
    # print(tf.argmax(labels, 1).shape)
    # Compute cross-entropy loss for subclasses
        .

    # your code start from here for step 3
    cross_entropy_loss_value = tf.nn.
        sparse_softmax_cross_entropy_with_logits
        (logits=subclass_logits, labels=tf.
        argmax(labels, 1))
    return cross_entropy_loss_value
train_and_evaluate_ESKD(cnn_model,
    compute_teacher_loss,i)
# NUM_EPOCHS = 12
print('Training student')
acu=train_and_evaluate(fc_model,
    compute_student_loss)
cnn_model.save('teacher_MNIST_ESKD_'+str(i)+'
    .h5')
fc_model.save('student_MNIST_ESKD_'+str(i)+'.
    h5')
y.append(acu)
```

```
import matplotlib.pyplot as plt
x=T
l=plt.plot(x,y)
plt.title('Test accuracy vs. Teacher Epoch
    Number')
plt.xlabel('Teacher_epoch_number')
plt.ylabel('test_accuracy')
plt.legend()
plt.show()
```

II. Task 2: Knowledge Distillation in MHIST Dataset

A. Question 1

1) SubQuestion a: **How can we adapt these models for the MHIST dataset using transfer learning? Talk about the Feature Extraction and Fine-Tuning processes during transfer learning.**

The core idea of Transfer learning is consists of taking features learned on one problem, and leveraging them on a new, similar problem. [3] Feature Extraction is
Fine tuning [4]

2) SubQuestion b: **What is a residual block in ResNet architectures?** The residual block is the significant and elemental component of ResNet, Residual Network, an example is shown in figure .

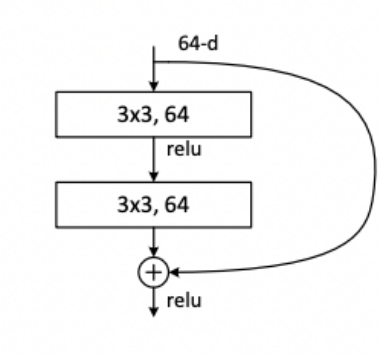

Fig. 10. Residual Block Example

A common sense of Convolutional Neural Network is that, when the depth of CNN becomes essentially deep, in which the network could be called DNN, Deep Neural Network. It will be hard to train the network model due to the problem of gradient disappearance (vanishing gradient problem) and gradient explosion. He et al. [5] proposed ResNet whose layer could get activated with skip connection, which is like a shortcut to the deeper layers. This structure is called residual block. For example, we assume that there's an input, $a^{[l]}$, two convolution layers, with a hidden value between, $a^{[l+1]}$, and an output, $a^{[l+2]}$, shown in the figure 11 [6].

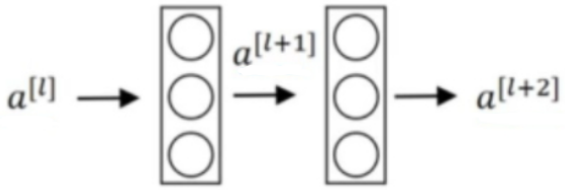

Fig. 11. 2-Layer Structure

In the classic CNN, we need to pass convolutional layer twice in order to compute $a^{[l+2]}$ by $a^{[l]}$, like the following formulas, where $g()$ stands for activation function, e.g. ReLU.

$$z^{[l+1]} = W^{[l+1]}a^{[l]} + b^{[l+1]}, a^{[l+1]} = g(z^{[l+1]})$$

$$z^{[l+2]} = W^{[l+2]}a^{[l+1]} + b^{[l+2]}, a^{[l+2]} = g(z^{[l+2]})$$

However, once we enable the skip connection, we are able to compute $a^{[l+2]}$ by $a^{[l]}$ with the following formula.

$$a^{[l+2]} = g(z^{[l+2]} + a^{[l]})$$

And the whole structure is a basic example of residual block. It will significantly solve the trouble of deep learning, and networks with the structure, residual block, could be able to contain more layers.

3) SubQuestion c: **What are the differences between the ResNetV1 and ResNetV2 architectures?**

ResNetV2 is an improved version of ResNetV1, and there are three major differences between ResNetV1 and ResNetV2. Firstly, the structure of residual block is different, shown in figure 12. [7] ResNetV1 use the structure of *Conv-BN-ReLU* and ResNetV2 use the structure of *BN-ReLu-Conv*.

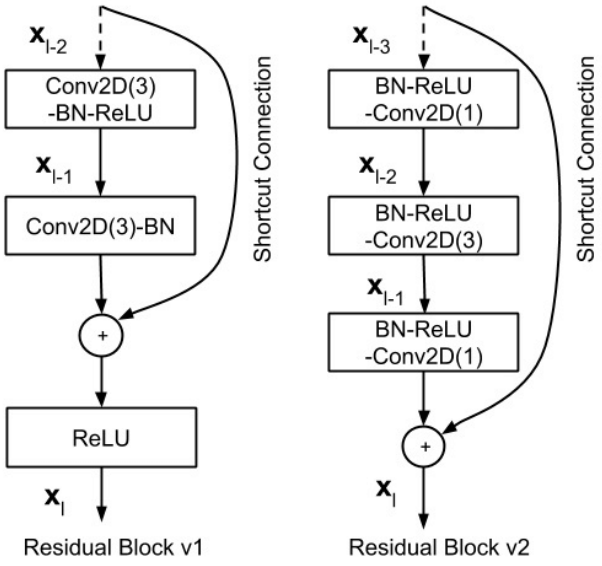

Fig. 12. Residual Block Structure of ResNetV1 vs. ResNetV2

Secondly, the stride strategy is different. ResNetV1 keep the size of feature map of *Block1(Conv2_*)*, and ResNetV2 keep the size of feature map of *Block4(Conv5_*)*. Thirdly, ResNetV2 enhance the regularization by using pre-activation, so the model will be trained faster.

4) SubQuestion d: **What are the differences between the MobileNetV1 and MobileNetV2 architectures?**

5) SubQuestion e: **How can ResNet architectures, regardless of model depth, overcome the vanishing gradient problem?** This question is partiality explained in the section of SubQuestion b. More theoretical explanation in general will be that the ResNet will "decrease" the number of layers, especially during back propagation procedure. The core reason of vanishing gradient problem is as the depth of network increases, the influence of input itself will finally decrease or increase at the output significantly. As a result, the input will have almost nothing to do with the result at this time, and the accuracy will drop gigantically after its increasing with the depth increases. So "shorten" the depth by using ResNet will solve the problem.

6) SubQuestion f: **Is MobileNetV2 a lightweight model? Why?**

B. Question 2

In this section, we introduce our implementation sub task 3,4,5,6,7,8,13 in Task 1 using MHIST dataset with two pre-trained network, ResNet50V2 and MobileNetV2. As there are a number of similarities between this section and Task 1, we would ignore some codes of some functions which is almost the same as the previous selection. We also finish more complete data analysis and preview during the procedure, so we are going to show the full procedure of implementation in this part, with divisions of stages in model training and knowledge distillation.

1) Data Importing and preprocessing:
2) Model Construction:
3) Train Facilities:
4) Model Training:
5) Data Analysis and Evaluation:

C. Question 3

Explain the effect of transfer learning and knowledge distillation in the performance of the student model. Do pre-trained weights help the teacher and student models perform well on the MHIST dataset? Does knowledge transfer from the teacher to the student model increase the student's performance?

REFERENCES

[1] Jang Hyun Cho and Bharath Hariharan. On the efficacy of knowledge distillation. In Proceedings of the IEEE/CVF international conference on computer vision, pages 4794–4802, 2019. http://openaccess.thecvf.com/content_ICCV_2019/papers/Cho_On_the_Efficacy_of_Knowledge_Distillation_ICCV_2019_paper.pdf.

[2] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531, 2(7), 2015. https://arxiv.org/abs/1503.02531.

[3] Transfer learning & fine-tuning, Keras developer guide, 2020, https://keras.io/guides/transfer_learning.

[4] Anusua Trivedi, Deep Learning Part 2: Transfer Learning and Fine-tuning Deep Convolutional Neural Networks, Revolutions. https://blog.revolutionanalytics.com/2016/08/deep-learning-part-2.html.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun; Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770-778

[6] Andrew Ng, Deeplearning.ai. https://www.deeplearning.ai/

[7] Le Huy Hien, Ngo V.H., Nguyen. (2020). Recognition of Plant Species using Deep Convolutional Feature Extraction. 11. 904-910.