

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

Abstract—This paper is a report for Project B 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

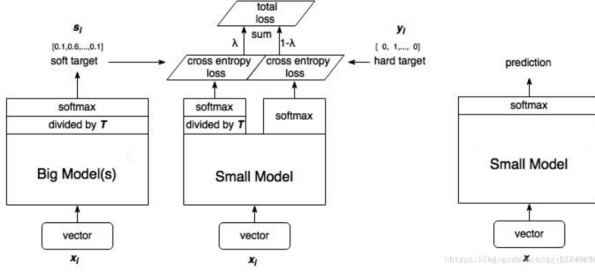


Fig. 1. Knowledge Distillation

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:

$$T \rightarrow \infty$$

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

$$T \rightarrow 0$$

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
=====")
cnn_model.summary()
```

The screenshot of teacher model is shown below: The stu-

===== 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 (MaxPooling2D)	(None, 11, 11, 64)	0
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

dent model contains a flatten layer and three dense layers. The code is listed as below:

```
# 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
=====")
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 third 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 teacher 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 and 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. The test accuracy for teacher model is 98.78%.

```
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.

```
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

And Table 1 is the test accuracy table for the hyperparameter tuning procedure.

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

TABLE I
RESULT TABLE FOR TUNING ALPHA AND TEMPERATURE

As is shown in the table, we think that the best alpha should be 16, and the best temperature is 0.9. The maximum accuracy

for student model is 98.08%. We think that the temperature T has impacts on the calculation of soft targets, with a higher temperature, the information of different predicted possibilities from different classes will be more balanced. The soft targets will receive their information more equally, as we have illustrated in question 1. And the task balance parameter α is used to adjust the weight of the teacher model transferring knowledge. If α is higher, the teacher model will be more important in the computation.

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 result table for the test accuracy under different temperatures is shown below.

Temperature	Alpha	Test Accuracy(%)
1	0.5	97.93
2	0.5	97.88
4	0.5	97.91
16	0.5	97.92
32	0.5	97.84
64	0.5	97.80

TABLE II
RESULT TABLE FOR DIFFERENT TEMPERATURES

The curve is shown here. To make the curve more clear and beautiful, we rebuild the x-axis scale which is the temperature. As is clearly shown in the picture, the accuracy decreases in

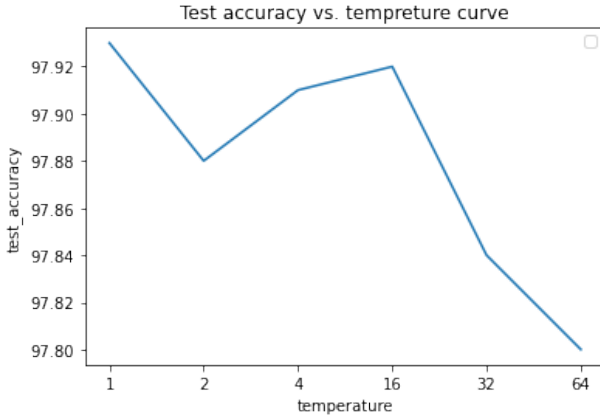


Fig. 7. Student Accuracy VS. Temperature

general, but it does not decrease a lot. We will analyze the reasons for it in detail.

The reason for decreasing is that: as we have stated above, with a high temperature, the influence of different classes will become more balanced. The predicted possibility which is the true label will bring less information to the model, so the accuracy is reduced. The reason for not decreasing so much is that: The dataset MNIST is a well-balanced dataset, the test accuracy for student model is high enough, the difference between student and teacher model is about 2%, so the accuracy will not decrease below the student model without knowledge distillation.

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.

As we can see here, the test accuracy of student model without knowledge distillation is 97.20%, while student models with knowledge distillation achieves 98.08% test accuracy at most. 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.) The result is consistent with the expected result.

H. Question 8

In this part, we compute the number of parameters and FLOPs(floating-point operations per second) in teacher model

```

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

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 1011466 parameters and FLOPs is 5.642779648 *times/second*. The student model with KD has 1238730 parameters, and FLOPs is 0.633838592 *times/second*. The student model without KD has 1238730 parameters, and FLOPs is 0.633838592 *times/second*. We find that student models have more parameters than teacher models, but student models' FLOPs is much smaller than teacher model. We think the result matches the truth of knowledge distillation. The student models have the same amount of parameters. This shows that the knowledge distillation does not have influence on the structure of the models. The student models have more parameters than teacher models because of their network structure. This is decided by the network construction.

The student models have the same FLOPs, because what the knowledge distillation do is changing the calculation of loss values, it'll not increase the times of floating computation, thus keeping the low complexity of the model training. The teacher model has a much higher FLOPs than student models because the model is more complex and contains convolutional layer which needs large amount of floating computation.

I. Question 9

In this part, we will talk about our XAI methodology used to build the explanation map for the models. We choose RISE to do the visualization, the code is highly adapted from the *xai_utils.py* in Project A, but with modifications to support 1-Channel Images. Besides, the calling function is also modified to fit in the format of data. A here is the modified function of RISE.

```

def create_random_mask(h=7,
                        w=7,
                        H=28,
                        W=28,
                        p_l=0.5,
                        resample=Image.BILINEAR):
    """
    Generates one random mask utilized in RISE
    inputs:
        h, w: initial size of binary mask
        H, W: final size of the upsampled mask
        p_l: probability of actiating pixels in
            the down-sampled masks.
        interp: upsampling technique.
    returns:
        mask: a smooth mask with the values in
            range [0,1] with size of HxW.
    """
    assert H > h, 'Masks should be resized to
        higher dimensions.'
    assert W > w, 'Masks should be resized to
        higher dimensions.'
    # create random binary h x w mask
    mask = np.random.choice([0, 1], size=(h, w)
        , p=[1 - p_l, p_l])

    # upsample mask to (h+H,w+W)
    mask = Image.fromarray(mask * 255.)
    mask = mask.resize((H + h, W + w), resample
        =resample)
    mask = np.array(mask)

    # randomly crop mask to HxW
    w_crop = np.random.randint(0, w + 1)
    h_crop = np.random.randint(0, h + 1)
    mask = mask[h_crop:H + h_crop, w_crop:W +
        w_crop]

    # normalize between 0 and 1
    mask /= np.max(mask)

    return mask

def RISE(img, model, class_index, N_MASKS
        =8000, H=28, W=28, C=1):
    """
    img: a 3-D input image
    model: a trained model
    class_index: The class of interest
    N_MASKS: The number of random masks to be
        generated
    H,W,C: The desired dimensions of the random
        masks
    """
    X = np.zeros(shape=(N_MASKS, H, W, C),
        dtype=np.float32)
    masks = np.zeros((N_MASKS, H, W), dtype=np.
        float32)
    for i in range(N_MASKS):
        m = create_random_mask(H=H, W=W)
        masks[i] = m
        # print(img)
        # x = img.numpy()
        x = img.copy()
        if C==1:
            x[:, :, 0] *= m

```

```

            if C==3:
                x[:, :, 0] *= m
                x[:, :, 1] *= m
                x[:, :, 2] *= m
            X[i] = x
        preds_masked = model.predict(X, verbose=0)
        sum_mask = np.zeros(masks[0].shape, dtype=
            np.float32)

        # np.einsum???
        for i, mask in enumerate(masks):
            # print(i)
            # print(mask)
            m = mask * preds_masked[i, class_index]
            sum_mask += m

        sum_mask -= np.min(sum_mask)
        sum_mask /= np.max(sum_mask)
        return sum_mask

```

Note that we add the condition of $C == 1$, and modify the default size and channel number of the function. To call this function, we have to use the following sentence, students are almost the same.

```

explanation_map_RISE_teacher = RISE(dataf[
    index].numpy(), cnn_model, class_index=np.
    argmax(label[index]), N_MASKS=1000)
explanation_map_RISE_teacher -=
    explanation_map_RISE_teacher.min()
explanation_map_RISE_teacher /=
    explanation_map_RISE_teacher.max()+10e-30

```

The plotting is quite simple, but only one thing have to notice is that we have to reshape the original image into 2-Dimensional array, as the `plt.imshow()` only accept 2-Dimensional images or 3-Dimensional images with channels of 3 or 4.

And the explanation map is shown here. We present 3 groups of explanation maps of 3 samples in the validation set.

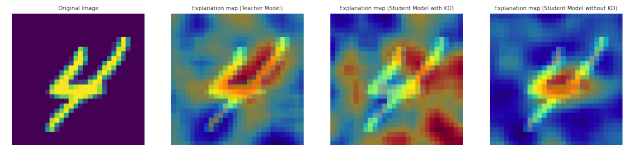


Fig. 10. Explanation maps (index=3)

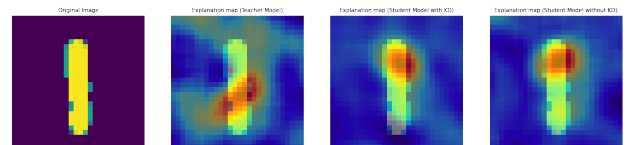


Fig. 11. Explanation maps (index=5)

According to the three groups of explanation maps above, we could easily see that the student with KD inherit the weight partiality from the teacher model, while it still slightly

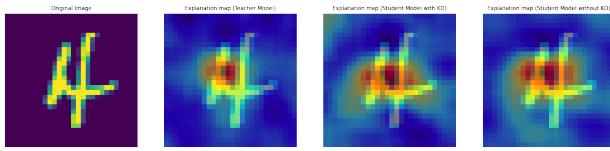


Fig. 12. Explanation maps (index=8)

keep some of its original highlighted region. Conclusively, the student with KD takes the feature as the combination of teacher model and origin trained model itself.

J. Question 10

In this part, we read Jang Hyun Cho and Bharath Hariharan's [1] paper. The novel idea is that the higher accuracy of teacher model does not necessarily mean that the corresponding student model will have higher accuracy. The reason is that mismatched capacity makes it difficult for the students to mimic the teacher model's behavior and prediction stably. For the same student model, if the teacher model is more complex and has higher accuracy, the performance of student model will become worse.

What's more, Teacher Assistant Knowledge Distillation (TAKD) is not an effective way to solve this problem. They propose that the only solution is to stop the training of the teacher model earlier before it converges. This kind of training method is named "Early-Stopping Knowledge Distillation" (ESKD).

K. Question 11

By stopping the teacher model training earlier before the convergence, it's likely that teacher model hasn't finished training and fitting into the training set. As a result, the solution of teacher model will be relatively simpler compared with the fully trained teacher models. So it'll be more likely for the student model to find an approximate solution similar to the simple teacher model solution with a low capacity.

By finding an approximate solution, the student model can predict a similar possibility as the teacher model, thus getting more knowledge from the teacher model than traditional knowledge distillation. As a result, ESKD improves the student model performance.

L. Question 12

The limitations are listed below:

- First, it's very hard to define when to stop the teacher model training. If we stop the teacher model too early, the performance of the teacher model is not good enough for the knowledge transferring. If we stop the teacher model too late, too many epochs will make the teacher model too complex and the student model will find it hard to get an approximate solution. As a result, the transferring result is far from good. So it's hard for us to decide the epoch number. If we calculate the number by trying every numbers, it's very time-consuming.

Solution: In the small models, we can try every possible epoch numbers and find the highest accuracy. For the bigger models, we can calculate the absolute value of the difference between current loss value and previous loss value, and set a criterion as the limit. If the loss value difference is less than the limit, it means the model is sufficient enough for transferring. This criterion should be defined by calculating on several different models.

- Second, the improvement is not enough from our perspective. The top-1 error for ResNet decreases for only 1.5%. We doubt whether it pays to do the ESKD, because it takes some time to compute the appropriate epoch and decide when to stop.

Solution: We think we should apply ESKD only to the larger models and larger datasets.

M. Question 13

In this part, we implement the Early-Stopping Knowledge Distillation. We redefine the *train_and_evaluate* function to fit in a new parameter '*epoch_num*'. We use it to control the running epoch number of the teacher model. In every epoch, it'll compute the loss values, the gradients and apply gradients to modify parameters in models. Then it'll calculate the accuracy using the test set. The code is listed below.

```
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

```

Since this model has only 12 epochs, we try every number for teacher model, and record the corresponding student model performance to decide which one is the best number. Here is the code for it. In the program, we rebuild the teacher model and student model first. (Note: If we don't redefine the *compute_teacher_loss* function, it'll finish with reporting bugs somehow.) Then we call the *train_and_evaluate_ESKD* function to do the training for teacher model and *train_and_evaluate* function to train student model and record the result of the test accuracy of student model in the final epoch.

```

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)

```

Table 3 is the accuracy result table for ESKD. The first column is the training epoch number, the second column is the test accuracy of student model in the final epoch accordingly. (Note: Here we set the Temperature T as 4, task balancer α as 0.5 and student training epoch as 12.)

Teacher_training_epoch	Test Accuracy(%)
1	97.34
2	97.13
3	97.49
4	97.88
5	97.74
6	97.54
7	97.66
8	97.81
9	97.46
10	97.61
11	97.70

TABLE III
RESULT TABLE FOR ESKD

To show the result clearly, we plot out the result with the following code.

```

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()

```

The result plot is shown here. We think the teacher model

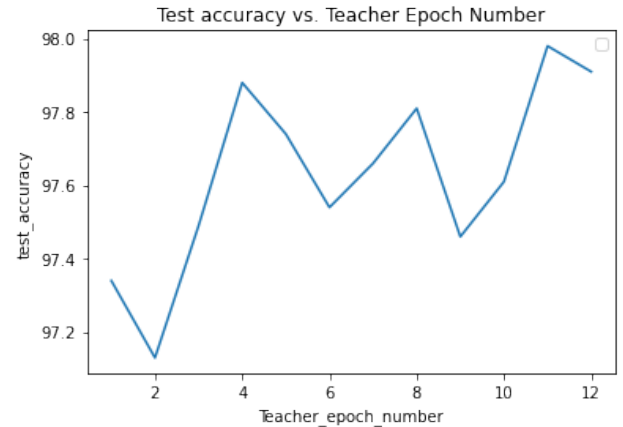


Fig. 13. Result Plot for ESKD

should stop at epoch 4. In the table, it's the best choice. Obviously, the test accuracy does not improve a lot, there are several possible reasons.

- First, the test accuracy is high enough, there cannot be a large improvement in the performance.

- Second, the MNIST dataset is a well balanced dataset and it's small and simple to predict. The solution of the teacher model is not beyond the capacity of the student model, so the student model can still learn enough knowledge from teacher model.

II. TASK 2: KNOWLEDGE DISTILLATION IN MHIST DATASET

A. Question 1

1) *SubQuestion a:* The core idea of Transfer learning is consists of taking features learned on one problem, and leveraging them on a new, similar problem, which is also known as using pre-trained models. Transfer learning is usually used in tasks where the data is too little to train a full-scale model.

[3] A typical steps will be as followed:

- 1) Get the layers from a pre-trained model.
- 2) Freeze all the layers to avoid destroying their information in future training.
- 3) Add some trainable layers on the top.
- 4) Train the new layers with MHIST to fit a new model.

Feature Extraction is typically done with convolution layers. In our approach, we could adapt feature extraction using the new layers we added on the top of the adapted layers of model. Fine tuning is an optional step of transfer learning which consists of unfreezing the entire or part of the model obtained above, and retraining it on the new data with a very low learning rate. This can potentially achieve meaningful improvements, by incrementally adapting the pre-trained features to the new data. [3] [4]

2) *SubQuestion b:* The residual block is the significant and elemental component of ResNet, Residual Network, an example is shown in figure 14.

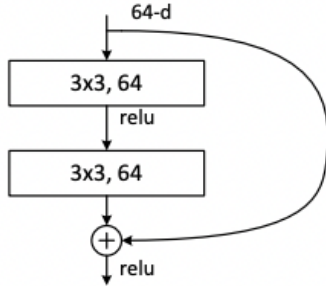


Fig. 14. Residual Block Example

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 15 [6].

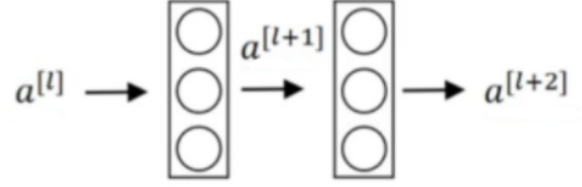


Fig. 15. 2-Layer Structure

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:* 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 16. [7] ResNetV1 use the structure of *Conv-BN-ReLU* and ResNetV2 use the structure of *BN-ReLU-Conv*.

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:* There are a few improvement MobileNetV2 made upon the previous version, MobileNetV1. MobileNetV1 have 28 trainable layers, along with average pooling layers and SoftMax layers. It use a stride of 2 to sampling the input with convolution. The hidden convolution layers are all novel structures called depth-wise separable convolution. Every convolution layer is followed by Batch Normalization Layer and ReLU activation layer. Last but not the least, the least full connection layer is without activation function. MobileNetV2 have two novel structure that MobileNetV1 doesn't have, Linear Bottleneck and Inverted Residual. It consists of 17 bottleneck layer, totally 54 trainable layer. The inverted residual will expand the dimension then decrease it in order to extract more feature and enhance the

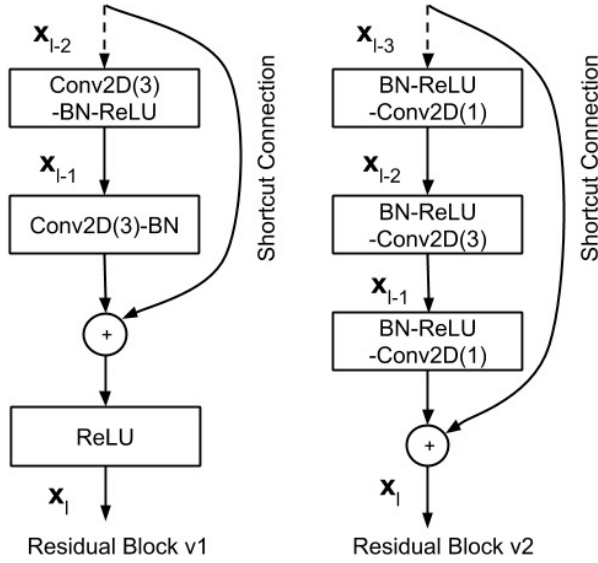


Fig. 16. Residual Block Structure of ResNetV1 vs. ResNetV2

propagation of gradient. In that, it decrease the size of model even more.

5) *SubQuestion e*: 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*: Yes, MobileNetV2 is definitely a lightweight model. Practically, in our approach, the MobileNetV2 have 3,540,986 parameters in total, while the ResNet 25,615,802 parameters totally, which is a quite big difference of the size (weight). Theoretically, a lightweight model can be defined as a "light-weighted" model, which means it contains some structures or facilities which could help the model keep the performance and reduce its size. One typical strategy of light-weighting model is to replace some large feature mapping convolution layer with some small size or grouped convolution layers, both MobileNetV1 and MobileNetV2 uses the strategy, whose detailed structure is discussed in the sub question d.

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 without transfer learning. 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*: The material of MHIST dataset we obtained is an annotation CSV file and a images folder consists of all the images in the dataset. For the first step, we could get a preview of the annotation file and the image data. We use DataFrame container of package *pandas* to extract the information of annotations, the format of annotations is like figure 17 and 18. And we can also easily use *plt* to import and plot the image easily, two examples is shown in figure 19 and 20.

Image Name	MHIST_aaa.png	0	\
Majority Vote Label	SSA		
Number of Annotators who Selected SSA (Out of 7)	6		
Partition	train		

Image Name	MHIST_aab.png	1	\
Majority Vote Label	HP		
Number of Annotators who Selected SSA (Out of 7)	0		
Partition	train		

Image Name	MHIST_aac.png	2	\
Majority Vote Label	SSA		
Number of Annotators who Selected SSA (Out of 7)	5		
Partition	train		

Image Name	MHIST_aae.png	3	4
Majority Vote Label	HP		SSA
Number of Annotators who Selected SSA (Out of 7)	1	5	
Partition	train	train	

Fig. 17. Some Example Data in Train Set of Annotation File

Image Name	MHIST_aag.png	5	\
Majority Vote Label	HP		
Number of Annotators who Selected SSA (Out of 7)	2		
Partition	test		

Image Name	MHIST_aah.png	6	\
Majority Vote Label	HP		
Number of Annotators who Selected SSA (Out of 7)	2		
Partition	test		

Image Name	MHIST_aaq.png	14	\
Majority Vote Label	SSA		
Number of Annotators who Selected SSA (Out of 7)	5		
Partition	test		

Image Name	MHIST_aar.png	15	22
Majority Vote Label	HP		HP
Number of Annotators who Selected SSA (Out of 7)	0	1	
Partition	test	test	

Fig. 18. Some Example Data in Test Set of Annotation File

```
raw_data = pd.read_csv(annotation_path)

train_data = raw_data.query("Partition == 'train'")
test_data = raw_data.query("Partition == 'test'")
print("Train Data Size", train_data.shape)
print("Test Data Size", test_data.shape)
```

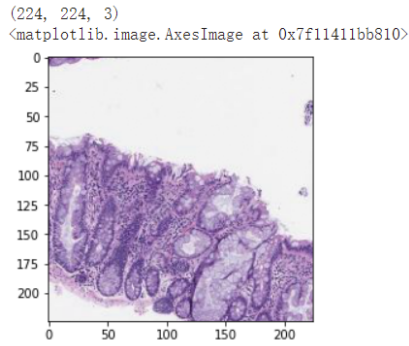


Fig. 19. An example image of Train Set

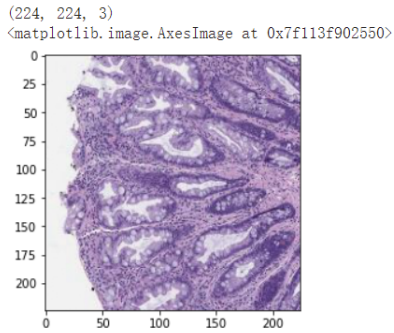


Fig. 20. An example image of Test Set

The data generator is constructed using ImageDataGenerator from keras, which is highly inspired by the HMT Task of Project A.

```
from keras.preprocessing.image import
    ImageDataGenerator
train_set_generator = ImageDataGenerator(
    rescale=1/255., shear_range=0.1,
    rotation_range=15, horizontal_flip=True,
    vertical_flip=True)
test_set_generator = ImageDataGenerator(
    rescale=1/255.)
```

Different from the implementation of HMT task, the dataset is not categorized by folder so it's better to use DataFrame as a source to config the instance of generators.

```
train_generator = train_set_generator.
    flow_from_dataframe(
        dataframe=train_data,
        x_col='Image Name',
        y_col='Majority Vote Label',
        directory='./images/',
        # subset='training',
        batch_size=32,
        shuffle=True,
        class_mode="categorical",
        target_size=(224,224)
    )

test_generator = test_set_generator.
    flow_from_dataframe(
```

```
dataframe=test_data,
x_col='Image Name',
y_col='Majority Vote Label',
directory='./images/',
# subset='validation',
batch_size=32,
shuffle=True,
class_mode="categorical",
target_size=(224,224)
)
```

2) *Model Construction:* We create the model by adapting the pre-trained model from keras, the code is as followed.

```
from tensorflow.keras.applications.resnet_v2
    import ResNet50V2
from tensorflow.keras.applications.
    mobilenet_v2 import MobileNetV2
from tensorflow.keras.layers import Dense
from tensorflow.keras import Sequential

teacher_model = Sequential()
teacher_model.add(ResNet50V2(
    classifier_activation = None, input_shape
    = (224,224,3)))
teacher_model.add(Dense(NUM_CLASSES))
print(teacher_model.summary())

student_model = Sequential()
student_model.add(MobileNetV2(
    classifier_activation=None, input_shape
    = (224,224,3)))
student_model.add(Dense(NUM_CLASSES))
print(student_model.summary())
```

The basic information of model is shown in figure 21 and 22.

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50v2_weights_tf_dim_ordering_tf_kernels.h5
102875136/102869326 [=====] - 1s 0us/step
102869326/102869326 [=====] - 1s 0us/step
Model: "sequential"

Layer (type)	Output Shape	Param #
resnet50v2 (Functional)	(None, 1000)	25613000
dense (Dense)	(None, 2)	2002

Total params: 25,615,002
Trainable params: 25,570,362
Non-trainable params: 45,440
None

Fig. 21. ResNet50V2 Model Summary

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobilenet_v2_weights_tf_dim_ordering_tf_kernels_1.0_224.h5
14540000/14536120 [=====] - 0s 0us/step
14540000/14536120 [=====] - 0s 0us/step
Model: "sequential_1"

Layer (type)	Output Shape	Param #
mobilenetv2_1.00_224 (Funcot	(None, 1000)	3539984
total)		
dense_1 (Dense)	(None, 2)	2002

Total params: 3,540,986
Trainable params: 3,506,974
Non-trainable params: 34,112
None

Fig. 22. MobileNetV2 Model Summary

3) *Train Facilities:* The major facility we have to apply is the loss function, which could be integrated from the Task 1. However, we slightly modified the train_and_evaluate function in order to test fined model under different temperature. The code is shown below.

```

def train_and_evaluate(model, compute_loss_fn,
    num_epochs, learning_rate, **kwargs):
    """Perform training and evaluation for a
        given model.

    Args:
    model: Main Instance of tf.keras.Model.
    compute_loss_fn: A function that computes
        the training loss given the
        images, and labels.
    num_epochs: Number of epochs to train for
    learning_rate: Optimizer learning rate
    kwargs: Passed through to loss fn
    """

    # your code start from here for step 4
    optimizer = tf.keras.optimizers.Adam(
        learning_rate=learning_rate)

    train_generator.reset()

    accuracy = 0

    for epoch in range(1, num_epochs + 1):
        # Run training.
        print('Epoch {}: '.format(epoch), end='')

        #for images, labels in mhists_train:
        for batch in range(TRAIN_BATCH_NUM):

            images, labels = train_generator.next
                ()

            with tf.GradientTape() as tape:

                loss_value = compute_loss_fn(model
                    , images, labels, **kwargs)

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

        # Run evaluation.
        num_correct = 0
        num_total = BATCH_SIZE*TEST_BATCH_NUM

        test_generator.reset()

        for batch in range(TEST_BATCH_NUM):
            images, labels = test_generator.next
                ()

            num_correct += compute_num_correct(
                model, images, labels)[0]
        accuracy = num_correct / num_total * 100
        print("Class accuracy: " + '{:.2f}%'.
            format(accuracy))

    return accuracy

```

4) *Model Training*: We did a large-scale experiment in this part, we divide the training into 7 groups, teacher model and temperature at 6 different degrees. Despite the teacher group consists of initial training and fined training, each student

group consists of four model trainings: student-kd-initial, student-kd-fined, student-scratch-initial, student-scratch-fined. So there is a pile of data we would like to present here. And each group is executed with the code as follows.

```

# Alpha = 0.5, Learning rate as table 1 in
    instruction

current_temperature = 64

print("\n=====Student(KD) Model-Initial
    =====")
student_kd_init_acc = train_and_evaluate(
    student_model, compute_student_loss,
    INIT_EPOCH_NUM, 1e-3, teacher_model=
        teacher_model, temperature=
            current_temperature, alpha=0.5)
student_model.save('student_kd_model_init_{}.
    h5'.format(current_temperature))
print("\n=====Student(KD) Model-Fine
    =====")
student_kd_fine_acc = train_and_evaluate(
    student_model, compute_student_loss,
    FINE_EPOCH_NUM, 1e-4, teacher_model=
        teacher_model, temperature=
            current_temperature, alpha=0.5)
student_model.save('student_kd_model_fine_{}.
    h5'.format(current_temperature))

print("\n=====Student(Scratch)-Initial
    =====")
student_nokd_init_acc = train_and_evaluate(
    student_model_nokd,
    compute_student_scratch_loss,
    INIT_EPOCH_NUM, 1e-3, temperature=
        current_temperature)
student_model_nokd.save('
    student_nokd_model_init_{}.h5'.format(
        current_temperature))
print("\n=====Student(Scratch)-Fine
    =====")
student_nokd_fine_acc = train_and_evaluate(
    student_model_nokd,
    compute_student_scratch_loss,
    FINE_EPOCH_NUM, 1e-4, temperature=
        current_temperature)
student_model_nokd.save('
    student_nokd_model_fine_{}.h5'.format(
        current_temperature))

print("\n\n=====SUMMARY
    =====")
print("Current Temperature: ",
    current_temperature)
print("Student(KD) Initial Accuracy: ",
    student_kd_init_acc)
print("Student(KD) Fine Accuracy: ",
    student_kd_fine_acc)
print("Student(Scratch) Initial Accuracy: ",
    student_nokd_init_acc)
print("Student(Scratch) Fine Accuracy: ",
    student_nokd_fine_acc)
print("=====")

```

The summary sections of output for teacher and each student

model with different temperature group are shown below.

```
Teacher Initial Accuracy: tf.Tensor(83.4375, shape=(), dtype=float32)
Teacher Fine Accuracy: tf.Tensor(84.375, shape=(), dtype=float32)
```

Fig. 23. The summary of accuracy of Teacher Model Group

```
=====SUMMARY=====
Current Temperature: 1
Student(KD) Initial Accuracy: tf.Tensor(63.333332, shape=(), dtype=float32)
Student(KD) Fine Accuracy: tf.Tensor(63.125004, shape=(), dtype=float32)
Student(Scratch) Initial Accuracy: tf.Tensor(72.708336, shape=(), dtype=float32)
Student(Scratch) Fine Accuracy: tf.Tensor(80.104164, shape=(), dtype=float32)
=====
```

(a) Temperature=1

```
=====SUMMARY=====
Current Temperature: 2
Student(KD) Initial Accuracy: tf.Tensor(63.437496, shape=(), dtype=float32)
Student(KD) Fine Accuracy: tf.Tensor(63.437496, shape=(), dtype=float32)
Student(Scratch) Initial Accuracy: tf.Tensor(69.6875, shape=(), dtype=float32)
Student(Scratch) Fine Accuracy: tf.Tensor(83.85417, shape=(), dtype=float32)
=====
```

(b) Temperature=2

```
=====SUMMARY=====
Current Temperature: 4
Student(KD) Initial Accuracy: tf.Tensor(63.229168, shape=(), dtype=float32)
Student(KD) Fine Accuracy: tf.Tensor(63.125004, shape=(), dtype=float32)
Student(Scratch) Initial Accuracy: tf.Tensor(68.333336, shape=(), dtype=float32)
Student(Scratch) Fine Accuracy: tf.Tensor(85.104164, shape=(), dtype=float32)
=====
```

(c) Temperature=4

```
=====SUMMARY=====
Current Temperature: 16
Student(KD) Initial Accuracy: tf.Tensor(63.437496, shape=(), dtype=float32)
Student(KD) Fine Accuracy: tf.Tensor(63.333332, shape=(), dtype=float32)
Student(Scratch) Initial Accuracy: tf.Tensor(68.4375, shape=(), dtype=float32)
Student(Scratch) Fine Accuracy: tf.Tensor(83.85417, shape=(), dtype=float32)
=====
```

(d) Temperature=16

```
=====SUMMARY=====
Current Temperature: 32
Student(KD) Initial Accuracy: tf.Tensor(63.229168, shape=(), dtype=float32)
Student(KD) Fine Accuracy: tf.Tensor(63.229168, shape=(), dtype=float32)
Student(Scratch) Initial Accuracy: tf.Tensor(69.375, shape=(), dtype=float32)
Student(Scratch) Fine Accuracy: tf.Tensor(84.270836, shape=(), dtype=float32)
=====
```

(e) Temperature=32

```
=====SUMMARY=====
Current Temperature: 64
Student(KD) Initial Accuracy: tf.Tensor(63.020832, shape=(), dtype=float32)
Student(KD) Fine Accuracy: tf.Tensor(63.333332, shape=(), dtype=float32)
Student(Scratch) Initial Accuracy: tf.Tensor(71.666664, shape=(), dtype=float32)
Student(Scratch) Fine Accuracy: tf.Tensor(82.395836, shape=(), dtype=float32)
=====
```

(f) Temperature=64

Fig. 24. The summary of accuracy of Student Model Group

Temperature experiment have the same conclusion as it in Task 1, ask temperature goes higher, the performance of model grows marginally.

5) *Data Analysis and Evaluation:* The model should be evaluated with metrics other than accuracy due to the unbalanced dataset. We've evaluate the F1-score as alternative. However, we think accuracy also makes sense in some cases, since in the same condition, the different accuracy could show the model performance qualitatively. In the report, we only includes the F1-score result, if you want to know the accuracy status, you can refer to our public repository.

```
this_model = load_model("./models/Task2/
student_kd_model_fine_16.h5")
```

```
y_pred = this_model.predict(x_test)
y_predicted = np.argmax(y_pred,axis=1)
```

```
print(classification_report(y_true=y_true,
y_pred=y_predicted, target_names=labels))
```

We could manually feed them into arrays and plot them in order to analyze the data, shown in figure 25,26and 27.

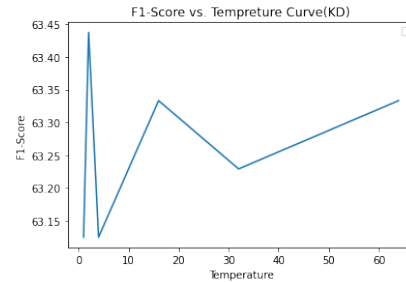


Fig. 25. F1-Score vs. Temperature Curve (KD)

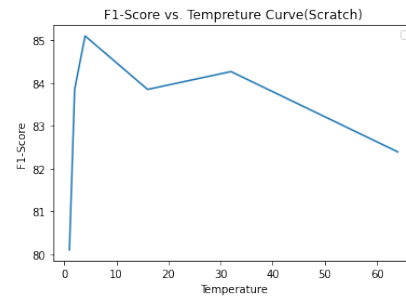


Fig. 26. F1-Score vs. Temperature Curve (Scratch)

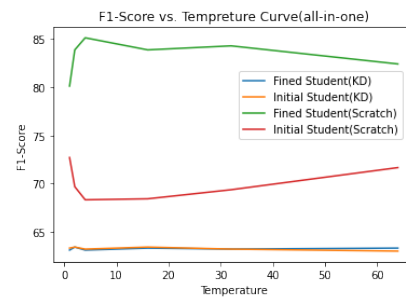


Fig. 27. F1-Score vs. Temperature Curve (All-in-one)

The plots indicate that the fined model are slightly improved its performance, but we're also surprising that Scratch groups have a significantly difference. We will discuss it in the next section.

One significant thing we would like to mention, is the conversion from EagerTensor object to Array. We adapted this function from Internet and use it to fit in the input requirement of classification_report().

```
def imgFlow2npArray(img_flow, img_sum,
img_size):
```

```

x = np.zeros(shape = (img_sum, img_size[0],
                      img_size[1], img_size[2]))
y = np.zeros(shape = (img_sum))
for image in img_flow:
    img_sum = img_sum + 1
    x[img_sum] = image[0][0] #image[0] save
                             image data in matrix, so needs to
                             del one dim
    y[img_sum] = image[1][0].tolist().index
    (1.) #image[1] save image label in
    matrix, so needs to del one dim as
    well
    if img_sum <= 0:
        break
return x, y
# Adapted from: https://blog.csdn.net/
qq_44980390/article/details/118029300

```

We also calculate the Flops of models, and the outcomes are the same for each group of models under different temperature, as shown below in figure 28, which tells us that KD will not influence the size of the model itself.

```

teacher model:
Parameter number:25615802
WARNING:tensorflow:From /usr/local/lib/python3.7/dist-packages
Instructions for updating:
Use `tf.compat.v1.graph_util.tensor_shape_from_node_def_name`
FLOPs:223.754962368
student model with KD:
Parameter number:3540986
FLOPs:19.686146304
student model without KD:
Parameter number:3540986
FLOPs:19.686146304

```

Fig. 28. Model Flops

The final sub-task is to apply the methodology we chose, which is Early-Stop KD. Basically we have the same conclusion as it is in Task 1 as well, but somehow the accuracy is highly influenced by the unbalanced dataset. However, it is still able to see that the performance improved as the number of epoches grows up till the optimal point, the result is shown in figure 29.

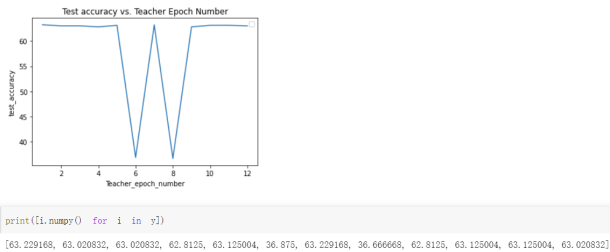


Fig. 29. ESKD Experiment

C. Question 3

We suppose that transfer learning and knowledge distillation are both the methodology that could improve the performance of student model. Each of the method could work well

individually, but there are some scenarios that the combination will also be possible to decrease the performance metrics. Transfer learning plays its role when there's something same in weight of the pretrained model and current model, when adapting weight of some layers of the pretrained model, the learning rate could be set lower and it's more easy to reach the optimal model weights, and also decrease the training time. Knowledge distillation obviously influence the performance of the student model. Other than adapting weights from all-layers or part of layers, the KD will pass its knowledge to student model and improve its performance, while decrease the size of model. But practically, when we tried to use the methodologies together, the weight might be influence for both, and result in an unsatisfying performance of student model. So in our approach, the answer of KD is no, but the pre-trained weights do help the process of learning. But it's more essential to have experiments on transfer learning in order to get better student model 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.