ECE1512 Project A: Knowledge Distillation for Building Lightweight Deep Learning Models in Visual Classification Tasks

Zhenhuan Sun
University of Toronto

I. Task 1

All models in this section are trained on free T4 GPU provided by Google Colab. The complete code can be found in my Github repository [1]. In this section, I only include segments of the code that are necessary for answering the questions in the project and explaining the results.

A. Question 1

(a) The purpose of using knowledge distillation is to transfer the knowledge from a large and sophisticated model to a model that is smaller and more suitable for deployment.

(b) The class probabilities produced by the sophisticated model is transferred as knowledge.

(c) The limit of softmax function with temperature parameter T can be expressed as

$$\lim_{T\to\infty} \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)} = \frac{\exp(0)}{\sum_j \exp(0)} = \frac{1}{K} \tag{1}$$

where K denotes the number of classes [2]. Thus, from this we can see that T is a scaling factor that will make the probability distribution across all classes become closer to a uniform distribution. Temperature parameter T is used to soften the probability distribution over classes, so that the class probabilities become less sharp and the differences in probabilities can be smoothed out. By using a large value of T, the differences between the probabilities of wrong classes are emphasized, the output class probabilities difference that used to be very small and have very little impact on the cross-entropy cost function will become larger and have a more pronounced impact during knowledge distillation. As a result, during knowledge distillation, the student model will be more capable of learning how to mimic teacher model's behavior to produce similar distribution of probabilities over classes for a same input, or in other words, how to think in the same way as the teacher model does. Through this, more information regarding how teacher model perceives the difference between classes and how it generalizes to different data can be transferred from teacher model to student model.

(d) The loss function in knowledge distillation comprises two parts, the first part, which is called distillation loss, is defined as

$$L^{(d)}(\mathbf{x}|T) = -\sum_{i=1}^{K} q_i(\mathbf{x}|T)\log(p_i(\mathbf{x}|T)) \tag{2}$$

where $q_i(\mathbf{x}|T)$ is the softened probability for class i given input vector $\mathbf{x}$ and distillation temperature T from the teacher model, and $p_i(\mathbf{x}|T)$ is the student model's softened probability for class i given same input vector and distillation temperature. Cross entropy is used to measure the difference between the student's softened probability distribution and the teacher's softened probability distribution over all classes.

The second part, which is sometimes referred as classification loss, is defined as

$$L^{(c)}(\mathbf{x}|T=1) = -\sum_{i=1}^{K} y_i \log(p_i(\mathbf{x}|T=1)) \tag{3}$$

where y_i is the ith element of the one-hot vector. Cross entropy is used to measure the difference between student model's unsoftened predicted probability distribution over different classes, which is computed by setting T to 1, and the true class probability distribution, i.e., one-hot encoded vector.

The combined loss function is defined as

$$L(\mathbf{x}|T) = \lambda T^2 \cdot L^{(d)}(\mathbf{x}|T) + (1-\lambda)\cdot L^{(c)}(\mathbf{x}|1) \tag{4}$$

where weight parameter (λ), i.e., task balance parameter, is introduced to combined the distillation loss and classification loss. In addition, to compensate for the gradient's reduction by a factor of $1/T^2$ due to the presence of T in the softmax function, the distillation loss is scaled by a factor of T^2 in the weighted sum.

The task balance parameter governs the relative contribution of distillation loss and classification loss in the overall knowledge distillation process. When λ equals to 1, the student model focuses only on distillation and is trained only using the knowledge from the teacher model. When λ equals to 0, the student model is trained regularly using labeled data.

(e) The use of soft targets allows student model to account for the relative probabilities of different classes in the teacher model's output and have a deeper understanding of how teacher model thinks about and generalizes on different data. Thus, by performing knowledge distillation from teacher model to student model, we are essentially letting teacher model teach the student model how to mimic its behavior to generalize to different data as it does. This is equivalent to the effect of regularization, where we add regularization term to prevent model from memorizing the training data, so that the model can generalize well.

B. Question 2 to 8

The code implementations for questions 2 to 8 can be found in my Github repository [1]. Each code segment in the repository is labeled with its function and a corresponding question number for easy navigation and reference.

For question 5, figure 1 showcases the plot of test accuracy for teacher and student models. Although different hyperparameters for distillation provides different student model performance, we found that, in general, task balance parameter $\alpha = 0.9$ and temperature parameter $T = 20$ provides better performance for student model.

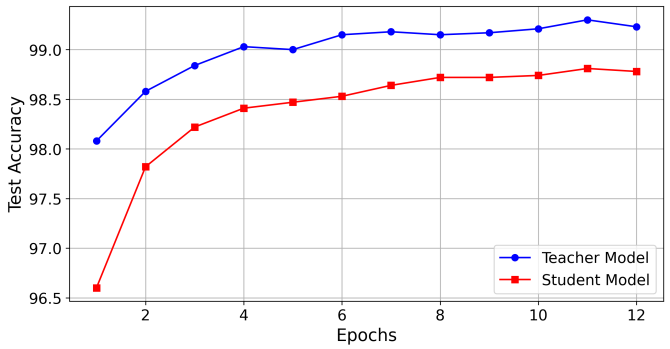

Fig. 1. Test accuracy vs. number of epochs for teacher and student models

For question 6, in order to better illustrate the impact of temperature hyperparameter in the test accuracy of student model, we computed the average test accuracy obtained over all epochs for each temperature, and a curve representing the average student test accuracy vs. temperature hyperparameters is plotted in figure 2. The test accuracy for each epoch and each temperature can also be found in [1].

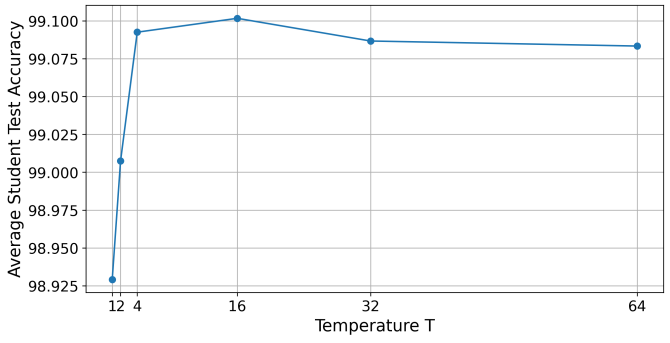

Fig. 2. Average test accuracy vs. temperature hyperparameters for student model

The plot indicates that, in general, as the value of temperature hyperparameter increases, the average test accuracy over all epochs also increases. This align with our expectations, as discussed in question 1.(c). The increase in temperature T will facilitate more information regarding how teacher model generalizes to different data to be transferred from teacher model to student model. Thus, better test accuracy can be expected from student model, as its decision making process become more similar to that of teacher model. However, as also indicated by figure 2, such increase in test accuracy does not persist with the continued increase of the value of temperature. The temperature scaled softmax function will eventually produce uniform distribution across all classes as we keep increasing temperature, and the class probabilities difference will once again become indistinguishable, resulting in less information to be transferred to from teacher model to student model.

For question 7, the test accuracy comparison between student model trained with knowledge distillation and student model trained without knowledge distillation is given by figure 3. We can see from this plot that the student model with knowledge distillation outperforms the student model without knowledge distillation by a noticeable margin.

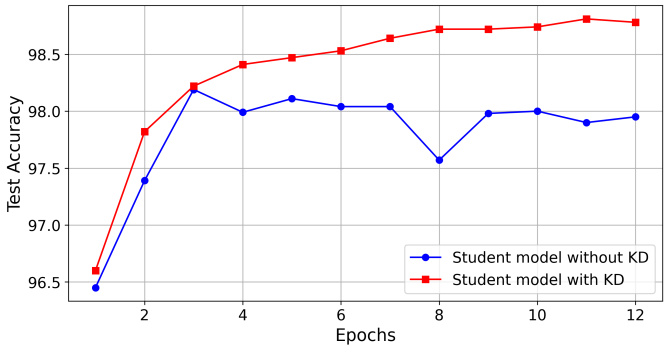

Fig. 3. Test accuracy vs. number of epochs for student model with KD and student model without KD

In question 8, we used Tensorflow built-in method $count\_params()$ to count the number of parameters in both teacher and student models. The visual comparison of the number of parameters for these two models is in figure 4. We used code in [3] to compute and compare the number of floating-point operations per second (FLOPs), the resulting visual comparison is given in figure 5. From these two plots, we can see that even though the student model has more parameters than the teacher model, it requires significantly fewer floating-point operations to make inference. This also align with our expectation, since the student model has relatively simpler architecture than teacher model.

C. Question 9 to 11

The paper I chose to read was subclass distillation [4].

(9) Building upon the idea of conventional knowledge distillation, subclass distillation still treats class probabilities as knowledge for distillation, however instead of using the probabilities of the main classes, it forces the teacher to divide existing classes into many subclasses, and the probability distribution across all subclasses is transferred from the teacher model to the student model as the knowledge. The intuition behind the use of subclasses is that subclasses may be able to capture finer-grained aspects or details within each main class, providing richer information for the student to learn from.

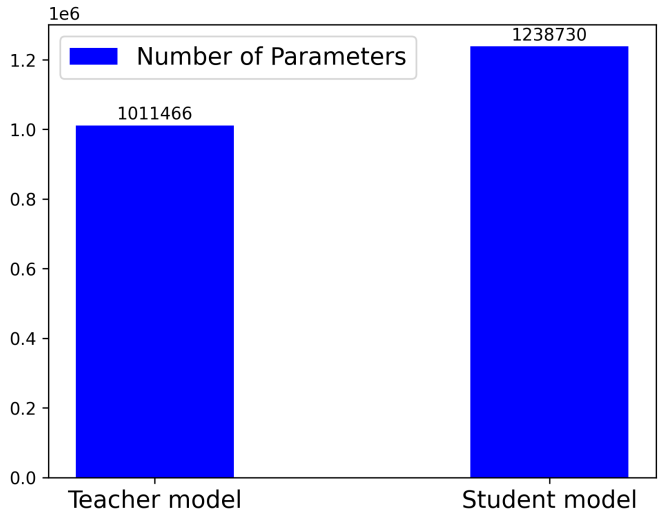

Fig. 4. Number of parameters comparison for teacher and student models

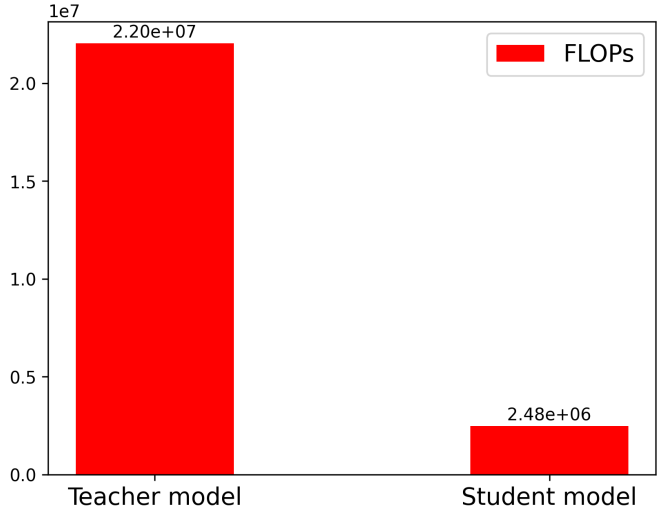

Fig. 5. Number of FLOPs comparison for teacher and student models

(10) In essence, subclass distillation takes the idea of knowledge distillation a step further by introducing subclasses to capture more fine-grained knowledge from the teacher model, enabling better transfer of knowledge to the student model. This is especially useful for classification problems that only have a few classes. The class probability distributions of a few class contains less information than those with many classes. The additional subclassess invented by the teacher model during subclass distillation might be able to discover finer hierarchical information from each main class and thus allows more information to be transferred from teacher model to student model.

(11) Introducing subclasses can increase the complexity of knowledge distillation process. For subclass distillation to be effective, more data may be required to represent each subclass adequately. In addition, it can be challenging to determine how many subclasses to use and how to define them. To address these issues, collecting more data can help decide on the optimal number of subclasses. Techniques like k-means clustering can also assist in defining meaningful subclasses.

D. Question 12

The code for questions 12 can be found in my Github repository [1]. The code structure for implementing subclass distillation largely resemble that of conventional knowledge distillation, albeit with several distinctions.

First of all, an additional hyperparameter called NUM_SUBCLASSES_PER_CLASS is introduced during model creation for teacher and student models. Thus, instead of having the original 10 classes in the MNIST dataset, the total number of classes and the number of units in the last dense layer of both model have increased to $10 * \text{NUM\_SUBCLASSES\_PER\_CLASS}$.

In addition, additional steps are required to compute the predicted probabilities for each main class due to the presence of subclasses. This can be done by reshaping the softmax probability tensor for all subclasses to shape of [batch_size, class, subclass] and summing over the subclass axis, which contains the probabilities of all subclasses for a class, to get the predicted probability for the class. The code implementation of this idea is included in the following code listing.

```
# It has shape [batch size, c x s]
subclass_logits = cnn_model(images,
training=True)

# Compute softmax probabilities across all
 classes and subclasses.
softmax_probs = tf.nn.softmax(
subclass_logits)

# Reshape probabilities to [batch_size, c,
 s].
softmax_probs = tf.reshape(softmax_probs,
[-1, 10, NUM_SUBCLASSES_PER_CLASS])

# Compute the sum of softmax probabilities
 across the subclasses for each class.
# The resulting shape is [batch_size, c]
summed_probs = tf.reduce_sum(softmax_probs
, axis=-1)
```

Listing 1. Python code for obtaining class probabilities from subclass probabiliities in subclass distillation

By replacing the softmax probability computation in the conventional knowledge distillation with the above process, we are able to compute the cross-entropy loss between one-hot encoded hard labels and class probability distribution for both teacher model and student model, and the evaluation method, i.e., $compute\_num\_correct(model, images, labels)$, can also benefit from this idea to compute softmax probability for each class in subclass distillation. As for distillation loss of student model, we simply just replaced the class probabilities

of teacher model from conventional knowledge distillation with the subclass probabilities, and used them as soft labels. The cross-entropy is computed between those soft labels and the subclass probabilities produced by the student model, through which student model will be guided to mimic the behaviour of the teacher model to produce similar subclass probability distribution.

Finally, the auxiliary loss that encourages the network to use all subclasses, i.e., assigning not negligible probabilities to all subclasses of a class, during subclass distillation is computed using built-in matrix manipulation function in Tensorflow, which is shown in the following code listing.

```
batch_size = tf.shape(
normalized_subclass_logits)[0]
T = 1.0

# Compute the temperature-scaled logits
product matrix
logits_product_matrix = tf.matmul(
normalized_subclass_logits, tf.transpose(
normalized_subclass_logits)) / T

# Compute the exponentiated logits product
 matrix
exp_logits_product_matrix = tf.exp(
logits_product_matrix)

# Compute the numerators: diagonal
elements of the matrix represent e^{\vec{
v_i}^T\vec{v_i}/T}
numerators = tf.linalg.diag_part(
exp_logits_product_matrix)

# Compute the denominators: mean of each
row
denominators = tf.reduce_mean(
exp_logits_product_matrix, axis=-1)

# Compute the auxiliary loss for each
vector in the batch
aux_losses = tf.math.log(numerators /
denominators)

# Compute the mean auxiliary loss across
the batch
L_aux = -tf.reduce_mean(aux_losses)
```

Listing 2. Python code for implementing auxiliary loss in subclass distillation

The resulting test accuracies for teacher and student models over all training epochs are plotted in figure 6. By comparing figure 1 and figure 6, we can see that the use of subclass distillation for MNIST dataset on the given two model architectures does not further improve the performance of student model. This is expected, since subclass distillation is most useful in cases where there are only a few classes. In our classification problem, there are 10 classes, which already allows a sufficient amount of information about the generalization tendencies of teacher model to be transferred to student model. Adding more subclasses will not further improve the classification performance but will incur additional computation complexity during knowledge distillation.

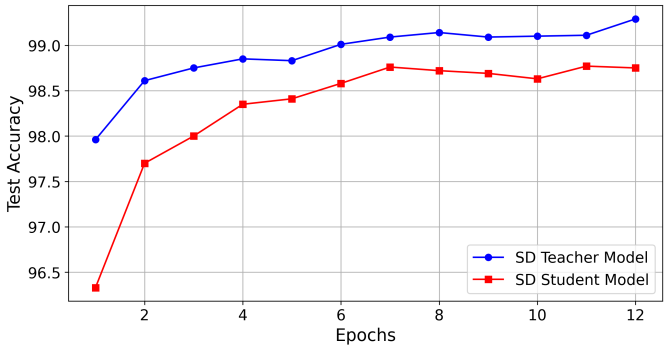

Fig. 6. Test accuracy vs. number of epochs for teacher and student models in subclass distillation

E. Question 13

For this question, we tried to use saliency maps [5] to highlight the most influential parts of the input for a prediction. The code for this question can be found in my Github repository [1]. We have referred to [6] during our implementation of saliency map computation. By comparing the saliency maps of teacher and student models, we tried to understand which parts of the input that different models tend to focus on to make their predictions. As shown in figure 7, the saliency map comparison among these three models does provide us some insights about the behavior of these three models and their similarities. The student model with knowledge distillation has a different saliency map from the student model without knowledge distillation, suggesting that the knowledge distillation process does influence the way the student model sees and interprets the input. In addition, it appears that the student model with knowledge distillation has mimicked some behaviour of the teacher model, albeit in a different manner. We found that the brighter regions in the teacher's saliency map, which represent regions of higher influence for the teacher's prediction, correspond to darker regions in the saliency map of student model with knowledge distillation, which represent regions of lesser influence for the student's prediction. In other words, the part of input that is most influential, e.g., bright region, to teacher's prediction is roughly the part that is least influential, e.g., dark, to the student model's prediction. The reason behind this is unclear, but my guess is that different models, i.e., teacher model and student model, have different architectures, which means they might focus on different aspects of the input. For instance, CNNs, i.e.,teacher model, tend to focus on local patterns, while fully connected networks, i.e., student with/without KD, might distribute their attention more globally. This can cause the saliency maps to look quite different.

II. Task 2

All models in this section are trained on free T4 GPU provided by Google Colab. The complete code can be found in my Github repository [7]. In this section, instead of using Tensorflow for implementation, we switched to PyTorch due to our greater familiarity and comfort working with PyTorch.

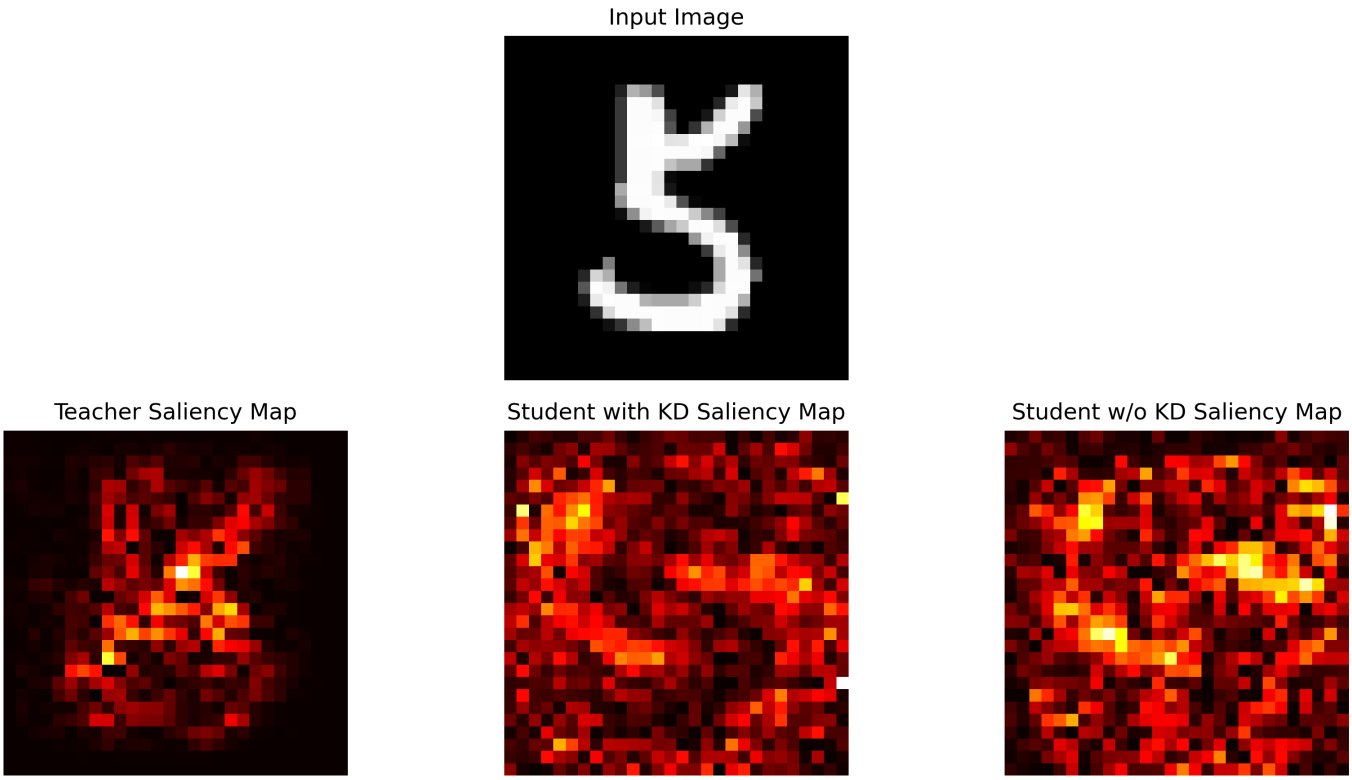

Fig. 7. Saliency maps for teacher model, student model with knowledge distillation, and student model without knowledge distillation

A. *Question 1*

(a) MHIST dataset contains images of colorectal polyps. Both pre-trained ResNet50 model and pre-trained MobileNetV2 model do not possess the capability to classify colorectal polyps, as they are pre-trained on ImageNet which does not contain any colorectal polyps examples. However, the first few layers of deep neural networks, especially those trained on ImageNet, can usually capture general features in images like edges, textures, and shapes. These features are often generic and can be beneficial for many other visual classification tasks. By using the architecture of ResNet50 or MobileNetV2 and their pre-trained weights, we can leverage their already-learned primitive features identifiers for our tasks. In addition, the final layer of deep neural networks is usually customized for the dataset they are trained on. For example, different dataset may contain different number of classes, and thus having different number of neurons in their last classification layer. Even though the MHIST dataset have different number of classes than ImageNet, by leveraging fine-tuning processes, we can replace the classification layers of ResNet50 and MobileNetV2 models with a customized classification layer tailored for MHIST, freeze the pre-trained weights in previous layers, and fine-tune the parameters on new classification layers. This allows us to obtain our task specific model in a more efficient way.

(b) Residual block is a fundamental unit in Residual Network architecture [8]. It usually contains 2 convolutional layers with batch normalization layer attached after each, activation functions, and most importantly a residual path that connect the input of block to the output, creating skip connection.

(c) Compared to ResNetV1 which has the following order within the residual block: Conv $\rightarrow$ BN $\rightarrow$ ReLU, the order within the residual block has changed to: BN $\rightarrow$ ReLU $\rightarrow$ Conv in ResNetV2. In addtion, in ResNetV1, ReLU activation is applied after the addition of the residual path. In ResNetV2, no activation is applied after addition.

(d) MobileNetV1 used depthwise separable convolutions to reduce computational cost and build lightweight deep neural networks. MobileNetV2 build upon this by adding linear bottlenecks between the layers and connecting bottlenecks with residual path, which forms a block called inverted residual block [9].

(e) This is due to the presence of residual connection in the residual block [10]. For example, we can express the function in residual block as $f(x) + x$, where $f(x)$ is the function performed by the original layers and $+x$ is the residual connection in the block. If we take the derivative of $f(x) + x$ with respect to x, we will get $\frac{d}{dx} f(x) + 1$. The vanishing gradient problem is usually occured during backpropogation where gredients are

repeatedly multiplied by some small number during the backpropogation along the $f(x)$ path, i.e., $\frac{d}{dx}f(x)$. The $+1$ gives the backpropogated gradient a path to avoid the processing done in $f(x)$ and flow uninterruptly, which essentially mitigate the problem of vanishing gradient.

(f) MobileNetV2 is considered as a lightweight model. It and its predecessor MobileNetV1 are designed for mobile devices with limited computation resources. Thus, its architecture, such as depthwise separable convolutions and inverted residual blocks, are tailored to reduce the number of FLOPs and parameters required to make prediction and provide a good balance between performance and computational efficiency.

B. Question 2

The code implementations for everything in questions 2 can be found in my Github repository [7]. Each code segment in the repository is labeled with its function for easy navigation and reference. Again, I only include segments of the code that are necessary for answering the questions in the project and explaining the results in this report.

During our implementation and testing of transfer learning in task 2, we found that sometimes it can lead to overfitting, especially when we unfreeze more weights in the layers prior to the final classification layer of the model. Thus, letting model to keep training over all fine tune epochs sometimes gives model poor generalization ability. To avoid this issue, we employed a method that evaluates the model's generalization ability throughout the training process and ensures the version of the model demonstrating the best generalization ability up to that point is preserved [11]. After training, only the model parameters that has the best generalization ability over training will be loaded and used. Such method is implemented as follow

```
    model_wrapper.model.eval()
    for inputs, labels in test_dataloader:
        inputs, labels = inputs.to(
model_wrapper.device), labels.to(
model_wrapper.device)
        output = model_wrapper.model(
inputs)
        loss = model_wrapper.criterion(
output, labels)
        test_loss += loss.item() * inputs.
size(0)

    test_loss = test_loss / len(
test_dataloader.dataset)
    print('Epoch: {} \tTest Loss: {:.6f}'.
format(epoch, test_loss))

    if test_loss <= test_loss_min:
        print('Test loss decreased ({:.6f}
 --> {:.6f}).  Model Parameters Saved'.
format(
            test_loss_min, test_loss))
        torch.save(model_wrapper.model.
state_dict(), save_path)
    test_loss_min = test_loss
```

Listing 3. Python code for evaluating model's generalization ability and saving model parameters

We use this method to train models throughout task 2.

To repeat question 5 of task 1 in task 2, we have first identified that a major difference between MHIST dataset and MNIST dataset is that MHIST dataset contains imbalanced data. Since test accuracy cannot comprehensively evaluate the performance of a model for imbalanced data, we evaluate the model's test accuracy for each class and used weighted F1 score alongside to get a relatively more comprehensive idea about the performance of the model on imbalanced data. The performance comparison between teacher model, which is trained by transfer learning, and student model, which is trained through knowledge distillation from the teacher model with $\alpha = 0.4$ and $temperature = 2$, is shown in table I.

TABLE I
PERFORMANCE COMPARISON BETWEEN TEACHER AND STUDENT MODELS

Metric	Teacher	Student
Test Accuracy (SSA)	75% (272/360)	49% (179/360)
Test Accuracy (HP)	87% (537/617)	88% (545/617)
Test Accuracy (Overall)	82% (809/977)	74% (724/977)
F1 Score (SSA)	0.764045	0.585925
F1 Score (HP)	0.864734	0.811616
Weighted F1 Score	0.827633	0.728454

To repeat question 6 of task 1 in task 2, instead of plotting a curve representing student model test accuracy vs. temperature hyperparmeters T, weighted F1 score is used in place of test accuracy to report performance. Figure 8 showcases the resulting plot. As we can see, when task balance parameter α is fixed at 0.5, temperature value of 1 allows the student model to achieve the highest weighted F1 score, indicating the model has achieved a good and balanced classification performance across all classes. However, the overall relationship between student model performance and temperature parameter is not as clear as the relationship found in task 1 question 6, and it varies across different runs. We think the increased complexity in model architecture has introduced more factors that directly or indirectly affect the model performance, which makes the relationship between model performance and temperature value harder to find.

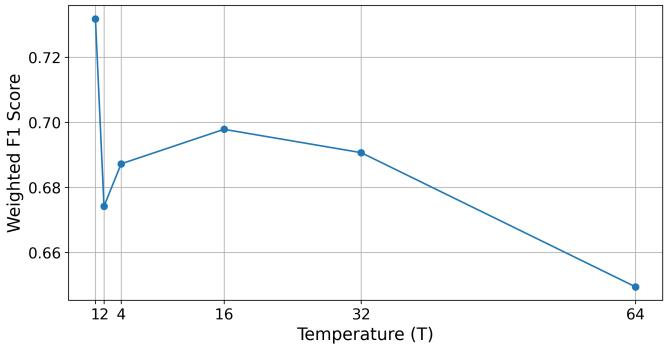

Fig. 8. Weighted F1 score vs. temperature hyperparameters for student model

To repeat question 7 of task 1 in task 2, the performance comparison between student model trained with knowledge distillation and student model trained without knowledge distillation but with transfer learning is given by table II. In general, student model trained with knowledge distillation can achieve a relatively more balanced classification performance across two classes compared to student model trained with transfer learning.

TABLE II
PERFORMANCE COMPARISON BETWEEN STUDENT MODEL WITH AND WITHOUT KD

Metric	Student with KD	Student w/o KD
Test Accuracy (SSA)	49% (179/360)	43% (158/360)
Test Accuracy (HP)	88% (545/617)	90% (561/617)
Test Accuracy (Overall)	74% (724/977)	73% (719/977)
F1 Score (SSA)	0.585925	0.550523
F1 Score (HP)	0.811616	0.813043
Weighted F1 Score	0.728454	0.716311

To repeat question 8 of task 1 in task 2, we used PyTorch built-in method $model.parameters()$ to count the number of parameters in both teacher and student models. The visual comparison of the number of parameters for these two models is in figure 9. We used $thop$ library in PyTorch to compute and compare the number of FLOPs, the resulting visual comparison is given in figure 10. From these two plots, we can see that the student model in task 2 has fewer parameters and also requires significantly fewer floating-point operations to make inference compared to the teacher model in task 2. Such outcome is expected as student model is based on MobileNetV2 model, which is a lightweight neural network model compared to ResNet50 model.

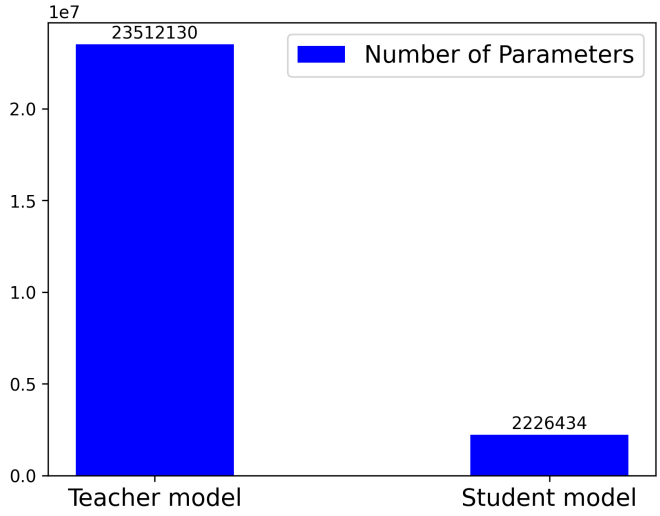

Fig. 9. Number of parameters comparison for teacher and student models

To repeat question 12 of task 1 in task 2, we reimplement subclass distillation in PyTorch. The implementation of subclass distillation in PyTorch is very similar to the TensorFlow implementation we discussed in the task 1 section. Thus,

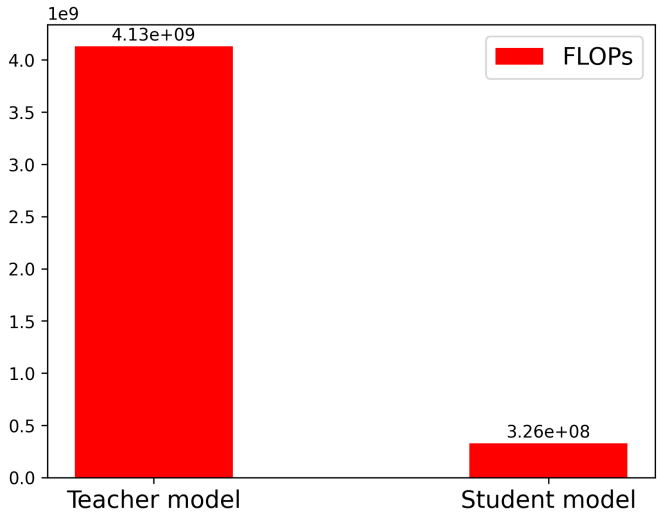

Fig. 10. Number of FLOPs comparison for teacher and student models

we omit the explanation of code here and direct reader to the Github repository [7] for more details. The performance comparison between teacher model and student model trained with subclass distillation is summarized in table III. Compared to table I, we can see a noticeable performance improvement on teacher model due to the use of subclass during training.

TABLE III
PERFORMANCE COMPARISON BETWEEN TEACHER AND STUDENT MODELS

Metric	Teacher	Student
Test Accuracy (SSA)	79% (287/360)	50% (180/360)
Test Accuracy (HP)	86% (536/617)	86% (535/617)
Test Accuracy (Overall)	84% (823/977)	73% (715/977)
F1 Score (SSA)	0.788462	0.578778
F1 Score (HP)	0.874388	0.803303
Weighted F1 Score	0.842726	0.720571

To repeat question 13 of task 1 in task 2, we implemented and adjusted the $compute\_saliency$ function in PyTorch to accommodate images that have three channels, since images in MHIST dataset have three channels. The resulting saliency map comparison for all three channels is shown in figure 11. From the figure, we can see that teacher model and student model with knowledge distillation identify roughly the same regions as the most influential parts of input to make a prediction across all three channels.

C. Question 3

By freezing most of the layer parameters on a pre-trained model and fine tuning only a few top layers, transfer learning allows the student model to leverage the primitive feature identifier of the model, that is learned from training on a large dataset, and focus on the learning of task specific features. This allows student model to be trained more efficiently and with relatively less data for a customized task. In addition, with the help of knowledge distillation, knowledge from an even larger model can be transferred to the student model,

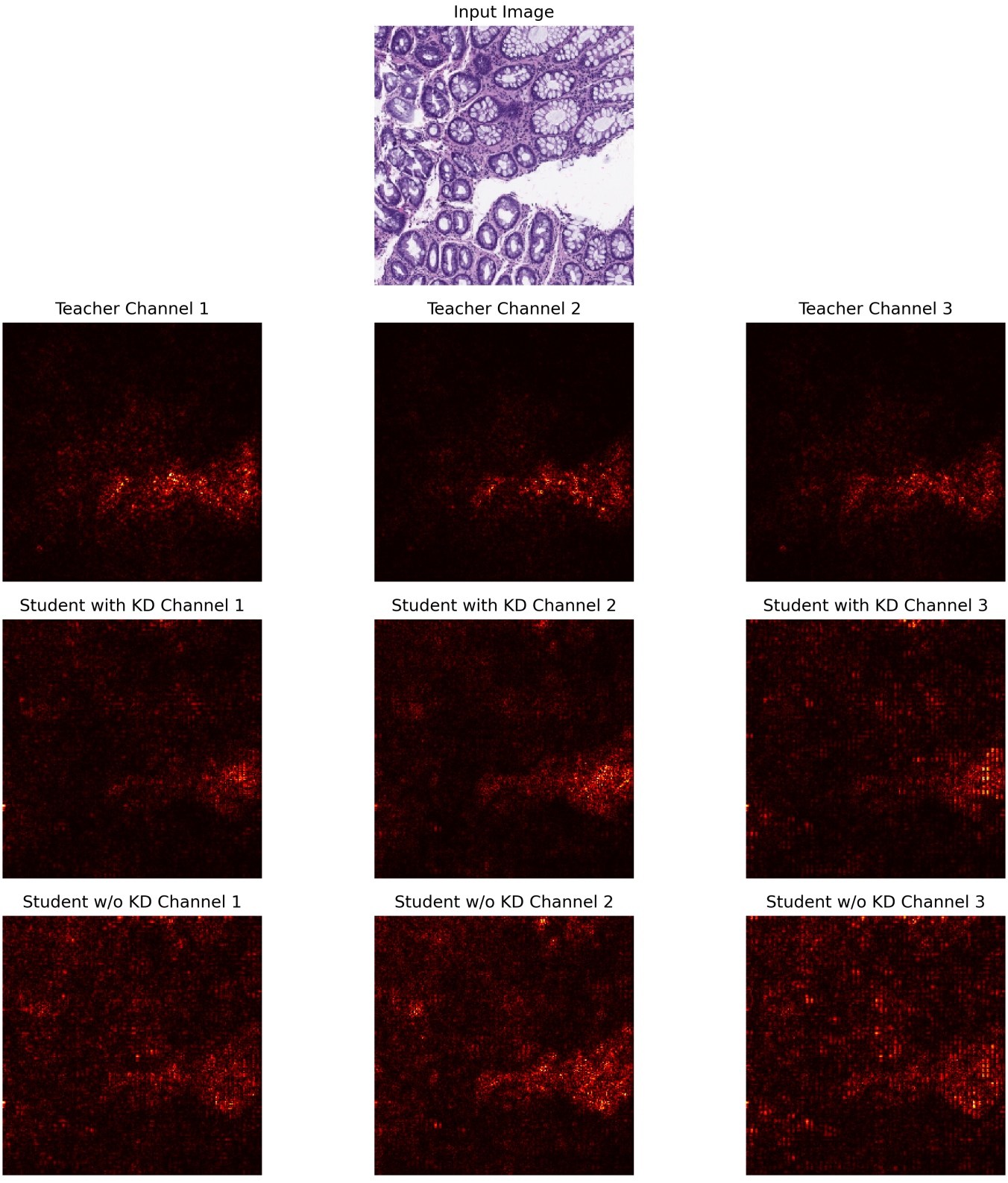

Fig. 11. Saliency maps comparison for teacher model, student model with knowledge distillation, and student model without knowledge distillation across all three channels

which essentially allows student model to use existing pre-trained model's knowledge to better mimic the behaviour of teacher model.

Judging by the results we obtained, we think pre-trained weights do help the teacher and student models perform well on the MHIST dataset, given that we only have very limited data for training. If we train ResNet50 and MobileNetV2 models from scratch using the data we have, we can anticipate overfitting and very poor generalization capability from them.

In our experiment, knowledge transfer does improve the performance of student model by a small margin, compared to student model from scratch. The weighted F1 score of student model after knowledge distillation from teacher model is 0.728, whereas the weighted F1 score of student model learned from scratch is 0.716, which suggests that knowledge distillation has provided a noticeable improvement on the performance of the student model.

REFERENCES

1 Sun, Z., "Task 1: Knowledge distillation in mnist dataset," https://github.com/sunhuanhuan920/ECE1512_2023F_ProjectRepo_Zhenhuan_Sun/blob/main/Project%20A/Task_1.ipynb, 2023.

2 Hinton, G., Vinyals, O., and Dean, J., "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

3 Tokusumi, "Keras-flops," https://github.com/tokusumi/keras-flops, 2020.

4 Müller, R., Kornblith, S., and Hinton, G., "Subclass distillation," *arXiv preprint arXiv:2002.03936*, 2020.

5 Simonyan, K., Vedaldi, A., and Zisserman, A., "Deep inside convolutional networks: Visualising image classification models and saliency maps," *arXiv preprint arXiv:1312.6034*, 2013.

6 Rastogi, A., "Visualizing neural networks using saliency maps in pytorch," https://medium.datadriveninvestor.com/visualizing-neural-networks-using-saliency-maps-in-pytorch-289d8e244ab4, 2020.

7 Sun, Z., "Task 2: Knowledge distillation in mhist dataset," https://github.com/sunhuanhuan920/ECE1512_2023F_ProjectRepo_Zhenhuan_Sun/blob/main/Project%20A/Task_2.ipynb, 2023.

8 He, K., Zhang, X., Ren, S., and Sun, J., "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

9 Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C., "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

10 Zhang, A., Lipton, Z. C., Li, M., and Smola, A. J., "Dive into deep learning," *arXiv preprint arXiv:2106.11342*, 2021.

11 Gangoly, S., "Cnn on cifar10 data set using pytorch," https://shonit2096.medium.com/cnn-on-cifar10-data-set-using-pytorch-34be87e09844, 2021.