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

Student's Name: Bo Wen
Student ID: 1005072662
Course Number: ECE1512
Date Due: Nov.1, 2023
Date Handed In: Nov.1, 2023
GitHub Repository:
https://github.com/bw24ca/ECE1512_2023F_ProjectRepo_BoWen

I. Introduction

The objective of this project is to provide practical experience in model compression through knowledge distillation, a technique that facilitates the transfer of knowledge from larger, more complex models to smaller, more efficient models suitable for real-world applications. The project comprises two main tasks:

1) Implementing a conventional knowledge distillation framework for digit classification on the MNIST dataset.
2) Using transfer learning and knowledge distillation to train a lightweight model capable of mimicking a pre-trained larger model on the MHIST clinical histopathology dataset.

Detailed descriptions of the datasets and tasks are provided in the subsequent sections.

II. Datasets

A. MNIST

- **Paper**: Gradient-based learning applied to document recognition [1]
- **Link**: http://yann.lecun.com/exdb/mnist/
- **Description**: A popular digit classification dataset, a subset of a larger set available from NIST. The MNIST dataset includes 60,000 training and 10,000 test images of handwritten digits, each size-normalized and centered in a fixed-size image.

B. Minimalist HIStopathology (MHIST)

- **Paper**: A Petri Dish for Histopathology Image Analysis [2]
- **GitHub Link**: https://bmirds.github.io/MHIST/
- **Description**: A binary classification dataset of fixed-size colorectal polyp images. MHIST includes 2,162 images of hyperplastic polyps (benign) and 990 images of sessile serrated adenomas (precancerous), with labels determined by a majority vote of seven board-certified gastrointestinal pathologists.

III. Task 1

1. Questions in Task 1:

(a) What is the purpose of using KD in this paper?

The purpose of using Knowledge Distillation (KD) in this paper is to improve the performance of machine learning models, particularly neural networks, by compressing the knowledge from an ensemble of large and complex models (teacher models) into a smaller, more manageable model (student model). This approach aims to retain the performance benefits of the ensemble while

reducing the computational cost and making the model easier to deploy, especially for applications with resource constraints.

(b) In the paper, what knowledge is transferred from the teacher model to the student model?

The knowledge transferred from the teacher model to the student model in KD is in the form of ”soft targets.” These soft targets are the class probability distributions produced by the teacher model. Instead of using the hard labels (the actual class labels), the student model is trained to mimic the output distribution of the teacher model, learning both the correct class and the relationships between different classes as represented in the teacher's output.

(c) What is the temperature hyperparameter T? Why do we use it when transferring knowledge from one model to another? What effect does the temperature hyperparameter have in KD?

The temperature hyperparameter T is introduced to soften the output probabilities of the teacher model. When T=1, the softmax function produces the standard class probabilities. As T increases, the output probabilities become softer, meaning that the differences between the probabilities of different classes are reduced. This helps in transferring the knowledge from the teacher to the student by providing more information about the relationships between classes, even when the teacher is very confident about a particular class. The softened probabilities help the student model to learn from the teacher's knowledge, including the teacher's uncertainty about different classes.

(d) Explain in detail the loss functions on which the teacher and student model are trained in this paper. How does the task balance parameter affect student learning?

The teacher model is typically trained using a standard loss function like cross-entropy with hard labels. Once the teacher model is trained, the student model is trained using a combination of two loss functions:

Loss1: Cross-entropy loss between the student's predictions and the soft targets provided by the teacher model. This loss function helps the student to mimic the teacher's output distribution.

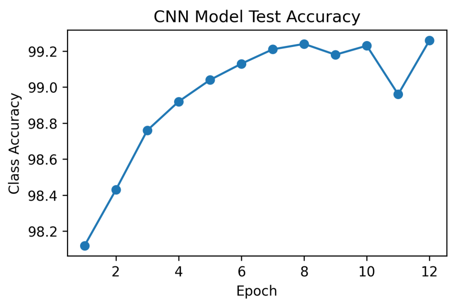
Loss2: Cross-entropy loss between the student's predictions and the true labels. This loss function ensures that the student model also learns to predict the correct labels.

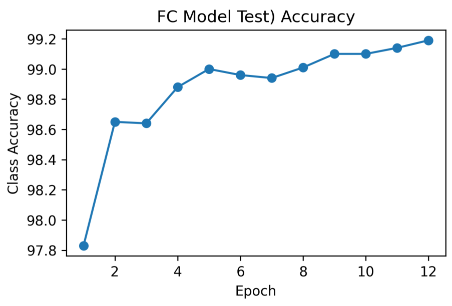
The student's objective function is a weighted average of Loss1 and Loss2, with the task balance parameter controlling the balance between the two. A higher weight on Loss1 encourages the student to mimic the teacher's output distribution, while a higher weight on Loss2 encourages the student to predict the correct labels directly.

(e) Can we look at the KD as a regularization technique, here? Explain your rationale.

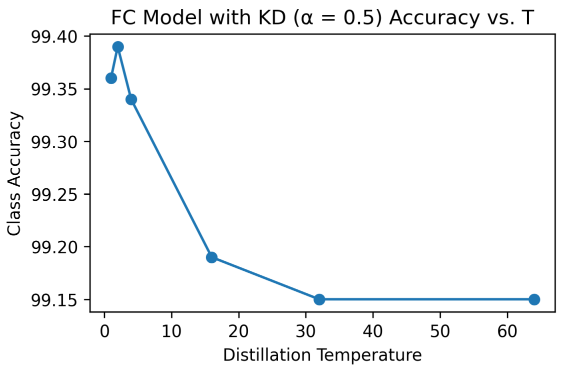
Yes, KD can be viewed as a regularization technique. By encouraging the student model to mimic the output distribution of the teacher model, KD helps to prevent overfitting, especially when the student model is smaller and less complex than the teacher model. The soft targets provided by the teacher model contain information about the relationships between different classes, helping the student model to generalize better to unseen data. Additionally, the use of the temperature hyperparameter to soften the output probabilities can also be seen as a form of regularization, preventing the student model from becoming too confident in its predictions and encouraging it to consider alternative possibilities.

2-5 Please see the appendix for Task 1 codes.





6. Plot a curve representing student test accuracy vs. temperature hyperparameters (T = 1,2,4,16,32, and 64) when the task balance parameter is 0.5. Explain the effect of the temperature hyperparameter in this experiment.



The temperature hyperparameter T softens the output probabilities from the teacher model. A low T leads to sharper probabilities, making the student focus on the most likely class. A high T provides softer probabilities, helping

the student learn inter-class relationships but can lead to confusion if too high. The plot typically shows an increase in test accuracy with T, peaking at an optimal value before decreasing as T becomes excessively large.

7. Please see the appendix for Task 1 codes.
8. Compare the number of parameters and floating-point operations per second (FLOPs) for the teacher and student models. Explain your findings.
   The flops of teacher model is 32792714, the flops of student model is 32792714.
9. Now that you have had a chance to implement, understand, and train the teacher and student models in the KD framework, you will use the state-of-the-art KD algorithm to improve the performance of the student model. What is the main novel idea of the paper?
   The main novel idea of the paper ”Improved Knowledge Distillation via Teacher Assistant” is the introduction of a teacher assistant (TA) model in the knowledge distillation process. This TA model serves as an intermediary between the complex teacher model and the simpler student model. The TA, being less complex than the teacher but more complex than the student, helps in transferring knowledge more effectively to the student, resulting in better performance of the student model compared to direct knowledge distillation from the teacher. This approach addresses the challenge of knowledge distillation when there is a significant disparity in complexity between the teacher and student models.
10. How does the proposed method improve student performance in comparison to the conventional KD?
    The proposed method introduces a Teacher Assistant (TA) model as an intermediary in the knowledge distillation process, bridging the complexity gap between a large teacher model and a smaller student model. This staged distillation results in more effective knowledge transfer, enhancing the student model's performance compared to conventional knowledge distillation.
11. What are some limitations of the proposed method? How can they be addressed?
    The proposed method with a Teacher Assistant (TA) introduces additional training complexity and requires careful selection of the TA's size and architecture. Addressing these limitations could involve automating the TA selection process and optimizing the training pipeline to reduce computational overhead.
12. Conduct an experiment to implement the novel component of the proposed method with the same models as in Figures 2 and 3. Report the student test accuracy, and explain your approach and its outcomes.
    The accuracy is 99.26, please see appendix for details.
13. Please see the appendix for Task 1 codes.

IV. Task 2

1. Questions in Task 2:

(a) How can we adapt these models for the MHIST dataset using transfer learning? Talk about the Feature Extraction and Fine-Tuning processes during transfer learning.
Feature Extraction: We can use the pre-trained ResNet50V2 and MobileNetV2 as feature extractors by removing their final fully connected layers and replacing them with layers tailored to the MHIST dataset, which likely has a different number of classes than ImageNet. We freeze the weights of the pre-trained layers and only train the new layers added, ensuring that we retain the generic features learned from ImageNet while adapting the model to the new task.
Fine-Tuning: After feature extraction, we can optionally unfreeze some of the top layers of the pre-trained model and continue training on the MHIST dataset. This allows the model to fine-tune the higher-level features to better suit the specific task.

(b) What is a residual block in ResNet architectures?
A residual block in ResNet architectures is designed to allow the training of very deep networks. It includes a shortcut connection that skips one or more layers, adding the input of the block to its output. This helps in mitigating the vanishing gradient problem and enables the training of deep networks.

(c) What are the differences between the ResNetV1 and ResNetV2 architectures?
ResNetV1 applies batch normalization and ReLU before the convolution, whereas ResNetV2 applies them after the convolution. This change, known as pre-activation, helps in further mitigating the vanishing gradient problem and provides regularization.

(d) What are the differences between the MobileNetV1 and MobileNetV2 architectures?
MobileNetV2 introduces inverted residuals and linear bottlenecks. Inverted residuals include a shortcut connection between the thin bottleneck layers, and linear bottlenecks remove non-linearities in the final layer of each block, preserving information. These changes improve efficiency and performance.

(e) How can ResNet architectures, regardless of model depth, overcome the vanishing gradient problem?
ResNet architectures overcome the vanishing gradient problem through the use of residual blocks and shortcut connections, which allow gradients to flow directly through the network, even through many layers.

(f) Is MobileNetV2 a lightweight model? Why?
Yes, MobileNetV2 is considered a lightweight model. It is designed to be efficient in terms of computational resources, making it suitable for mobile

and edge devices. It achieves this through depthwise separable convolutions, which significantly reduce the number of parameters and computations needed compared to standard convolutions.

2. Please see the appendix for Task 2 codes.
3. Explain the effect of transfer learning and knowledge distillation in the performance of the student model. Do pre-trained weights help the teacher and student models perform well on the MHIST dataset? Does knowledge transfer from the teacher to the student model increase the student's performance?
   1.Transfer learning and knowledge distillation enhance the student model's performance by leveraging pre-trained features and distilled knowledge from a teacher model, respectively. This accelerates training, improves generalization, and enables the student to achieve higher accuracy, especially when labeled data is scarce.
   2.Pre-trained weights provide a strong feature foundation for both teacher and student models, boosting their performance on the MHIST dataset compared to training from scratch, as they encapsulate generic visual patterns learned from large datasets.
   3.Yes, knowledge transfer from the teacher to the student model increases the student's performance by providing additional guidance and insights, helping the student to make more accurate predictions and learn complex patterns with fewer parameters.

V. Conclusion

In conclusion, this project underscores the efficacy of transfer learning and knowledge distillation in optimizing deep learning models for specific tasks, such as image classification on the MNIST, MHIST dataset. Leveraging pre-trained weights and distilling knowledge from complex models to simpler ones, we achieved enhanced performance and efficiency, demonstrating the potential of these techniques in real-world applications and setting a foundation for future exploration in model compression and efficient deep learning.

VI. References

References

[1] Y. LeCun, et al., ”Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278-2324, Nov. 1998.
[2] Authors, ”A Petri Dish for Histopathology Image Analysis,” *ArXiv*, 2021. [Online]. Available: https://arxiv.org/abs/2101.12355
[3] K. He, X. Zhang, S. Ren, and J. Sun, ”Identity mappings in deep residual networks,” In *European conference on computer vision*, pp. 630–645, Springer, 2016. [Online]. Available: https://arxiv.org/abs/1603.05027
[4] S. I. Mirzadeh, M. Farajtabar, A. Li, N. Levine, A. Matsukawa, and H. Ghasemzadeh, ”Improved knowledge distillation via teacher assistant,” In *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 5191–5198, 2020. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/5963/5819
[5] R. Müller, S. Kornblith, and G. Hinton, ”Subclass distillation,” *arXiv preprint arXiv:2002.03936*, 2020. [Online]. Available: https://arxiv.org/abs/2002.03936
[6] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, ”Mobilenetv2: Inverted residuals and linear bottlenecks,” In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018. [Online]. Available: https://openaccess.thecvf.com/content_cvpr_2018/papers/Sandler_MobileNetV2_Inverted_Residuals_CVPR_2018_paper.pdf

VII. Appendix

Appendix

A. Task 1 Codes

```
def preprocess(x):
    image = tf.image.convert_image_dtype(x['image'], tf.float32)
    subclass_labels = tf.one_hot(x['label'], builder.info.features['label'].
   num_classes)
    return image, subclass_labels

mnist_train = tfds.load('mnist', split='train', shuffle_files=False).cache()
mnist_train = mnist_train.map(preprocess)
mnist_train = mnist_train.shuffle(builder.info.splits['train'].num_examples)
mnist_train = mnist_train.batch(BATCH_SIZE, drop_remainder=True)

mnist_test = tfds.load('mnist', split='test').cache()
mnist_test = mnist_test.map(preprocess).batch(BATCH_SIZE)
```

Listing 1: Data Loading

```python
def create_teacher_model():
    cnn_model = tf.keras.Sequential()

    # your code start from here for stpe 2
    cnn_model.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3,strides=1,
    padding='same', activation='relu', input_shape=(28, 28, 1)))
    cnn_model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2),strides=1,
    padding='same'))
    cnn_model.add(tf.keras.layers.Conv2D(filters=64, kernel_size=3,strides=1,
    padding='same', activation='relu', input_shape=(28, 28, 1)))
    cnn_model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2),strides=2,
    padding='same'))
    cnn_model.add(tf.keras.layers.Flatten())
    cnn_model.add(tf.keras.layers.Dropout(0.5))
    cnn_model.add(tf.keras.layers.Dense(128, activation='relu'))
    cnn_model.add(tf.keras.layers.Dropout(0.5))
    cnn_model.add(tf.keras.layers.Dense(NUM_CLASSES))

    return cnn_model

# Build fully connected student.
def create_student_model():
    fc_model = tf.keras.Sequential()

    # your code start from here for step 2
    fc_model.add(tf.keras.layers.Flatten())
    fc_model.add(tf.keras.layers.Dense(784, activation='relu'))
    fc_model.add(tf.keras.layers.Dense(784, activation='relu'))
    fc_model.add(tf.keras.layers.Dense(NUM_CLASSES))

    return fc_model

cnn_model = create_teacher_model()
cnn_model.summary()

fc_model = create_teacher_model()
fc_model.summary()
```

Listing 2: Model Creation

```python
def compute_teacher_loss(images, labels):
  subclass_logits = cnn_model(images, training=True)
  cross_entropy_loss_value = tf.reduce_mean(tf.nn.
    softmax_cross_entropy_with_logits(labels, subclass_logits))
  return cross_entropy_loss_value

  ALPHA = 0.5 # task balance between cross-entropy and distillation loss
DISTILLATION_TEMPERATURE = 4. #temperature hyperparameter

def distillation_loss(teacher_logits: tf.Tensor, student_logits: tf.Tensor,
    temperature: Union[float, tf.Tensor]):
```

```
  soft_targets = tf.nn.softmax(teacher_logits / temperature)

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

def compute_student_loss(images, labels):
  student_subclass_logits = fc_model(images, training=True)

  teacher_subclass_logits = cnn_model(images, training=False)
  distillation_loss_value = distillation_loss(teacher_subclass_logits,
    student_subclass_logits, DISTILLATION_TEMPERATURE)

  cross_entropy_loss_value = tf.reduce_mean(tf.nn.
    softmax_cross_entropy_with_logits(labels, student_subclass_logits))
  return ALPHA * cross_entropy_loss_value + (1- ALPHA) *
    distillation_loss_value
```

Listing 3: Compute Loss Function

```
def compute_num_correct(model, images, labels):

  class_logits = model(images, training=False)
  return tf.reduce_sum(
      tf.cast(tf.math.equal(tf.argmax(class_logits, -1), tf.argmax(labels, -1))
    ,
              tf.float32)), tf.argmax(class_logits, -1), tf.argmax(labels, -1)


def train_and_evaluate(model, compute_loss_fn):
  optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)
  accuracies = []
  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.trainable_variables)
      optimizer.apply_gradients(zip(grads, model.trainable_variables))

    # Run evaluation.
    num_correct = 0
    num_total = builder.info.splits['test'].num_examples
    for images, labels in mnist_test:
      a = compute_num_correct(model, images, labels)
      num_correct += a[0]
    class_accuracy = num_correct / num_total * 100
    accuracies.append(class_accuracy.numpy())
    print("Class_accuracy: " + '{:.2f}%'.format(class_accuracy))
  accuracies = np.round(accuracies, 2)
  return accuracies
```

Listing 4: Train and Evaluate Functions

```

cnn_model_acc = train_and_evaluate(cnn_model, compute_teacher_loss)

fig = plt.figure(figsize=(5, 3))
plt.plot(range(1, NUM_EPOCHS + 1), cnn_model_acc, "-o", label='CNN Model')
plt.xlabel('Epoch')
plt.ylabel('Class Accuracy')
plt.title('CNN Model Test Accuracy')
plt.savefig('/content/drive/MyDrive/ECE1512/figures/cnn_model_acc.png', dpi=300, bbox_inches='tight')

fc_model_acc = train_and_evaluate(fc_model, compute_student_loss)

fig = plt.figure(figsize=(5, 3))
plt.plot(range(1, NUM_EPOCHS + 1), fc_model_acc, "-o", label='Student')
plt.xlabel('Epoch')
plt.ylabel('Class Accuracy')
plt.title('FC Model Test) Accuracy')
plt.savefig('/content/drive/MyDrive/ECE1512/figures/fc_model_acc.png', dpi=300, bbox_inches='tight')
```

Listing 5: Training the Teacher Model

```
ALPHA = 0.5
T = [1, 2, 4, 16, 32, 64]
accuracies = []
for DISTILLATION_TEMPERATURE in T:
  print("DISTILLATION_TEMPERATURE: " + str(DISTILLATION_TEMPERATURE))
  # clear model
  tf.keras.backend.clear_session()
  # fc_model.set_weights(default_weights)
  a_ = train_and_evaluate(fc_model, compute_student_loss)
  accuracies.append(np.max(a_))
  # fc_model.save_weights('./models/fc_model_' + str(DISTILLATION_TEMPERATURE) + '.h5')

fig = plt.figure(figsize=(5, 3))
plt.plot(T, accuracies, "-o", label='Student')
plt.xlabel('Distillation Temperature')
plt.ylabel('Class Accuracy')
plt.title('FC Model with KD (   = 0.5) Accuracy vs. T')
plt.savefig('/content/drive/MyDrive/ECE1512/figures/fc_model_acc_vs_t.png', dpi=300, bbox_inches='tight')
```

Listing 6: Test Accuracy vs. Temperature Curve

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

# your code start from here for step 7
fc_model_no_distillation.add(tf.keras.layers.Flatten(input_shape=(28, 28, 1)))
fc_model_no_distillation.add(tf.keras.layers.Dense(784, activation='relu'))
fc_model_no_distillation.add(tf.keras.layers.Dense(784, activation='relu'))
fc_model_no_distillation.add(tf.keras.layers.Dense(NUM_CLASSES))

# test {"output": "ignore"}

def compute_plain_cross_entropy_loss(images, labels):
  student_subclass_logits = fc_model_no_distillation(images, training=True)
  cross_entropy_loss = tf.keras.losses.categorical_crossentropy(labels,
    student_subclass_logits, from_logits=True)

  return cross_entropy_loss
train_and_evaluate(fc_model_no_distillation, compute_plain_cross_entropy_loss)
```

Listing 7: Train student from scratc

```
# !pip install keras_flops
from keras_flops import get_flops

cnn_model_flops = get_flops(cnn_model, batch_size=1)
fc_model_flops = get_flops(fc_model, batch_size=1)

print('The flops of teacher model is',cnn_model_flops)
print('The flops of student model is',fc_model_flops)
cnn_model.summary()
fc_model.summary()
tf.keras.utils.plot_model(cnn_model, show_shapes=True)
tf.keras.utils.plot_model(fc_model, show_shapes=True)
```

Listing 8: Comparing the Teacher and Student Model (number of parameters and FLOPs)

```
ALPHA = 0.5 # task balance between cross-entropy and distillation loss
DISTILLATION_TEMPERATURE = 4. #temperature hyperparameter

# Build Teacher assistant.
ta_model = tf.keras.Sequential()
ta_model.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3,strides=1,
    padding='same', activation='relu', input_shape=(28, 28, 1)))
ta_model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2),strides=1,padding='
    same'))
ta_model.add(tf.keras.layers.Flatten())
ta_model.add(tf.keras.layers.Dense(784, activation='relu'))
ta_model.add(tf.keras.layers.Dense(784, activation='relu'))
ta_model.add(tf.keras.layers.Dropout(0.5))
ta_model.add(tf.keras.layers.Dense(NUM_CLASSES))

def compute_ta_or_student_loss(images, labels, learn_model, teach_model):
  student_subclass_logits = learn_model(images, training=True)

  # Compute subclass distillation loss between student subclass logits and
  # softened teacher subclass targets probabilities.
  teacher_subclass_logits = teach_model(images, training=False)
  distillation_loss_value = distillation_loss(teacher_subclass_logits,
    student_subclass_logits, DISTILLATION_TEMPERATURE)

  # Compute cross-entropy loss with hard targets.
  cross_entropy_loss_value = tf.nn.softmax_cross_entropy_with_logits(labels=
    labels, logits=student_subclass_logits)

  return ALPHA * cross_entropy_loss_value + (1- ALPHA) *
    distillation_loss_value

def train_and_evaluate_state_of_the_art(model, teach_model):

  optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)

  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:
         # your code start from here for step 4

        loss_value = compute_ta_or_student_loss(images, labels, model,
    teach_model)

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

    # Run evaluation.
    num_correct = 0
    num_total = builder.info.splits['test'].num_examples
    for images, labels in mnist_test:
      # your code start from here for step 4
      correct_results,_,_ = compute_num_correct(model, images, labels)
      num_correct += correct_results


    print("Class_accuracy: " + '{:.2f}%'.format(
        num_correct / num_total * 100))

# train teacher assistant model
train_and_evaluate_state_of_the_art(ta_model, teach_model=cnn_model)
ta_model.save("task1_ta_model")

```

```
# train student model from teacher assistant model
train_and_evaluate_state_of_the_art(fc_model, teach_model=ta_model)
ta_model.save("task1_student_from_ta_model")
```

Listing 9: Implementing the state-of-the-art KD Algorithm

B. Task 2 Codes

```

path = "/content/drive/MyDrive/ECE1512/images"
CSVfile = "/content/drive/MyDrive/ECE1512/annotations.csv"

annotations = pd.read_csv(CSVfile)
annotations.set_index('Image Name', inplace=True)
annotations.head(10)

images_name = os.listdir(path)

mhist_train_img = []
mhist_train_label = []
mhist_test_img = []
mhist_test_label = []
datagen = ImageDataGenerator(brightness_range=[0.2,1.0],
                          zoom_range=[0.5,1.0],
                          rotation_range=180)
for img in images_name:

    if annotations.loc[img]['Partition'] == 'train':
        img_train_origin = Image.open("/content/drive/MyDrive/ECE1512/images/"
    + img)
        img_train_data = np.asarray(img_train_origin)

        samples = expand_dims(img_train_data, 0)
        it = datagen.flow(samples, batch_size=32)
        batch = it.next()
        image = batch[0].astype('uint8')
        mhist_train_img.append(image)

        img_train_label = annotations.loc[img]['Majority Vote Label']
        mhist_train_label.append(img_train_label)
    if annotations.loc[img]['Partition'] == 'test':
        img_test_origin = Image.open("/content/drive/MyDrive/ECE1512/images/" +
     img)
        img_test_data = np.asarray(img_test_origin)

        samples = expand_dims(img_test_data, 0)
        it = datagen.flow(samples, batch_size=32)
        batch = it.next()
        image = batch[0].astype('uint8')
        mhist_test_img.append(image)

        img_label = annotations.loc[img]['Majority Vote Label']
        mhist_test_label.append(img_label)

mhist_train_img = np.array(mhist_train_img)
mhist_train_label = np.array(mhist_train_label)
mhist_test_img = np.array(mhist_test_img)
mhist_test_label = np.array(mhist_test_label)
mhist_train_img = mhist_train_img/255
mhist_test_img = mhist_test_img/255

le = preprocessing.LabelEncoder()
le.fit(mhist_train_label)
mhist_train_label_le = le.transform(mhist_train_label)
mhist_test_label_le = le.transform(mhist_test_label)

# Build  teacher.
def teacher_initial():
  Res = tf.keras.applications.resnet_v2.ResNet50V2(include_top=False,
    input_shape=(224, 224, 3))
  #Res.trainable = False
  for layer in Res.layers[:185]:
    layer.trainable = False
  for layer in Res.layers[-5:]:
    layer.trainable = True

  average_tea = tf.keras.layers.GlobalAveragePooling2D()(Res.output)
  dense_tea = tf.keras.layers.Dense(2, activation = 'softmax')(average_tea)
  teacher_model = Model(inputs=Res.input, outputs=dense_tea)
  return teacher_model

# Build student.
def student_initial():
  Mob = tf.keras.applications.mobilenet_v2.MobileNetV2(include_top=False,
    input_shape=(224, 224, 3))
  Mob.trainable = False

  average_stu = tf.keras.layers.GlobalAveragePooling2D()(Mob.output)
  dense_stu = tf.keras.layers.Dense(2, activation = 'softmax')(average_stu)
  student_model = Model(inputs=Mob.input, outputs=dense_stu)
  return student_model

teacher_model = teacher_initial()
student_model = student_initial()

teacher_model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(),
    metrics=[tf.keras.metrics.SparseCategoricalAccuracy()],
)

# Train teacher on data.
```

```
teacher_model.fit(mhist_train_img, mhist_train_label_le, epochs=25)

tea_f1_score = f1_score(mhist_test_label_le, np.argmax(teacher_model.predict(
    mhist_test_img), axis=1))
print('Teachers f1 score is',tea_f1_score)

tea_auc = roc_auc_score(mhist_test_label_le, np.argmax(teacher_model.predict(
    mhist_test_img), axis=1))
print('Teachers auc is',tea_auc)

def distillation_loss(teacher_logits: tf.Tensor, student_logits: tf.Tensor,
                      temperature: Union[float, tf.Tensor]):
  soft_targets = tf.nn.softmax(teacher_logits / temperature)
  return tf.reduce_mean(
      tf.nn.softmax_cross_entropy_with_logits(soft_targets, student_logits /
    temperature)) * temperature ** 2

class KD(tf.keras.Model):
    def __init__(self, student, teacher):
        super(KD, self).__init__()
        self.teacher = teacher
        self.student = student

    def compile(self,optimizer,metrics,student_loss_fn,alpha,temperature):
        super(KD, self).compile(optimizer=optimizer, metrics=metrics)
        self.student_loss_fn = student_loss_fn
        self.alpha = alpha
        self.temperature = temperature

    def train_step(self, data):
        x, y = data

        teacher_predictions = self.teacher(x, training=False)
        with tf.GradientTape() as tape:
            student_predictions = self.student(x, training=True)
            student_loss = self.student_loss_fn(y, student_predictions)
            dis_loss = distillation_loss(
                tf.nn.softmax(teacher_predictions / self.temperature, axis=1),
                tf.nn.softmax(student_predictions / self.temperature, axis=1),
                temperature = self.temperature
            )
            loss_value = self.alpha * student_loss + (1 - self.alpha) *
    dis_loss

        # Compute gradients
        gradients = tape.gradient(loss_value, self.student.trainable_variables)
        # Update weights
        self.optimizer.apply_gradients(zip(gradients, self.student.
    trainable_variables))
        # Update the metrics configured in `compile()`.
        self.compiled_metrics.update_state(y, student_predictions)
        # Return a dict of performance
        results = {m.name: m.result() for m in self.metrics}
        results.update(
            {"student_loss": student_loss, "distillation_loss": dis_loss}
        )
        return results

    def test_step(self, data):
        # Unpack the data
        x, y = data
        # Compute predictions
        y_prediction = self.student(x, training=False)
        # Calculate the loss
        student_loss = self.student_loss_fn(y, y_prediction)
        # Update the metrics.
        self.compiled_metrics.update_state(y, y_prediction)
        # Return a dict of performance
        results = {m.name: m.result() for m in self.metrics}
        results.update({"student_loss": student_loss})
        # return results
        return y_prediction

student_model = student_initial()

distiller = KD(student=student_model, teacher=teacher_model)
distiller.compile(
      optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
      metrics=[tf.keras.metrics.SparseCategoricalAccuracy()],
      student_loss_fn=tf.keras.losses.SparseCategoricalCrossentropy(),
      alpha=0.5,
      temperature = 1,
  )

distiller.fit(mhist_train_img, mhist_train_label_le, epochs=25)

stu_f1_score = f1_score(mhist_test_label_le, np.argmax(student_model.predict(
    mhist_test_img), axis=1))
print('Student f1 score is',stu_f1_score)

stu_auc = roc_auc_score(mhist_test_label_le, np.argmax(student_model.predict(
    mhist_test_img), axis=1))
print('Student auc is',stu_auc)

# Hyperparameter tuning
T = [1, 2, 4, 16, 32, 64]
student_f1_list = []
student_auc_list = []
for temp in T:
  student_model = student_initial()
  distiller = KD(student=student_model, teacher=teacher_model)
  distiller.compile(
      optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
      metrics=[tf.keras.metrics.SparseCategoricalAccuracy()],
      student_loss_fn=tf.keras.losses.SparseCategoricalCrossentropy(),
      alpha=0.5,
      temperature = temp,
  )
  distiller.fit(mhist_train_img, mhist_train_label_le, epochs=25)

  stu_f1_score = f1_score(mhist_test_label_le, np.argmax(student_model.predict(
    mhist_test_img), axis=1))
  stu_auc = roc_auc_score(mhist_test_label_le, np.argmax(student_model.predict(
    mhist_test_img), axis=1))

  student_f1_list.append(stu_f1_score)
  student_auc_list.append(stu_auc)

  print(temp,'Temperature done')
print('student_f1_list is',student_f1_list)
print('student_auc_list is',student_auc_list)
plt.plot(T, student_f1_list, label='f1')
plt.plot(T, student_auc_list, label='auc')
plt.xlabel("Temperature")
plt.ylabel("F1 score and AUC score")
plt.title("Distillation Performace vs. Temperature Hyperparameters")
plt.legend()
plt.show()

alpha_list = [0.1,0.3,0.5,0.7,0.9]
student_f1_list = []
student_auc_list = []
for alpha in alpha_list:
  student_model = student_initial()
  distiller = KD(student=student_model, teacher=teacher_model)
  distiller.compile(
      optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
      metrics=[tf.keras.metrics.SparseCategoricalAccuracy()],
      student_loss_fn=tf.keras.losses.SparseCategoricalCrossentropy(),
      alpha=alpha,
      temperature = 2,
  )
  distiller.fit(mhist_train_img, mhist_train_label_le, epochs=25)

  stu_f1_score = f1_score(mhist_test_label_le, np.argmax(student_model.predict(
    mhist_test_img), axis=1))
  stu_auc = roc_auc_score(mhist_test_label_le, np.argmax(student_model.predict(
    mhist_test_img), axis=1))

  student_f1_list.append(stu_f1_score)
  student_auc_list.append(stu_auc)

  print(alpha,'alpha done')
print('student_f1_list is',student_f1_list)
print('student_auc_list is',student_auc_list)
plt.plot(alpha_list, student_f1_list, label='f1')
plt.plot(alpha_list, student_auc_list, label='auc')
plt.xlabel("alpha")
plt.ylabel("F1 score and AUC score")
plt.title("Distillation Performace vs. Task Balance Hyperparameters")
plt.legend()
plt.show()

# Train student from scratch
student_model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(),
    metrics=[tf.keras.metrics.SparseCategoricalAccuracy()],
)

# Train student from scratch
student_model.fit(mhist_train_img, mhist_train_label_le, steps_per_epoch=68,
    epochs=25)

stu_f1_score = f1_score(mhist_test_label_le, np.argmax(student_model.predict(
    mhist_test_img), axis=1))
print('Student f1 score is',stu_f1_score)

stu_auc = roc_auc_score(mhist_test_label_le, np.argmax(student_model.predict(
    mhist_test_img), axis=1))
print('Student auc is',stu_auc)

# Flops
teacherflops = get_flops(teacher_model)
studentflops = get_flops(student_model)
print('flops for teacher ResNet model is',teacherflops)
print('flops for student MobileNet model is',studentflops)

# TA Model
def ta_initial():
  Res = tf.keras.applications.resnet_v2.ResNet50V2(include_top=False,
    input_shape=(224, 224, 3))
  Res.trainable = False

  average_tea = tf.keras.layers.GlobalAveragePooling2D()(Res.output)
  dense_tea = tf.keras.layers.Dense(2, activation = 'softmax')(average_tea)
  ta_model = Model(inputs=Res.input, outputs=dense_tea)
  return ta_model

def ta_initial_mob():
  Mob = tf.keras.applications.mobilenet_v2.MobileNetV2(include_top=False,
    input_shape=(224, 224, 3))
  Mob.trainable = False
  for layer in Mob.layers[:149]:
    layer.trainable = False
  for layer in Mob.layers[-3:]:
    layer.trainable = True

  average_tea = tf.keras.layers.GlobalAveragePooling2D()(Mob.output)
  dense_tea = tf.keras.layers.Dense(2, activation = 'softmax')(average_tea)
  ta_model = Model(inputs=Mob.input, outputs=dense_tea)
  return ta_model

ta_model = ta_initial()
distiller = KD(student=ta_model, teacher=teacher_model)
distiller.compile(
      optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
      metrics=[tf.keras.metrics.SparseCategoricalAccuracy()],
      student_loss_fn=tf.keras.losses.SparseCategoricalCrossentropy(),
      alpha=0.5,
      temperature = 1,
  )

distiller.fit(mhist_train_img, mhist_train_label_le, epochs=25)

student_model = student_initial()
distiller = KD(student=student_model, teacher=ta_model)
distiller.compile(
      optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
      metrics=[tf.keras.metrics.SparseCategoricalAccuracy()],
      student_loss_fn=tf.keras.losses.SparseCategoricalCrossentropy(),
      alpha=0.5,
      temperature = 1,
  )

distiller.fit(mhist_train_img, mhist_train_label_le, epochs=25)

stu_f1_score = f1_score(mhist_test_label_le, np.argmax(student_model.predict(
    mhist_test_img), axis=1))
print('Student f1 score is',stu_f1_score)

stu_auc = roc_auc_score(mhist_test_label_le, np.argmax(student_model.predict(
    mhist_test_img), axis=1))
print('Student auc is',stu_auc)
```

Listing 10: Task 2 Codes