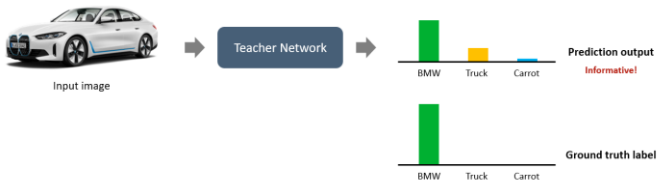
ASSIGNMENT REPORT

I. Vanilla Knowledge Distillation

1. Algorithm explanation

The pioneer work of knowledge distillation was proposed by Hinton at el. [1] which transfers the knowledge from the cumbersome (teacher) model to the small (student) model. After training, the small model with more effective computation can achieve a comparable performance to the larger one. The idea to transfer knowledge from the teacher model is using the output distribution of class probabilities of the teacher as soft targets to train the student model:

Previously, in classification, we only consider the groundtruth label of training data (in one-hot vector) to supervise the prediction outputs of student model. However, the only information we get from such one-hot vector [0, 1, 0, …, 0] is which class the current image should belong to. We regard such labels as hard targets. Meanwhile, the soft targets, which are the outputs of softmax of teacher logits are more informative. For example, from the prediction output of the teacher below, we can understand that there is a small chance a "BMW car" is mistaken for a "truck" (orange < green) while the possibility a "BMW car" is mistaken for a "carrot" (in blue) is many times smaller.



The paper [1] proposes a new softmax function with temperature parameter T:

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

q as the class probabilities output and z as the output logit of the model. For traditional softmax function (T=1), the probability of the correct class is usually much higher while the rest are close to zero. For this reason, we may not obtain a very informative output. If we increase the value of T, the output distribution will become softer, which will give us more information from the teacher's model.

According to [1], if we use both groundtruth labels (hard targets) and teacher's output prediction (soft targets), we can achieve the best performance.

The overall loss of vanilla knowledge distillation:

$$\mathbf{L = \alpha \times CrossEntropy(\sigma(z_s, T = 1), label) + \beta \times KLDivergence(\sigma(z_s, T = t), \sigma(z_t, T = t)) \times T^2}$$

α, β : loss weights (usually choose β larger than α). $CrossEntropy()$: classification loss. $KLDivergence()$: Use Kullback–Leibler divergence loss as metric of distillation loss, which will encourage the student to produce similar output probability distribution with the teacher model during training. σ : softmax function. z_s, z_t : student logits and teacher logits respectively.

Because "the magnitudes of the gradients produced by the soft targets scale as 1/ T^2" [1]. In case we use both hard and soft targets here, we should multiply the KL divergence by T^2.0

2. Training code

My student model is trained on Kaggle. The code is quite long, I will not paste the corresponding code while explaining them here (please refer to the submitted ipynb notebook).

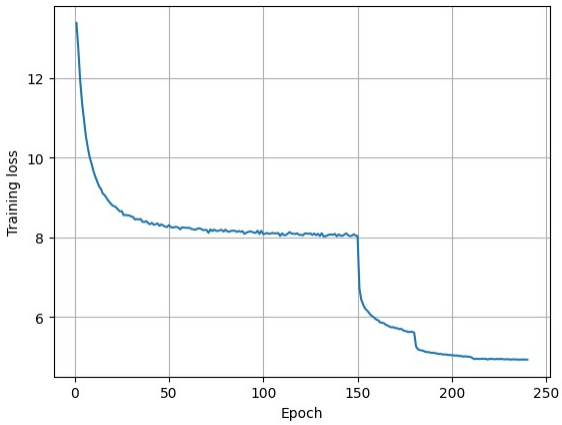
The KDLoss() class defines the knowledge distillation loss term mentioned above, where given the temperature (T=4.0 in this work), it will take in the output logits of student and teacher model (shape of batch_size x class_number) and return the KD loss of the mini batch:

We define necessary parameters and training function in **KDTrainer() class**: Batch size = 64. Number of epoch = 240. Initial learning rate = 0.05 and will be decreased by multiplying by 0.1 at epoch number 150, 180, 210. SGD optimizer is used with momentum = 0.9 and weight decay = 5e-4.

$\alpha = 1.0, \beta = 0.9, T = 4.0$ (refer to the loss function described above)

Regarding **train_student() function**, self.teacher is our large model (ResNet32x4) and self.student is our small model (ResNet8x4). Since we only use the teacher to produce soft targets, we set the evaluation mode and disable the gradient computation of the teacher model. We iterate through each batch of images and calculate the loss for each time. The train_loss variable saves the average loss for the whole training set we used. I used train_loss_arr to save the history of training losses for visualizing the learning curve at the end of training phase.

3. Results



Overall, the training loss decreasing overtime proved that my vanilla KD has been implemented properly. Choosing suitable learning rate is very important. Towards the end of the training phase, we need smaller learning rates so that the model can converge to smaller loss.

```
trainer.load_student_checkpoint(CKPT_PATH)
accuracy = trainer.evaluate_student()

print(f"Student model test accuracy: {accuracy:.3f} %")
print(f"Is above threshold performance? {accuracy > 72.2}")
```

```
Student model test accuracy: 73.110 %
Is above threshold performance? True
```

II. Improved Knowledge Distillation

1. Algorithm explanation

In the vanilla KD algorithm, we transferred knowledge from teacher to student model by exploring the relation between different classes for each input sample, which could be regarded as instance level or inter-class relation. It is worth to exploit that the prediction distribution of multiple samples for each class is also informative too and

can be regarded as intra-relations. For example, given 3 input samples of a car, a truck and carrots, the score corresponding to the carrot class in the carrot sample should be higher than in the car and truck samples.

Considering the fact that there is a large discrepancy between the teacher (large) and the student (small) model, the usage of KL divergence to force the student outputs match exactly with the teacher outputs seems to be too demanding. Also, what we really care for while transferring the knowledge may be the "relations" rather than the "values". Therefore, using a more relaxed metric that can also represents the "relations" we want to transfer is a good idea.

Overall, we have the final loss for new KD algorithm as below:

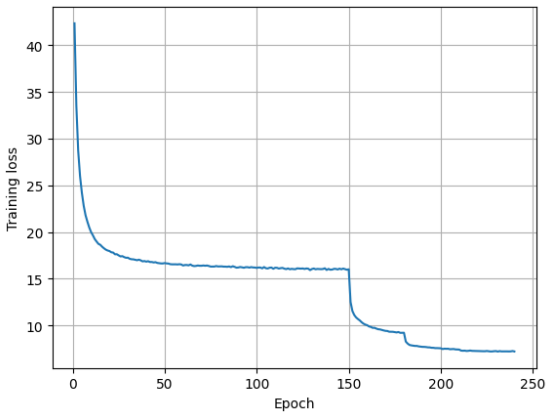
$$\mathbf{L} = \mathbf{0.1} \times \mathbf{CrossEntropy}(\boldsymbol{\sigma}(\mathbf{z_s}, \mathbf{T} = \mathbf{1}), \mathbf{label}) + \mathbf{0.9} \times (\boldsymbol{\alpha} \times \mathbf{L_{inter}}(\mathbf{p_s}, \mathbf{p_t}) + \boldsymbol{\beta} \times \mathbf{L_{intra}}(\mathbf{p_s}.\mathbf{transpose}(), \mathbf{p_t}.\mathbf{transpose}()) \times \mathbf{T^2}$$

Where p_s, p_t : output of softmax with temperature T. L_{inter}, L_{intra} : defined by the distance metric in [2]

2. Training code

The training code is the same as previous experiments, with same parameters except the loss function we defined in NewLoss() class, which follows the above formula. Here, the inter-loss and intra-loss weights, which are $\boldsymbol{\alpha}, \boldsymbol{\beta} = \mathbf{2}$.

3. Results




```
trainer.load_student_checkpoint(CKPT_PATH)
accuracy = trainer.evaluate_student()

print(f"Improved student model test accuracy: {accuracy:.3f} %")
```

```
Improved student model test accuracy: 76.100 %
```

III. References

[1] G. Hinton et al. Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531, 2015.

[2] T. Huang et al. Knowledge Distillation from A Stronger Teacher. NeurIPS, 2022.

[3] https://intellabs.github.io/distiller/knowledge_distillation.html