Knowledge Distillation

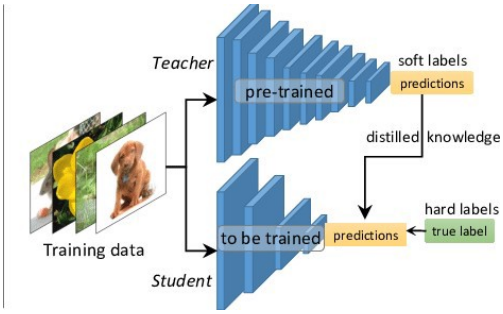
What is Knowledge Distillation -

It is a type of model compression technique which is used to train a smaller network (whose main purpose is for deployment so needs faster prediction) which has lesser parameters to train using a fairly large complex model, this is done by defining the custom loss function and unique technique which is discussed below.

Why do we need Knowledge Distillation ?

In cases where we need to deploy the model, we can't wait for long enough time for the model to give out results after training, just because the model is complex and have complex parameters. Techniques like Transfer learning don't work because they too need to have large layers similar to the parent model and has complex weights. So KD is a technique that provides equal or higher performance for a smaller model whose knowledge is distilled from a fairly large complex model.

Structure of the Knowledge Distillation technique -



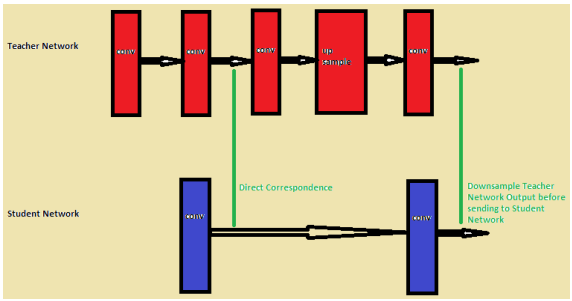
There are 2 components of the structure -

- Teacher Model - This is the complex model that is trained on the entire dataset, this model has high accuracy and high number of parameters to train
- Student Model - This is the simpler model which is trained using the Teacher model and the structure of this model can be totally different from the teacher one.

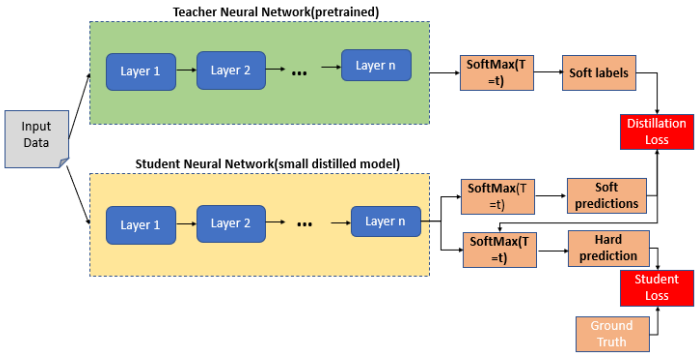
The Knowledge Distillation tries to reduce the KL divergence between the two models.

Steps for Knowledge Distillation

- Train the Teacher Network - This complex is first trained over GPU and entire dataset for the best accuracy possible. Here note that this model is trained using the modified Softmax function (containing the Temperature)
- Establish Relation between Teacher and Student - A correspondence needs to be established between the intermediate outputs of the student network and the teacher network. This can be done by directly passing the output layers of the teacher network to the student network or some kind of augmentation can be done.



- Forward Pass through the Teacher network - Pass the data through the teacher network to get all intermediate outputs and then apply data augmentation to the same.
- Back-propagation of the student Model - The data is passed through the student model and loss is calculated using both the predictions of the soft target of the student model and the parent model which is combined using a hyper parameter Alpha. This loss is used to back-propagate the network and let the model learn the exact outcome of the teacher model.



What are Soft-Targets ?

Soft targets use the logits, the inputs to the final softmax rather than the softmax's probabilities as the targets for learning the small model.

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

Note this is a modification of the softmax function, that includes the Temperature hyperparameter.

Soft targets have high entropy providing more information than one hot encoded hard target. Soft targets have less variance in the gradient between training cases allowing student networks to be trained on much less data than the Teacher using a much higher learning rate i. e. it says which 2 looks like 3's and which looks like 7's.

The lesser variance allows the smaller Student model to be trained on much smaller data than the original cumbersome model and with a much higher learning rate

Temperature -

We know that the knowledge of the Student model comes from the Teacher model and if the latter model is confident about the prediction then, the outcome would be similar to the true - One Hot encoding given. So a technique was needed to reduce this confident value predicted by the Teacher model (Note this should be done before computing the softmax). So before we compute the softmax, we divide the scores of each class by a constant known as the temperature.Higher temperatures produce softer probabilities which provides a stronger signal to the student for learning from the teacher.

Code for the Distillation class needed

Note - This code is taken from the official documentation of Keras.

```
# Defining the Distiller class that will override the keras.Model
class Distiller(keras.Model):

  # Initializing the class
    def __init__(self, student, teacher):
        super(Distiller, self).__init__()
        self.teacher = teacher
        self.student = student

    # Overriding the compile function of the keras.Model
    def compile(
        self,
        optimizer,
        metrics,
        student_loss_fn, # Student loss function
        distillation_loss_fn, # Distillation loss function
        alpha=0.1, # How much weightage must be given to the distiller loss function
        temperature=3,
    ):
        #Initializing the variables
        super(Distiller, self).compile(optimizer=optimizer, metrics=metrics)
        self.student_loss_fn = student_loss_fn
        self.distillation_loss_fn = distillation_loss_fn
        self.alpha = alpha
        self.temperature = temperature

    #Overriding the train_step function of keras.Model
    def train_step(self, data):

        x, y = data

        # Forward pass of teacher
        teacher_predictions = self.teacher(x, training=False)

        # Here Gradient Tape is used for calculating the gradients of any variabe in the graph
        with tf.GradientTape() as tape:
```

```
            # Forward pass of student
            student_predictions = self.student(x, training=True)

            # Compute losses
            student_loss = self.student_loss_fn(y, student_predictions)

            # The Distillation Loss function
            # Here Temperature is the constant divided so as to create a smoother prediction instead of a hard prediction
            distillation_loss = self.distillation_loss_fn(
                tf.nn.softmax(teacher_predictions / self.temperature, axis=1),
                tf.nn.softmax(student_predictions / self.temperature, axis=1),
            )
            # Use of the Alpha as a weighted average
            loss = self.alpha * student_loss + (1 - self.alpha) * distillation_loss

        # Compute gradients
        trainable_vars = self.student.trainable_variables
        gradients = tape.gradient(loss, trainable_vars)

        # Update weights
        self.optimizer.apply_gradients(zip(gradients, trainable_vars))

        # 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": distillation_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
```

Here we over-ride 3 functions of the keras.Model class - Compile, train_step, test_step

After which the code can be run as follows -

```
#compile distiller
distiller.compile(optimizer=keras.optimizers.Adam(),
                  metrics=[keras.metrics.SparseCategoricalAccuracy()],
                  student_loss_fn=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                  distillation_loss_fn=keras.losses.KLDivergence(),
                  alpha=0.3,
                  temperature=7)

# Distill teacher to student
distiller.fit(x_train, y_train, epochs=5)
# Evaluate student on test dataset
distiller.evaluate(x_test, y_test)
```

If we run these codes we see that the Model where knowledge was distilled from a larger model had an accuracy of 97% whereas the one trained from scratch has 59%.

Conclusion -

So knowledge distillation is a better substitute of Transfer Learning as the smaller model requires less parameters and achieves similar accuracies to the teacher model. This technique of model compression can be used in cases where ML models need to be deployed as they are both space and time efficient giving about the same accuracy.