DVCLive and Checkpoints in Deep Learning

When working on Deep Learning scenarios, we can generally expect our models to take a long time to train. In such a case, it will be useful to get intermittent updates for our training process. Imagine training a model for an entire day, only to find out it's not performing well at the very end. We'd rather find out that our model's performance isn't improving during our training.

DVC has a feature specifically designed for these deep learning scenarios. We can define checkpoints in our training stage to create a snapshot of our model at that point in time. These checkpoints usually correspond to training epochs.

DVC will save our model weights at each checkpoint and we can log all metrics and results. Code and data changes are also tracked, and a new commit is created for each checkpoint. This means that an experiment can contain multiple checkpoints, and we can refer back to specific checkpoints at any time. This can be useful when we want to pinpoint at which checkpoint metrics start diverging, for example. We could simply revert back to an optimal checkpoint, and continue our experimentation from there.

To register checkpoints in our training stage, we use a library called DVCLive. It integrates with popular machine learning frameworks such as Keras and Tensorflow and lets us automatically store metrics through a callback with DVCLive.

So how do we use DVCLive and set up our checkpoints? First, we import the library into our training script. We then use `live.log()` to write the metrics we want to log to our output file. `live.next_step()` indicates to DVC where our next epoch starts.

```
# train.py

from dvclive import Live

live = Live("training_metrics")

for epoch in range(NUM_EPOCHS):
    train_model(...)
    metrics = evaluate_model(...)

    for metric_name, value in metrics.items():
        live.log(metric_name, value)

    live.next_step()
```

Here, `training_metrics` is the file to which DVCLive will log our metrics. We then specify this file as a metrics file in our DVC pipeline, as we explored earlier in module 5:

```
stages:
  train:
    cmd: python train.py
    deps:
      - train.py
    metrics:
      - training_metrics.json:
          cache: false
    plots:
      - training_metrics/scalars:
          cache: false
```

We can visualize these metrics with `dvc plots`. We can also automatically generate an HTML report in our training script with `live.make_report()`:

```
# train.py

from dvclive import Live

live = Live("training_metrics")

for epoch in range(NUM_EPOCHS):
    train_model(...)
    metrics = evaluate_model(...)

    for metric_name, value in metrics.items():
        live.log(metric_name, value)

    live.make_report()
    live.next_step()
```

In this scenario, DVCLive will automatically create a `report.html` file in the same directory as our metrics file. This file will be updated at the end of each epoch, allowing us to visualize our progress over time.