**Quantum Transfer Learning (Logbook)**

(Rithwik Bhardwaj, Saniya Nazeer, Arushi Tibrewal, Noah Kanter)

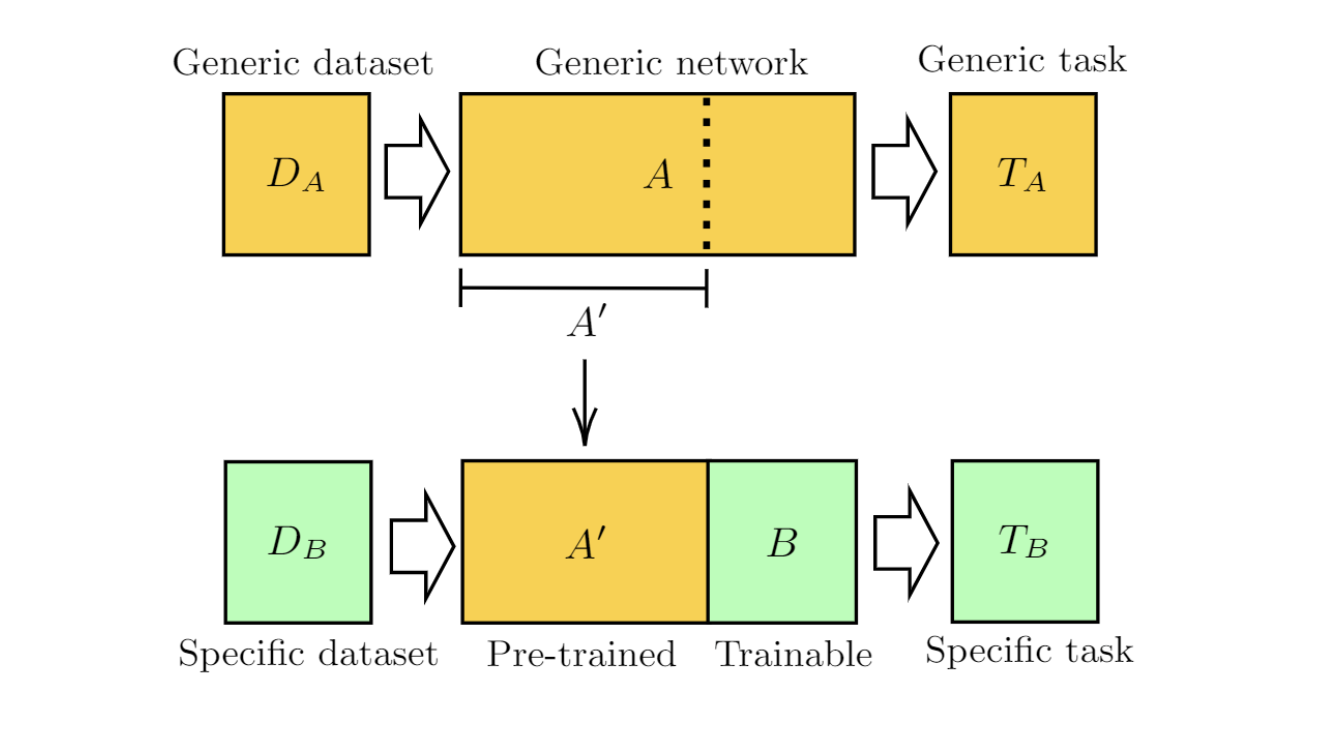
**Overview**

In this project we apply a machine learning method, known as *transfer learning*, to an image classifier based on a hybrid classical-quantum network.

**Introduction**

Transfer learning is a well-established technique for training artificial neural networks, which is based on the general intuition that if a pre-trained network is good at solving a given problem, then, with just a bit of additional training, it can be used to also solve a related problem.

This idea can be formalized in terms of two abstract networks A and B, independently from their quantum or classical physical nature.



**Transfer Learning**

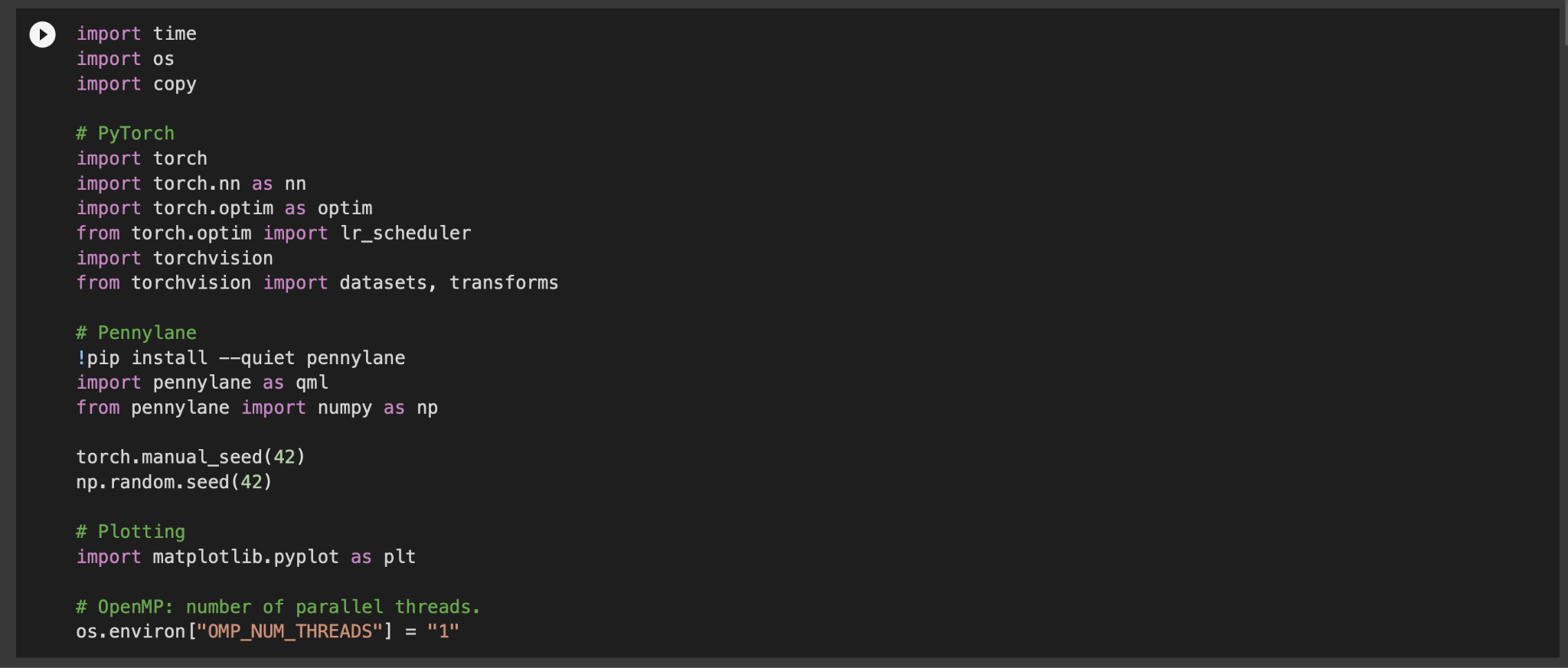
Transfer learning is a machine learning technique in which a pretrained model (source model) on a large dataset is used as a starting point for training a new model for a related task. We take a network that has been pre-trained on a dataset for a given task. Then, we cut off/remove some of the final layers and then we connect a new trainable network at the end of the first network that we had. We need to keep the weights of our first network constant (i.e., we freeze the layers in the first network), and train the final block for a new task of our interest.

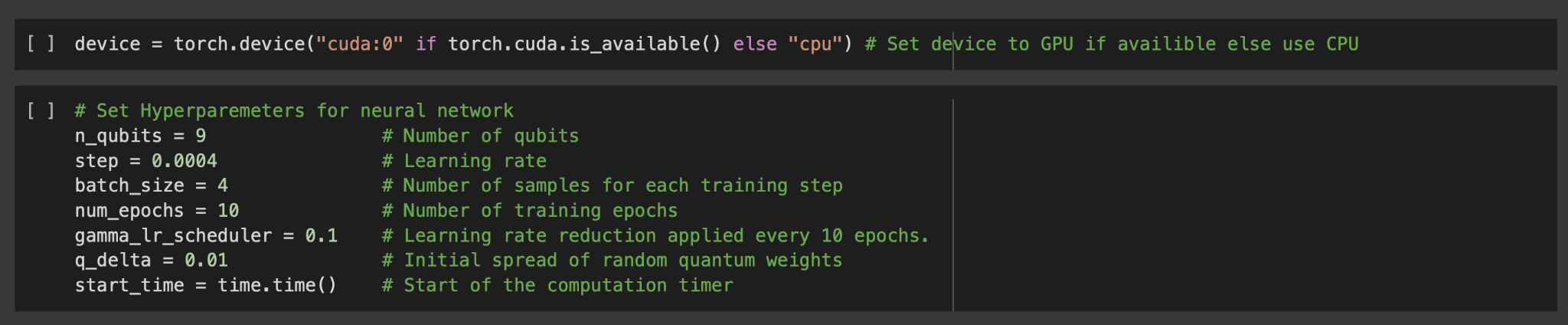
**Classical to Quantum Transfer Learning**

We use ResNet18 as our pre-trained network (a deep residual neural network introduced by Microsoft) which is pre-trained on our images dataset. ResNet takes in a cropped 244x244 normalized image. Due to technical limitations of quantum circuits we decided to use a classical convolutional neural network (CNN) to reduce the size of the input data. Using the ResNet18 pretrained network (with frozen weights) we reduced the input image from 244x244 resolution to a vector of size 512.

After loading ResNet, we used three trainable quantum layers to classify images as images of bees or ants. Since we freeze the classical layers (this means that the weights cannot be modified further so the information contained in them does not change), only the quantum layers can learn the image features and thus, the network's accuracy depends on the quantum layers.

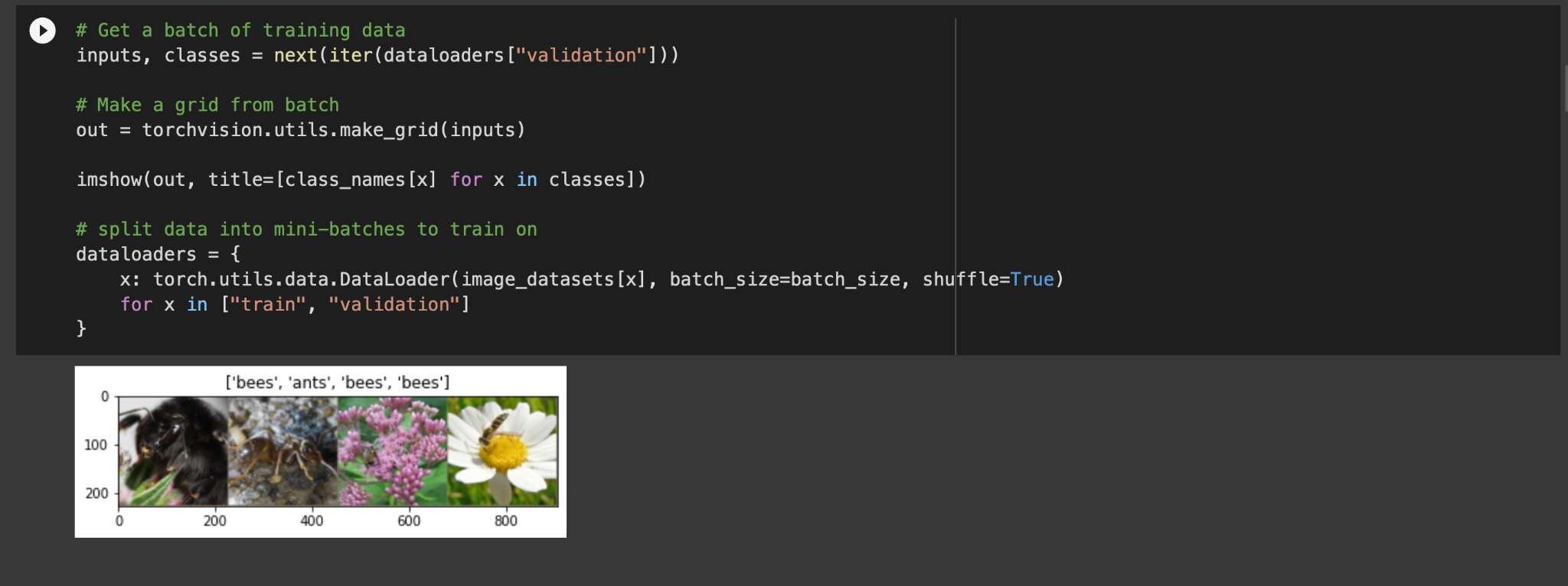
**Code step-by-step walkthrough**

First we import the necessary modules and load our dataset. 

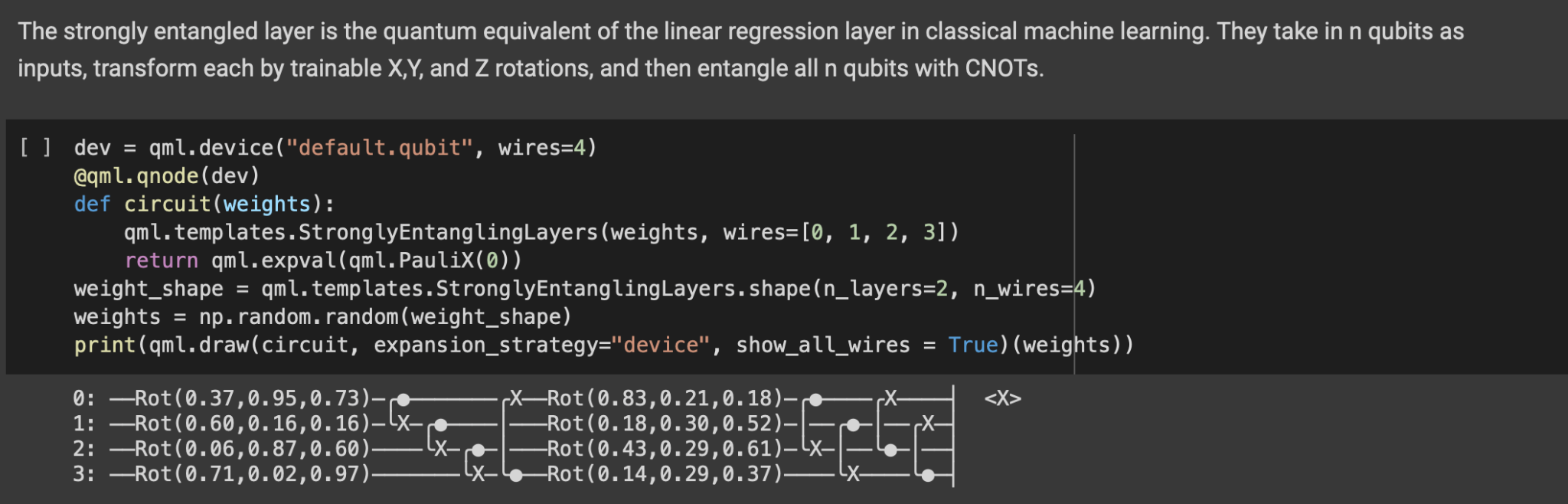
Then, we define our quantum device that we’ll use to build our quantum layers and we set the hyperparameters for our neural network. 

Since Resnet18 is a pretrained network it has a set input size and shape so we transform and normalize the training/testing data.

Then we take a batch of the training data and make a grid from the batch and we split the data into mini-batches to train our model on.



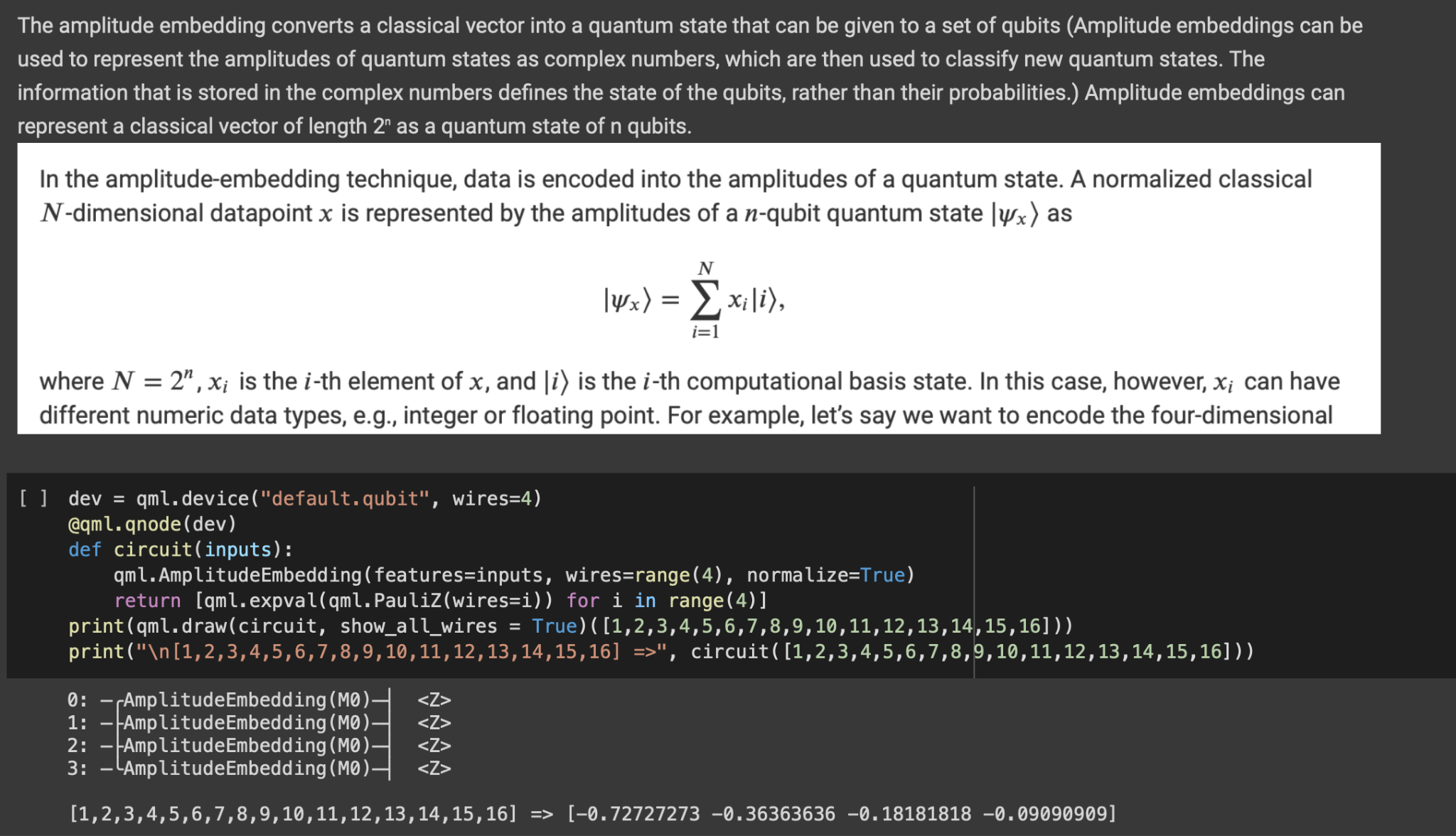
StronglyEntanglingLayers given an input of n qubits perform a X,Y, and Z rotation of trainable weight on each n qubit before using the CNOT gate to entangle the n qubits.



**Amplitude embeddings** are a technique used to encode classical data into the amplitudes of a quantum state. This enables quantum operations to be performed on the classical data, allowing us to use quantum computers to process and analyze classical data.

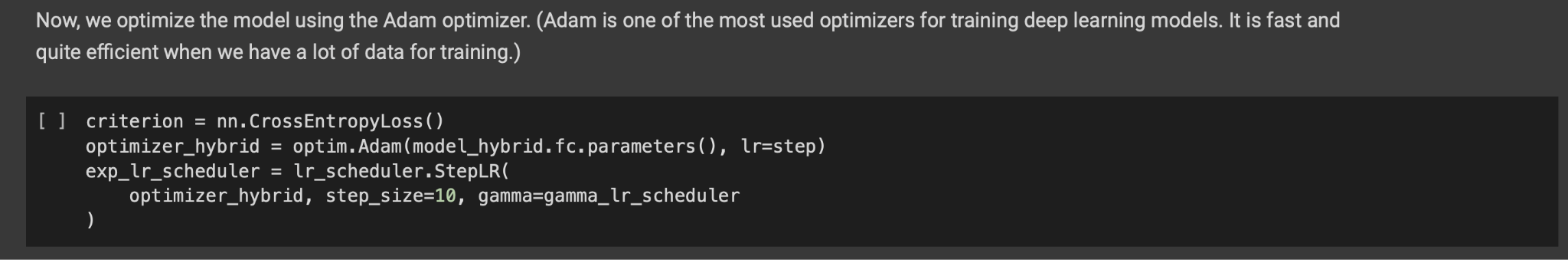
Amplitude embeddings work by mapping a classical dataset to the amplitudes of a quantum state. Each amplitude represents a data point in the dataset, and the magnitude of the amplitude corresponds to the weight or importance of the data point in the overall dataset.

Once the classical data is encoded into the quantum state, we can perform quantum operations on it.



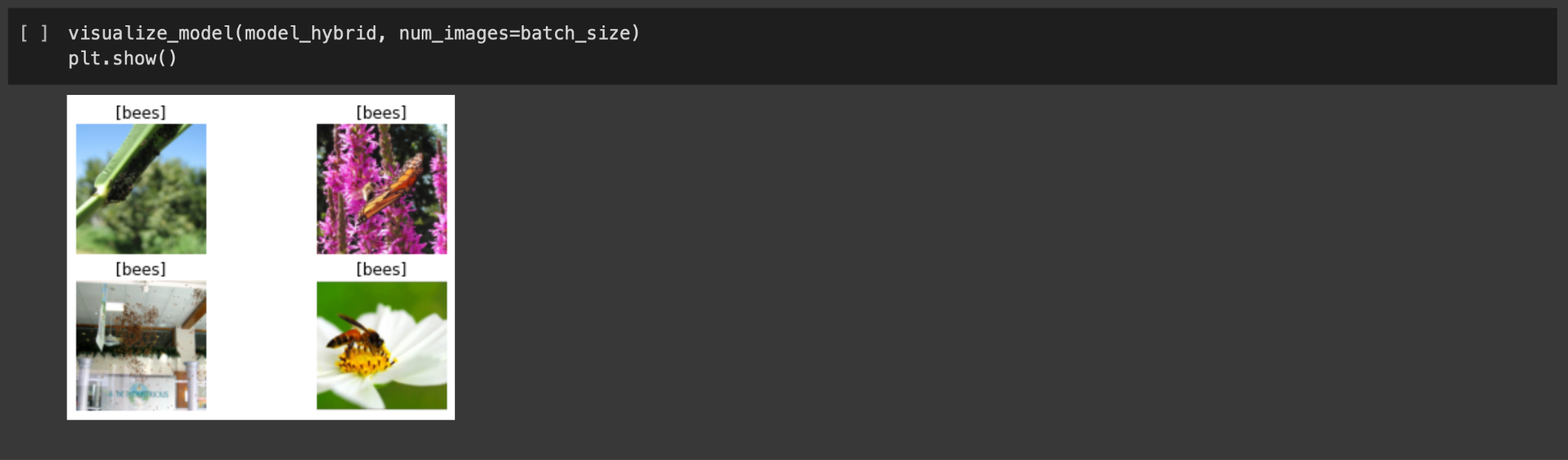
After that, we load ResNet, and use three trainable quantum layers to classify images as images of bees or ants. We also freeze the classical layers so that the weights cannot be modified further and only the quantum layers can learn the image features and thus, the network's accuracy depends on the quantum layers.

Then we use the Adam optimizer to optimize our model.



And finally we train the model!

After performing a visualization of the model, here’s an example of our output:



**Test with different hyperparameters**

|  |  |
| --- | --- |
| Depth | Trial |
| 2:2:2 | 0.5686 |
| 4:4:4 | 0.5686 |
| 6:6:6 | 0.6405 |
| 8:8:8 | 0.6601 |
| 10:10:10 | 0.6797 |
| 12:12:12 | 0.7451 |
| 15:15:15 | 0.7712 |
| 20:20:20 | 0.8627 |
| 15:10:8 | 0.6601 |
| 20:15:10 | 0.8105 |
| 20:18:15 | 0.8039 |
| 25:20:15 | 0.8758 |

|  |  |
| --- | --- |
| Depth | Trial |
| 2:2:2 | 0.5686 |
| 8:8:8 | 0.6601 |
| 15:15:15 | 0.7712 |
| 20:20:20 | 0.8627 |
| 25:20:15 | 0.8758 |

Methodology Info

* Trial = best accuracy on validation data after 20 epochs (may not be last epoch)
* Learning rate = 0.0004
* Batch size = 4
* Depth= (depth of quantum layer 1):(depth of quantum layer 2):(depth of quantum layer 3)
* Numpy/Pytorch Seed = 42

**References**

Our GoogleColab link: <https://colab.research.google.com/drive/1fb3KLoYkI0EoFWnoOBAYHRCs_eUgPqsr?authuser=2>

Pennylane Documentation: <https://docs.pennylane.ai/en/stable/>

Pennylane Quantum Transfer Learning: <https://pennylane.ai/qml/demos/tutorial_quantum_transfer_learning.html>

(And a special mention to ChatGPT:) )