Homework 3 - Answer

For more details see the docs

The first part of the notebook, just checks that the requisite packages are installed.

We then move onto defining the model

```
[ ]
    class MyModel(nn.Module):
        def __init__(self):
            super(MyModel, self).__init__()

            self.conv1 = nn.Conv2d(in_channels=1, out_channels=2, kernel_size=5, stride=2)
            self.conv2 = nn.Conv2d(in_channels=2, out_channels=3, kernel_size=5, stride=2)

            self.relu = nn.ReLU()

            self.d1 = nn.Linear(48, 48)
            self.d2 = nn.Linear(48, 10)

        def forward(self, x):
            # 32x1x28x28 => 32x32x26x26
            x = self.conv1(x)
            x = self.relu(x)
            x = self.conv2(x)
            x = self.relu(x)

            # flatten => 32 x (32*26*26)
            x = x.flatten(start_dim = 1)

            # 32 x (32*26*26) => 32x128
            x = self.d1(x)
            x = self.relu(x)

            # logits => 32x10
            logits = self.d2(x)

            return logits


    circuit = MyModel()
```

Thanks to Pedro : @pedrocoelho22
who pointed out that the comments are incorrect and should be

```
conv1: 32x1x28x28 => 32x2x12x12,
conv2: 32x2x12x12 => 32x3x4x4,
flatten: 32x3x4x4 => 32 x (3x4x4=48),
d1: 32x48 => 32x48,
d2: 32x48 => 32x10 (edited)
```

```
    shape = [1, 28, 28]
    # After training, export to onnx (network.onnx) and create a data file (input.json)
    x = 0.1*torch.rand(1,*shape, requires_grad=True)

    # Flips the neural net into inference mode
    circuit.eval()

        # Export the model
    torch.onnx.export(circuit,               # model being run
                      x,                     # model input (or a tuple for multiple inputs)
                      model_path,            # where to save the model (can be a file or file-like object)
                      export_params=True,        # store the trained parameter weights inside the model file
                      opset_version=10,          # the ONNX version to export the model to
                      do_constant_folding=True,  # whether to execute constant folding for optimization
                      input_names = ['input'],   # the model's input names
                      output_names = ['output'], # the model's output names
                      dynamic_axes={'input' : {0 : 'batch_size'},    # variable length axes
                                    'output' : {0 : 'batch_size'}})

    data_array = ((x).detach().numpy()).reshape([-1]).tolist()

    data = dict(input_data = [data_array])

        # Serialize data into file:
    json.dump( data, open(data_path, 'w' ))
```

We serialise the inputs for later

```
[ ] cal_path = os.path.join("calibration.json")

    data_array = (torch.rand(20, *shape, requires_grad=True).detach().numpy()).reshape([-1]).tolist()

    data = dict(input_data = [data_array])

    # Serialize data into file:
    json.dump(data, open(cal_path, 'w'))


    ezkl.calibrate_settings(cal_path, model_path, settings_path, "resources")
```

We can now create the circuit and do the rest of gthe setup

```
[ ]  res = ezkl.compile_circuit(model_path, compiled_model_path, settings_path)
     assert res == True
```

```
[ ]  # srs path
     res = ezkl.get_srs( settings_path)
```

```
[ ]  # now generate the witness file

     res = ezkl.gen_witness(data_path, compiled_model_path, witness_path)
     assert os.path.isfile(witness_path)
```

ezkl.setup will give us the proving and verification keys

```
[ ]
     # HERE WE SETUP THE CIRCUIT PARAMS
     # WE GOT KEYS
     # WE GOT CIRCUIT PARAMETERS
     # EVERYTHING ANYONE HAS EVER NEEDED FOR ZK



     res = ezkl.setup(
             compiled_model_path,
             vk_path,
             pk_path,

         )

     assert res == True
     assert os.path.isfile(vk_path)
     assert os.path.isfile(pk_path)
     assert os.path.isfile(settings_path)
```

We can now create the proof

```
[ ]  # GENERATE A PROOF


     proof_path = os.path.join('test.pf')

     res = ezkl.prove(
             witness_path,
             compiled_model_path,
             pk_path,
             proof_path,

             "single",
         )

     print(res)
     assert os.path.isfile(proof_path)
```

and verify it

```
[ ]  # VERIFY IT

      res = ezkl.verify(
              proof_path,
              settings_path,
              vk_path,

          )

      assert res == True
      print("verified")
```

Terminology

[stride]