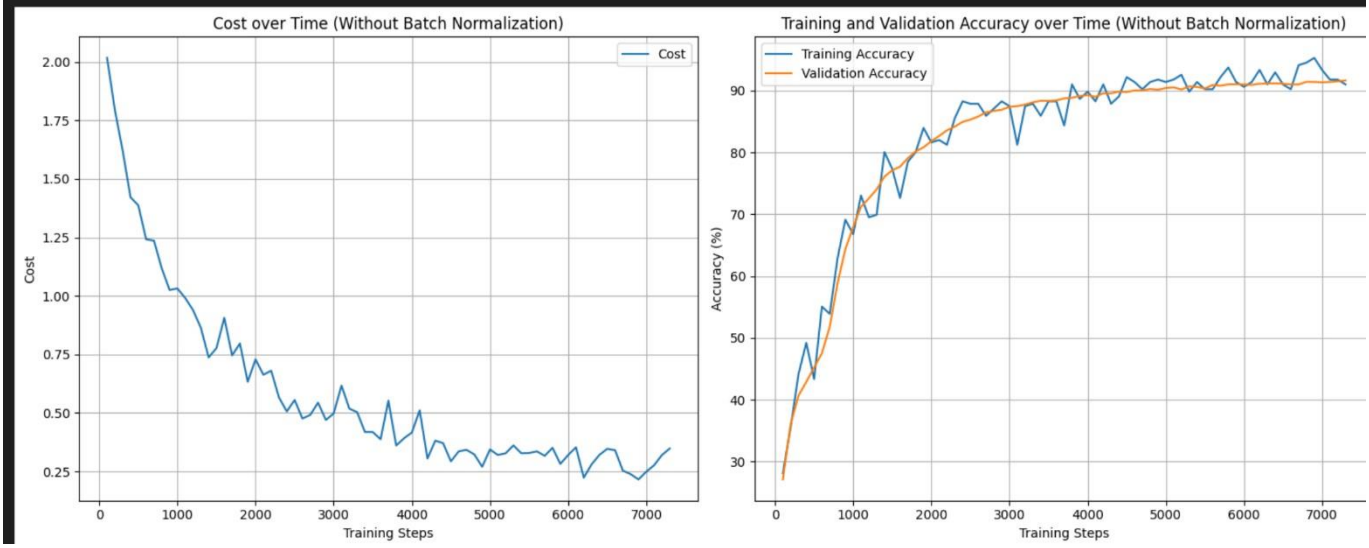
We used a batch size of 256

4. Performance of the neural network on the MNIST dataset without batch normalization

```
Final Training Accuracy: 91.02%
Final Validation Accuracy: 91.66%
Final Test Accuracy: 91.87% , Number of Iterations: 7362
Total Training Time: 64.54 seconds
```

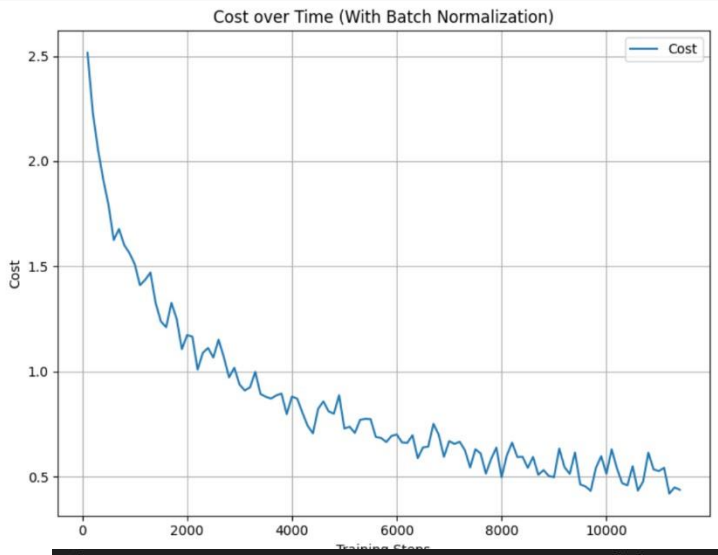


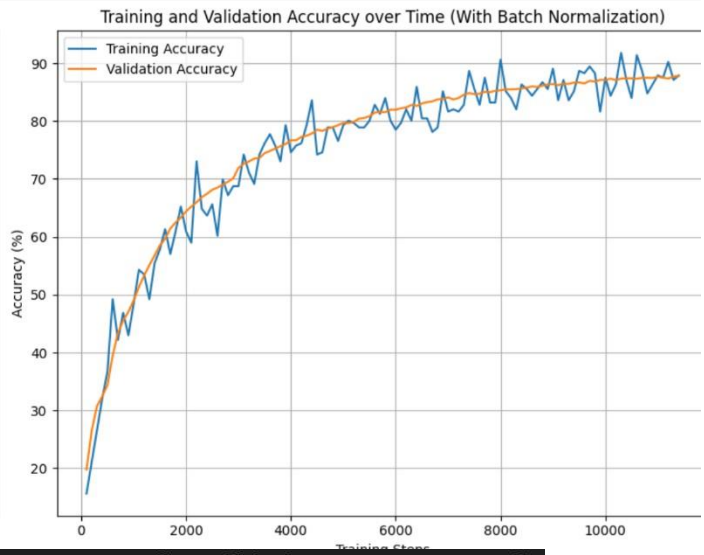

```
After 100 steps: cost = 2.017472505569458 , Training accuracy = 28.12% , Validation accuracy = 27.12%
After 200 steps: cost = 1.7943065166473389 , Training accuracy = 35.55% , Validation accuracy = 36.08%
After 300 steps: cost = 1.6206823587417603 , Training accuracy = 44.14% , Validation accuracy = 40.68%
After 400 steps: cost = 1.4219380617141724 , Training accuracy = 49.22% , Validation accuracy = 42.90%
After 500 steps: cost = 1.3868895769119263 , Training accuracy = 43.36% , Validation accuracy = 45.25%
After 600 steps: cost = 1.241807460784912 , Training accuracy = 55.08% , Validation accuracy = 47.53%
After 700 steps: cost = 1.2362000942230225 , Training accuracy = 53.91% , Validation accuracy = 51.82%
After 800 steps: cost = 1.1176578998565674 , Training accuracy = 62.89% , Validation accuracy = 58.88%
After 900 steps: cost = 1.025376796722412 , Training accuracy = 69.14% , Validation accuracy = 64.38%
After 1000 steps: cost = 1.0319569110870361 , Training accuracy = 66.80% , Validation accuracy = 67.97%
After 1100 steps: cost = 0.9909242391586304 , Training accuracy = 73.05% , Validation accuracy = 71.22%
After 1200 steps: cost = 0.9395040273666382 , Training accuracy = 69.53% , Validation accuracy = 72.58%
```

```
After 6100 steps: cost = 0.3530588150024414 , Training accuracy = 91.41% , Validation accuracy = 90.94%
After 6200 steps: cost = 0.22318917512893677 , Training accuracy = 93.36% , Validation accuracy = 91.12%
After 6300 steps: cost = 0.2800513207912445 , Training accuracy = 91.02% , Validation accuracy = 91.18%
After 6400 steps: cost = 0.32051968574523926 , Training accuracy = 92.97% , Validation accuracy = 91.19%
After 6500 steps: cost = 0.34651613235473633 , Training accuracy = 91.02% , Validation accuracy = 91.14%
After 6600 steps: cost = 0.3407195508480072 , Training accuracy = 90.23% , Validation accuracy = 91.00%
After 6700 steps: cost = 0.2535265386104584 , Training accuracy = 94.14% , Validation accuracy = 91.02%
After 6800 steps: cost = 0.2380819469690323 , Training accuracy = 94.53% , Validation accuracy = 91.43%
After 6900 steps: cost = 0.2156086564064026 , Training accuracy = 95.31% , Validation accuracy = 91.42%
After 7000 steps: cost = 0.24883034825325012 , Training accuracy = 93.36% , Validation accuracy = 91.36%
After 7100 steps: cost = 0.27647864818573 , Training accuracy = 91.80% , Validation accuracy = 91.41%
After 7200 steps: cost = 0.31984034180641174 , Training accuracy = 91.80% , Validation accuracy = 91.53%
After 7300 steps: cost = 0.34791556000709534 , Training accuracy = 91.02% , Validation accuracy = 91.66%
```

5. Performance of the neural network on the MNIST dataset with batch normalization

```
Final Training Accuracy: 87.89%
Final Validation Accuracy: 87.79%
Final Test Accuracy: 87.95% , Number of Iterations: 11460
Total Training Time: 115.59 seconds
```





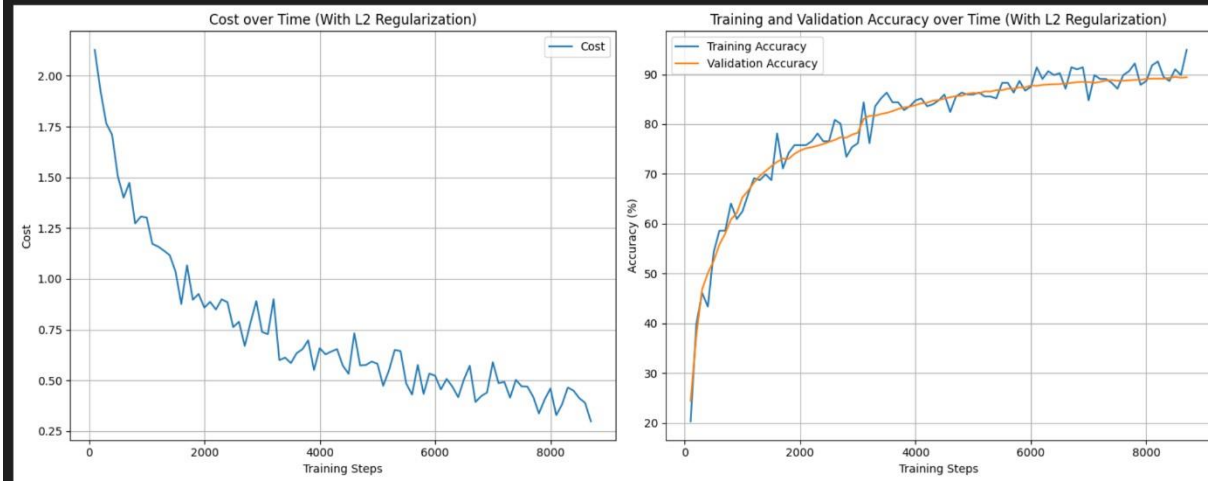

```
After 100 steps: cost = 2.5153262615203857 , Training accuracy = 15.62% , Validation accuracy = 19.77%
After 200 steps: cost = 2.2326087951660156 , Training accuracy = 21.09% , Validation accuracy = 26.50%
After 300 steps: cost = 2.0544686317443848 , Training accuracy = 26.56% , Validation accuracy = 30.75%
After 400 steps: cost = 1.9128262996673584 , Training accuracy = 32.03% , Validation accuracy = 32.43%
After 500 steps: cost = 1.7918856143951416 , Training accuracy = 36.72% , Validation accuracy = 34.38%
After 600 steps: cost = 1.624979853630066 , Training accuracy = 49.22% , Validation accuracy = 39.47%
After 700 steps: cost = 1.6775791645050049 , Training accuracy = 42.19% , Validation accuracy = 43.40%
After 800 steps: cost = 1.6013789176940918 , Training accuracy = 46.88% , Validation accuracy = 45.52%
After 900 steps: cost = 1.5626966953277588 , Training accuracy = 42.97% , Validation accuracy = 47.04%
After 1000 steps: cost = 1.5107836723327637 , Training accuracy = 48.44% , Validation accuracy = 49.01%
After 1100 steps: cost = 1.4100792407989502 , Training accuracy = 54.30% , Validation accuracy = 51.33%
After 1200 steps: cost = 1.4357725381851196 , Training accuracy = 53.52% , Validation accuracy = 53.32%
```

```
After 10300 steps: cost = 0.4686680734157562 , Training accuracy = 91.80% , Validation accuracy = 87.34%
After 10400 steps: cost = 0.45785951614379883 , Training accuracy = 87.11% , Validation accuracy = 87.40%
After 10500 steps: cost = 0.5490660667419434 , Training accuracy = 83.98% , Validation accuracy = 87.38%
After 10600 steps: cost = 0.4332536458969116 , Training accuracy = 91.41% , Validation accuracy = 87.32%
After 10700 steps: cost = 0.4761410653591156 , Training accuracy = 88.67% , Validation accuracy = 87.47%
After 10800 steps: cost = 0.6137319803237915 , Training accuracy = 84.77% , Validation accuracy = 87.50%
After 10900 steps: cost = 0.5339111685752869 , Training accuracy = 86.33% , Validation accuracy = 87.46%
After 11000 steps: cost = 0.5259843468666077 , Training accuracy = 87.89% , Validation accuracy = 87.66%
After 11100 steps: cost = 0.54160076379776 , Training accuracy = 87.50% , Validation accuracy = 87.53%
After 11200 steps: cost = 0.4189155697822571 , Training accuracy = 90.23% , Validation accuracy = 87.34%
After 11300 steps: cost = 0.4482972025871277 , Training accuracy = 87.11% , Validation accuracy = 87.72%
After 11400 steps: cost = 0.4371049702167511 , Training accuracy = 87.89% , Validation accuracy = 87.79%
```

It can be observed that both models achieved similar results, but the model without normalization achieved slightly better outcomes. Additionally, we can see that the running time and the number of iterations to convergence were better in the model without normalization.

Performance of the neural network on the MNIST dataset with L2 norm regularization (and without batch normalization)

```
Final Training Accuracy: 94.92%
Final Validation Accuracy: 89.40%
Final Test Accuracy: 89.39% , Number of Iterations: 8719
Total Training Time: 76.89 seconds
```




```
After 100 steps: cost = 2.1285810470581055 , Training accuracy = 20.31% , Validation accuracy = 24.43%
After 200 steps: cost = 1.9254838228225708 , Training accuracy = 39.84% , Validation accuracy = 37.62%
After 300 steps: cost = 1.7659335136413574 , Training accuracy = 46.09% , Validation accuracy = 46.86%
After 400 steps: cost = 1.7100316286087036 , Training accuracy = 43.36% , Validation accuracy = 50.12%
After 500 steps: cost = 1.5059601068496704 , Training accuracy = 54.30% , Validation accuracy = 52.53%
After 600 steps: cost = 1.3998212814331055 , Training accuracy = 58.59% , Validation accuracy = 55.83%
After 700 steps: cost = 1.4735468626022339 , Training accuracy = 58.59% , Validation accuracy = 58.03%
After 800 steps: cost = 1.2728341817855835 , Training accuracy = 64.06% , Validation accuracy = 60.86%
After 900 steps: cost = 1.3070603609085083 , Training accuracy = 60.94% , Validation accuracy = 62.17%
After 1000 steps: cost = 1.302426815032959 , Training accuracy = 62.50% , Validation accuracy = 65.31%
After 1100 steps: cost = 1.1719776391983032 , Training accuracy = 66.02% , Validation accuracy = 66.67%
After 1200 steps: cost = 1.1590176820755005 , Training accuracy = 69.14% , Validation accuracy = 68.28%
After 1300 steps: cost = 1.1386746168136597 , Training accuracy = 68.75% , Validation accuracy = 69.60%
```

```
After 7700 steps: cost = 0.4192414879798889 , Training accuracy = 90.62% , Validation accuracy = 88.81%
After 7800 steps: cost = 0.33655664324760437 , Training accuracy = 92.19% , Validation accuracy = 88.88%
After 7900 steps: cost = 0.40652623772621155 , Training accuracy = 87.89% , Validation accuracy = 88.88%
After 8000 steps: cost = 0.4599655270576477 , Training accuracy = 88.67% , Validation accuracy = 89.09%
After 8100 steps: cost = 0.32857513427734375 , Training accuracy = 91.80% , Validation accuracy = 89.13%
After 8200 steps: cost = 0.38104674220085144 , Training accuracy = 92.58% , Validation accuracy = 89.14%
After 8300 steps: cost = 0.46476566791534424 , Training accuracy = 89.45% , Validation accuracy = 89.12%
After 8400 steps: cost = 0.44928067922592163 , Training accuracy = 88.67% , Validation accuracy = 89.28%
After 8500 steps: cost = 0.4120746850967407 , Training accuracy = 91.02% , Validation accuracy = 89.47%
After 8600 steps: cost = 0.3892616927623749 , Training accuracy = 89.84% , Validation accuracy = 89.32%
After 8700 steps: cost = 0.2977752089500427 , Training accuracy = 94.92% , Validation accuracy = 89.40%
```

When L2 regularization is applied to a neural network, the weights typically have smaller magnitudes than those in a network without this regularization. This occurs because L2 regularization adds a

penalty term to the loss function. As a result, the optimization process favors smaller weights, nudging them closer to zero. This approach helps mitigate overfitting by lessening the impact of large weight values

```
W2 without L2 Regularization:
 tensor([[-0.4438, -0.1523, -0.1350, -0.0263, -0.2624, -0.4538, -0.4320, -0.7610,
          0.1793, -0.0931, -0.1109,  0.0882, -0.3653, -0.2756,  0.1382, -0.2535,
         -0.2882, -0.3597,  0.0319,  0.2193],
        [-0.2849, -0.2664, -0.2305,  0.3626, -0.3032,  0.2081, -0.1543, -0.2135,
         -0.3184, -0.3123,  0.1690, -0.2084, -0.3254, -0.0632,  0.6803, -0.3379,
          0.1481,  0.2552, -0.1848,  0.5876],
        [-0.0775,  1.1009, -0.1740,  0.1221, -0.3866,  0.3087,  0.4993,  1.0119,
         -0.0399,  0.7984,  0.5880, -0.9777, -0.4120, -0.2918, -0.0367,  0.0600,
          0.7174,  0.2424,  0.6228, -0.0955],
        [-0.7470, -0.0594, -0.1712,  0.2700,  0.7730, -0.0378,  0.8773,  0.3372,
          0.1346,  0.7311, -0.0447,  1.5563, -0.5677, -0.4514, -0.7692, -0.8191,
         -0.2513,  1.1189,  0.4773, -0.1900],
        [-0.0252,  0.1118, -0.0772, -0.0525, -0.3180,  0.6591,  0.0487, -0.5450,
         -0.0661,  0.0844, -0.0207, -0.2458, -0.1742,  0.1964,  0.5259,  0.1290,
         -0.1973,  0.3755, -0.3879,  0.0778],
        [ 0.9394, -0.2453,  0.8037, -0.5214,  0.1822, -0.0789,  0.8026,  0.2330,
          0.5356,  0.7876, -0.6745, -0.2413,  0.6731, -1.0602,  0.0791, -0.6435,
         -0.8194,  0.6004, -0.0961,  0.5460],
        [ 0.4645, -0.1014,  0.6856,  0.0267,  0.0991, -0.0761,  0.5638, -0.2932,
         -0.9153, -0.5744,  0.5134, -0.3664,  0.3113, -0.3004,  0.1659,  0.3019,
          0.4427, -0.0416, -0.1615,  0.7582]])
W2 with L2 Regularization:
 tensor([[ 1.8852e-01, -5.2870e-02,  9.3713e-01,  6.4815e-01,  6.4485e-01,
         -6.0578e-01, -1.5861e-01, -7.0808e-01, -9.4088e-02, -5.7287e-01,
         -5.8583e-01, -1.4005e-01,  1.0193e+00, -4.8314e-01, -7.9298e-01,
         -2.2627e-02,  3.3845e-01,  8.4184e-02,  3.5135e-01, -8.7830e-02],
        [-7.8218e-02, -4.8631e-03, -5.0608e-01,  9.1436e-02,  7.7414e-01,
          1.2093e-01, -5.4691e-02, -1.9179e-01, -4.4979e-01, -3.5007e-01,
         -5.9851e-01,  9.3949e-04,  3.4288e-01,  2.1228e-01,  3.0095e-01,
          9.4592e-01, -3.1100e-01, -6.2781e-02,  3.3212e-01, -4.9908e-01],
        [ 7.2783e-02,  6.6466e-01,  5.7261e-01, -4.8983e-01, -1.7683e-01,
          4.6497e-01,  6.8149e-01,  2.4658e-01, -6.9730e-01,  5.9250e-01,
         -7.4513e-02,  8.2996e-01,  2.5081e-01, -9.5800e-03, -2.4178e-01,
          4.0236e-01, -1.0373e+00, -3.3303e-01, -8.4459e-01,  3.6414e-01],
        [-5.6001e-03,  2.4936e-01,  9.1086e-02,  3.7966e-01,  6.2430e-02,
         -3.3878e-02,  3.1139e-01, -5.9757e-01,  5.3284e-01,  6.4602e-01,
          4.6928e-01, -7.3013e-01, -4.3912e-01,  6.8954e-01,  7.5512e-01,
          3.5279e-01, -2.0460e-01,  6.3283e-01, -7.9076e-01, -4.5610e-01],
        [ 1.0355e-01,  4.8079e-01,  7.5022e-02,  2.3402e-01,  6.7340e-01,
          3.8636e-01, -7.3115e-01,  8.6886e-02, -5.2131e-01,  4.2397e-02,
          3.1340e-01, -2.2990e-01, -1.2187e-01,  5.0991e-01, -3.2773e-01,
         -3.3654e-01, -1.3044e-01, -5.2710e-01,  3.3161e-01,  7.9963e-02],
        [-3.5974e-03,  2.5774e-02, -4.7427e-01,  2.9698e-01,  4.9919e-02,
         -3.6317e-01, -5.6009e-02, -1.5877e-01,  5.3796e-02, -2.9979e-01,
          1.6954e-01,  4.5411e-01, -1.4529e-02, -8.2394e-02, -4.9863e-02,
         -2.4606e-02, -6.1657e-01,  2.9774e-01, -2.0812e-01, -5.9820e-01],
        [ 5.8004e-01, -8.4168e-01,  4.8937e-01,  1.7208e-01, -4.5197e-01,
          1.0773e-01,  9.7345e-01, -7.6744e-01, -4.5663e-01, -3.5588e-01,
          1.3368e-01, -4.9097e-01,  4.8024e-01,  6.8015e-02,  7.9226e-02,
          1.5029e-02, -4.3084e-01, -1.7033e-01, -1.9341e-01,  5.0469e-01]])
W3 without L2 Regularization:
```

```
W1 without L2 Regularization:
 tensor([[-0.0033, -0.0646,  0.0846,  ..., -0.0413, -0.0240, -0.020
         [ 0.0554, -0.0674,  0.0350,  ...,  0.1142, -0.0033,  0.0028
         [-0.0170,  0.0163,  0.0081,  ..., -0.0014, -0.0256, -0.0201
         ...,
         [ 0.0392, -0.1084,  0.0170,  ..., -0.0559, -0.0166, -0.0002
         [-0.0563, -0.0635,  0.0463,  ...,  0.1385, -0.0507,  0.0740
         [ 0.0364,  0.0004,  0.0351,  ...,  0.1153,  0.0583, -0.0104
W1 with L2 Regularization:
 tensor([[ 4.6502e-02,  2.8364e-03, -1.5262e-02,  ...,  4.5615e-02,
          -4.6492e-03, -1.3208e-02],
         [-2.6803e-02,  1.9742e-02, -1.3920e-03,  ..., -3.1583e-02,
           5.2967e-02,  3.3497e-02],
         [-1.8338e-02,  2.6448e-02, -1.8762e-02,  ...,  4.1301e-02,
          -7.4314e-03, -6.9892e-04],
         ...,
         [-5.1882e-02, -6.0071e-03, -3.8433e-02,  ..., -1.1502e-02,
           5.6794e-02,  1.5542e-02],
         [-4.8847e-02, -4.7234e-02, -6.9126e-03,  ..., -3.9564e-02,
          -3.2503e-02, -6.5591e-02],
         [ 9.7060e-05,  2.2879e-02, -7.7942e-02,  ...,  1.5218e-02,
          -6.4338e-02,  1.1747e-03]])
```

```
W3 without L2 Regularization:
 tensor([[ 0.5287,  0.3028, -0.2189,  0.0954,  0.7524,  2.0926,  1.0771],
         [-0.2403,  0.5528, -0.0909,  2.0498,  0.2308, -0.4480, -0.5828],
         [ 0.2760, -0.4577, -0.5679,  0.0767,  0.6722, -0.1729,  0.2517],
         [ 0.6682, -0.7413,  2.2948, -0.3142, -1.3136, -0.5379,  0.0587],
         [ 0.8370, -0.6868,  1.0707, -1.0289, -0.7453, -1.0837,  1.0211]])
W3 with L2 Regularization:
 tensor([[ 0.0451,  0.8968,  1.2394, -0.7926,  0.0316, -0.0888,  1.0349],
         [-0.9306,  0.5165,  0.1905,  1.8337,  0.6779, -0.5420,  0.5771],
         [ 0.6026, -0.6003, -0.0970, -0.7404, -0.3870,  1.1313,  1.0870],
         [ 0.9726,  0.1095, -0.6713,  0.9779,  0.0455, -0.0576,  0.7475],
         [ 0.2375, -0.5473,  1.3415,  0.2447, -0.7963, -0.2560, -0.4562]])
W4 without L2 Regularization:
 tensor([[-0.3931, -0.4030,  0.1238,  1.3074, -0.9816],
         [-0.8724,  0.8151, -0.2784, -1.3790,  2.0160],
         [ 0.4610, -1.0851, -1.0027,  0.0747,  1.1319],
         [ 1.6155, -0.7866, -1.3071, -0.4275, -0.5643],
         [-1.3169,  1.1998,  0.2906,  0.2878,  0.0860],
         [ 1.2791,  0.4213,  0.2769, -0.2984, -1.2462],
         [-1.3263, -0.6208, -0.0271,  1.1964,  0.2642],
         [ 0.6531,  1.6554,  0.2762, -1.2104, -0.1867],
         [-0.0513,  1.2730, -0.2289,  0.0843, -1.2463],
         [ 0.7541,  0.1849,  0.9604,  0.0177,  0.0559]])
W4 with L2 Regularization:
 tensor([[ 0.7936, -1.8108, -0.8285, -0.1741,  0.6040],
         [ 0.7737, -0.6839,  0.7405,  0.7905, -1.8221],
         [ 1.5728, -0.1172, -0.1882, -1.1996, -0.7834],
         [-0.1701,  0.3225, -0.4256, -0.5388,  0.7295],
         [-0.8382, -0.7928,  1.0809, -0.3982,  0.6494],
         [-1.1891,  1.8534,  0.2853, -0.8275,  0.3501],
         [ 0.8489,  0.0578,  0.5798, -0.1231, -0.8671],
         [ 0.1040,  1.4700, -0.5826,  0.6994, -1.6151],
         [-0.3792,  1.2414,  0.3333,  0.2636, -0.2695],
         [-0.5749, -0.7123, -0.8173,  1.3924,  0.1319]])
```

To enable the code to support L2 normalization, we have modified the 'compute_cost', and the 'L_layer_model' functions. The changes allow us to penalize the model based on the

lambda hyperparameter (which we set to 0.1 in our experiment). The formulas we used are taken from the lectures.

```python
def compute_cost(AL, Y, parameters, lambd):
    """
    Computes the cost function with L2 regularization.

    Args:
        AL (torch.Tensor): Probability vector corresponding to the label predictions, shape
        Y (torch.Tensor): True "label" vector, shape (num_classes, m).
        parameters (dict): Dictionary containing the model parameters.
        lambd (float): Regularization parameter.

    Returns:
        cost (torch.Tensor): Cross-entropy cost with L2 regularization.
    """
    m = Y.shape[1]

    # Cross-entropy loss
    cross_entropy_cost = -torch.sum(Y * torch.log(AL + 1e-8)) / m

    # L2 regularization cost
    L2_regularization_cost = 0
    if lambd != 0:
        L = len(parameters) // 2  # Number of layers
        for l in range(1, L + 1):
            L2_regularization_cost += torch.sum(torch.square(parameters["W" + str(l)]))
        L2_regularization_cost = (lambd / (2 * m)) * L2_regularization_cost

    cost = cross_entropy_cost + L2_regularization_cost
    return cost
```