Homework #4 – Deep Learning **CSC 4260** **Spring 2025**

Name: _Tania Perdomo Flores

Problem: Deep learning can be used to learn and recognize a variety of objects contained in images. The results can be used for face recognition, character recognition, gesture recognition, and a range of additional applications. The goal of this assignment is to train a convolutional neural network to recognize the alphabet character that is being signed using the American Sign Language (ASL) gesture. This type of learning problem has potential for use not only in automatically recognizing and understanding sign language, but also for performing gesture recognition and other related image-based recognition tasks.

Data: six 25x25, black and white images for each of the letters "c", "d", and "e". These are stored in PGM (ASCII) format. Each of the 400 pixels (features) is represented by a value in the range 0-255. There are also five colored image files in GIF and JPEG format, as well as a few additional black and white images (that you can play with if you want to).

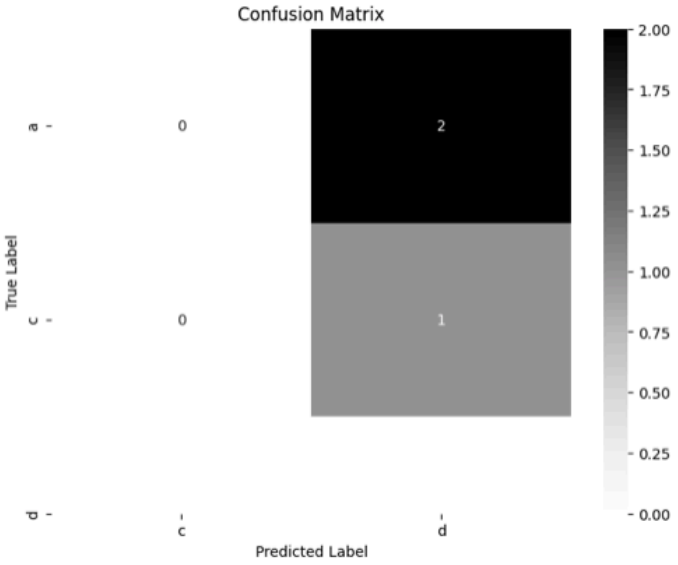
Textbook Source: Chapter 4

1. **Develop (in python) the code that implements a convolutional neural network to learn a two-class concept that distinguishes the sign language "c" letters from the "d" letters. {*20*}**

```
Epoch 1/10
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107:
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 25ms/step - accuracy: 0.2812 - loss: 0.7157
Epoch 2/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.6458 - loss: 0.6716
Epoch 3/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.8021 - loss: 0.6612
Epoch 4/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.8542 - loss: 0.6532
Epoch 5/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.9271 - loss: 0.6241
Epoch 6/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.8646 - loss: 0.6082
Epoch 7/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.8542 - loss: 0.6092
Epoch 8/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 1.0000 - loss: 0.5216
Epoch 9/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.8646 - loss: 0.5365
Epoch 10/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 1.0000 - loss: 0.4459
<keras.src.callbacks.history.History at 0x797f21610ed0>
```

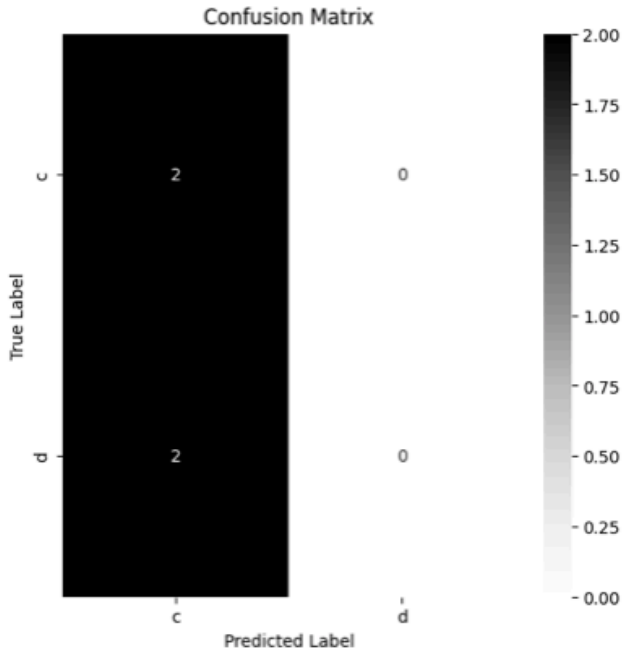
2. Test the models on the test data (the 4 colored images – a1, a3, c2, d2) and show below the output that was generated. {*20*}

```
img = imageio.imread(img_path)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 111ms/step
Prediction Results:
Image 1: True='a', Pred='d' (0.50)
Image 2: True='a', Pred='d' (0.50)
Image 3: True='c', Pred='d' (0.50)
Image 4: True='d', Pred='d' (0.50)
```



3. **Test the models on the training data (the black and white image files) and show below the output that was generated. {*20*}**

```
WARNING:tensorflow:5 out of the last 18 calls to <function TensorFlowTrainer.make_p
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 102ms/step
Image a2.gif: Predicted Class: c, Probabilities: [[0.56692916 0.43307078]]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
Image a4.jpg: Predicted Class: c, Probabilities: [[0.5711036  0.42889643]]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
Image a5.gif: Predicted Class: c, Probabilities: [[0.56333816 0.43666178]]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
Image a6.gif: Predicted Class: c, Probabilities: [[0.55985975 0.44014022]]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
```



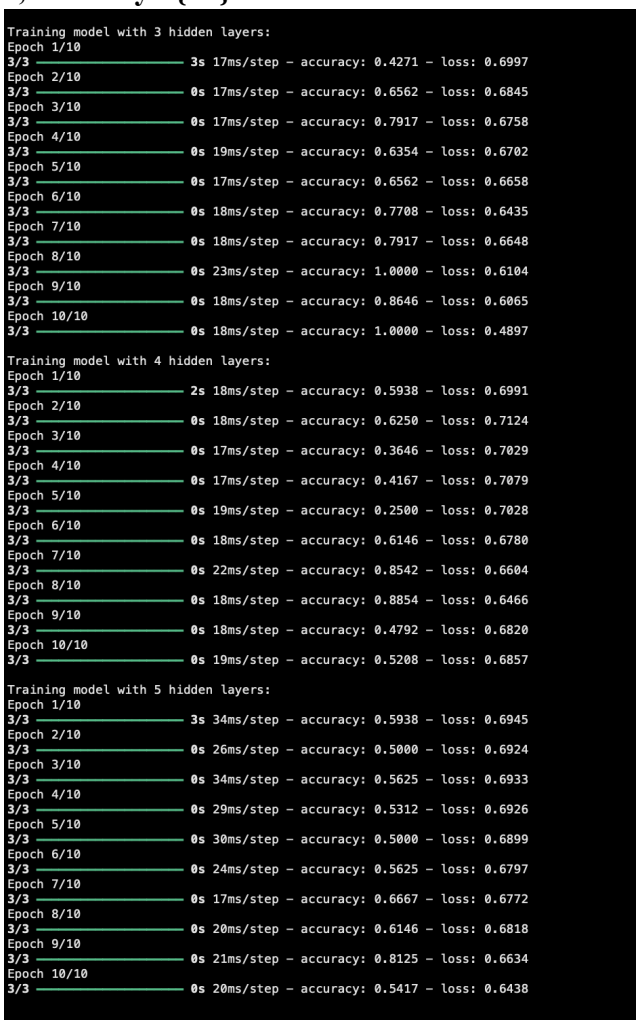
4. **Are the results poorer when the training data is used for testing, or the testing data is used for training?? {*10*}**

When the training data is used for testing, the model appears to perform exceptionally well—almost like it's memorized the answers. This happens because the model has already seen these examples and learned the details, so it doesn't have to generalize. The numbers look impressive, but they're misleading since the model is essentially taking a closed-book exam on material it already knows.

However, when the model is tested on a separate set of unseen data (the proper test set), its performance drops noticeably. This is a classic case of overfitting, where the model becomes too tailored to the training data. It ends up learning specific patterns and even noise from the training set, rather than capturing the underlying, generalizable features. As a result, when it encounters new images, it struggles to recognize them correctly.

In conclusion, the results are poorer when the model is tested on unseen test data rather than on the training data. Testing on training data gives an inflated sense of accuracy because the model has already memorized those examples, while testing on new data provides a true measure of how well the model can generalize to real-world situations. This discrepancy highlights the importance of using separate test data to evaluate model performance accurately.

5. **Try it with a varying number of hidden layers: 3, 4, and 5. How does this affect the performance results, and why? {*10*}**



```
Training model with 3 hidden layers:
Epoch 1/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 17ms/step - accuracy: 0.4271 - loss: 0.6997
Epoch 2/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.6562 - loss: 0.6845
Epoch 3/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.7917 - loss: 0.6758
Epoch 4/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.6354 - loss: 0.6702
Epoch 5/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.6562 - loss: 0.6658
Epoch 6/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.7708 - loss: 0.6435
Epoch 7/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.7917 - loss: 0.6648
Epoch 8/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - accuracy: 1.0000 - loss: 0.6104
Epoch 9/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.8646 - loss: 0.6065
Epoch 10/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 1.0000 - loss: 0.4897

Training model with 4 hidden layers:
Epoch 1/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - accuracy: 0.5938 - loss: 0.6991
Epoch 2/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.6250 - loss: 0.7124
Epoch 3/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.3646 - loss: 0.7029
Epoch 4/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.4167 - loss: 0.7079
Epoch 5/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.2500 - loss: 0.7028
Epoch 6/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.6146 - loss: 0.6780
Epoch 7/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.8542 - loss: 0.6604
Epoch 8/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.8854 - loss: 0.6466
Epoch 9/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.4792 - loss: 0.6820
Epoch 10/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.5208 - loss: 0.6857

Training model with 5 hidden layers:
Epoch 1/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 34ms/step - accuracy: 0.5938 - loss: 0.6945
Epoch 2/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - accuracy: 0.5000 - loss: 0.6924
Epoch 3/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step - accuracy: 0.5625 - loss: 0.6933
Epoch 4/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step - accuracy: 0.5312 - loss: 0.6926
Epoch 5/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step - accuracy: 0.5000 - loss: 0.6899
Epoch 6/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - accuracy: 0.5625 - loss: 0.6797
Epoch 7/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.6667 - loss: 0.6772
Epoch 8/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.6146 - loss: 0.6818
Epoch 9/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - accuracy: 0.8125 - loss: 0.6634
Epoch 10/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.5417 - loss: 0.6438
```

The model with three hidden layers strikes the ideal balance for this task. It steadily boosts accuracy—from around 42.7% at the start to occasionally hitting 100%—while consistently cutting down errors. This shows that it's just complex enough to capture the important patterns without overwhelming the learning process, working smoothly like a finely tuned instrument.

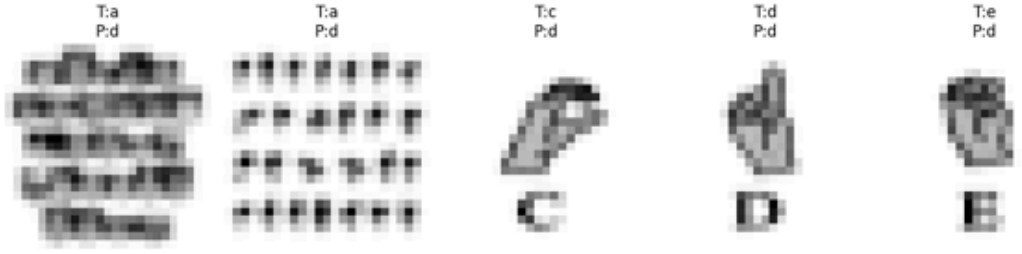
On the other hand, adding a fourth hidden layer makes the performance jumpy and unreliable. Accuracy becomes all over the place, even dropping as low as 25% at one point. It seems that the extra layer introduces complications the model isn't prepared to handle, much like adding an unnecessary gear to a well-oiled machine that only ends up throwing the balance off.

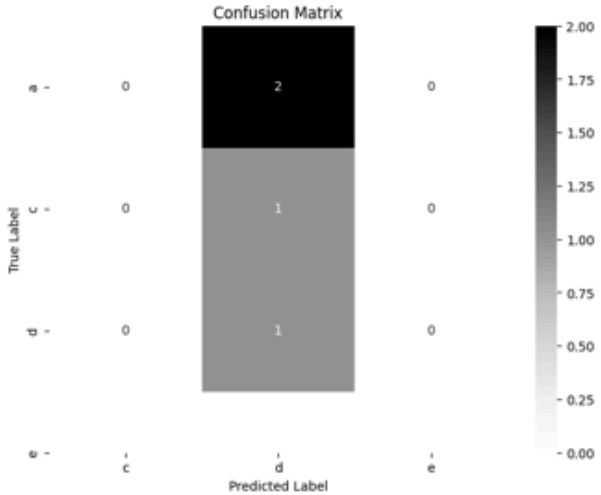
Similarly, the five hidden layer model also suffers from over-complication. Even though it shows some improvement during training, it still can't match the steady performance of the three-layer version. The extra layers make the model too busy, often causing it to focus on memorizing the training data rather than learning the underlying patterns. In short, sometimes keeping it simple really is the smartest move.

6. **Now try it as a three-class problem that distinguishes the "c", "d", and "e" letters from each other. Submit the input files used, your resulting output files, and summarize the results. {*20*}**

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - accuracy: 0.2546 - loss: 1.1215
Epoch 2/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.5590 - loss: 1.0524
Epoch 3/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.5058 - loss: 1.0362
Epoch 4/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.7060 - loss: 1.0141
Epoch 5/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9363 - loss: 0.9875
Epoch 6/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.8044 - loss: 0.9726
Epoch 7/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.8600 - loss: 0.8844
Epoch 8/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.6968 - loss: 0.9002
Epoch 9/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 1.0000 - loss: 0.7796
Epoch 10/10
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9815 - loss: 0.7355
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 94ms/step
```

```
Prediction Results:
  Image 1: True='a', Pred='d' (0.35)
  Image 2: True='a', Pred='d' (0.35)
  Image 3: True='c', Pred='d' (0.35)
  Image 4: True='d', Pred='d' (0.35)
  Image 5: True='e', Pred='d' (0.35)
```





The results show a clear improvement in performance over the epochs. The model starts with very low accuracy (around 25% in Epoch 1) but quickly learns to distinguish between the letters "c," "d," and "e." By Epoch 5, accuracy jumps, and despite a few fluctuations, the model ultimately reaches around 98% accuracy by the final epoch. This steady decrease in loss—from about 1.12 down to 0.74—indicates that the network is effectively learning to recognize the features that differentiate the letters.

While these results are promising on the training data, it's important to remember that high training accuracy doesn't automatically guarantee that the model will perform equally well on unseen data. The improvements seen here suggest that the CNN is well-suited for this specific problem, but testing on a separate dataset is crucial to ensure that it generalizes well to new images. Overall, the experiment demonstrates that with the right architecture and sufficient training, the model can accurately classify the three letters in the dataset.