STA414: HW3 Q2

Date: March 22, 2021
First Name: Mahrukh
Last Name: Niazi
Student Id: 1003948204

Q2.1: Complete the code.

In [1]:
```python
def affine_backward(grad_y, x, w):
    """ Computes gradients of affine transformation.
    Hint: you may need the matrix transpose np.dot(A, B).T = np.dot(B, A) and (A.T).T = A
    :param grad_y: Gradient from upper layer
    :param x: Inputs from the hidden layer
    :param w: Weights
    :return: A tuple of (grad_h, grad_w, grad_b)
        WHERE
        grad_x: Gradients wrt. the inputs/hidden layer.
        grad_w: Gradients wrt. the weights.
        grad_b: Gradients wrt. the biases.
    """
    ###########################################################################
    # TODO:                                                                   #
    # Complete the function to compute the gradients of affine                #
    # transformation.                                                         #
    ###########################################################################
    grad_x = np.dot(grad_y, w.T)
    grad_w = np.dot(x.T, grad_y)
    grad_b = np.sum(grad_y, axis = 0)
    ###########################################################################
    #                            END OF YOUR CODE                             #
    ###########################################################################
    return grad_x, grad_w, grad_b

def relu_backward(grad_y, x):
    """ Computes gradients of the ReLU activation function wrt. the unactivated inputs.
    :param grad_y: Gradient of the activation.
    :param x: Inputs
    :return: Gradient wrt. x
    """
    ###########################################################################
    # TODO:                                                                   #
    # Complete the function to compute the gradients of relu.                 #
    ###########################################################################
    grad_x = grad_y * (x > 0)
    ###########################################################################
    #                            END OF YOUR CODE                             #
    ###########################################################################
    return grad_x

def nn_update(model, eta):
    """ Update NN weights.
    :param model: Dictionary of all the weights.
    :param eta: Learning rate
    :return: None
    """
    ###########################################################################
    # TODO:                                                                   #
    # Complete the function to update the neural network's parameters.        #
    # Your code should look as follows                                        #
    # model["W1"] = ...                                                       #
    # model["W2"] = ...                                                       #
    # ...                                                                     #
    ###########################################################################
    model["W1"] = model["W1"] - eta * model['dE_dW1']
    model["W2"] = model["W2"] - eta * model['dE_dW2']
    model["W3"] = model["W3"] - eta * model['dE_dW3']
    model["b1"] = model["b1"] - eta * model['dE_db1']
    model["b2"] = model["b2"] - eta * model['dE_db2']
    model["b3"] = model["b3"] - eta * model['dE_db3']

    ###########################################################################
    #                            END OF YOUR CODE                             #
    ###########################################################################
    return
```

Q2.2: Generalization.

Using default hyperparameters: **learning rate = 0.01, batch size = 100, num_epochs = 1000, num_hiddens = [16, 32]**

Plot of training and validation cross entropy loss and accuracy:

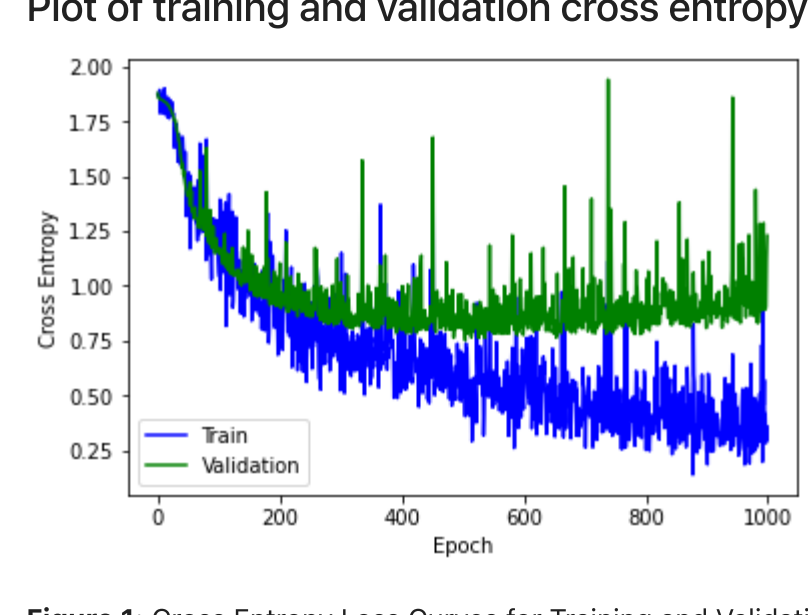


Figure 1: Cross Entropy Loss Curves for Training and Validation Set

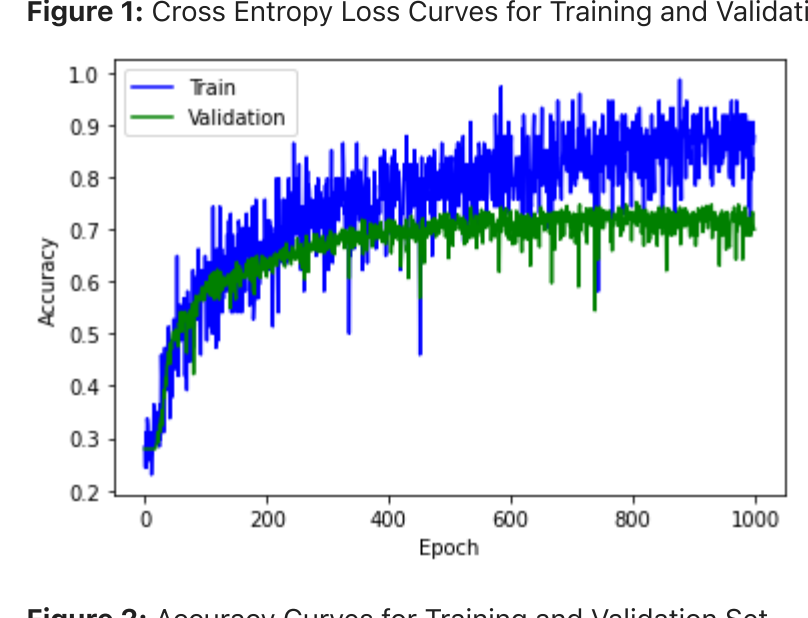


Figure 2: Accuracy Curves for Training and Validation Set

Plot of training and validation errors is given by the accuracy plot in Figure 2 since there is a 1-1 correspondence between accuracy and error.

Final training and validation errors:

Table 1: Final Statistics when training model using learning rate = 0.01, batch size = 100, num_epochs = 1000, num_hiddens = [16, 32]

Measure	Training Set	Validation Set	Test Set
Cross-Entropy	0.54485	1.22866	1.09753
Accuracy	0.79223	0.69928	0.69870

The final training error is 1 - 0.79223 = 0.20777.
The final validation error is 1 - 0.69928 = 0.30072.

Comments on network's performance:

The training accuracy plot **(Figure 2)** depicts percentage of data in current dataset that were labelled with the correct class and the validation accuracy plot **(Figure 2)** depicts the precision on randomly selected images from different classes. Essentially, the training accuracy is based on images that the network model has already been trained/learned on so the model can over fit to the noise in the trained data.

As we can see from the accuracy plot **(Figure 2)**, the training set has greater accuracy (therefore lower error) than that of the validation set. This is also seen from the statistics: training accuracy is 0.79223, which is greater than the validation accuracy 0.69928. This suggests that the current model will perform with about 70% accuracy on new data. Since there is increasingly greater training accuracy than validation accuracy, it suggests that the model fits the training set better, but it is losing its ability to predict on new data (validation set), so the model is likely starting to fit noise and overfitting the data (high variance). So the network's performance differs on the training set as it overfits the training data as opposed to the validation data set.

For the network to perform/generalize better, the validation accuracy should be equal or slightly less than the training accuracy.

Q2.3: Optimization.

Changing learning rate parameter:

keeping other default hyperparameters constant: batch size = 100, num_epochs = 1000, num_hiddens = [16, 32]

Using learning Rate = 0.001:

Table 2: Final Statistics when training model using learning rate = 0.001, batch size = 100, num_epochs = 1000, num_hiddens = [16, 32]

Measure	Training Set	Validation Set
Cross-Entropy	1.10543	1.12684
Accuracy	0.60670	0.57995

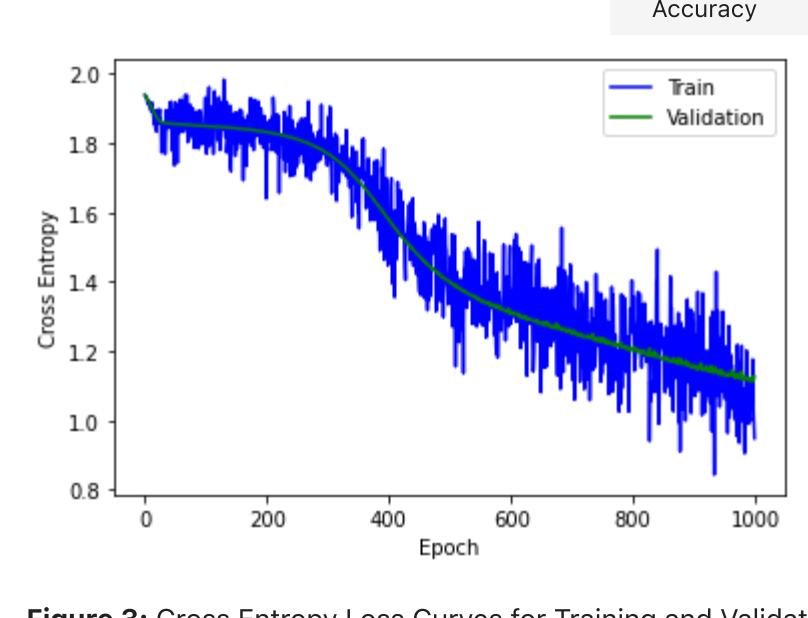


Figure 3: Cross Entropy Loss Curves for Training and Validation Set using **learning rate = 0.001**

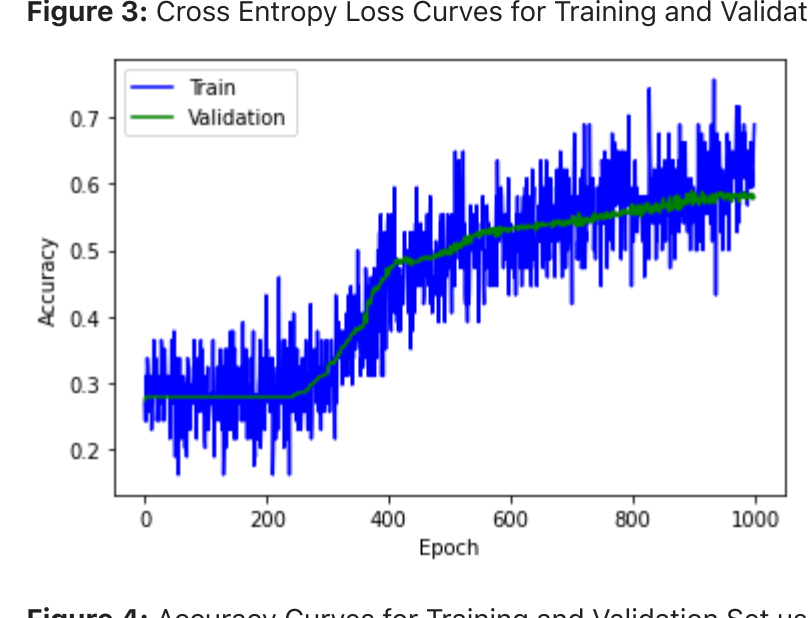


Figure 4: Accuracy Curves for Training and Validation Set using **learning rate = 0.001**

Using learning rate = 0.01:

Table 3: Final Statistics when training model using learning rate = 0.01, batch size = 100, num_epochs = 1000, num_hiddens = [16, 32]

Measure	Training Set	Validation Set
Cross-Entropy	0.38604	1.07142
Accuracy	0.86396	0.71599

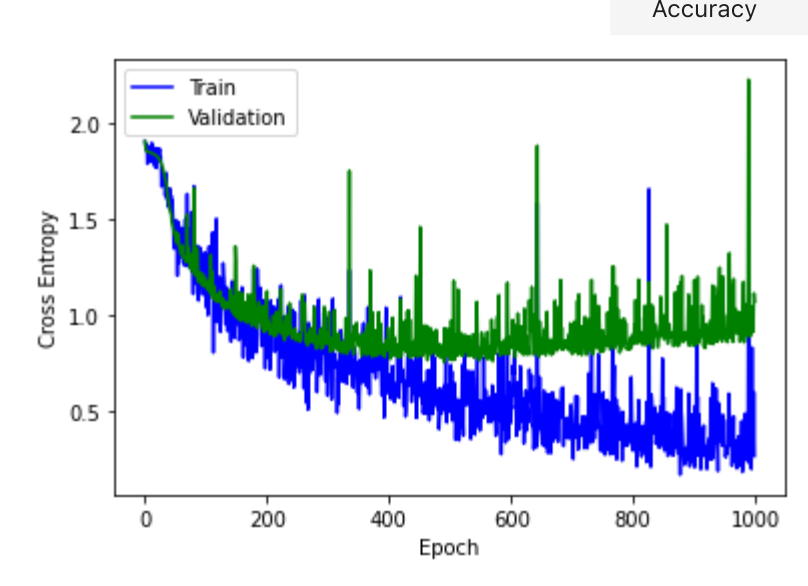


Figure 5: Cross Entropy Loss Curves for Training and Validation Set using **learning rate = 0.01**

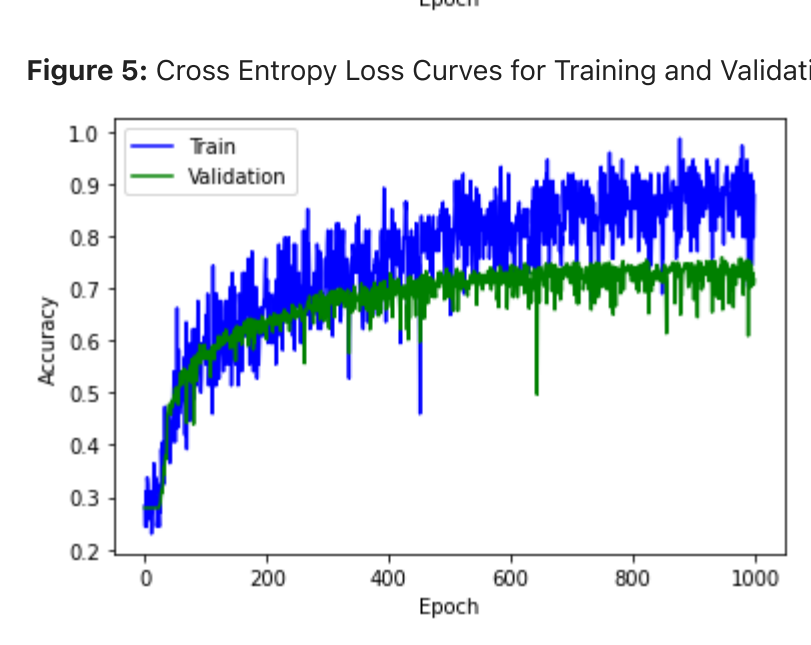


Figure 6: Accuracy Curves for Training and Validation Set using **learning rate = 0.01**

Using learning rate = 0.5:

Table 4: Final Statistics when training model using learning rate = 0.5, batch size = 100, num_epochs = 1000, num_hiddens = [16, 32]

Measure	Training Set	Validation Set
Cross-Entropy	1.86108	1.85905
Accuracy	0.28542	0.27924

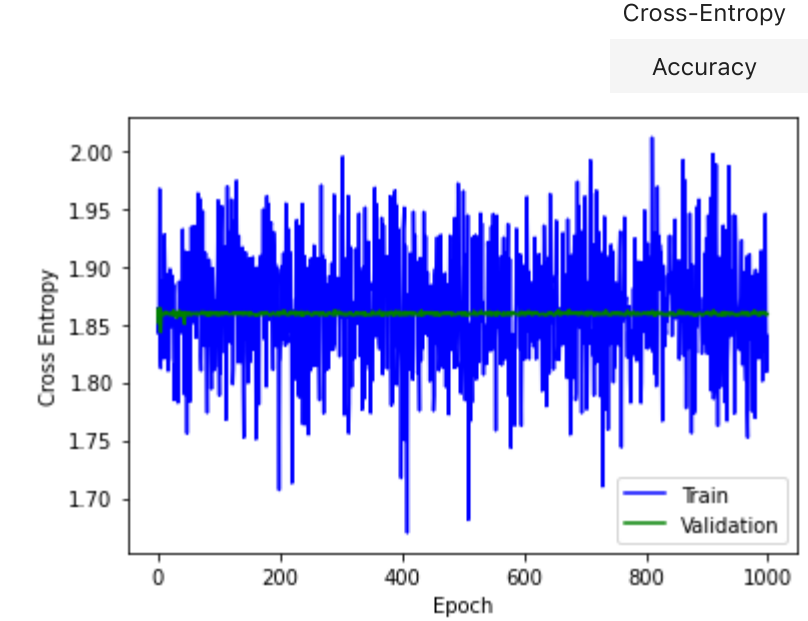


Figure 7: Cross Entropy Loss Curves for Training and Validation Set using **learning rate = 0.5**

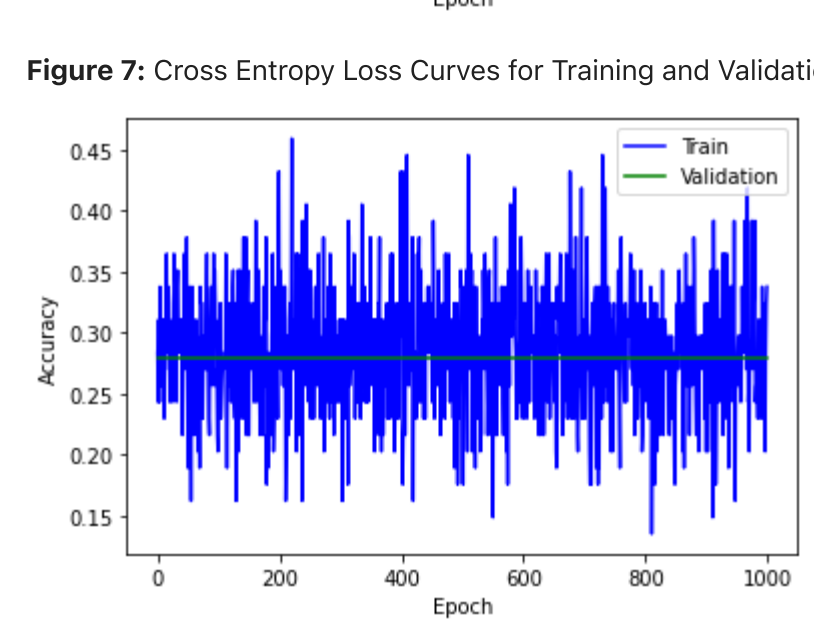


Figure 8: Accuracy Curves for Training and Validation Set using **learning rate = 0.5**

Effect of learning rate on convergence properties:

In general, a smaller learning rate can allow the model to reach a better optimal set of weights but takes significantly longer to train. As such, the model takes longer to converge so will require more training epochs. The model can also get stuck in the process if the learning rate is too small so an optimal solution is not reachede. A larger learning rate can result in rapid changes, thus requiring less training epochs, but can converge too quickly to a suboptimal solution. These properties are shown in the figures produced above **(Figures 3-8)**.

Looking at cross entropy **(Figures 3, 5, 7)**, we see that with a low learning rate **(Figure 3)**, the improvements in the loss function are slower and more linear, and the model does not converge to an optimal solution. This is seen with both the training and validation set. With a high learning rate **(Figure 7)**, the loss oscillates over training epochs for both training and validation set, which suggests that the trained weights begin to diverge. With a moderate learning rate **(Figure 5)**, the loss function converges more quickly to a minimum.

Looking at accuracy **(Figures 4, 6, 8)**, we see that with a moderate learning rate **(Figure 6)** the validation accuracy is the highest. This is also seen in the final statistics: validation accuracy for learning rate 0.01 is the highest (0.71599) and training error is the lowest (0.13604) **(Table 3)**. This is followed by learning rate 0.001 having the second highest validation accuracy and second lowest training error **(Table 2)**, and learning rate 0.5 performing the worst for these statistics **(Table 4)**.

Choosing best learning rate parameter:

Based on these results, the <u>best learning rate is 0.01</u> since the model converges within 1000 epochs and the validation accuracy is the highest and training error is the lowest.

Changing batch size parameter:

keeping other default hyperparameters constant: learning rate = 0.01, num_epochs = 1000, num_hiddens = [16, 32]

Using batch_size = 10:

Table 5: Final Statistics when training model using learning rate = 0.01, batch size = 10, num_epochs = 1000, num_hiddens = [16, 32]

Measure	Training Set	Validation Set
Cross-Entropy	0.00994	2.05463
Accuracy	1.00000	0.74463

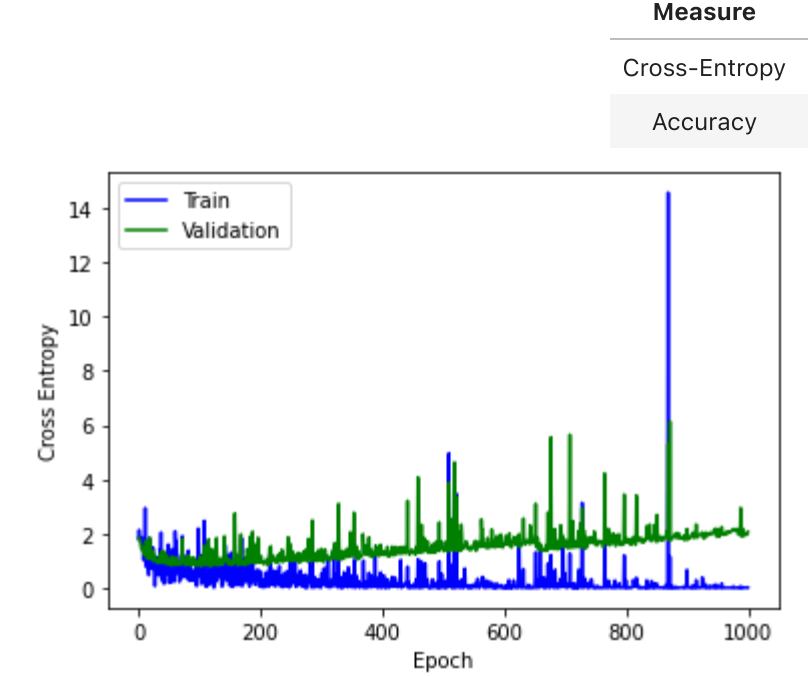


Figure 9: Cross Entropy Loss Curves for Training and Validation Set using **batch_size = 10**

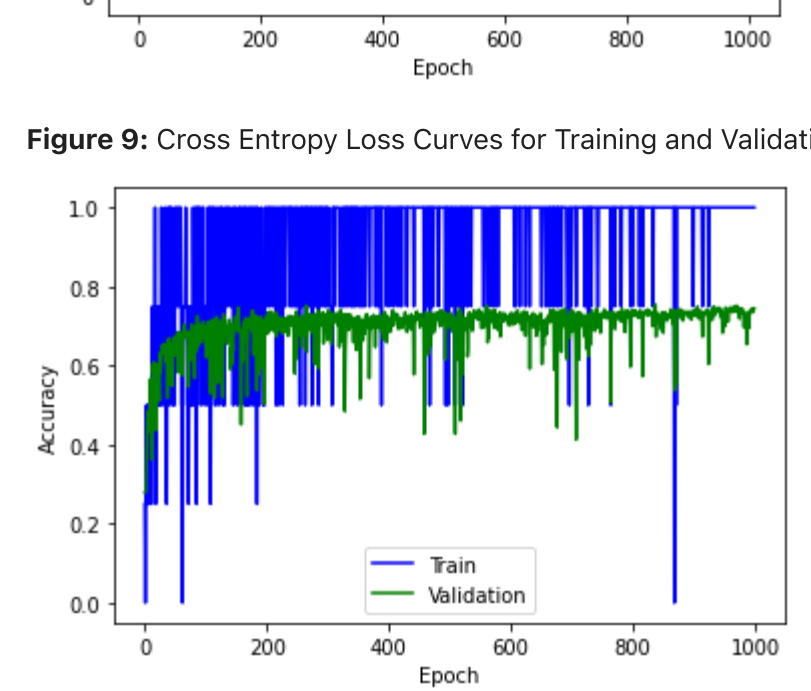


Figure 10: Accuracy Curves for Training and Validation Set using **batch_size = 10**

Using batch_size = 100:

Table 6: Final Statistics when training model using learning rate = 0.01, batch size = 100, num_epochs = 1000, num_hiddens = [16, 32]

Measure	Training Set	Validation Set
Cross-Entropy	0.38604	1.07142
Accuracy	0.86396	0.71599

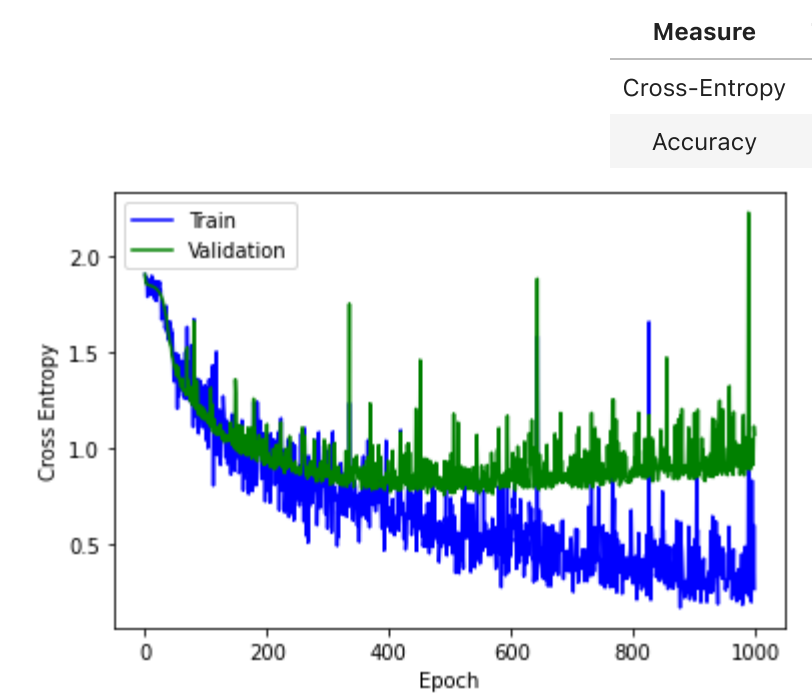


Figure 11: Cross Entropy Loss Curves for Training and Validation Set using **batch_size = 100**

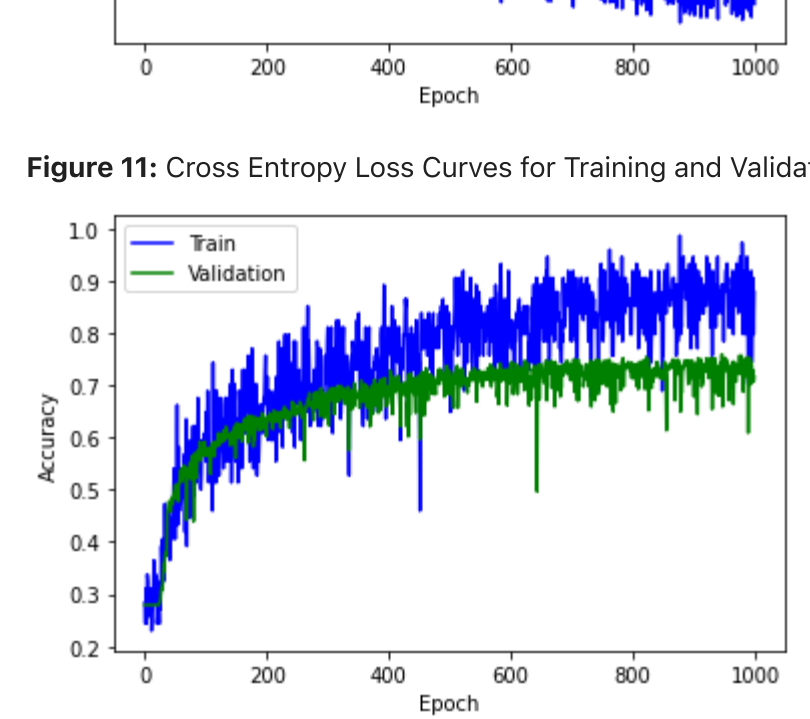


Figure 12: Accuracy Curves for Training and Validation Set using **batch_size = 100**

Using batch_size = 1000:

Table 7: Final Statistics when training model using learning rate = 0.01, batch size = 1000, num_epochs = 1000, num_hiddens = [16, 32]

Measure	Training Set	Validation Set
Cross-Entropy	1.06620	1.09098
Accuracy	0.61440	0.58711

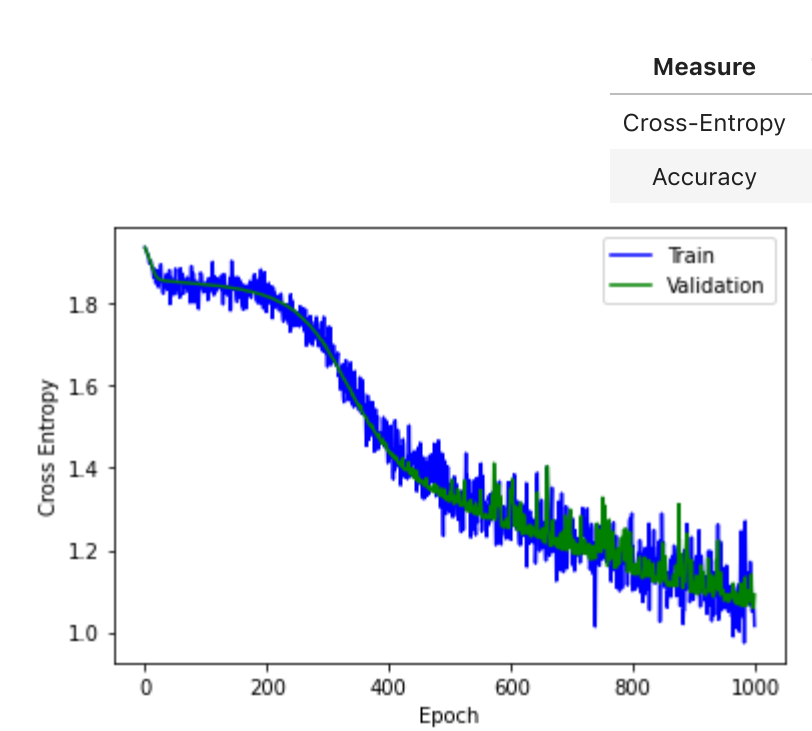


Figure 13: Cross Entropy Loss Curves for Training and Validation Set using **batch_size = 1000**

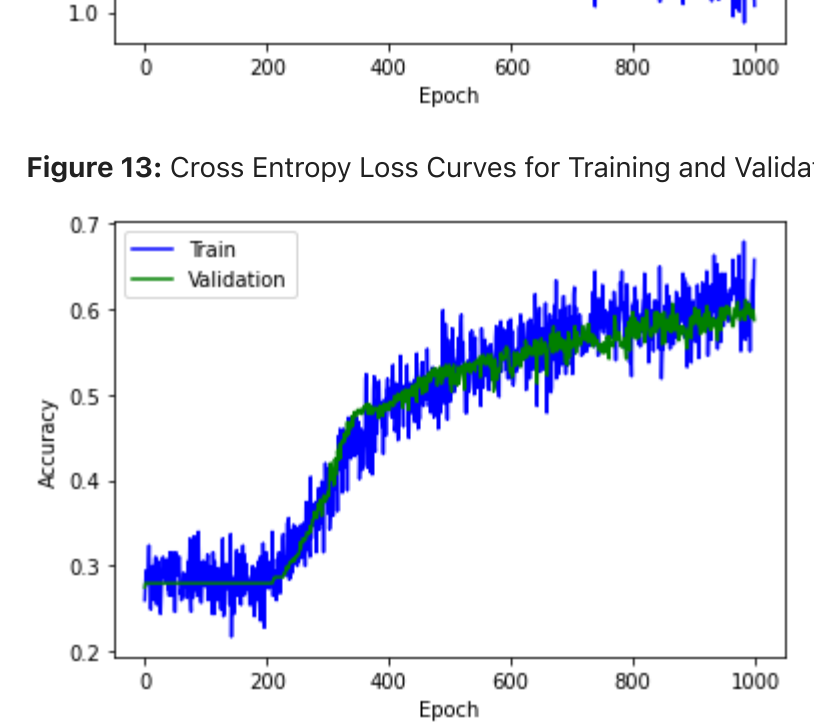


Figure 14: Accuracy Curves for Training and Validation Set using **batch_size = 1000**

Effect of batch size on convergence properties:

In general, a too large batch size will lead to poor generalization of results (low accuracy). Smaller batch sizes have faster convergence to good solutions but too small of a batch size may not converge to a global optima. These properties are shown in the figures produced above **(Figures 9-14)**.

Looking at cross entropy **(Figures 9, 11, 13)**, we see that with a too small batch size has the highest minimum validation loss **(Figure 9)**, followed by the largest batch size **(Figure 13)**, and the moderate batch size with the lowest minimum validation loss **(Figure 11)**. Thus, the smallest batch size has the worst minimum validation loss and the moderate batch size has the best minimum validation loss. Moreover, the smallest batch size **(Figure 9)** does not converge to the minimum validation loss, as it slightly starts increasing instead, and the moderate batch size **(Figure 11)** does converge to a minimum validation loss. We also see that the large batch size **(Figure 13)** has the slowest training loss decrease and takes more epochs to converge to the minimum validation loss.

Looking at accuracy **(Figures 10, 12, 14)**, we see that the largest batch size **(Figure 14)** has the lowest validation accuracy and highest training error, followed by the moderate batch size **(Figure 12)**, and the smallest batch size **(Figure 10)** with the highest validation accuracy and no training error.

Choosing best batch size parameter:

Based on these results, the <u>best batch size rate is 100</u> since the model converges within 1000 epochs to a minimum validation loss (highest and lowest batch size did not converge), and has higher validation accuracy than that of the larger batch size.

Q2.4: Model architecture.

Changing number of nodes in hidden layer 1:

keeping other default hyperparameters constant: learning rate = 0.01, num_epochs = 1000, batch_size = 100

Using num_hiddens = [2,32]:

Table 8: Final Statistics when training model using learning rate = 0.01, batch size = 100, num_epochs = 1000, num_hiddens = [2, 32]

Measure	Training Set	Validation Set	Test Set
Cross-Entropy	1.28166	1.30509	1.34574
Accuracy	0.52401	0.52267	0.51688

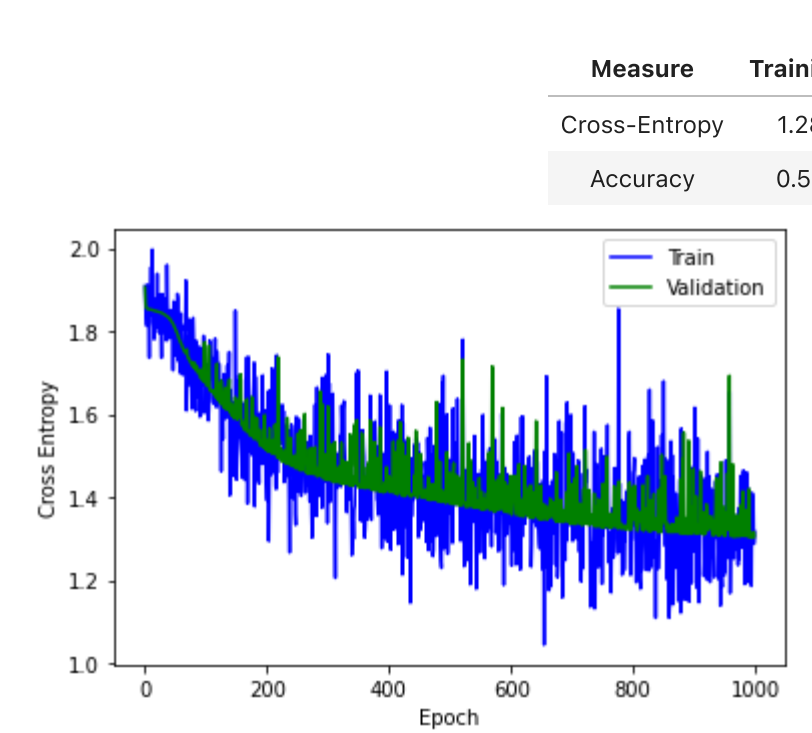


Figure 15: Cross Entropy Loss Curves for Training and Validation Set using **num_hiddens = [2,32]**

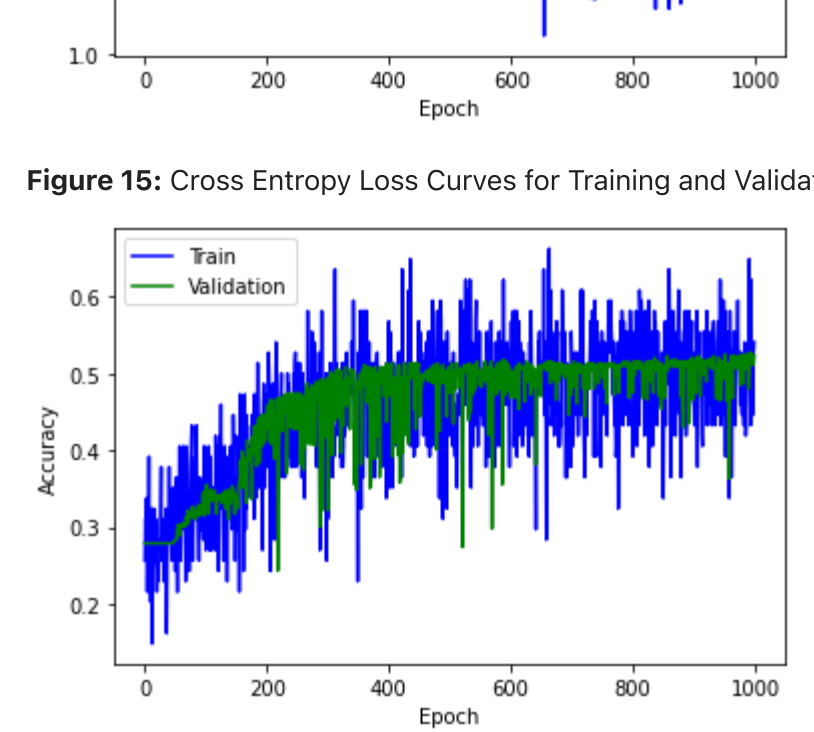


Figure 16: Accuracy Curves for Training and Validation Set using **num_hiddens = [2,32]**

Using num_hiddens = [20, 32]:

Table 9: Final Statistics when training model using learning rate = 0.01, batch size = 100, num_epochs = 1000, num_hiddens = [20, 32]

Measure	Training Set	Validation Set	Test Set
Cross-Entropy	0.18778	0.86982	0.74201
Accuracy	0.94013	0.76850	0.77922

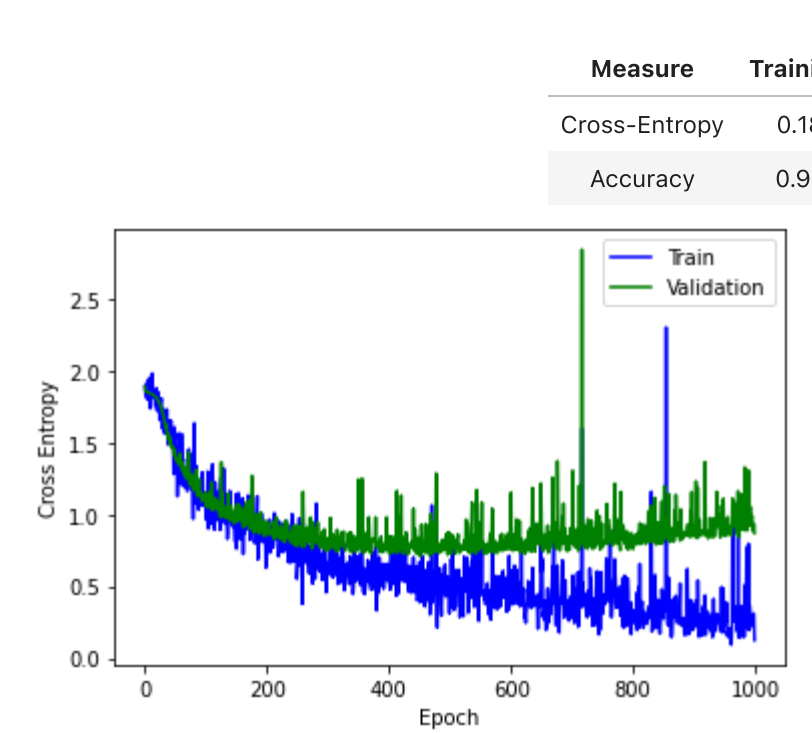


Figure 17: Cross Entropy Loss Curves for Training and Validation Set using **num_hiddens = [20,32]**

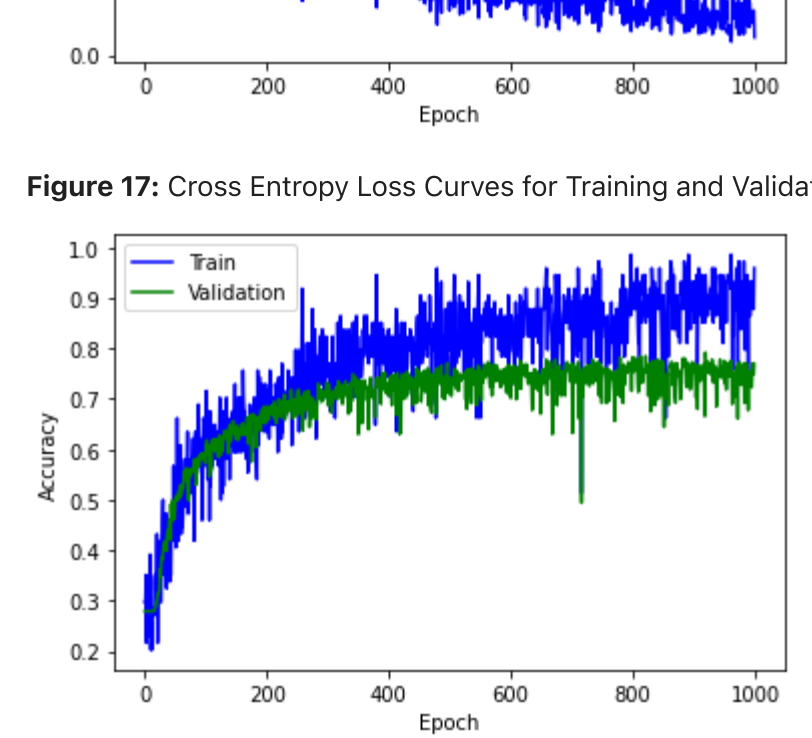


Figure 18: Accuracy Curves for Training and Validation Set using **num_hiddens = [20,32]**

Using num_hiddens = [80, 32]:

Table 10: Final Statistics when training model using learning rate = 0.01, batch size = 100, num_epochs = 1000, num_hiddens = [80, 32]

Measure	Training Set	Validation Set	Test Set
Cross-Entropy	0.07029	0.98573	0.73545
Accuracy	0.98785	0.74463	0.80260

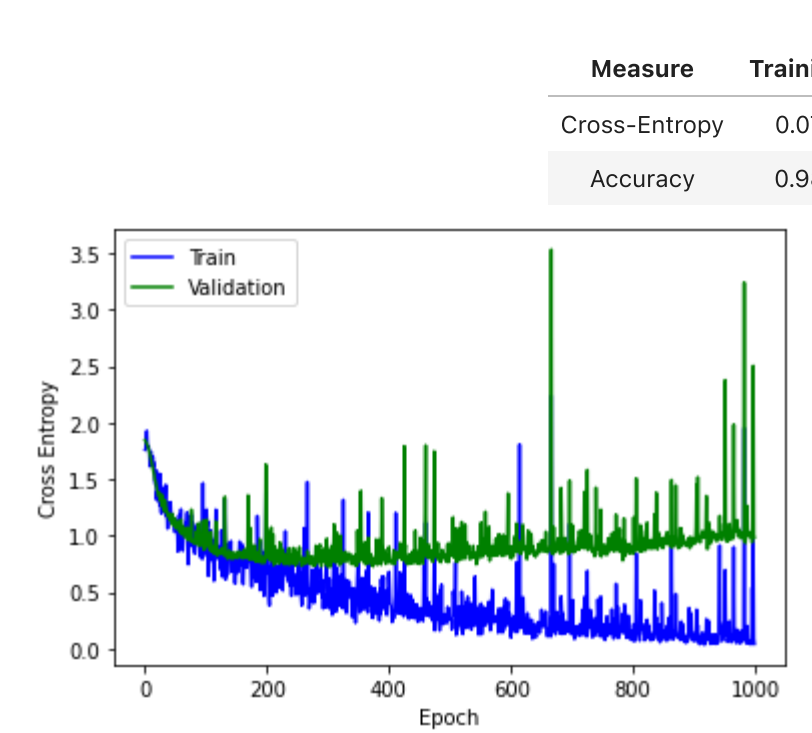


Figure 19: Cross Entropy Loss Curves for Training and Validation Set using **num_hiddens = [80,32]**

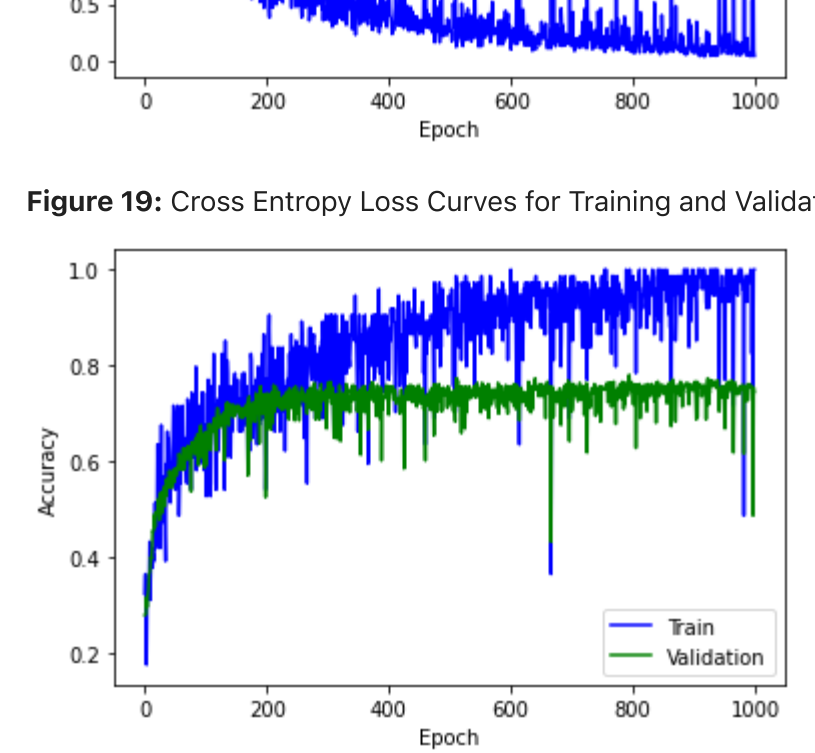


Figure 20: Accuracy Curves for Training and Validation Set using **num_hiddens = [80,32]**

Changing number of nodes in hidden layer 2:

keeping other default hyperparameters constant: learning rate = 0.01, num_epochs = 800, batch_size = 100

Using num_hiddens = [16, 2]:

Table 11: Final Statistics when training model using learning rate = 0.01, batch size = 100, num_epochs = 800, num_hiddens = [16, 2]

Measure	Training Set	Validation Set	Test Set
Cross-Entropy	1.26028	1.32261	1.36377
Accuracy	0.53053	0.51790	0.50909

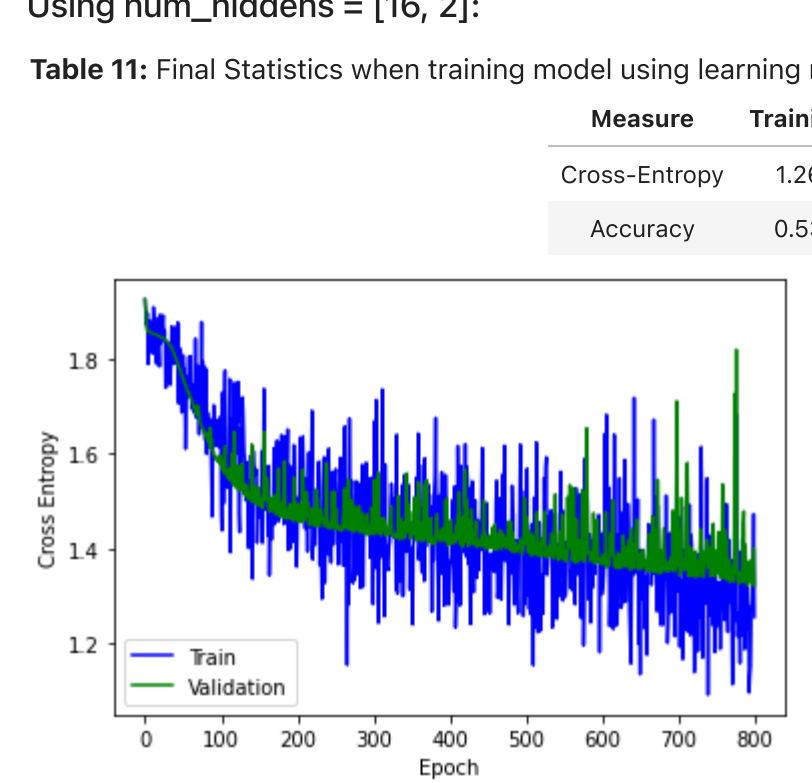


Figure 21: Cross Entropy Loss Curves for Training and Validation Set using **num_hiddens = [16,2]**

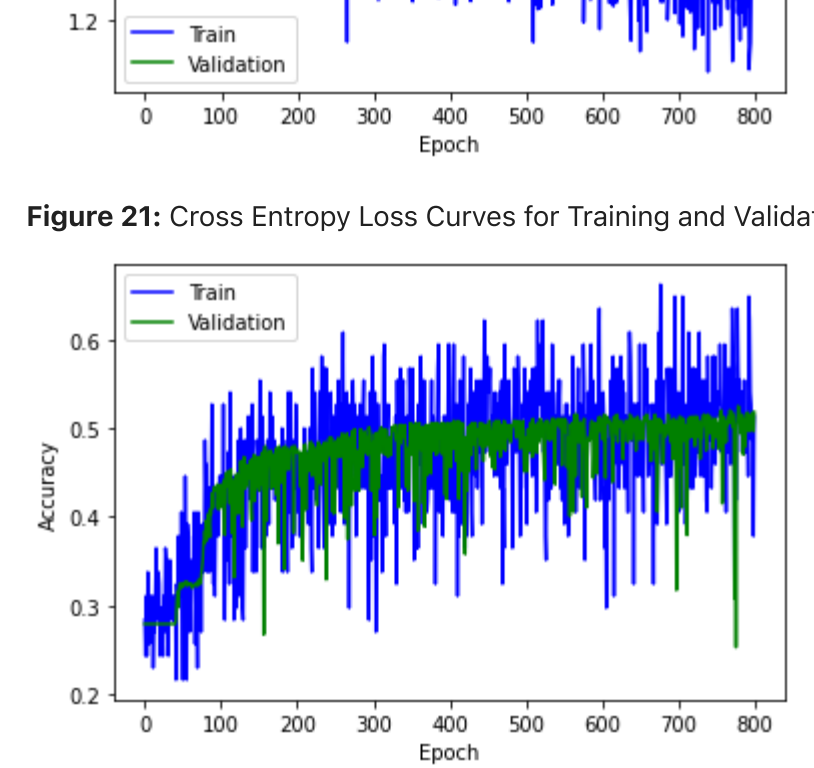


Figure 22: Accuracy Curves for Training and Validation Set using **num_hiddens = [16,2]**

Using num_hiddens = [16, 20]:

Table 12: Final Statistics when training model using learning rate = 0.01, batch size = 100, num_epochs = 800, num_hiddens = [16, 20]

Measure	Training Set	Validation Set	Test Set
Cross-Entropy	0.25838	0.77781	0.71475
Accuracy	0.91612	0.75656	0.75065

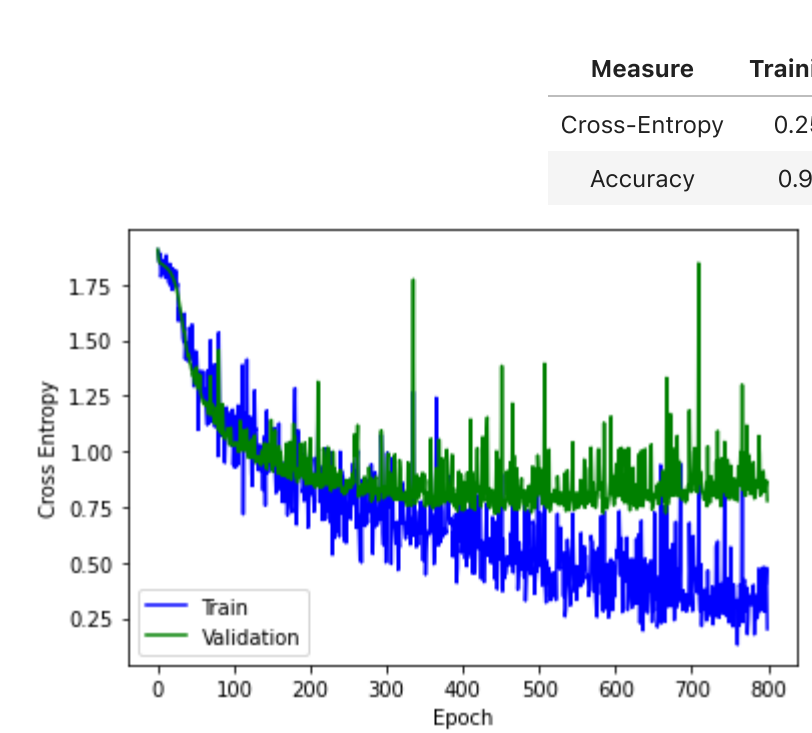


Figure 23: Cross Entropy Loss Curves for Training and Validation Set using **num_hiddens = [16,20]**

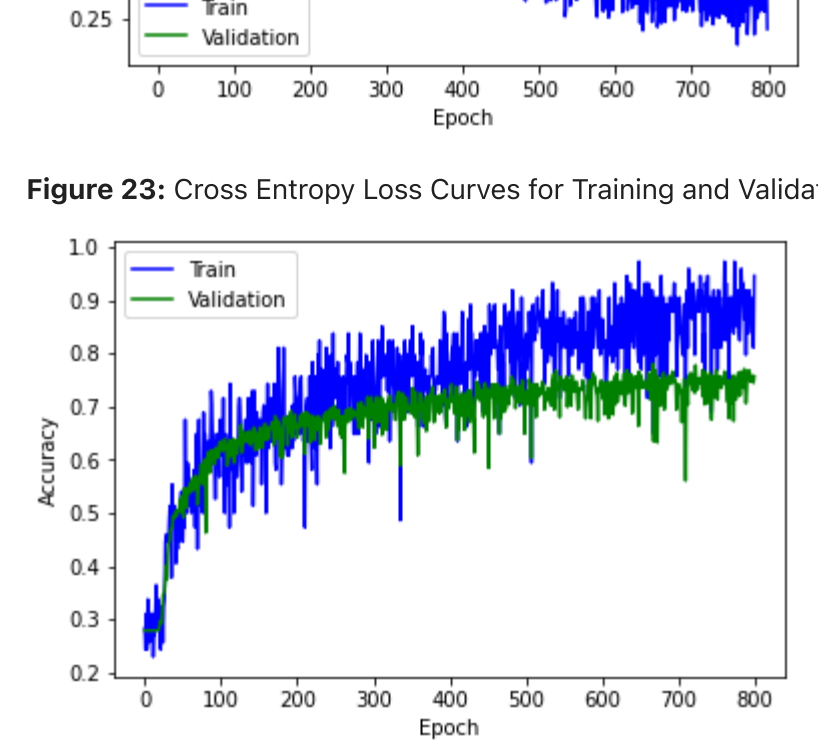


Figure 24: Accuracy Curves for Training and Validation Set using **num_hiddens = [16,20]**

Using num_hiddens = [16, 80]:

Table 13: Final Statistics when training model using learning rate = 0.01, batch size = 100, num_epochs = 800, num_hiddens = [16, 80]

Measure	Training Set	Validation Set	Test Set
Cross-Entropy	0.21647	0.84025	0.72777
Accuracy	0.93568	0.74702	0.76883

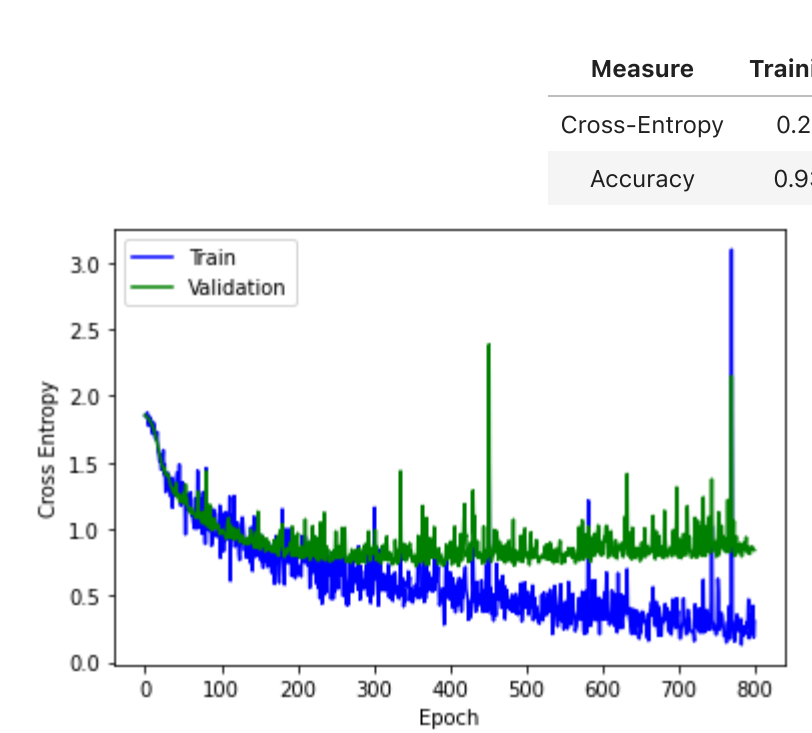


Figure 25: Cross Entropy Loss Curves for Training and Validation Set using **num_hiddens = [16,80]**

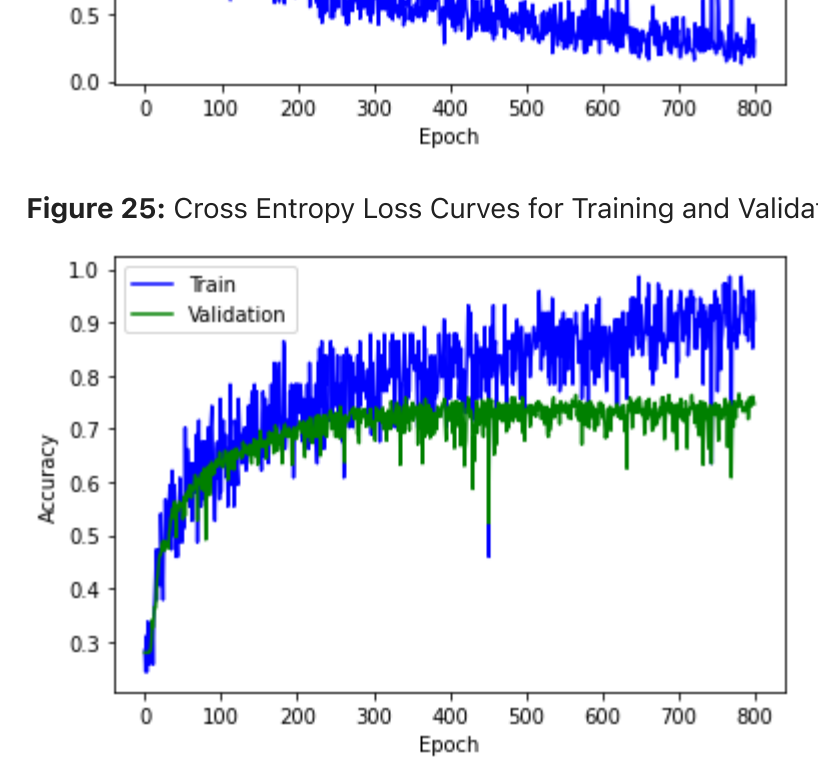


Figure 26: Accuracy Curves for Training and Validation Set using **num_hiddens = [16,80]**

Comment on the effect of this modification on the convergence properties, and the generalization of the network:

For hidden layer 1, we see from the final statistics and in the figures produced that 20 neurons **(Figure 18)** yields the highest validation accuracy of about 77%, followed by a 74% validation accuracy for 80 neurons **(Figure 20)**, and lowest validation accuracy of about 52% for 2 neurons **(Figure 16)**. The training error was the highest for 2 neurons **(Figure 16)**, followed by 20 neurons **(Figure 18)**, and 80 neurons with the lowest training error **(Figure 20)**. 2 neurons has the highest minimum validation loss of 1.30509 **(Table 8)**, followed by 80 neurons with minimum validation loss of 0.98573 **(Table 10)**, and 20 neurons with the lowest minimum validation loss of 0.86982 **(Table 9)**. Thus, 2 neurons achieves worse validation lossess than the higher number neurons, and does not converge. As we can see in **(Figure 19)**, the validation loss starts to slightly increase 400 epochs onwards for 80 neurons, which suggests that the model overfits.

Based on these statistics, 20 neurons produces the best generalization of the network as it converges to a minimum validation loss and has the highest validation accuracy.

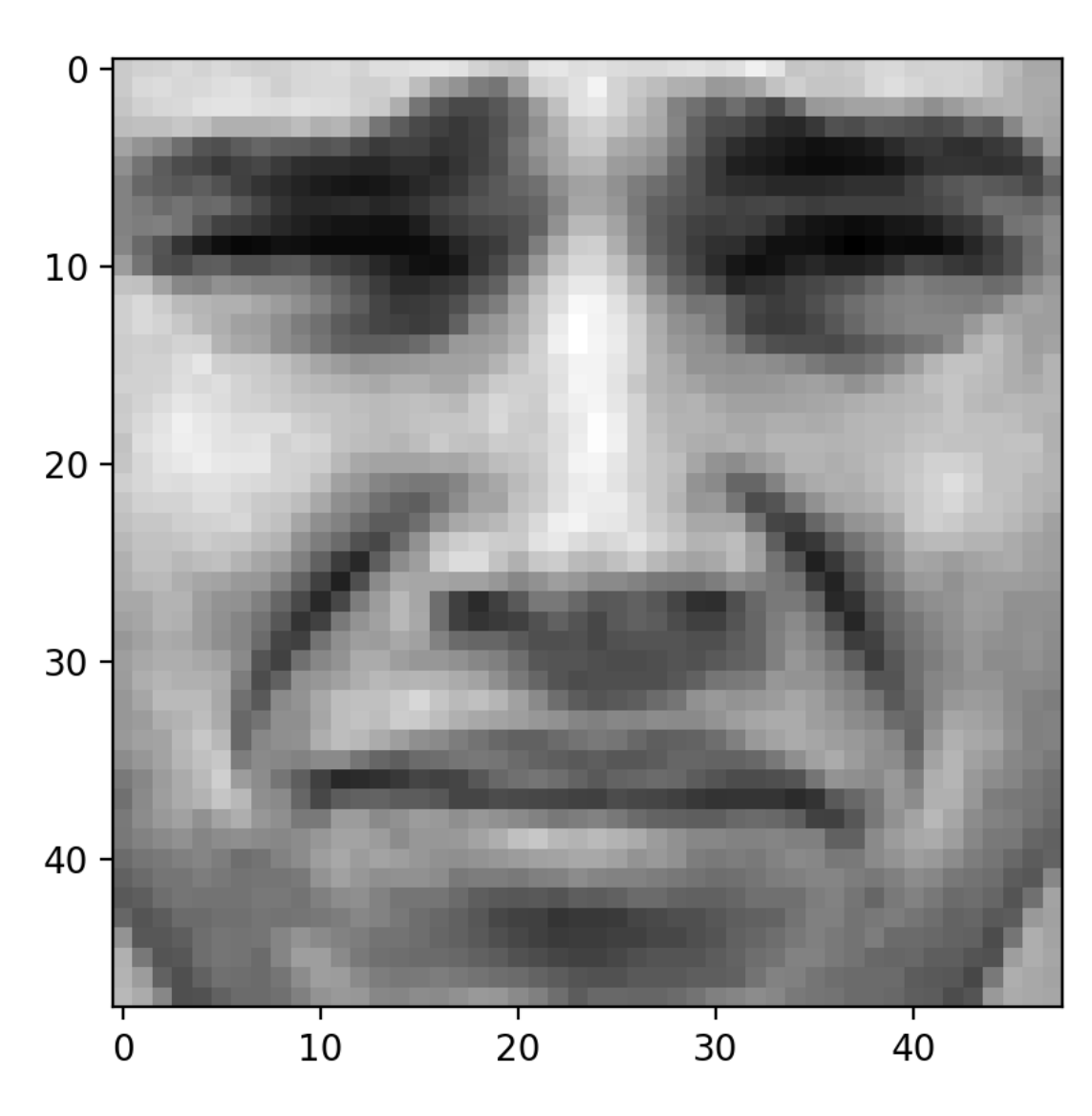
From these modifications, we can see that a small number of hidden neurons may not converge and can underfit the data, so there is poor generalization of the network. Too large of a number of hidden neurons may start to overfit the data, so it also leads to poor generalization of the network. Thus, a moderate number of hidden neurons is best to increase generalization of the network and allow convergence.

For hidden layer 2, we see from the final statistics and in the figures produced that 20 neurons **(Figure 24)** yields the highest validation accuracy of about 77%, followed by a 75% validation accuracy for 80 neurons **(Figure 26)**, and lowest validation accuracy of about 52% for 2 neurons **(Figure 22)**. The training error was the highest for 2 neurons **(Figure 22)**, followed by 20 neurons **(Figure 24)**, and 80 neurons with the lowest training error **(Figure 26)**. 2 neurons has the highest minimum validation loss of 1.32261 **(Table 11)**, followed by 80 neurons with minimum validation loss of 0.84025 **(Table 13)**, and 20 neurons with the lowest minimum validation loss of 0.77781 **(Table 12)**. Thus, 2 neurons achieves worse validation lossess than the higher number neuroons, and does not converge. As we can see in **(Figure 23)** and **(Figure 25)**, the loss converges for both 20 and 80 neurons respectively; however, since 20 neurons produces a higher validation accuracy, it provides better generalization of the network.

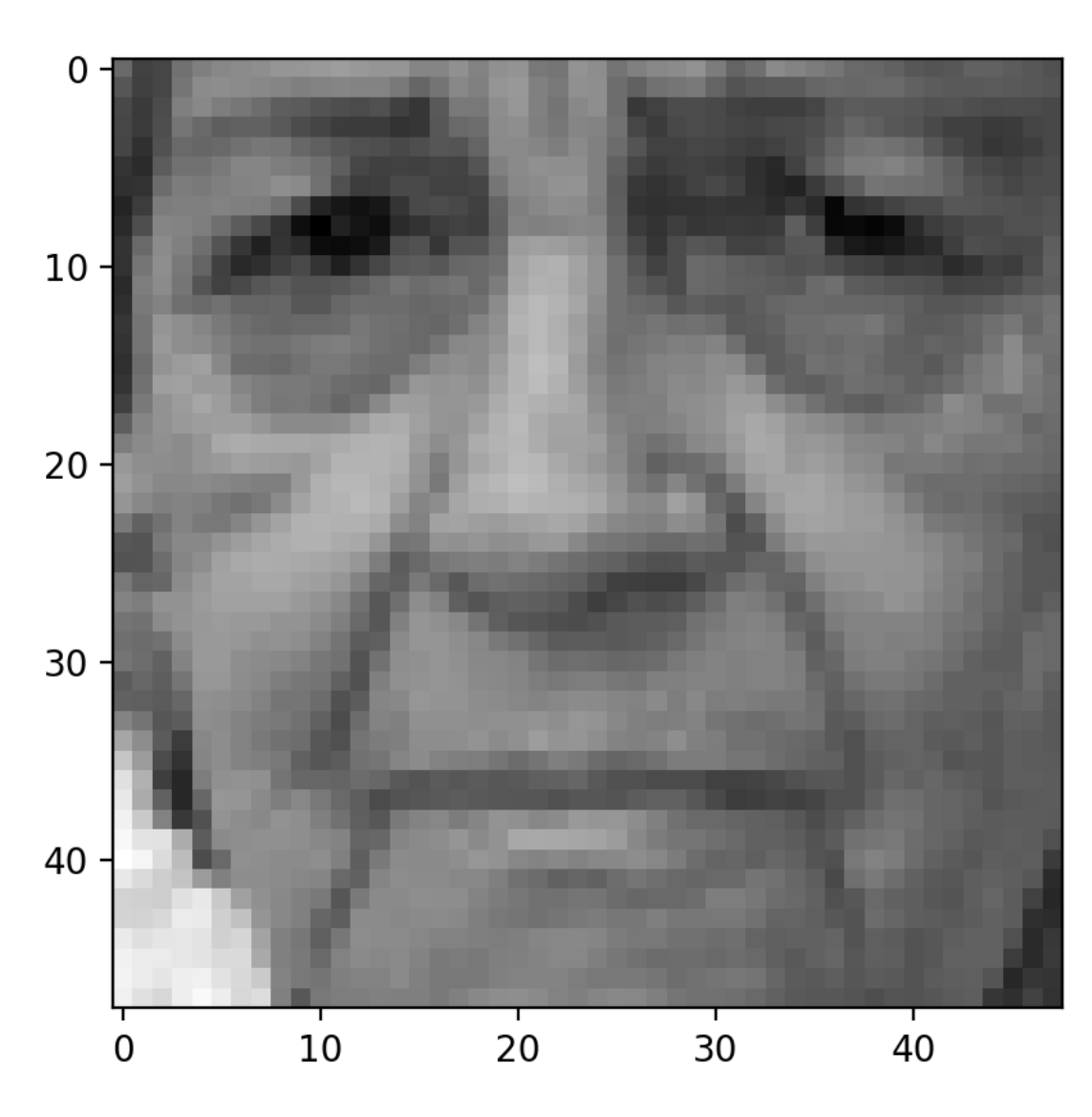
From these modifications, we see the same general results as explained for hidden layer 1 where too few neurons may not converge, and too many neurons may start to overfit the data. In this case for hidden layer 2, 80 neurons does show slight overfitting as the validation loss begins to increase slightly from 450 epochs onwards.

Q2.5: Network Uncertainty.

Sample 1:
Actual Class: 4 [Happy]
Predicted Class: 1 [Anger]



Sample 2:
Actual Class: 0 []
Predicted Class: 4 [Happy]



Sample 3:
Actual Class: 2 [Disgust]
Predicted Class: 2 [Disgust]

Sample 4:
Actual Class: 6 [Surprise]
Predicted Class: 5 [Sad]

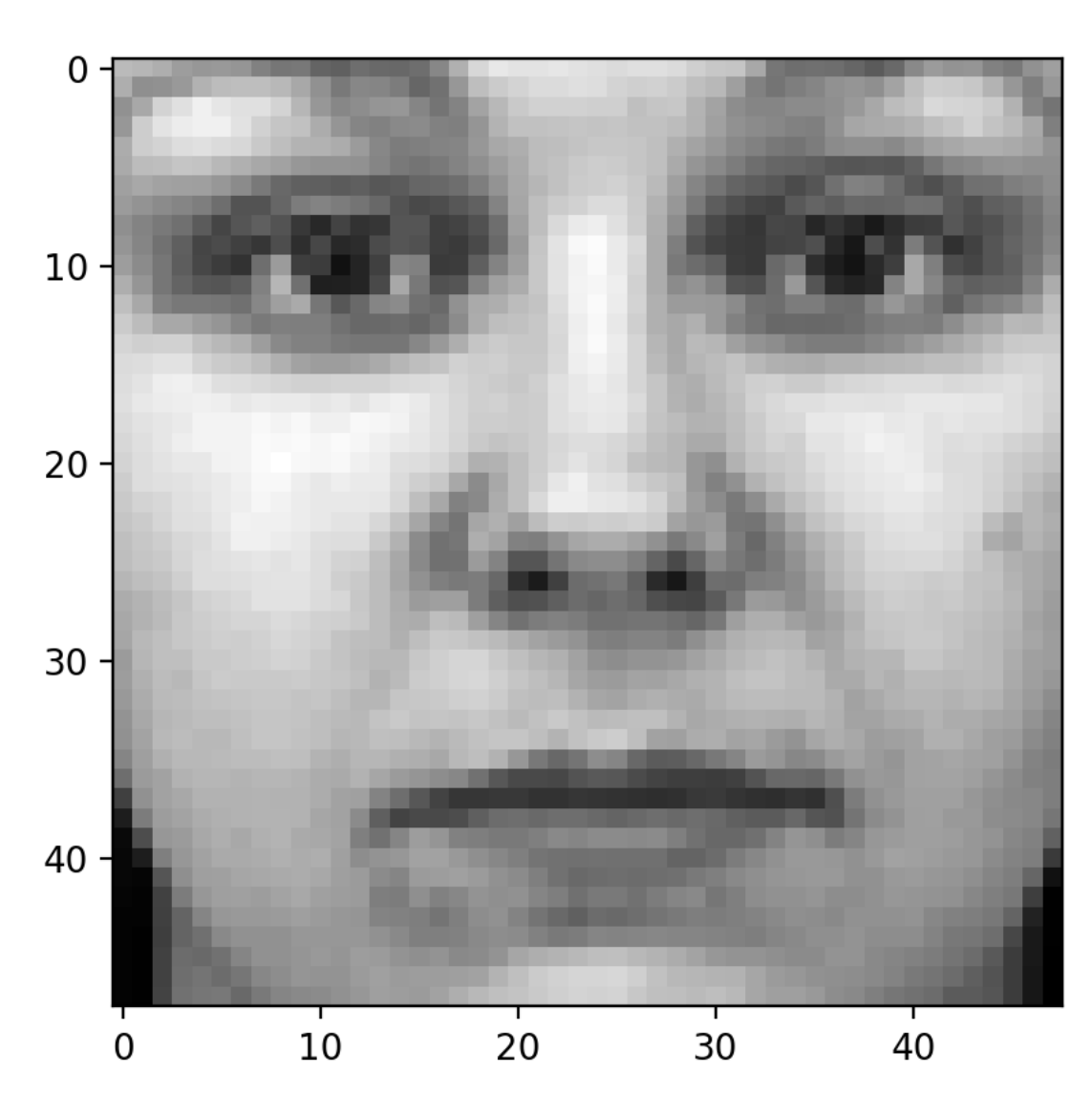


Figure 27: Plot showing four face images where neural network is not confident of the classification output. Threshold level of 0.5. Default hyperparameters used: learning rate = 0.01, num_epochs = 1000, batch_size = 100, and model architecture: [16,32]

For sample 1, the model confuses anger and happy as the image shows many face lines (usually when a person is angry there are a lot of face lines). For sample 2, the image is of an elderly person with many face lines which makes predicting the expression more difficult. For sample 4, this image is hard to distinguish for a human as well. It is unclear if the expression is suprise, sad, or neutral.

The classifier will not necessarily be correct even if it outputs the correct top scoring class since samples can be misclassifed with uncertainty as well. A better approach would be to use the top-k classes with more certainty.

With regards to generalization, it seems that this model does not generalize well with the Angry and Happy class as those classes have the highest number of uncertain predictions.