Christopher Kang
cskang2

ECE 417 MP 3: Convolutional Neural Network

1.Introduction

In this MP, we are given the problem in attempting to identify whether the image given is an elephant (represented by label = 0) or a lionfish (represented by label = 1). This is essentially a binary classification and to do this task we will be using a simple convolutional neural net (CNN) in order to help us solve this problem. Our CNN is compromised of, a convolutional layer (the layer where we apply our filters and help our network learn the corresponding features of the elephant/lionfish), a pooling layer (taking the best features obtained within a block of convolutional outputs), a flatten block (essentially just reshaping the pooled layer into the appropriate shape for the input to the next block), 2 FC blocks (connecting our input block to the hidden layer or essentially doing a matrix operation between the two), and ReLU /Sigmoid blocks (activation functions for our network).

For each epoch, we input our mini-batch into our network (forward propagation) and then through our back propagation, we will then update each of weights inputted into the network (in this case there are 3 weight values that we need to update so 3 back propagation procedure will be needed in this step). Once we did this procedure for all our batch sets, the program will then feed the test data into the network and report the loss/test accuracy for that epoch. We then repeat this procedure until we reached our desired epoch.

2. Methods

$$\mathcal{A}(n_1, n_2) = \left\{ \begin{array}{l} (m_1, m_2): \\ n_1 M \leq m_1 < (n_1 + 1)M, \\ n_2 M \leq m_2 < (n_2 + 1)M \end{array} \right\}$$

The above equation represents the max pooling layer where M in this case represents stride along with the size of our box of convolution outputs. Essentially, max pooling is simply trying to find the best feature that represents a specific region of the image. This will help us save time in our future computations as we only to consider few of the convolutional outputs rather than the entire convolution outputs on the entire region of the image. If our stride isn't too small or too big, max pooling should be able to pick out key features of the image relevant to our classification.

$$\frac{\partial}{\partial x}\left(\frac{-Y\log\left(\frac{1}{1+e^{-x}}\right)-(1-Y)\log\left(1-\frac{1}{1+e^{-x}}\right)}{N}\right)=-\frac{e^{x}(Y-1)+Y}{N(e^{x}+1)}$$

One of the major portions of the network was the back-propagation procedure. In this implementation, instead of computing dL/dsigma and dsigma/dF2, dL/dF2 was computed to help reduce the amount of lines needed in the code. The equation above represents the partial derivative of the cross-entropy loss function with respect to X (which in this case represents F2). As seen in the equation, the entire sigmoid is in the equation along with the input of F2. Additionally, the equation had to be modified in order to prevent Python from outputting an infinity value when computing e^x. So, the final equation ended up being:

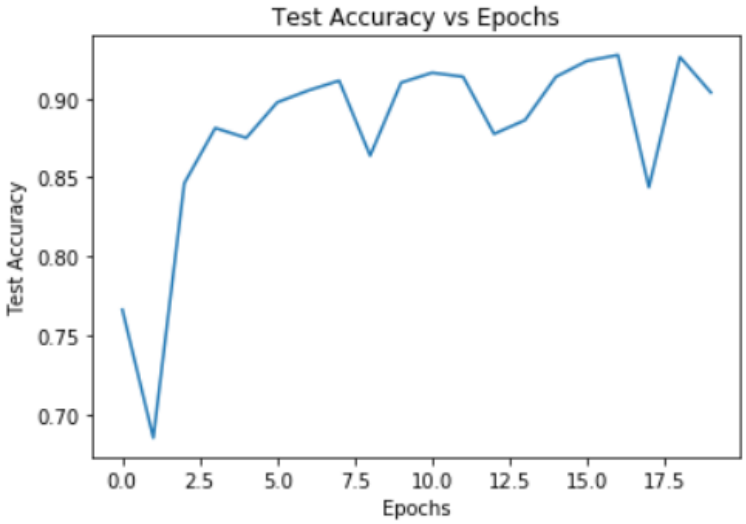
Christopher Kang
cskang2

dL / dF2 = (Y-1) + Y * e^-F2 / N (1 + e^-F2)

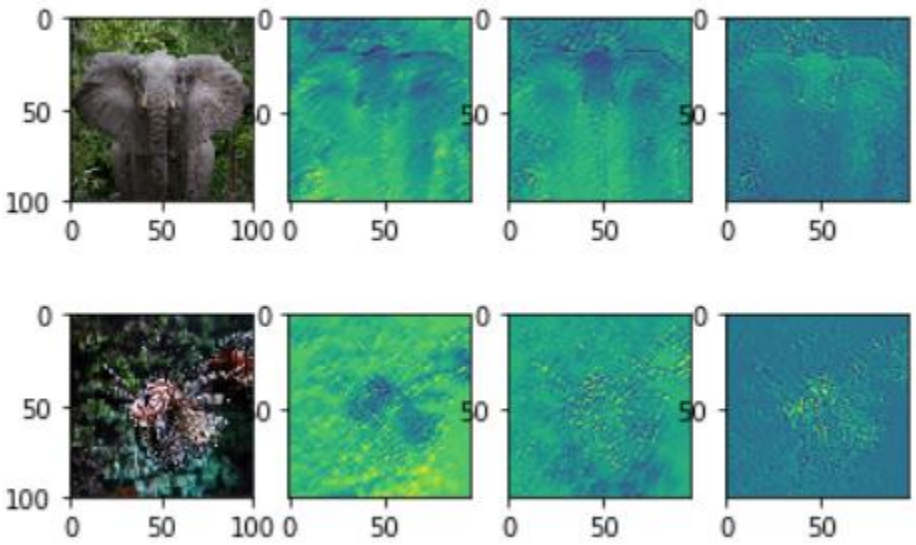
This allowed python to instead compute e^-x as 0 which doesn't affect the remaining part of the equation. The rest of the back propagation is done by simply taking the partial derivative in each block of the network until we reach to our desired weight block.

3. Results

Test Accuracy vs Epochs



Convolution Between W0 and Input Images



Christopher Kang
cskang2

Confusion Matrix

	Predicted: Elephant (0)	Predicted Lionfish (1)
Actual: Elephant (0)	92.75% (Correct)	7.25% (Incorrect)
Actual: Lionfish (1)	15.50% (Incorrect)	84.50% (Correct)

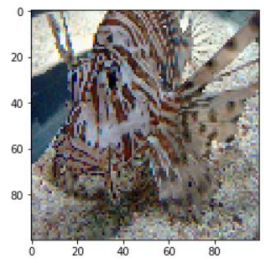
4. Discussion

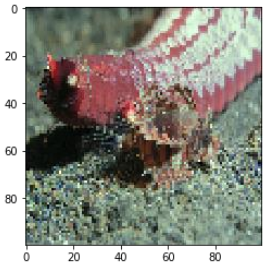
Before testing our network out, we normalized each of the pixels value in all the images (simply by just dividing by 255). Feature scaling was done in order to speed up our gradient descent convergence and additionally prevent the network from converging onto a specific region of pixels. As mentioned within the documentation, mini-batches were done by randomizing the indices and taking batch size indices.

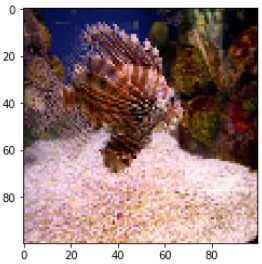
As seen in the Training Accuracy vs Epoch graph, the training accuracy varied within the first few epochs but after a couple of epochs, the training accuracy was able to stabilize between the high 80's and low 90's.

In the convolution between W0 and the input images, the features that the network attempted to learn were the outlines of each of the animals provided. Features such as the body, trunk (elephant), spike features (lion fish) was what the network seemed to learn over time as well. As seen

As shown in the confusion matrix, the network was able to accurately most of the images pertaining to elephants and lionfish. The network seemed to have trouble classifying the lionfish class. Some of the images that the network classified as elephant were







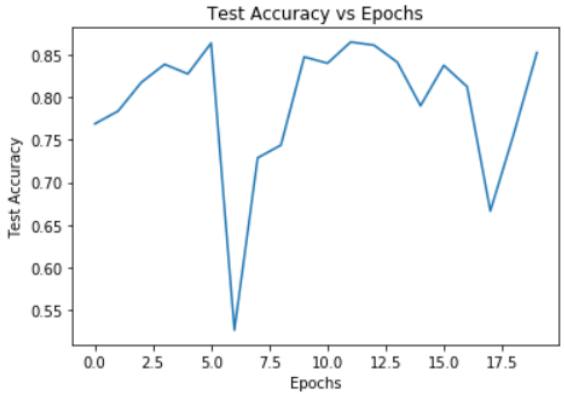
The first image could have been the network identifying the lion fish as potentially elephant ear. The second image was due to the background containing a shape like an elephant's trunk. The third image could be due to the lionfish position outlining as an elephant general outline.

5. Extra Credit

There were two methods implemented in order to tune the network: these two being L2 regularization and decaying gradient descent.

L2 regularization is a way to prevent the network from being over trained and this is accomplished by adding: Loss = Loss + lambda / 2 * m summation (Weights^2). What this is doing is making the larger weights having a greater impact to the loss compared the smaller weights. Since our loss function now

has this regularization term, we must update our gradient descent equations by adding: lambda/m * W to each of the respective weight's gradient descent calculation.

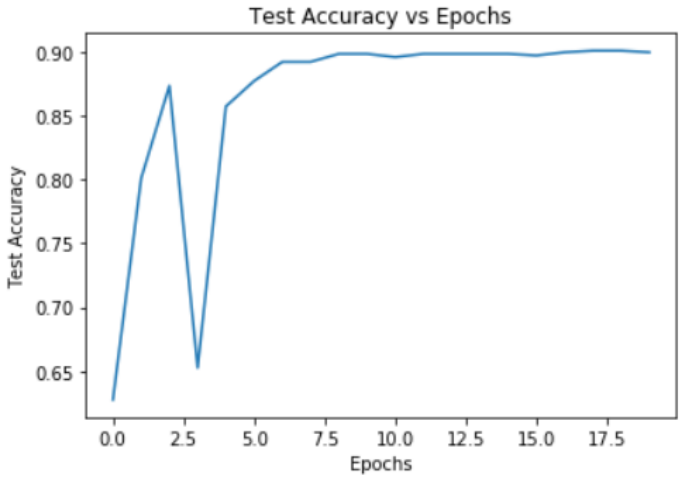


The accuracy is lower than the original network and there seems to be more dips in the test accuracy which indicates that the network is attempting to prevent the network from overfitting. Though this may also imply that our network wasn't really overfitting on any of the features and that this regularization may have not been needed to improve the network.

Decaying gradient descent is a way to have the network's learning rate be dependent on the current epoch. Since the provided test accuracy vs epoch seemed to have random drops in accuracy and has a curve similar to that of exponential graph, the learning rate was then updated by:

lr = initial lr * e^(-Beta * curr_epoch)

Where beta is our hyperparameter and initial lr is our initial learning rate.



As seen in the graph of Test Accuracy vs Epochs, there was only one dip during the training portion and the curve ended up being smoother towards the end.