Introduction

The problem that I'm working on in this project is using a convolutional neural network, which is used to identify images in order to try and correctly predict which pictures of human skin have cancer. The pictures are split into cancerous and non-cancerous and mostly contain pictures that look like bruises and lesions. Identifying cancerous growths on the skin is an extremely important issue because early detection can be the difference maker between living and dying or living and living with restrictions. Throughout their lifetimes, people overall have about a 40% chance to develop cancer and being able to have a model that predicts if skin in a picture has cancer with great accuracy can be extremely helpful in improving lifespans and overall quality of life.

Analysis

This data was solely pictures that are pretty easy to discern what they are with the human eye, so they didn't require any manual analysis, especially since I'm not knowledgeable at all on what cancer looks like. As for cleaning through code, the first thing I did was create a function using the os module in order to take images from their various file paths and put them into a dataframe with the appropriate cancer or non cancer labels. In my head it seemed much simpler but the actual code for it took me longer than I would have liked to figure out.

For preprocessing and formatting all the data I used the imagedatagenerator package from keras. It is super helpful because you can do rescaling and setting various parameters for augmentation all within the same step. Various parameters for the flow from the dataframe function I used determine potential rotations of the image, shifts either going left or right to leave out some of the image, zooming in or out among other things. The purpose of is to artificially make the data more varied which can help the model learn better and predict better

Methods

For my own neural network I used various layers and each one had a purpose within training the model to identify if within a picture there is skin cancer. The convolutional layers work to make it so the model can detect features within the photos that are important such as edges and texture within pictures and these are important because size, shape and texture shown on the pictures are significant to detecting if skin cancer is present or not. The relu activation function in every convolutional layer works by outputting 0 whenever a number less than 0 is inputting and by outputting whatever the number is if the output is >0. What this does is introduce non-linearity and helps the network to form complex relationships. Batch normalization helps by normalizing the data and essentially helping the neural network train faster than if we didn't use it. The max pooling layer downsampled the images and helps the network recognize features better. The flattening layers flatten a two dimensional summary of the features into one dimension as some of the layers require a one dimensional vector. The dense layers assist the convolutional layers by further processing the data and relationships discovered. The dropout layer is interesting because

what it does, denoted by the .2 is that for any data point, there is a 20% chance it is changed to zero and not considered.

While this sounds counterproductive, this actually helps to reduce overfitting which is when the model performs better on data that it has already seen or training data, compared to data that it hasn't seen yet or testing data. This is important because when we create any sort of machine learning model we want it to be able to perform equally well on unseen and seen data. Finally, there is a single neuron at the end of the network with the sigmoid activation, which is best suited for binary problems as we are trying to determine cancerous or non cancerous, that is our best choice.
For the fine tuned model, after doing some reading around I decided to use mobilenetV2. A big part of why I chose this model was because it was optimized to run on mobile devices, so I sort of assumed it would be very well optimized while still getting results and that was exactly what happened.

When using a fine tuned model, I essentially download a pre-trained machine learning model from another source and essentially remove the "head" or the part of the model solely responsible for predictions and we replace it with what we think is the best thing for the specific problem we are trying to solve. I used global average pooling, which essentially averages values over various feature maps because I know a model like this is very complex because it is trained on 14 million images and the problem I am trying to solve is rather simple. Then I have a dense layer, just like the other model with relu activation and the purpose of that layer is to learn as much as possible from the averages of feature maps done in the previous layer. Finally, there is a layer with a single neuron and sigmoid activation for prediction, exactly the same as in my handmade neural network.

For both models I used binary cross entropy as the loss because that is the most appropriate one for a binary classification problem and I used the Adam optimizer because it tends to converge in the quickest amount of time and I knew that since I didn't have too many samples that was very important

Results

Overall, both models performed very similarly, but this doesn't mean they're equal. It took my handmade model 50 epochs to reach the same performance; the fine tuned model only took 10 epochs to achieve similar measures of accuracy. Additionally my handmade model took more than five times longer to train than the fine tuned one, which means that mine is much less efficient and took five times as long to converge. Regardless of their epochs, both models performed in the low 90% and had loss of around .25 which I am pretty satisfied with. I tested training my handmade model for only ten epochs and could get a maximum of 80% accuracy and pretty high loss, which tells me a lot about the significant time and effort people who make machine learning models need to put in.

Reflection

My biggest problem for this assignment was creating my handmade model so it works well, doesn't take a ton of epochs to converge and so it isn't super expensive. I've struggled with this basically since we started working with neural networks and from last project, where I learned less can be more, I tried to apply that to this one and that didn't really work which sort of led to me just doing guess and check work, which led to me taking so long on this assignment. With every major take home assignment I feel like I've ran into this issue and from this time what I learned is that image detection and classification is inherently complex regardless of how simple or complicated the images themselves may seem.