Justin Teufel

HW 5 - Histopathologic Cancer Detection

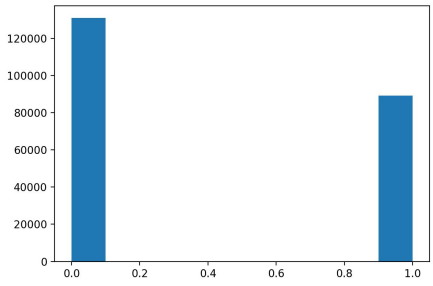
https://github.com/jteufel/histopathologic-cancer-detection

Description

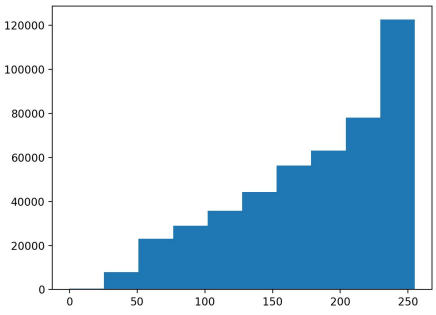
In the Histopathologic Cancer Detection dataset, we are provided with a large number of small pathology images to classify into one of two categories - whether or not it contains a pixel of tumor tissue. We are provided with a set of images paired with the correct classifications to use for training a model of our choosing. The training dataset consists of approximately 220,000 samples, while the testing set consists of approximately 57,000 images to classify.

Exploratory Data Analysis

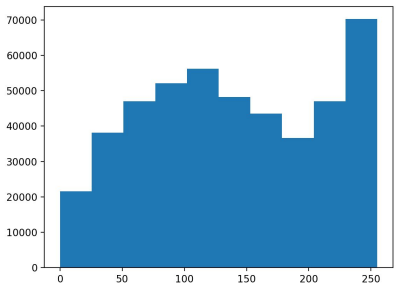
I used the exploratory Data Analysis as an opportunity to pull some quick metrics from the dataset, such as the distribution of the two classifications, as well as the prevalence of certain RGB pixel values. Below is the distribution of image classifications for the training set:



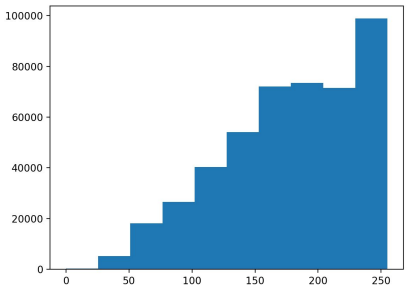
R Pixel Distribution



G Pixel Distribution



B Pixel Distribution



Architecture

For my predictive model, I chose to use a Convolutional Neural Network (CNN). A CNN consists of a series of *convolutional layers* - at each of these layers, a series of multiplication (dot-product) operations are performed on a tensor (initially consisting of the image pixel values) using a *filter*. *Pooling layers* can be placed in-between the convolution layers and allow for a reduction in the output size of the convolution layer. The output of the convolution process is an abstracted tensor in which the 'shape' is reduced. Convolutional layers work well for image processing because the process of convoluting pixel data allows for patterns in an image to be recognized, while the complexity/size of the data can be reduced. After the convolution layers, the convolved data can be passed into fully connected 'dense' layers, from which the final classification is output.

My final CNN model consisted of 2 convolutional layers with 1 pooling layer in between them, proceeded by 2 'dense' layers - the first of with a

ReLU activation function applied to the output of the first. Below is a more detailed description of each layer:

1. **Convolution Layer 1:**
    - Convolutional layer takes an input of size 96,96,3 (image dimensions) and uses a size 3,3 filter for the convolution. The output is size 94,94,16 - the 16 corresponds to the number of channels.
2. **Pooling Layer:**
    - The pooling layer takes the output of the convolution layer and reduces it using a 2,2 filter. The output is of size 47,47,16.
3. **Convolution Layer 2:**
    - A second convolution layer is applied to the output of the pooling layer - it also uses a 3,3 filter but produces 32 output channels, so the resulting data is of size 47,47,32.
4. **Flattening Layer:**
    - Reshapes the output data to a single array, for processing in the dense layer. Output is an array of size 64800.
5. **Dense Layer with ReLU Activation:**
    - Consists of a dense layer and an activation layer. The dense layer consists of 32 ‘nodes’, with connections to each of the 64800 inputs (for a total of 2073632 parameters). The hidden activation layer then applies a rectified linear unit (ReLU) function to calculate the weighted outputs.
6. **Dense Layer 2:**
    - The final dense layer consists of 10 nodes from the 32 input nodes (for a total of 330 parameters).

Results

The final CNN architecture was the result of several train-test iterations on a reduced portion of the dataset (5000 sample images). Each iteration training iteration consisted 20 Epochs.

Iteration 1:

My first model consisted of 4 convolution layers with a pooling layer between each (3 total), along with the 2 dense layers at the end:

Layer	**Output Shape**	**Param #**
(Conv2D)	(94, 94, 16)	448
(MaxPooling2D)	(47, 47, 16)	0
(Conv2D)	(45, 45, 32)	4640
(MaxPooling2D)	(22, 22, 32)	0
(Conv2D)	(20, 20, 64)	18496
(MaxPooling2D)	(10, 10, 64)	0
(Conv2D)	(8, 8, 128)	73856
(Flatten)	(8192)	0
(Dense)	(32)	262176
(Dense)	(10)	330

Results:

Epoch 1/20 resulted in an accuracy of **.5856**. Epoch 10/20 resulted in an accuracy of **.6038**. Epoch 20/20 resulted in an accuracy of **.6040**. Output time was **416 seconds**.

Iteration 2:

My second model consisted of 3 convolution layers with a pooling layer between each (2 total), along with the 2 dense layers at the end:

Layer	**Output Shape**	**Param #**
(Conv2D)	(94, 94, 16)	448
(MaxPooling2D)	(47, 47, 16)	0
(Conv2D)	(45, 45, 32)	4640

(MaxPooling2D)	(22, 22, 32)	0
(Conv2D)	(20, 20, 64)	18496
(Flatten)	(25600)	0
(Dense)	(32)	819232
(Dense)	(10)	330

Results:

Epoch 1/20 resulted in an accuracy of **.5856**. Epoch 10/20 resulted in an accuracy of **.7238**. Epoch 20/20 resulted in an accuracy of **.9790**. Output time was **380 seconds**.

Iteration 3:

My third model consisted of 2 convolution layers with a single pooling layer between each, along with the 2 dense layers at the end:

Layer	**Output Shape**	**Param #**
(Conv2D)	(94, 94, 16)	448
(MaxPooling2D)	(47, 47, 16)	0
(Conv2D)	(45, 45, 32)	4640
(Flatten)	(64800)	0
(Dense)	(32)	2073632
(Dense)	(10)	330

Results:

Epoch 1/20 resulted in an accuracy of **.5688**. Epoch 10/20 resulted in an accuracy of **.9936**. Epoch 20/20 resulted in an accuracy of **.9982**. Output time was **347 seconds**.

Based on the results outlined above, the gradual removal of the convolution layers resulted in a significant rise in testing accuracy. This is likely due to the increased input size into the first dense layer (2073632 parameters in the final iteration as supposed to 262176 parameters in the first iteration). In the first and second iterations, with the additional convolution layers, we decided to gradually increase the output channels in an attempt to allow for more pattern recognition opportunities. This however did not result in any noticeable accuracy increases, and also seemed to increase the fitting time.

To see the competition results, please follow the GitHub link above.

Conclusion

The final model shows that Convolutional Neural Networks achieve very high levels of accuracy when used for image recognition. However, too much convolution can result in lower levels of accuracy.