Project 2

Image Classification with Convolutional Neural Networks

David Huson Joel Ward

April 16, 2023

Abstract

Computer vision is a popular field in today's world. From facial recognition to object detection and everything in between, they all use some form of computer vision. In this paper we will focus on a specific subfield in computer vision, namely Image Classification. The dominant method of performing image classification is to use some form of a convolutional network. In this paper we will explore many properties of such a network to determine how altering each property affects the performance of the image classification model.

1. Problem Description

For AI to grow to become a benefit to society in everyday life, a key component for many agents is in their ability to differentiate between different types of objects when using a camera as a source of perception. This project attempts to solve this problem for a specific scenario - being able to distinguish if an image contains a human, or a horse (that is to say, *classify* an image). To demonstrate our solution, an image is pulled from a dataset of artificially generated pictures of both horses and humans [5], and the AI model will run over these images and attempt to classify them accordingly. This will allow a user to conduct experiments in the viability of basic image recognition concepts.

2. Solution Method

The common method to solve the problem of image classification is to use what is known as a Convolutional Neural Network (CNN). Likewise, we also researched this approach.

2.1 Convolutional Neural Networks

A CNN is exactly what it sounds like, a neural network where at least one of the layers performs a mathematical operation known as a convolution. In CNNs we use discrete convolutions, which takes the input as a matrix and passes a smaller matrix over it (usually referred to as the kernel) [2]. It then performs a matrix multiplication between the kernel and the portion of the input matrix that overlaps the kernel. The values in the kernel have a distinct effect on the features which are extracted from the input matrix. A diagram of this is shown below.

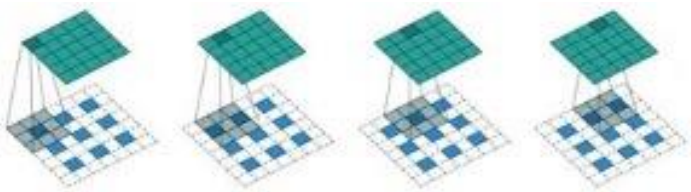

Figure 1. Here the green grid is the activation layer and the grid below it is the input to a convolution or pooling layer. We note that for each cell in the activation layer, there is a shadow over a portion of the input layer. This shadow is the kernel.

2.2 Our Method

For our project we chose the default kernel values provided by the Keras library [1], however these values can have a significant impact on the outcome of a convolutional layer. For instance, a Gaussian kernel will apply a Gaussian blur to the photo. There are a number of hyperparameters we used Keras Tuner [6] to find optimized values for, such as the number of filters (kernels) in a convolutional layer (Fig. 1.4) , stride of the pooling layer (Fig 1.3) , the size of the kernel (Fig. 1.5) and the number of convolutional layers themselves (Fig 1.1). All these results will be discussed in the next section. In summary we used a specific configuration of convolutional layers, pooling layers, and dense layers with optimized hyperparameters to achieve a successful image classification result. These pooling layers are responsible for feature extraction. They do this by passing a kernel over some input matrix and extracting the maximum value in that kernel and passing it to the activation layer. All our convolution layers use a ReLU activation function, and our output layer uses the sigmoid activation function.

2.3 Our Architecture

After running the trials with Keras Tuner as outlined above, we landed on the following architecture. We used a combination of convolution layers in series with max pooling layers, finally ending with a fully connected layer (dense layer) which leads into the output layer, as shown in Figure 2 below. This is loosely based on the architecture described in the Google Developers ML Practicum on Machine Learning [3].

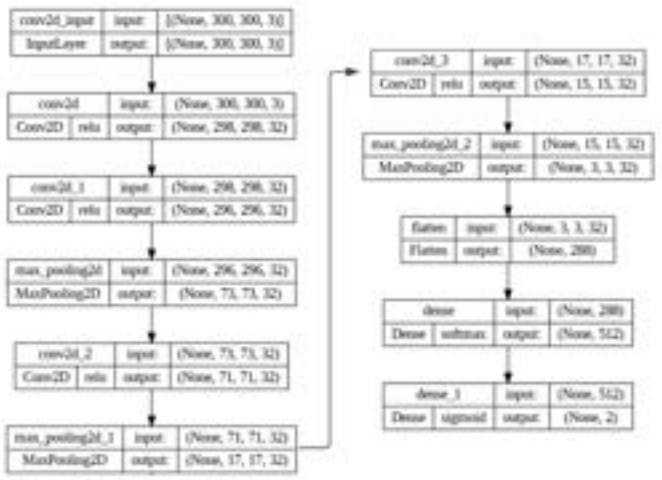

Figure 2. Chart of the best architecture found by experimentation.

3. Description of Results

After running multiple experiments, each examining the effect of changing a singular parameter of our model, we found some interesting results. All our experiments were run under the assumption that the most important metric of the outcome of a singular model is its accuracy when run over the test dataset. Each of the various models first trained against the same dataset and were then run against the same validation data set. Thus, the accuracies represented below represent as close to real-world, practical results as possible.

Initially, we experimented with the number of Convolutional Layers. From our testing, we found that the number of layers that resulted in the most accurate results was 6 Convolutional Layers. The remaining trials we ran, as well as their respective accuracies can be seen below, in Figure 3.

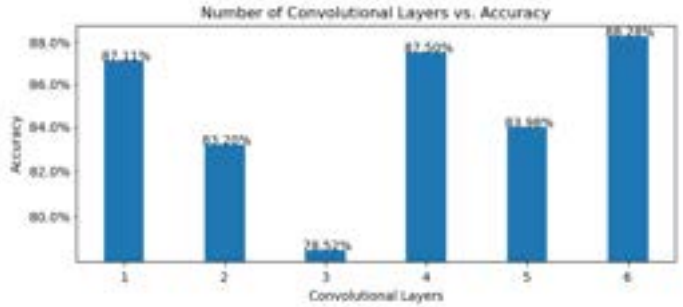

Figure 3. A graph showing the accuracy of the model on the validation data as a function of the number of convolutional layers.

Next, we tested various kernel sizes with respect to pooling layers versus the resulting accuracies. For this experiment, we found the most accurate Max Pooling kernel size to have been 6, though the potential for other kernel sizes to overtake that result exists under certain conditions. These other iterations may be seen in Figure 4.

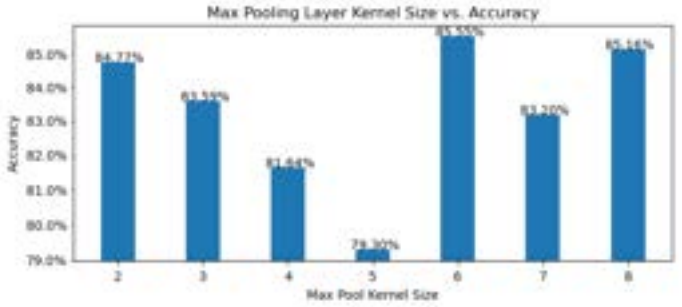

Figure 4. The graph showing validation accuracy as a function of the size of the max pooling layer's kernel.

On a similar front, we also explored pooling layer strides. Our results showed that a kernel stride for max pooling layers of 5 to have been the most accurate, though the data trends seem to indicate further performance improvements may have been obtained by continuing to increase the stride. This data trend is exemplified in Figure 5.

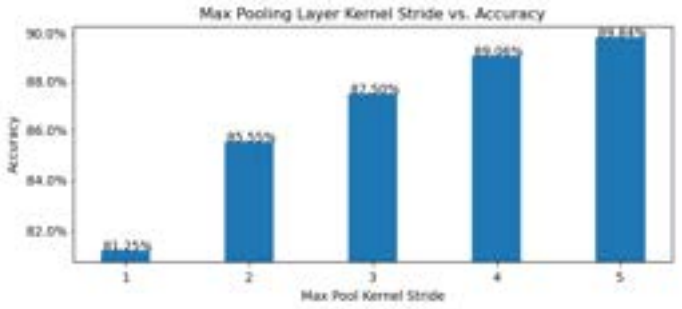

Figure 5. The graph showing the validation accuracy as a function of the Max pooling layer's kernel stride. The stride is the number of pixels the kernel will move at each iteration.

We then tested altering the number of filters per layer, which is the dimensionality of the output space for that layer. For this we found the best configuration was an output dimension with 32 channels. As shown in Figure 6 below.

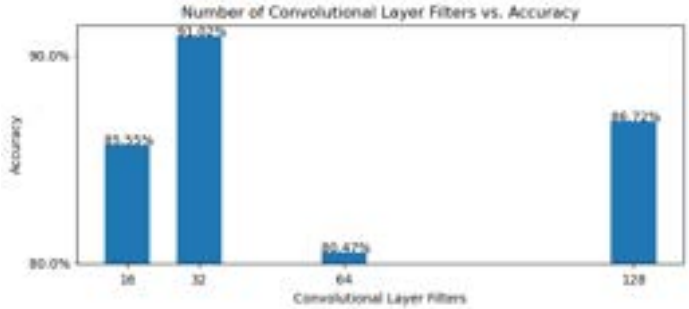

Figure 6. The graph showing validation accuracy as a function of the dimensionality of the output space.

Also, we ran trials to find the best kernel size for the Convolutional Layers. Interestingly, our results showed that the most optimal kernel size was 3. This kernel size produced a greater accuracy by a notably large factor. See Figure 7 below for the exact margins.

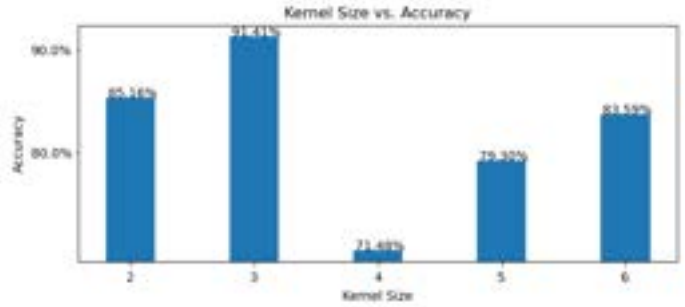

Figure 7. The graph of the validation accuracy as a function of the convolutional layer's kernel size.

Lasty, we experimented to find what size dense layer before the output layer provided the best validation accuracy. We ran tests on layer sizes starting with a size equal to that of the output layer (two) and increased my powers of two until we reached a layer size of 1024 nodes. After all trials were completed, we found the best configuration to be a size of 512 nodes. See Figure 8 below for more details.

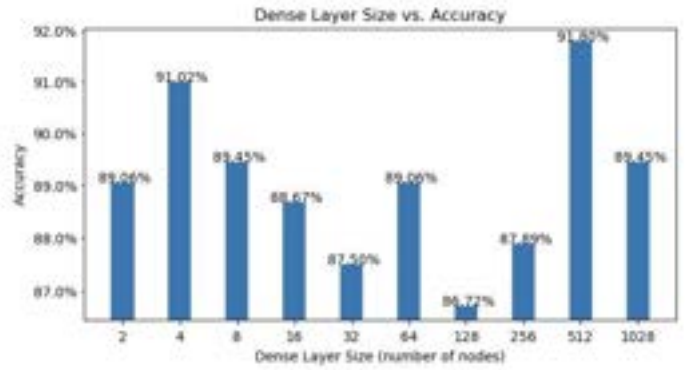

Figure 8. The graph of the validation accuracy as a function of the number of nodes in the dense layer before the output layer.

4. Conclusion

In this paper, we explored the impacts of tweaking various parameters of a convolutional neural network to find the maximum accuracy over a basic image classification problem.

4.1 Our Results

Our best model boasts an accuracy of 91.80% on the validation dataset, which is quite strong in a vacuum. However, in testing the final model with data from the internet, we found some weaknesses in it. These weaknesses can largely be attributed to the size of our training dataset, which only has 1027 training images (500 horses, 527 humans). There also exists a bias in our training data. Namely, the dataset is biased towards full body images of humans. Due to this bias, if given an image of a human in which only a portion of the body is visible, say their head, it will misclassify the image. Which leads us to ponder how we can improve the accuracy of our model on more general data.

4.2 Next Steps

Our data shows that the single variable that makes the most impact when changed is the Kernel Size of one or more convolutional layer(s). Our experiments focused on the effects of changing a single parameter at a time, however it may be beneficial to allow the tuning of several parameters simultaneously to find the best overall configuration. More excitingly, experimenting with various boosting methods may also lead to improvements.

Additionally, the dataset we tested with was relatively small, and improvements to accuracies could likely be seen when performing data augmentation to boost the number of training data points.

Furthermore, our current model is only designed to support a specific size and shape for input images. It would be much more beneficial to the average application of a CNN to allow a variety of image sizes to be used.

And finally, a larger and more varied training dataset would likely see substantial gains in accuracy. This could be achieved by open sourcing the dataset to allow input from the general public, as well as mining data from the internet.

References

[1] K*eras API reference*. (n.d.). https://keras.io/api/

[2] Lacroix, S. F., & Peters, A. *Convolution*. Wikipedia. https://en.wikipedia.org/wiki/Convolution, 2023

[3] Google Developers. *ML Practicum: Image Classification | Machine Learning*. https://developers.google.com/machine-learning/practica/image-classification

[4] Szeliski, R. *Computer Vision: Algorithms and Applications*, 2022

[5] *TensorFlow Dataset Catalog*. TensorFlow. https://www.tensorflow.org/datasets/catalog/horses_or_humans, 2022

[6] O'Malley, Tom, Bursztein, Elie, Long, James, Chollet, François, Jin, Haifeng, Invernizzi, Luca et al., KerasTuner, https://github.com/keras-team/keras-tuner, 2019