CECS 456 Final Project Report: Animal Pictures of 10 Different Categories Taken From Google Images

Jorge Gutierrez
California State University, Long Beach
jorge.gutierrez01@student.csulb.edu

May 15, 2024

GitHub Link
https://github.com/SquidCephalopod/CECS-456-Final-Project/blob/main/CECS_456_Final_Project.ipynb

1 Introduction

The classification learning task is used in machine learning applications to determine which discrete category a certain example belongs to. This learning task, specifically the multiclass classification task, is used in this project to help identify which animal is in a provided picture example. I do not use the other classification task, binary classification, since this dataset consists of multiple classes and not just two classes. The goal of this project is to design and train a deep learning model using an animal dataset that contains pictures of different categories of animals taken from google images. A neural network sequential model is used to do so, along with splitting the dataset into a training and validation subset, as well as other methods and features which will be discussed in the following sections. All of these are brought together to allow the deep learning model to use multiclass classification to predict what the animal is in the picture correctly.

2 Dataset and Related Work

The dataset we will be using is the Animals-10 (Animal Pictures of 10 Different Categories Taken from Google Images) dataset, which is a dataset that contains about 28k medium quality animal images belonging to 10 categories: dog, cat, horse, spider, butterfly, chicken, sheep, cow, squirrel, and elephant. The image count for each category varies from 2,000 to 5,000 images. All of these images have been collected from “google images” and have all been checked by a human. This dataset has also been used to test different image recognition networks, which had results of about 80% accuracy for some convolutional neural networks and 98% accuracy for Google Inception. One issue that I found with this dataset however, is that the image counts for each category have a large range, which could easily result in a skewed dataset. This is something that is not wanted when training a deep learning model since it can result in less accuracy due to the model having a bias for those classes with more images. For example, if the model is unsure what animal is in the picture, with the skewed dataset it will just assume it is from the class with the most images due to that being the more probable answer, even if it is incorrect. This would result in the model

favoring the classes with more images compared to those with less images. I overcame this by using data standardization through the inclusion of a normalization layer, which is a technique used in machine learning to ensure that the dataset has a consistent distribution through shifting and scaling inputs, resulting in more stability and generalization.

3 Methodology

I decided to create a simple classifier that uses a data standardization layer, a data augmentation layer, convolutional layers, max pooling, and dropout. My simple sequential convolutional neural network model consists first of a data augmentation layer, which is used to modify the already existing data in the dataset to create even more data to train the model with and reduce overfitting. This is followed by the normalization layer to rescale the dataset to make sure it is as evenly distributed as possible, preventing a skewed dataset. Next is the first 2D convolutional layer that has a filter size of 3x3, 16 filters, no padding, and ReLU as the activation function. After this is the first 2D max pooling layer, which involves taking the maximum values of each filter to extract important features and reduce the dimensions of the data. Next is the second convolutional layer with a filter size of 3x3, 32 filters, no padding, and ReLU as the activation function. Again follows another max pooling layer. Then the third and final convolutional layer comes with a filter size of 3x3, 64 filters, no padding, and ReLU as the activation function. After this is the third and final max pooling layer as well. After this follows a dropout layer with the value “0.2”, meaning that this layer drops a fifth of the given units, which would help solve any overfitting issues the model may have. Before the fully connected layers comes the flatten layer, which acts as a bridge between the convolutional/pooling layers, which extract features, and the fully connected layers, which perform the classification task. The last two layers are the fully connected layers used to perform the classification task. When it comes to compiling the model, the “adam” optimizer was used, short for “Adaptive Moment Estimation” which minimizes the loss function during training, the sparse categorical cross-entropy loss was used, and accuracy was used as the metrics.

4 Experimental Setup

Since the dataset was not already split up into training and validation sets, I designated 20% of the training data as validation data. It was important to sample the dataset since the distribution was skewed, so I took 20% of the images of each class. The experiments were run using the “adam” optimizer, with a batch size of 64. To train this model, the entire network as a whole was trained for 10 epochs using both the training and validation datasets. The model was tested on a random entry from the dataset to make sure that it could correctly identify an animal from a picture, and it was able to do so on the first try, identifying a horse correctly. The model was then tested with a data augmented image as well, to make sure that the “data_augmentation” layer was working like it was supposed to and that the model was able to identify an animal, even if the image was upside down. The model was also able

to successfully identify an animal from an augmented image. From here, I went on to actual experimentations by testing the model with 25 randomly selected images from the dataset. The results of these experiments can be seen in the image below and will be further discussed in the following "Measurements" section. For your information, the images appear as they do below because there was a compatibility issue with the image sizes to display them. Without this issue, we would be able to see images of animals clearly. However, this does not interfere with the model, it just interferes with displaying the images correctly to the user.

5 Measurements

In the results seen in the image above, you can see that the model seems to get about half of the images correct and the other half incorrect (the label "Class" is the expected output and the label "Pred" is the model's prediction). Although this may seem bad at first, it actually accurately depicts our model. This is because when we take a look at the "Evaluation" portion of our model, where we evaluated the accuracy and loss of our model, we can see that the training accuracy is about 67%, the validation accuracy is about 67% too, the training loss is about 99%, and that the validation loss is also about 99%. When it comes to predicting images, the model has about a 67% chance of predicting the animal in the image correctly. This may not be the best yes, but it lines up with why the output image shows the model getting about half of the predictions right and the other half wrong.

6 Results Analysis, Intuitions, and Comparison

As stated in the section before, the 67% accuracy results are not the best when other models exist that can have a 99% accuracy measurement. There can be multiple reasons as to why the model's accuracy is not as high as it could be, and I believe some of those could be that the model was not trained through enough epochs, more training data is needed, the wrong optimizer or loss function was used, or perhaps the entire model as a whole is not the most efficient model for this specific use case. Something that could cause lower accuracy however, that I know is not the issue, is that the dataset is not skewed, showing the data standardization layer working as it is supposed to. The reason I know this is because from the predictions that are incorrect, there is not a common animal that the model seems to resort to, meaning that there isn't one class overpowering the rest. I also don't believe that increasing the amount of epochs would increase the accuracy since the last few epochs seemed to show the

accuracy starting to plateau, meaning that it was not going to keep increasing with more epochs and it was going to stagnate at around 67% for any more epochs. Needing more training data could've been the issue for the low accuracy, which could be figured out by training the model again with a bigger piece of the dataset or a bigger dataset overall. However, ultimately I believe it comes down to the neural network model I chose to go with for this use case. There is most likely another model I could've gone with that could've resulted in more accuracy. However, having this completed model, I can now use it as a means of comparison and as a benchmark. With this threshold to beat, I can try other models that I believe would be better suited for this use case, and I would be able to determine if they are or not depending on if they result in more accuracy than this current model.

7 Conclusion

It is important to be knowledgeable of as many deep learning models as possible before tackling a project like this. The reason being is because an incorrect model can be selected, or a weaker model as in my case, which can result in more drastic issues than getting the animal in a picture incorrect. Take the "detecting a rare disease" example from class for example. Selecting an incorrect model and/or model parameters for that use case can result in some positive cases going by unnoticed as negative cases, which can be severely harmful to someone. Completing this project helped me realize the importance of the multiple amounts of models, which is similar to why there are multiple types of algorithms, since not one model can be used for every use case. However, in doing research to complete this project, I did find out that there are some pre-trained models that can be applied to your specific use case. Although this may seem like there are universal models, these pre-trained models still have to be altered to fit specifically to what you need it to do, again enforcing the importance of the multiple models and the necessity to pick the correct one.

8 Contributions

This project was completed by myself.