Troy Waters

Student ID # 029451303

CECS 456 Section 03

Github: https://github.com/EnragedWaters/CECS-456-Final-Project
Topic: Project # 4 - 10 Categories of Animal Pictures

Introduction

The focus of this project was developing a working, practical use case of convolutional neural networks (CNN's.) Specifically, in the use of a dataset with over 25,000 animal photos organized into 10 various classes, such as cats, dogs, and butterflies. Using data preprocessing to optimize the image dataset alongside using modern python libraries to create a self sufficient model capable of identifying images of animals and classifying them accordingly, all within the constraints of environment – a model was created from scratch to reach an accuracy of nearly 80%. This report will go into depth of the various details such as the image set itself, how the problem was approached, and the methods used to tackle the creation of the CNN model.

Dataset and Related Work

Before we begin the deep dive into all the particulars, we'll start with the dataset itself. This is a collection of 28,000 images of roughly the size of roughly 255x255 pixels. Images vary from cats, to dogs, and even to butterflies. On top of that, there's also various lighting scenarios such as daytime and night, alongside an infinite possibilities of angles, perspectives, and background noise. To help simplify the process, the images were downscaled to half their original size, specifically 128x128 pixels. From there, the pixel values were normalized between a value range of [0, 1] to work within the constraints of Google collab. It should be noted that not all images were created equal, and so the process of downscaling and normalizing allowed all the images to be uniform in size before beginning the training and testing of accuracy.

Methodology

The nature of this dataset points towards our usage of a CNN model : multi-classification. To streamline the process, TensorFlow and Keras python libraries were used to develop a CNN model. First and foremost, data processing was handled sequentially. Once images were resized to 128x128, they were few into the Input layer. Since the images are .JPG in nature, they contain 3 channels – Red, Green, and blue. The input data was then passed through two convolution layers. One with 32 filters, another with 64. Each filter applied a kernel to the input image in order to detect a pattern of some sort. These patterns vary: some are the "edges" between the animal and background or the features of the animal. The first convolution layer handled most of these basic features. The deciphering fur patterns from smooth patterns – was the purpose of the second convolution layer. The Rectified Linear Unit (ReLU) activation function was added to give the model the ability to plot the data in a non-linear fashion in order to more accurately classify complex images. Another key feature for each convolution layer was the usage of Max Pooling. Using max pooling allowed for the simplification of feature maps so that the data could be more optimally processed and generalized for further prediction. In the long run this allow prevents overfitting since complex feature maps could be misread, especially in the first convolution layer whose purpose is to detect simple features, not the image as a whole. This is further supported by the use of Flatten() to convert 2D feature maps into a single dimension vector for the next layer. This next layer contains 128 neurons to handle these feature maps, and a dropout of 50% was used to prevent overfitting. In the final layer, only 10 neurons are used, one for each of the 10 animal classes. Using softmax activation function, a final list is produced of the probability of the image being a certain animal, and the highest probability is used.

To further improve the overall accuracy of the model, the data needed to be generalized for future classification. Using data augmentation, the usage of a class known as ImageDataGenerator was used. It's main purpose is to rescale, randomly rotate, and flip images. Data was augmented in batches, reserving roughly 30% of the data for validation. With the usage of categorical cross-entropy alongside the Adam optimizer for handling the learning rate, it was the ideal approach for multi-class classification. It's also worth noting that Checkpoints were used save the best models based of validation accuracy, and to also account for any interrupts while the model ran on Google collab. At random, 20 epochs were chosen for reaching the best possible accuracy while handling batches of 32 images at a time to allow the model to not be overloaded. The combination of max pooling, drop outs, the data generator, and convolution layers were all used with the mindset of efficiency of memory since we cannot always account for the hardware the model is ran on.

Experimental Setup

For the creation of the CNN to be possible, a variety of tools and libraries were used. TensorFlow, Numpy, Keras, Matplotlib, and OpenCV were used to train the model and visualize it's performance. These libraries are built with scalability in mind and server as the perfect grounds for testing modular-sized machine learning models on Google Collab. The GPU acceleration of Google collab proved ideal in comparison to a personal work computer, which tends to prioritize the CPU. And given NumPy is used to exclusively handle matrices, Google Collab proved ideal since it can also manage RAM more optimally.

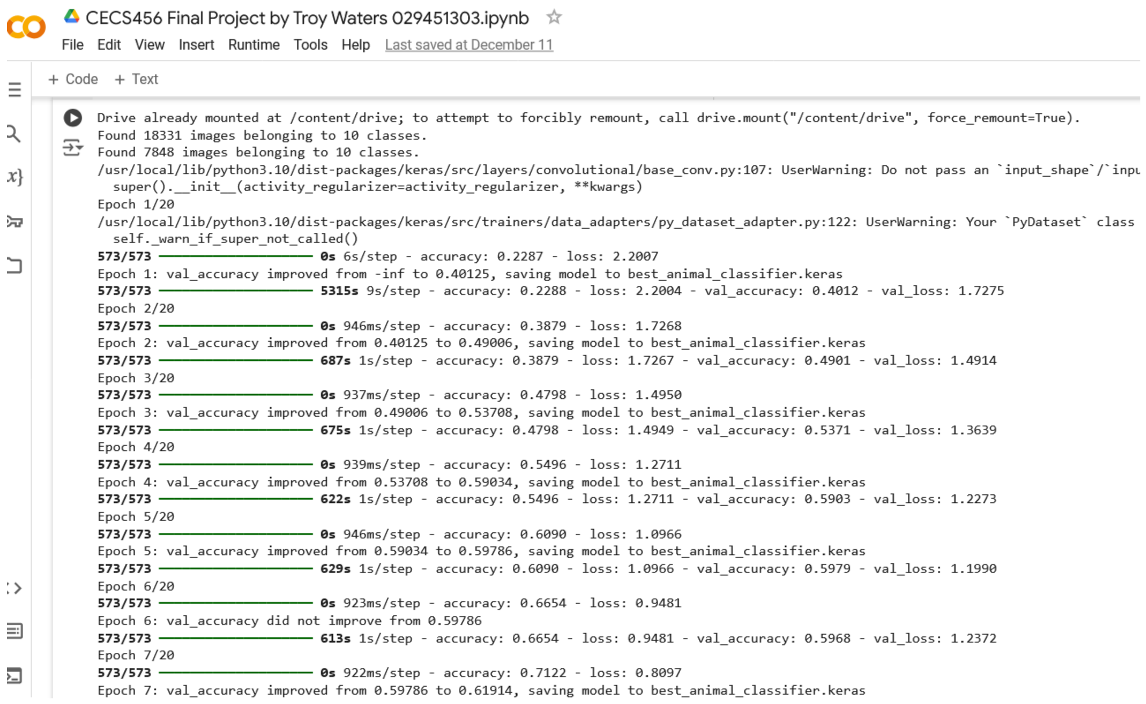
In most images, their RGB (Red, green, blue) value ranged from 0 to 255. Since computers can handle decimal calculations faster than large integers, these values were normalized between a range of 0 to 1 to generalize the entire image's pixel values. Training allocated for 70% of the dataset, 15% used for validation, and the remaining 15% were used for testing the model and evaluating it's performance.

Measurement

Despite the multi-class nature of this dataset, only two major measurements were tracked: accuracy and loss. Since this model needs to be generalized for any animal image given as input, accuracy is the only measurement we can focus on to further improve it's ability to generalize. Loss is handled by comparing the model's output and true animal class with the use of categorical cross-entropy loss function.

Result Analysis, Intuitions, and Comparisons

The training of the model took place overnight over the course of 20 epochs. The first two epochs showed minimal accuracy – of roughly 0.228 and 0.2288 with a loss of 2.2007 and 1.7268 respectively. In a fascinating fashion, the 3rd epoch showed hope: an accuracy of 0.3879. The trend continued as it reached the 5th epoch as an accuracy of 0.6090 and eventually reaching the 8th epoch with an impressive 0.7439 accuracy for it's limitations at only a loss of 0.5673. However, Google collab has stopped the model on the 10th epoch due to restrictions of RAM. However, given the 10th epoch had only reached 0.7873 accuracy, it is safe to assume the model was achieving it's peak performance since the diminishing returns were beginning to show.


CECS456 Final Project by Troy Waters 029451303.ipynb
File Edit View Insert Runtime Tools Help Last saved at December 11
+ Code + Text

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
Found 18331 images belonging to 10 classes.
Found 7848 images belonging to 10 classes.
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`inpu
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/20
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:122: UserWarning: Your `PyDataset` class
  self._warn_if_super_not_called()
573/573 ━━━━━━━━━━━━━━━━━━━━ 0s 6s/step - accuracy: 0.2287 - loss: 2.2007
Epoch 1: val_accuracy improved from -inf to 0.40125, saving model to best_animal_classifier.keras
573/573 ━━━━━━━━━━━━━━━━━━━━ 5315s 9s/step - accuracy: 0.2288 - loss: 2.2004 - val_accuracy: 0.4012 - val_loss: 1.7275
Epoch 2/20
573/573 ━━━━━━━━━━━━━━━━━━━━ 0s 946ms/step - accuracy: 0.3879 - loss: 1.7268
Epoch 2: val_accuracy improved from 0.40125 to 0.49006, saving model to best_animal_classifier.keras
573/573 ━━━━━━━━━━━━━━━━━━━━ 687s 1s/step - accuracy: 0.3879 - loss: 1.7267 - val_accuracy: 0.4901 - val_loss: 1.4914
Epoch 3/20
573/573 ━━━━━━━━━━━━━━━━━━━━ 0s 937ms/step - accuracy: 0.4798 - loss: 1.4950
Epoch 3: val_accuracy improved from 0.49006 to 0.53708, saving model to best_animal_classifier.keras
573/573 ━━━━━━━━━━━━━━━━━━━━ 675s 1s/step - accuracy: 0.4798 - loss: 1.4949 - val_accuracy: 0.5371 - val_loss: 1.3639
Epoch 4/20
573/573 ━━━━━━━━━━━━━━━━━━━━ 0s 939ms/step - accuracy: 0.5496 - loss: 1.2711
Epoch 4: val_accuracy improved from 0.53708 to 0.59034, saving model to best_animal_classifier.keras
573/573 ━━━━━━━━━━━━━━━━━━━━ 622s 1s/step - accuracy: 0.5496 - loss: 1.2711 - val_accuracy: 0.5903 - val_loss: 1.2273
Epoch 5/20
573/573 ━━━━━━━━━━━━━━━━━━━━ 0s 946ms/step - accuracy: 0.6090 - loss: 1.0966
Epoch 5: val_accuracy improved from 0.59034 to 0.59786, saving model to best_animal_classifier.keras
573/573 ━━━━━━━━━━━━━━━━━━━━ 629s 1s/step - accuracy: 0.6090 - loss: 1.0966 - val_accuracy: 0.5979 - val_loss: 1.1990
Epoch 6/20
573/573 ━━━━━━━━━━━━━━━━━━━━ 0s 923ms/step - accuracy: 0.6654 - loss: 0.9481
Epoch 6: val_accuracy did not improve from 0.59786
573/573 ━━━━━━━━━━━━━━━━━━━━ 613s 1s/step - accuracy: 0.6654 - loss: 0.9481 - val_accuracy: 0.5968 - val_loss: 1.2372
Epoch 7/20
573/573 ━━━━━━━━━━━━━━━━━━━━ 0s 922ms/step - accuracy: 0.7122 - loss: 0.8097
Epoch 7: val_accuracy improved from 0.59786 to 0.61914, saving model to best_animal_classifier.keras
```

The limitations of Google collab have been mentioned before but it's important to reiterate how it affects results. Initially, the model was ran using the more original size of 255x255, but the model was quickly terminated for overusing RAM on Google collab. Because images were downscaled to 128x128 to meet this RAM limitation, it's reasonable to say a lot of image features which could classify an animal more accurately were lost in the input layer. The usage of only two layers also plays heavily into this, since one was used for simple features, and a second layer for complex features. If a 3rd, intermediate layer for deciphering slightly more complex features, the accuracy would be improved immensely. The use of data augmentation did help offset this cost to performance since it used the same 128x128 image without having to start a new iteration or require more epochs. It's also possible that the batch size of 32 images could have been a major tipping point in training, since a larger batch would drastically change how the filtering and validation was handled. Overall, an accuracy of nearly 80% on real-world images is an excellent foundation for future improvement.

Conclusions

The versatility of CNNs cannot be further overstated in this demonstration on this dataset of over 25,000 animal images. From the handling of simple patterns such as edges and lines to the complexity of backgrounds and textures, all within the constraints of Google collab proved to be quite the challenge. The combination of pooling, dropout, layers, data augmentation, and down-scaling allowed the CNN model to generalize image data in a reasonable 20 epochs trained in the course of a full day to result in an accuracy of nearly 80%. Challenges include working with a limited dataset, the overall quality of that dataset, Google collab limitations, and complexity constraints. However, the main takeaway from this project was the creation of a model from scratch. The largest leap in furthering the development of an image classifier would be taking full advantage of existing pre-trained models, since it can be argued that 25,000 images isn't enough to fully develop a generalized model. The next step would be to shift the resource management to a more local environment with hardware capable of handling the growing complexity and RAM demands of the model as it continues to train. This would also allow for larger batch sizes, less down-sampling, more convolution layers to balance the complexity between layers, and adjusting learning rates.

GitHub: https://github.com/EnragedWaters/CECS-456-Final-Project