Using Machine Learning to classify Natural Images to 8 classes

By: Ethan Liu

Github Link: https://github.com/WebForks/456_final_project

Introduction

In this report, I explore using the convolutional neural network (cnn) model in order to classify images into 8 different categories: airplane, car, cat, dog, flower, fruit, motorbike, and person. In this project we use deep learning and image processing to construct the natural image classifier. The dataset (from kaggle.com/datasets/prasunroy/natural-images), organized into separate directories for each category, serves as the foundation for training and validating the model. This report highlights the use of deep learning for real-world classification tasks, demonstrating the power of CNNs in recognizing and differentiating complex visual patterns.

Dataset and related work

The Natural Images dataset contains labeled images distributed across eight categories: airplane, car, cat, dog, flower, fruit, motorbike, and person. Each category is represented by a substantial number of samples, providing sufficient data for training and validation. The images from the dataset are taken from the following links below

Airplane images obtained from http://host.robots.ox.ac.uk/pascal/VOC
Car images obtained from https://ai.stanford.edu/~jkrause/cars/car_dataset.html
Cat images obtained from https://www.kaggle.com/c/dogs-vs-cats
Dog images obtained from https://www.kaggle.com/c/dogs-vs-cats
Flower images obtained from http://www.image-net.org
Fruit images obtained from https://www.kaggle.com/moltean/fruits
Motorbike images obtained from http://host.robots.ox.ac.uk/pascal/VOC
Person images obtained from http://www.briancbecker.com/blog/research/pubfig83-lfw-dataset

Methodology

To create the model for our image classification, we created a convolutional neural network (CNN) to process the image data and calculate the probabilities for the eight categories.

Data Augmentation and Preprocessing

We first start with rescaling and changing all the pixel values so that they are normalized from range [0, 1] and then divided by 255.

We then use techniques like rotating, width and height shifting, zooming, and horizontal flipping in order to expand the dataset amount and enhance the generalization.

Finally we divide the dataset into 80% of training and 20% validation subsets in order to see the model performance on the unseen data.

Model Architecture

The CNN model is changed in multiple ways such as the input layer being resized to 150x150 pixels with red, green, and blue.

Three convolutional layers with different filter counts of 32, 64, and 128 are also used. Each layer. After setting up the layers and finishing processing the data images, we set up the model compilation by using the adam optimizer for more efficient and adaptive learning as well as use accuracy to evaluate the model performance during the training and validation.

Experimental setup

Data Preprocessing

The data images are resized to 150x150 pixels. The data augmentation was implemented using Keras ImageDataGenerator with transformations like rotating it by 20 degrees, increasing the width and height shifting by 20 percent, zooming in and out by 20 percent, and flipping the images horizontally. We then split the dataset with 80% for training and 20% for the validation subsets.

Model Architecture

The CNN model is comprised of a Conv2D layer with 3 convolutional layers with 32, 64, and 128 filters each followed by a max pooling layer. It also contains a Dropout layer of 0.5 to prevent overfitting, a Dense layer with 512 units and ReLU activation as well as an Output Layer with 8 units and softmax activation.

Training Parameters

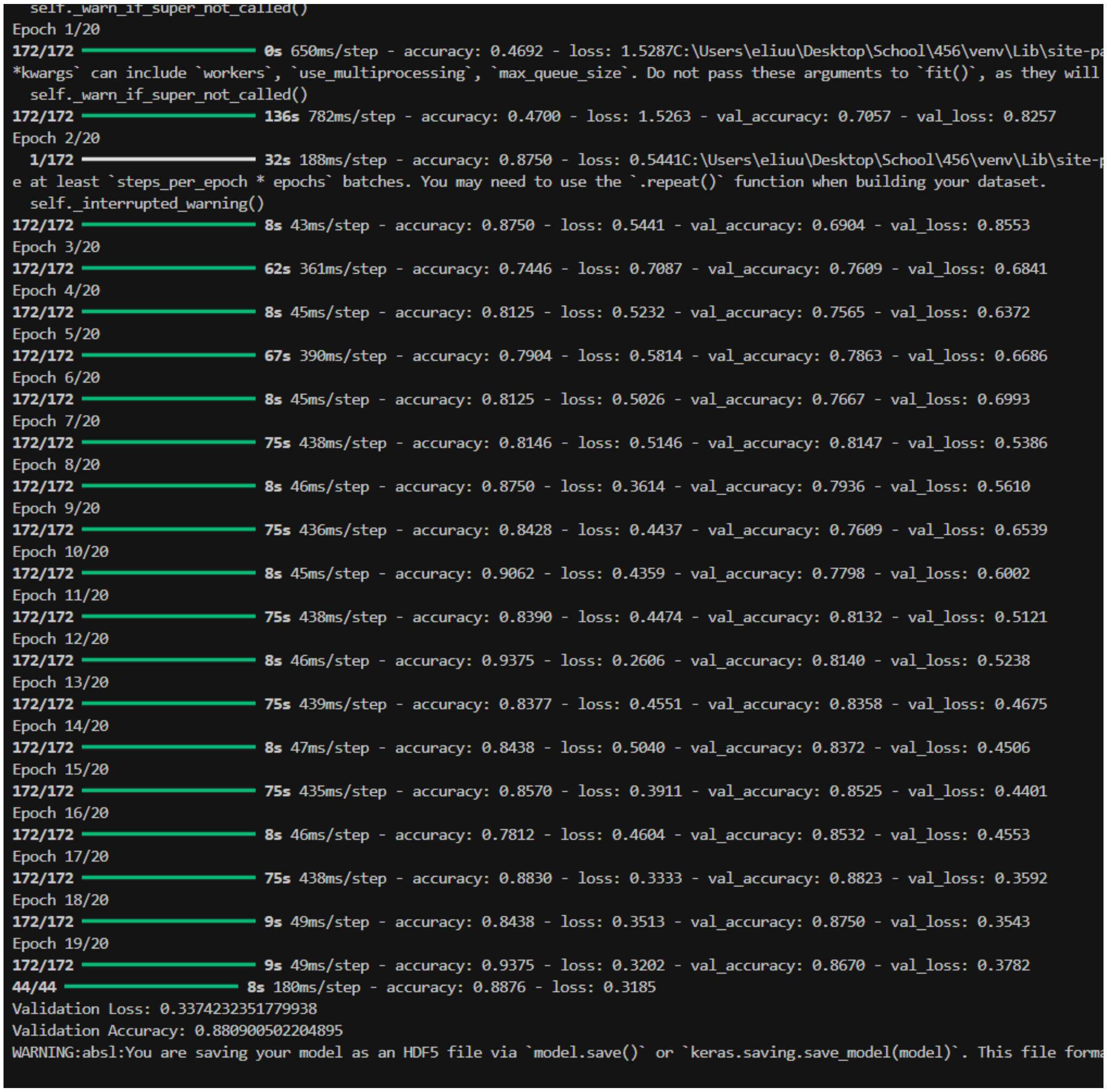
The training process uses batch processing where they are fed to train the model in batches of 32 images. The model is trained for 20 epochs which allows it sufficient iterations for learning patterns from the data. It uses the Adam optimizer at a learning rate of 0.001 and uses real time augmentation using Keras Image Data Generator.

Measurement

Performance is measured using training accuracy which is the accuracy of the model on the training set, validation accuracy which is the accuracy of the model on the validation set, and validation loss which is the cross entropy loss on the validation set.

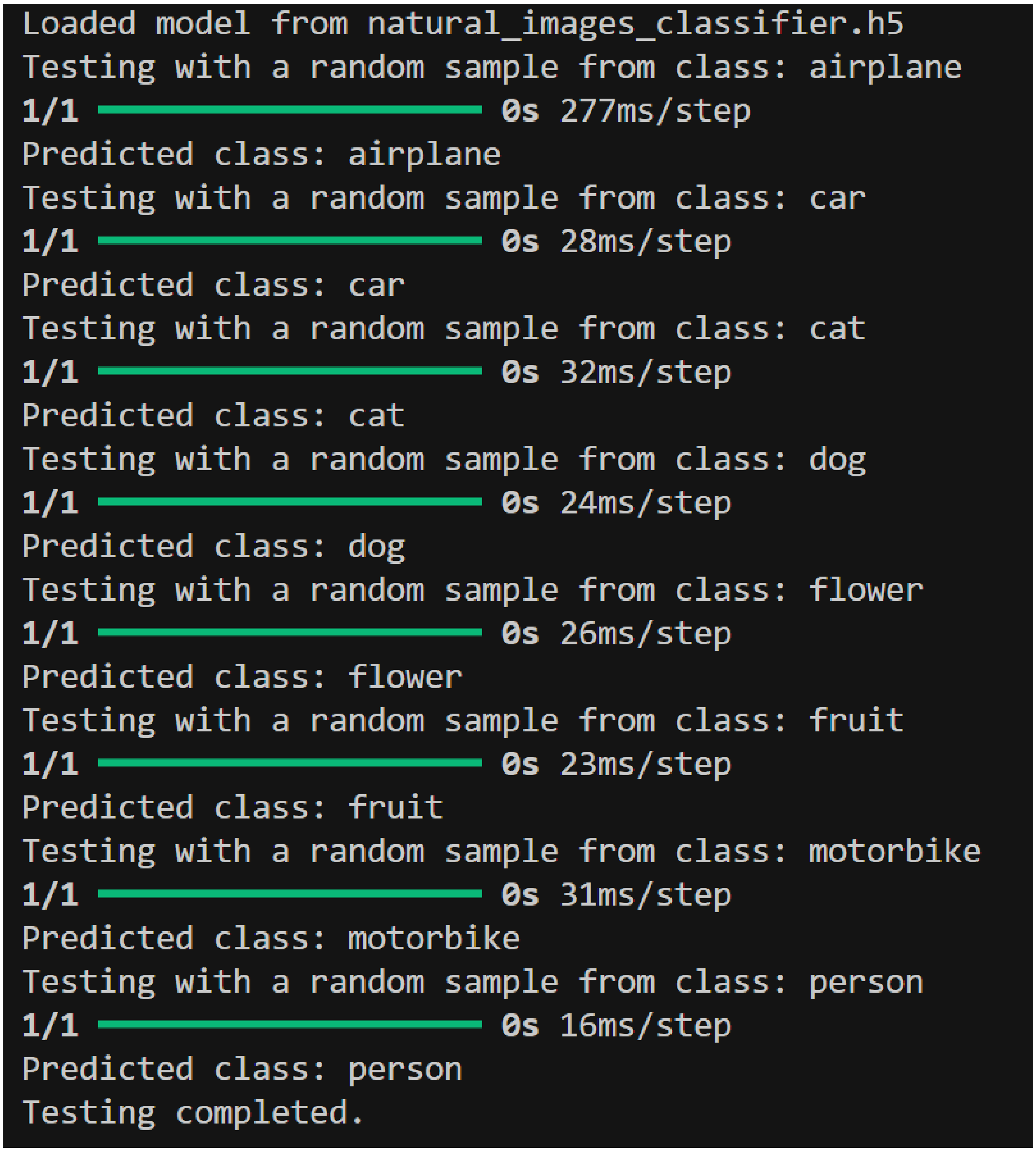
Result analysis intuitions and comparisons

The model achieves the following



```
  self._warn_if_super_not_called()
Epoch 1/20
172/172 ━━━━━━━━━━━━━━━━━━━━ 0s 650ms/step - accuracy: 0.4692 - loss: 1.5287C:\Users\eliuu\Desktop\School\456\venv\Lib\site-pa
*kwargs` can include `workers`, `use_multiprocessing`, `max_queue_size`. Do not pass these arguments to `fit()`, as they will
  self._warn_if_super_not_called()
172/172 ━━━━━━━━━━━━━━━━━━━━ 136s 782ms/step - accuracy: 0.4700 - loss: 1.5263 - val_accuracy: 0.7057 - val_loss: 0.8257
Epoch 2/20
  1/172 ━━━━━━━━━━━━━━━━━━━━ 32s 188ms/step - accuracy: 0.8750 - loss: 0.5441C:\Users\eliuu\Desktop\School\456\venv\Lib\site-p
e at least `steps_per_epoch * epochs` batches. You may need to use the `.repeat()` function when building your dataset.
  self._interrupted_warning()
172/172 ━━━━━━━━━━━━━━━━━━━━ 8s 43ms/step - accuracy: 0.8750 - loss: 0.5441 - val_accuracy: 0.6904 - val_loss: 0.8553
Epoch 3/20
172/172 ━━━━━━━━━━━━━━━━━━━━ 62s 361ms/step - accuracy: 0.7446 - loss: 0.7087 - val_accuracy: 0.7609 - val_loss: 0.6841
Epoch 4/20
172/172 ━━━━━━━━━━━━━━━━━━━━ 8s 45ms/step - accuracy: 0.8125 - loss: 0.5232 - val_accuracy: 0.7565 - val_loss: 0.6372
Epoch 5/20
172/172 ━━━━━━━━━━━━━━━━━━━━ 67s 390ms/step - accuracy: 0.7904 - loss: 0.5814 - val_accuracy: 0.7863 - val_loss: 0.6686
Epoch 6/20
172/172 ━━━━━━━━━━━━━━━━━━━━ 8s 45ms/step - accuracy: 0.8125 - loss: 0.5026 - val_accuracy: 0.7667 - val_loss: 0.6993
Epoch 7/20
172/172 ━━━━━━━━━━━━━━━━━━━━ 75s 438ms/step - accuracy: 0.8146 - loss: 0.5146 - val_accuracy: 0.8147 - val_loss: 0.5386
Epoch 8/20
172/172 ━━━━━━━━━━━━━━━━━━━━ 8s 46ms/step - accuracy: 0.8750 - loss: 0.3614 - val_accuracy: 0.7936 - val_loss: 0.5610
Epoch 9/20
172/172 ━━━━━━━━━━━━━━━━━━━━ 75s 436ms/step - accuracy: 0.8428 - loss: 0.4437 - val_accuracy: 0.7609 - val_loss: 0.6539
Epoch 10/20
172/172 ━━━━━━━━━━━━━━━━━━━━ 8s 45ms/step - accuracy: 0.9062 - loss: 0.4359 - val_accuracy: 0.7798 - val_loss: 0.6002
Epoch 11/20
172/172 ━━━━━━━━━━━━━━━━━━━━ 75s 438ms/step - accuracy: 0.8390 - loss: 0.4474 - val_accuracy: 0.8132 - val_loss: 0.5121
Epoch 12/20
172/172 ━━━━━━━━━━━━━━━━━━━━ 8s 46ms/step - accuracy: 0.9375 - loss: 0.2606 - val_accuracy: 0.8140 - val_loss: 0.5238
Epoch 13/20
172/172 ━━━━━━━━━━━━━━━━━━━━ 75s 439ms/step - accuracy: 0.8377 - loss: 0.4551 - val_accuracy: 0.8358 - val_loss: 0.4675
Epoch 14/20
172/172 ━━━━━━━━━━━━━━━━━━━━ 8s 47ms/step - accuracy: 0.8438 - loss: 0.5040 - val_accuracy: 0.8372 - val_loss: 0.4506
Epoch 15/20
172/172 ━━━━━━━━━━━━━━━━━━━━ 75s 435ms/step - accuracy: 0.8570 - loss: 0.3911 - val_accuracy: 0.8525 - val_loss: 0.4401
Epoch 16/20
172/172 ━━━━━━━━━━━━━━━━━━━━ 8s 46ms/step - accuracy: 0.7812 - loss: 0.4604 - val_accuracy: 0.8532 - val_loss: 0.4553
Epoch 17/20
172/172 ━━━━━━━━━━━━━━━━━━━━ 75s 438ms/step - accuracy: 0.8830 - loss: 0.3333 - val_accuracy: 0.8823 - val_loss: 0.3592
Epoch 18/20
172/172 ━━━━━━━━━━━━━━━━━━━━ 9s 49ms/step - accuracy: 0.8438 - loss: 0.3513 - val_accuracy: 0.8750 - val_loss: 0.3543
Epoch 19/20
172/172 ━━━━━━━━━━━━━━━━━━━━ 9s 49ms/step - accuracy: 0.9375 - loss: 0.3202 - val_accuracy: 0.8670 - val_loss: 0.3782
44/44 ━━━━━━━━━━━━━━━━━━━━ 8s 180ms/step - accuracy: 0.8876 - loss: 0.3185
Validation Loss: 0.3374232351779938
Validation Accuracy: 0.880900502204895
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file forma
```

We manage to get an 88% validation accuracy as well as a 33% validation loss.



```
Loaded model from natural_images_classifier.h5
Testing with a random sample from class: airplane
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 277ms/step
Predicted class: airplane
Testing with a random sample from class: car
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
Predicted class: car
Testing with a random sample from class: cat
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
Predicted class: cat
Testing with a random sample from class: dog
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
Predicted class: dog
Testing with a random sample from class: flower
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
Predicted class: flower
Testing with a random sample from class: fruit
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
Predicted class: fruit
Testing with a random sample from class: motorbike
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
Predicted class: motorbike
Testing with a random sample from class: person
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
Predicted class: person
Testing completed.
```

Through the testing of our model we manage to see that it correctly classifies the images.

Conclusion

This project successfully demonstrated the use of convolutional neural networks for image classification on the Natural Images dataset. The model's high accuracy and ability to generalize highlight the effectiveness of CNNs in image recognition.

Contributions

Ethan Liu: I wrote the code for final_project.py as well as testing.py. I also wrote the steps for the README and created the requirements.txt. I also wrote the final project report