CECS 456 Final Project: Natural Images with 8 classes

Le Duong (026469324)

California State University, Long Beach

Le.duong@student.csulb.edu

Link to GitHub Repo: https://github.com/blueisderp/CECS456Final

***Abstract*— This paper introduces a deep learning (DL) machine model for object image detection. More specifically, a Convolutional neural network model (CNN) is used that specializes in being able to classify natural images from a dataset into one of eight classifications. This paper discusses the techniques used to design and construct the CNN model and resulting analysis based on the model's performance during training, valuation, and testing.**

***Keywords*— Machine Learning, Deep Learning, CNNs, Object Image Detection**

I. Introduction

Machine Learning (ML) has become a cornerstone of modern digital technologies that enables computers to learn from data to make predictions in the real world without the need for it to be explicitly hard coded. Especially with the rise of popular consumer AIs such as OpenAI's ChatGPT and Google's Project Gemini, the concept of machine learning has come into the spotlight of technological innovation in a way unseen before. But this technology was prevalent long before any of the major consumer AIs changed the way the general thinks about Machine Learning. For example, something as unassuming as an email spam filter utilizes the same fundamental core principles and techniques in ML that allows it to determine whether an incoming email is spam or not. However, one of the more popular uses for ML in recent years is object image detection. That is, in the broadest sense the ability of a ML model to be able to identify a distinct object or pattern from an image. This has a wide array of uses in the modern world such as for emergency collision detection systems in self-driving cars to facial recognition software to enable machines to accurately identify certain human emotions. For this experiment, I designed and constructed a ML object detection model that specializes in being able to identify objects that can be classified in a wide array of categories.

II. Dataset & Related Work

The dataset used in this experiment to train the ML model is the Natural Images with 8 classes from Kaggle, a data science and ML platform under parent company Google. This training dataset was chosen due to the larger number of classifications (8 to be exact) available and the more diverse variety among classifications which are commonplace everyday objects. These classifications are: airplane, car, cat, dog, flower, fruit, motorbike, and person. For future iterations and expansions of this experiment, this dataset serves as a great foundation for general purpose object image detection models.

III. Methodology

The ML model used for this experiment is a Convolutional neural network (CNN) which is composed of multiple layers of 'neurons' interconnected front to back (with layers connected to the layer immediately in front and/or behind it) to allow the propagation of the training process throughout the neural network. And since CNN models utilize regularized weights over neuron connections, it makes them ideal for image classification problems (such as the one posed by

the dataset used here) but does not encounter the Vanishing gradient problem during backpropagation that other neural networks do. However, CNNs are susceptible to overfitting wherein their accuracy during training may be high, but fall off drastically when run against valuation and testing datasets. This is because the CNN model has become so overtuned to its training dataset, that it is unable to pick up and learn the underlying pattern. Thus, when confronted with new unseen data, the model could have wildly inaccurate predictions. Of course there are ways to combat this, such as increasing the training dataset, reducing the number of parameters, and regularization to penalize large weights.

IV. Experimental Setup

After importing the dataset and extracting the images from it, I sorted each image so that the size dimensions so that every image in the dataset was uniform (in this case a 28x28 image) to make it easier for the model to train with. The images were randomly split up into three sets: the training set, valuation set, and test set. The training set would be the images primarily used during training while the valuation set would be images used to test the CNN models accuracy (how often the model predicts the right outcome) and loss (how far off predicted outputs from the model are from actual target values) at the end of every training epoch. As seen below (Figure 1.1), the CNN model was constructed in sequential order as the filter size of each convolutional layer grows exponentially until the last output is flattened. It is then fed into two dense (fully-connected) layers where regularization is applied using dropout and L2 regularization to prevent the model overfitting the training dataset. Finally, the last layer reduces the output to one of the eight classifications mentioned earlier in the dataset. The model was then compiled and trained over the course of five epochs (training cycles) where each time the model would train on the training dataset and then be run against the valuation set to see how accurate it is when seeing new images.

Figure 1.1 Constructing CNN Model

```
# Build CNN Model

# initialize the CNN model
cnn = tf.keras.models.Sequential()

# convolutional layers
cnn.add(tf.keras.layers.Conv2D(filters=64,
kernel_size=3,    activation="relu",    padding="same",
input_shape=[28, 28, 3]))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2,
strides=2))
cnn.add(tf.keras.layers.Conv2D(filters=128,
kernel_size=3, activation="relu", padding="same"))
cnn.add(tf.keras.layers.Conv2D(filters=128,
kernel_size=3, activation="relu", padding="same"))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2,
strides=2))
cnn.add(tf.keras.layers.Conv2D(filters=256,
kernel_size=3, activation="relu", padding="same"))
cnn.add(tf.keras.layers.Conv2D(filters=256,
kernel_size=3, activation="relu", padding="same"))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2,
strides=2))

# flatten layer
cnn.add(tf.keras.layers.Flatten())

# dense layers with L2 regularization
cnn.add(tf.keras.layers.Dense(units=128,
activation="relu",
kernel_regularizer=tf.keras.regularizers.l2(0.01)))
cnn.add(tf.keras.layers.Dropout(0.5))
cnn.add(tf.keras.layers.Dense(units=64,
activation="relu",
kernel_regularizer=tf.keras.regularizers.l2(0.01)))
cnn.add(tf.keras.layers.Dropout(0.5))

# output layer
cnn.add(tf.keras.layers.Dense(units=len(classes),
activation="softmax"))
```

V. Measurements

Although initial results did not yield very positive results due to overfitting, these issues were mostly resolved due to some of the implementation of some of the solutions mentioned in the previous section. As seen in the final results in Figure 2.1, training accuracy remained relatively high (~85%) while valuation accuracy did not fall that far behind (~70%). And when time came for the model to be finally run against the test set, it had an accuracy of

84% which is a reasonably positive performance considering the relatively small training dataset it was training on (Figure 2.2). Both the valuation and test accuracy are indications that the CNN model can (most of the time) accurately classify new images it has not seen before. However, in both training and valuation the model seemed to have high loss meaning when it made incorrect predictions, they were much farther off the target value than intended.

Figure 2.1 Results of Training and Valuation

```
Epoch 1/5
45/45 [==============================] - 60s 1s/step -
loss: 0.6621 - accuracy: 0.8492 - val_loss: 1.1406 -
val_accuracy: 0.7364
Epoch 2/5
45/45 [==============================] - 62s 1s/step -
loss: 0.5975 - accuracy: 0.8582 - val_loss: 1.1984 -
val_accuracy: 0.6341
Epoch 3/5
45/45 [==============================] - 62s 1s/step -
loss: 0.5706 - accuracy: 0.8648 - val_loss: 1.0172 -
val_accuracy: 0.6821
Epoch 4/5
45/45 [==============================] - 64s 1s/step -
loss: 0.4893 - accuracy: 0.8815 - val_loss: 0.8828 -
val_accuracy: 0.7210
Epoch 5/5
45/45 [==============================] - 66s 1s/step -
loss: 0.4382 - accuracy: 0.8981 - val_loss: 0.6660 -
val_accuracy: 0.8043
```

Figure 2.2 Results of Testing

```
44/44 [==============================] - 4s 100ms/step
- loss: 0.6222 - accuracy: 0.8442
Total loss on Testing Set: 0.6222281455993652
Accuracy of Testing Set: 0.8442028760910034
```

VI. Results Analysis

The results of this experiment seem to suggest the ability of the CNN model to 'hit the mark' a majority of the time. This is once again reflected when the model was prompted to predict the first three images of the test set, which it did accurately (see Figure 3.1). However, in the rarer cases in which the model does get an incorrect prediction, it tends to 'miss the mark' with a much higher gap of error than should be tolerated. This suggests that the model has an under-bias where it under-fits the data and is making more simplistic assumptions in its predictions (especially during training). This could possibly be remedied by once again increasing the training dataset size or decreasing regularization. It is safe to say however, that the CNN model type used in this experiment is the more preferable neural network compared to other types because its layers utilize ReLU activation functions as opposed to Sigmoid or Tanh which are both susceptible to the Vanishing gradient problem.

Figure 3.1 Model predicting first three images from test dataset

```
1/1 [==============================] - 0s 55ms/step
Predicted Results:
Image 1: airplanes
Image 2: cat
Image 3: flower

Real Labels:
Image 1: airplanes
Image 2: cat
Image 3: flower
```

VII. Conclusion

In conclusion, the overall efforts and results of these experiments could be considered a success. The CNN model discussed in this paper does fulfill its intended function to a high enough degree that warrants it being a good foundation for future experiments or research for image object identification and classification models. ML models in this specific field will only grow in the future.

VIII. Contributions

This experiment was designed and conducted by Le Duong. For reference to the code used in this experiment, please access the GitHub link at top of this report. Special thanks to Kaggle for their dataset used in training this CNN model.