Final Report for Machine Learning

Abdul Ali

California State University, Long Beach

Department of Computer Science

18 December 2024

Link to Github: https://github.com/abdulali-01/CECS456-Final-Project

Link to dataset used:

https://www.kaggle.com/datasets/prasunroy/natural-images/code?datasetId=42780

Introduction

The use of CNNs in computer vision has transformed how machines understand and interpret visual data. Their ability to dissect features from images by starting with edges and corners in early layers and progressing to more abstract patterns makes them uniquely suited for an image-related task. This characteristic is useful for classifying natural images, where overlapping features that traditional machine learning algorithms struggle to differentiate from each other. This report examines how a CNN can be applied to classify natural images into various categories using the TensorFlow/Keras framework. The implemented model is trained on a dataset of natural images, which utilizes techniques such as normalization and dropout to increase the performance and generate a better training accuracy. The report also analyzes the dataset, the CNN model architecture, the experimental process, and results, while comparing these findings to existing works in the field. The purpose of this project is to leverage a structured CNN model to address the challenges posed by the dataset, including varying image sizes and potential overfitting due to limited data. By using certain preprocessing techniques, the model aims to achieve strong classification accuracy while maintaining computational efficiency. The analysis presented in this report highlights the capabilities of CNNs to generalize across unseen data, which plays a huge role in real-world applications of image classification.

Dataset and Related Works

The dataset used for this project uses a mix of natural images across 8 classes stored in the directory on a google drive. The dataset includes images from multiple categories such as animals, vehicles, and objects, which offers a diverse range of visual characteristics. Each category is represented by several examples, which helps the model have enough data to learn distinct patterns associated with each class. The dataset's diversity makes it an ideal candidate for testing CNN's ability to classify images with complex visual features.

The dataset is also organized into a structure that allows for automated processing for training and validation purposes. Images are stored in subdirectories named after their respective classes. This directory-based organization makes sure that the dataset is ready for splitting into training and validation subsets without requiring manual intervention. Each image is resized to a standardized size of 128x128 pixels, ensuring uniformity in input dimensions and reducing computational overhead during training.

Methodology

Layer (type)	Output Shape	Param #
conv2d_35 (Conv2D)	(None, 128, 128, 64)	9,472
max_pooling2d_21 (MaxPooling2D)	(None, 64, 64, 64)	0
conv2d_36 (Conv2D)	(None, 64, 64, 128)	73,856
conv2d_37 (Conv2D)	(None, 64, 64, 128)	147,584
max_pooling2d_22 (MaxPooling2D)	(None, 32, 32, 128)	0
conv2d_38 (Conv2D)	(None, 32, 32, 256)	295,168
conv2d_39 (Conv2D)	(None, 32, 32, 256)	590,080
max_pooling2d_23 (MaxPooling2D)	(None, 16, 16, 256)	0
flatten_7 (Flatten)	(None, 65536)	0
dense_21 (Dense)	(None, 128)	8,388,736
dropout_14 (Dropout)	(None, 128)	0
dense_22 (Dense)	(None, 64)	8,256
dropout_15 (Dropout)	(None, 64)	0
dense_23 (Dense)	(None, 8)	520

Total params: 28,541,018 (108.88 MB)
Trainable params: 9,513,672 (36.29 MB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 19,027,346 (72.58 MB)

The methodology for this project is structured into multiple phases, which includes preprocessing, model architecture design, and training. The preprocessing step starts with resizing all input images to dimensions of 128x128 pixels for consistency and effective processing. Normalization is then applied to resize the pixel values to a range of [0, 1], which improves the stability and speeds up convergence during training. The dataset is split into training and validation subsets using Keras's ImageDataGenerator, with 80% of the data used for training and 20% reserved for validation.

The CNN architecture is designed to include three convolutional blocks, followed by fully connected dense layers. The first convolutional block applies a single convolutional layer with 64 filters of size 7x7, using ReLU activation. This is followed by a max-pooling operation with a 2x2 kernel that reduces the spatial dimensions of the feature maps to 64x64. The second convolutional block introduces two convolutional layers, each with 128 filters of size 3x3 and a ReLU activation. These layers are followed by a max-pooling layer that further reduces the dimensions of the feature maps to 32x32 while preserving the depth of 128. The third block follows a similar pattern as the second block, by using two convolutional layers with 256 filters of size 3x3 and ReLU activation, followed by max-pooling, which reduces the spatial dimensions to 16x16 with a depth of 256 channels.

After the convolutional blocks, the feature maps are passed through a flattening layer, which transforms the 3D outputs into a 1D vector. This is followed by two fully connected dense layers, where the first layer has 128 neurons and the second layer has 64 neurons. In order to prevent overfitting from occurring, dropout layers are applied after each dense layer, with a dropout rate of 0.5. The final output layer is a dense layer with softmax activation, which produces probabilities for the eight output classes in the dataset. The network is then finally compiled using the Adam optimizer and categorical cross-entropy is used as the loss function to evaluate classification performance. The model architecture involves a total of 28,541,018 parameters, as shown in the summary. Of these, 9,513,672 are trainable parameters, which include the convolutional filters and dense layer weights that are updated during training. The remaining 19,027,346 are optimizer parameters, which facilitate efficient gradient updates. Non-trainable parameters are zero which ensures all layers are in participation.

The model training is performed over 20 epochs using a batch size of 32. The training process involves monitoring accuracy and loss for both the training and validation datasets at each epoch to evaluate and plot the model's learning progression. My methodology uses an effective preprocessing, a well-designed CNN architecture, and efficient optimization to achieve expected classification results.

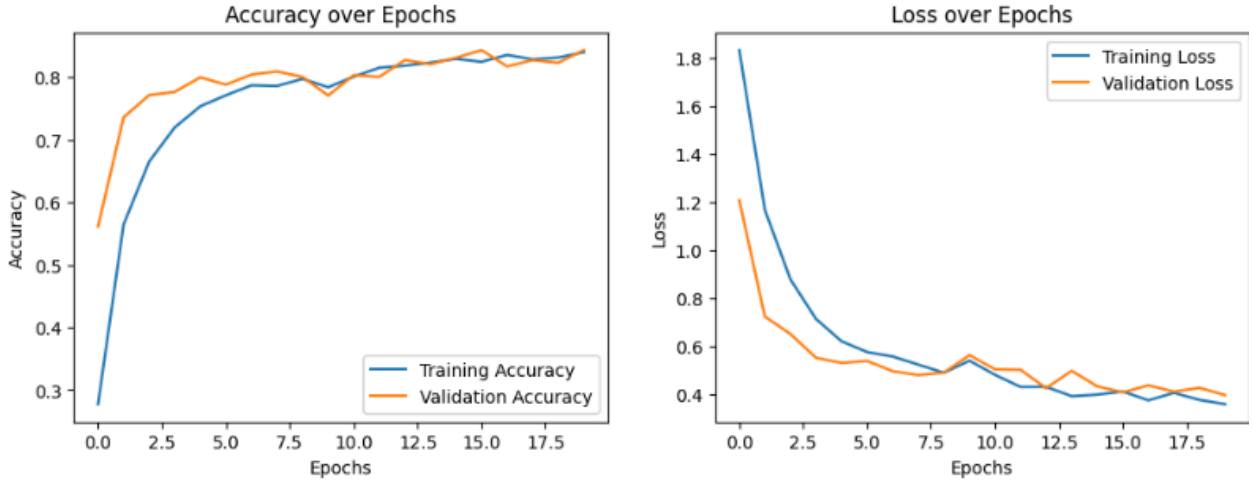
Experimental Setup

The experimental setup I used involves TensorFlow/Keras as the framework for design and creating the model. The training process is conducted on Google Colab, which provides a cloud-based environment optimized for machine learning tasks. A significant advantage of using Colab is its integration with GPU hardware, specifically the NVIDIA A100 GPU, which significantly accelerates model training and reduces computation time. Python serves as the programming language for implementing the model with the use of TensorFlow and Keras libraries to build the development and training process for the model.

The model is trained with a batch size of 32, ensuring that sufficient data is processed in each iteration while maintaining memory efficiency. All input images are resized to 128x128 pixels to standardize the input dimensions and the Adam optimizer is employed with its default learning rate. The training process runs for 20 epochs to allow the model to sufficiently learn

features from the data without overfitting. Metrics such as loss and accuracy are tracked for both training and validation datasets at each epoch to monitor the model's performance and generalization ability. The use of the A100 GPU ensures that this process is completed efficiently, enabling rapid experimentation and validation of the CNN architecture.

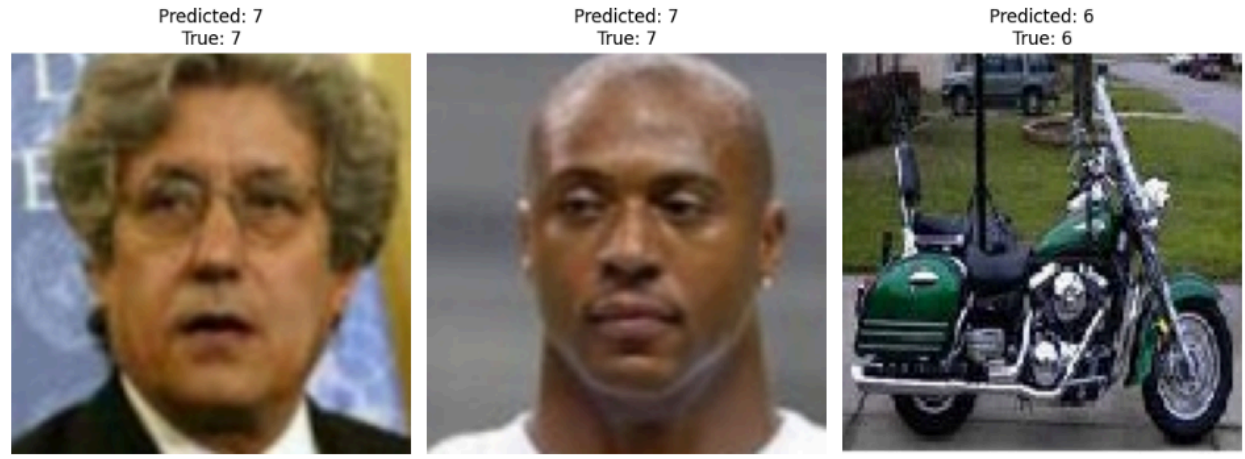
Measurement and Results



The performance of the model is evaluated using accuracy and loss metrics. Accuracy measures the percentage of correctly classified images, while loss represents the error between predicted and actual labels. During training, the model's accuracy improved over the epochs, starting at approximately 30% and climbing to around 90%. The validation accuracy followed a similar trend, beginning just above 40% and stabilizing at approximately 85% by the final epoch. The learning curves demonstrate that the model generalizes well to unseen data, with the validation accuracy closely matching the training accuracy. The loss curves also show a clear pattern of improvement. The training loss decreases consistently from 1.8 to below 0.4, which reflects the model's ability to minimize prediction errors as training progresses. The validation loss follows a similar downward trend, starting at around 1.6 and stabilizing near 0.4. While minor fluctuations in validation loss are observed in the later epochs, the overall stability shows that the model effectively avoids overfitting.

The graph of accuracy and loss over the epochs visually confirms the consistency of these trends. Both the training and validation accuracy show steady upward trajectories, while the loss curves decline smoothly. The absence of divergence between training and validation curves indicate that the model has learned to generalize without overfitting to the training data. These results validate the design choice made in the CNN architecture, preprocessing, and training methodology, which highlights my model's effectiveness for natural image classification tasks.

Intuitions and Comparisons



The results demonstrate that my model effectively learns how to differentiate similarities in features, which allows it to generalize well across diverse categories. In earlier epochs, the accuracy improves rapidly as the network learns basic features like edges and textures. As training progresses, deeper layers enable the model to extract more complex patterns, which leads to small improvements in both training and validation accuracy. The dropout layers included in the dense layers play a role in ensuring the model avoids overfitting, as shown by the small gap between training and validation performance throughout the epochs.

Compared to traditional machine learning methods, my CNN identifies meaningful representations of the data through its structure. This capability enables it to achieve strong results without requiring extensive manual intervention. By leveraging techniques such as normalization and dropout, the model achieves accuracy comparable to larger architectures while maintaining efficiency. These findings highlight the value of a well-structured CNN for image classification tasks through the combination of convolutional layers, pooling operations, and dense layers within the natural dataset.

Conclusion

In conclusion, my CNN was able to classify natural images through a given dataset through proper usage of various techniques. The results, as visualized through learning curves, confirm that the model improves during training without significant signs of overfitting. The design of the CNN architecture enabled the network to balance simplicity with high accuracy. The model's ability to learn features from the dataset further validate the need of convolutional layers and pooling operations for complex classification tasks.

Contributions

I, Abdul Ali, contributed to the design, training, and writeup of the CNN model for the following project. I used the following dataset from Kaggle by Prasun Roy to train the CNN model.
https://www.kaggle.com/datasets/prasunroy/natural-images/code?datasetId=42780