Image Classification using Convolutional Neural Networks (CNNs)

Jyotiska Saha(MST03-0056)

July 31, 2024
Submitted to Meta Scifor Technologies

Meta Scifor Technologies
Script. Sculpt. Socialize

UNDER GUIDIANCE OF
Urooj Khan

Contents

Abstract

Image classification is a pivotal task in computer vision, where the objective is to assign images to one of several predefined categories. This project addresses the multi-class classification problem using Convolutional Neural Networks (CNNs), focusing on seven distinct classes: bikes, cars, cats, dogs, flowers, horses, and humans. CNNs are highly effective for this task due to their capability to automatically learn and extract hierarchical features from raw image data. The dataset utilized consists of 1803 images, which were preprocessed through resizing, normalization, and data splitting into training, validation, and testing sets. The CNN model implemented includes multiple convolutional layers, batch normalization, max pooling, and dropout to enhance generalization and prevent overfitting. The model's performance is evaluated based on loss and accuracy metrics, demonstrating its effectiveness in classifying images into the specified categories.

1 Introduction

Image classification is a fundamental task in computer vision, where the goal is to categorize images into one of several predefined classes. This report focuses on using CNNs for multi-class image classification, which classifies images into seven distinct categories: bikes, cars, cats, dogs, flowers, horses, and humans. CNNs are particularly well-suited for this task due to their ability to automatically learn and extract features from raw image data.

2 Convolutional Neural Network and its Architecture

Convolutional Neural Networks (CNNs) are a specialized type of artificial neural network designed to process and analyze structured grid-like data, such as images. CNNs are particularly well-suited for computer vision tasks due to their ability to automatically and adaptively learn spatial hierarchies of features from the input images. The architecture of CNNs typically consists of a sequence of layers, each serving a specific function. The primary layers include:

- **Convolutional Layers:** These are the core building blocks of a CNN. A convolutional layer applies a set of filters (also known as kernels) to the input image, producing a feature map. Each filter is designed to detect specific features such as edges, corners, or textures. The operation involves a convolution process where the filter slides over the input data, performing element-wise multiplications and summing the results. The key parameters in a convolutional layer include the number of filters, filter size, stride (the step size of the filter), and padding (how the edges are handled).

- **Activation Functions:** Following each convolutional layer, an activation function is applied element-wise to introduce non-linearity into the model. This allows the network to learn more complex patterns. The most commonly used activation function is the Rectified Linear Unit (ReLU), which replaces all negative pixel values in the feature map with zero, effectively introducing non-linearity and helping with model training.

- **Pooling Layers:** These layers are used for downsampling the spatial dimensions of the feature maps, reducing the computational load and controlling overfitting. Pooling layers operate independently on each feature map. The most common type is MaxPooling, which selects the maximum value within a defined window, thus retaining the most prominent features. Another type is Average Pooling, which calculates the average value within the window.

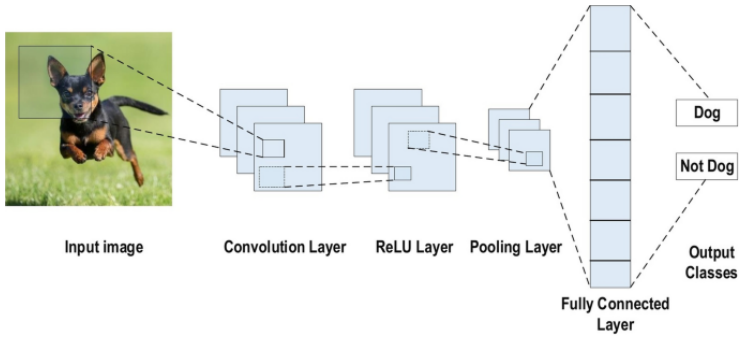


Figure 1: Example of CNN Architecture.

- **Batch Normalization:** This layer normalizes the output of the previous layers by subtracting the batch mean and dividing by the batch standard deviation. Batch normalization helps in stabilizing and accelerating the training process by reducing internal covariate shift, thus allowing for higher learning rates and faster convergence.
- **Dropout:** Dropout is a regularization technique used to prevent overfitting. It works by randomly setting a fraction of input units to zero during training. This forces the network to learn redundant representations of the data, improving generalization.
- **Fully Connected (Dense) Layers:** After the series of convolutional and pooling layers, the feature maps are flattened into a one-dimensional vector. This vector is then passed through one or more fully connected layers. These layers, also known as dense layers, perform the final classification by combining the extracted features and making decisions about the class labels.
- **Softmax Layer:** The final layer in a CNN used for multi-class classification is often a softmax layer. It outputs a probability distribution over the classes, with each value representing the model's confidence that the input belongs to a particular class. The class with the highest probability is chosen as the model's prediction.

CNNs leverage the spatial hierarchies of features through the convolutional layers, allowing them to detect low-level features such as edges in earlier layers and more abstract features like objects or shapes in deeper layers. This hierarchical feature learning, along with weight sharing and local connectivity, makes CNNs highly effective for tasks involving image and visual data. Additionally, advancements in CNN architectures, such as residual connections and inception modules, have further improved their performance and scalability for complex tasks.

3 Why CNN and not ANN?

While Artificial Neural Networks (ANNs) can be used for image classification, Convolutional Neural Networks (CNNs) offer several key advantages:

- **Parameter Efficiency:** CNNs use weight sharing in convolutional layers, applying the same filters to different parts of the image. This reduces the number of parameters compared to ANNs, which connect every pixel to every neuron and require more weights.
- **Local Connectivity:** Unlike ANNs where each neuron connects to all neurons in the previous layer, CNNs connect only to a local region. This captures spatial features more efficiently.
- **Hierarchical Feature Learning:** CNNs learn simple features in early layers and combine them into complex patterns in deeper layers, allowing them to recognize intricate details.
- **Translation Invariance:** CNNs can recognize objects regardless of their position in the image, thanks to pooling layers that make the network robust to changes in position and scale.
- **Efficient Computation:** Convolutional and pooling operations are computationally efficient and can be parallelized, making CNNs suitable for large-scale datasets and high-resolution images.
- **Generalization and Regularization:** Techniques like dropout and batch normalization are integrated into CNNs, helping to improve generalization and reduce overfitting.

In summary, CNNs offer significant advantages over traditional ANNs for image classification, including reduced parameter count, better spatial feature extraction, and improved efficiency and generalization.

4 Dataset and Preprocessing

The dataset used in this project comprises images from seven categories: bikes, cars, cats, dogs, flowers, horses, and humans. The total number of images is 1,803, distributed as follows:

- **Bikes:** 365 images
- **Cars:** 420 images
- **Cats:** 202 images
- **Dogs:** 202 images
- **Flowers:** 210 images
- **Horses:** 202 images
- **Humans:** 202 images

The images were collected from Kaggle and underwent the following preprocessing steps:

- **Resizing:** All images were resized to a uniform dimension of 256x256 pixels.
- **Normalization:** Pixel values were normalized to a range of 0 to 1 to improve the convergence rate of the neural network.
- **Data Splitting:** The dataset was split into training, validation, and testing sets with ratios of 70%, 20%, and 10%, respectively.

5 Model Building and Implementation

The CNN model was implemented using Keras with TensorFlow as the backend. The architecture consists of the following layers:

1. Convolutional Layers:

- **Conv2D (32 filters, 3x3 kernel):** The first convolutional layer applies 32 filters with a kernel size of 3x3 to the input images, utilizing the ReLU activation function. This layer is followed by a BatchNormalization layer, which normalizes the output of the convolution to stabilize and accelerate training. A MaxPooling2D layer is then applied to reduce the spatial dimensions of the feature maps, followed by a Dropout layer with a rate of 0.3 to prevent overfitting.

```
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(256, 256, 3), kernel_regularizer=l2(0.01)))
model.add(BatchNormalization())
model.add(MaxPooling2D())
model.add(Dropout(0.3))

model.add(Conv2D(64, (3, 3), activation='relu', kernel_regularizer=l2(0.01)))
model.add(BatchNormalization())
model.add(MaxPooling2D())
model.add(Dropout(0.3))

model.add(Conv2D(128, (3, 3), activation='relu', kernel_regularizer=l2(0.01)))
model.add(BatchNormalization())
model.add(MaxPooling2D())
model.add(Dropout(0.3))

model.add(Flatten())

model.add(Dense(512, activation='relu', kernel_regularizer=l2(0.01)))
model.add(Dropout(0.5))
model.add(Dense(7, activation='softmax'))
```

Figure 2: Model Building

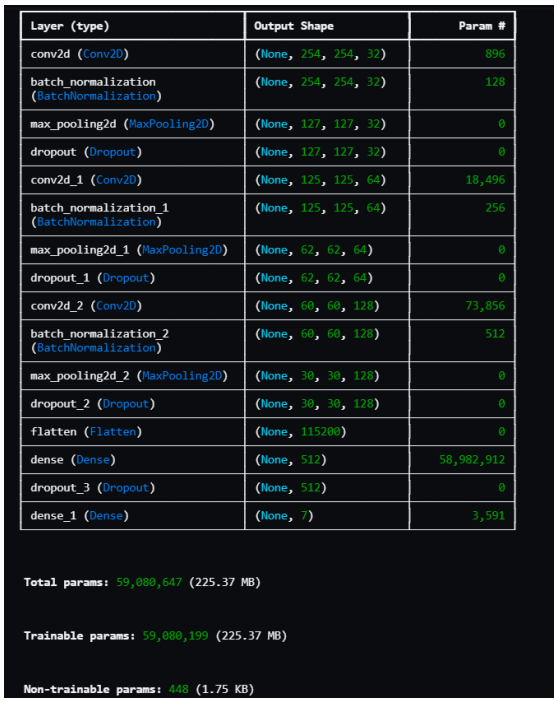
- **Conv2D (64 filters, 3x3 kernel):** The second convolutional layer applies 64 filters with a 3x3 kernel size, again using the ReLU activation function. It is followed by BatchNormalization, MaxPooling2D, and Dropout layers to further process the features and reduce overfitting.
- **Conv2D (128 filters, 3x3 kernel):** The third convolutional layer applies 128 filters with a 3x3 kernel size, with ReLU activation, BatchNormalization, MaxPooling2D, and Dropout layers. This layer helps in capturing more complex features from the images.

2. Fully Connected Layers:

- **Flatten:** The Flatten layer transforms the 3D output of the last convolutional layer into a 1D vector, which is then fed into the fully connected (dense) layers.
- **Dense (512 units):** A fully connected layer with 512 units and ReLU activation function. This layer learns high-level features and patterns from the flattened data, with a L2 regularization term applied to prevent overfitting. A Dropout layer with a rate of 0.5 is included to further reduce overfitting.
- **Dense (7 units):** The final dense layer has 7 units, corresponding to the number of output classes. It uses the softmax activation function to produce probabilities for each class.

Regularization and Optimization: The model employs L2 regularization on the convolutional and dense layers to penalize large weights and improve generalization. Dropout layers are used to randomly omit a fraction of neurons during training to reduce overfitting.

Overall, this architecture is designed to effectively learn and classify images into one of the seven categories by leveraging deep convolutional layers to extract features and fully connected layers to perform the classification.
Below is the model summary:



Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 254, 254, 32)	896
batch_normalization (BatchNormalization)	(None, 254, 254, 32)	128
max_pooling2d (MaxPooling2D)	(None, 127, 127, 32)	0
dropout (Dropout)	(None, 127, 127, 32)	0
conv2d_1 (Conv2D)	(None, 125, 125, 64)	18,496
batch_normalization_1 (BatchNormalization)	(None, 125, 125, 64)	256
max_pooling2d_1 (MaxPooling2D)	(None, 62, 62, 64)	0
dropout_1 (Dropout)	(None, 62, 62, 64)	0
conv2d_2 (Conv2D)	(None, 60, 60, 128)	73,856
batch_normalization_2 (BatchNormalization)	(None, 60, 60, 128)	512
max_pooling2d_2 (MaxPooling2D)	(None, 30, 30, 128)	0
dropout_2 (Dropout)	(None, 30, 30, 128)	0
flatten (Flatten)	(None, 115200)	0
dense (Dense)	(None, 512)	58,982,912
dropout_3 (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 7)	3,591

Total params: 59,080,647 (225.37 MB)

Trainable params: 59,080,199 (225.37 MB)

Non-trainable params: 448 (1.75 KB)

Figure 3: Model Summary

6 Model Training

The model was trained on the preprocessed dataset using the Adam optimizer and categorical cross-entropy loss. The dataset was split into training, validation, and testing sets, with typical ratios of 70%, 20%, and 10% respectively. Total 40 epoch (iteration) were used for training. The training process involved monitoring the validation loss to avoid overfitting.
After 40 epochs,

- **Training Accuracy:** 0.9763
- **Training Loss:** 2.7373

- **Validation Accuracy:** 0.7784
- **Validation Loss:** 3.7087

Here is the Loss and Accuracy diagram:

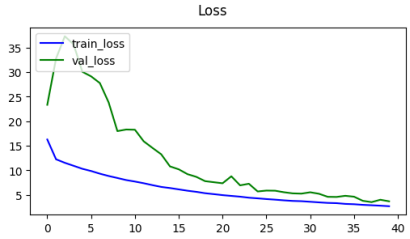


Figure 4: Loss Diagram

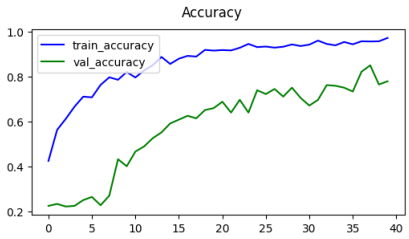


Figure 5: Accuracy Diagram

7 Results

The trained CNN model achieved a high level of accuracy(82%) on the testing set, demonstrating its effectiveness in classifying images across the seven categories. The results included metrics such as accuracy, precision, recall to evaluate the model's performance across different classes. Key findings and observations, along with sample images of correctly and incorrectly classified instances, were analyzed.
Results of test data are given below:

```
Precision: 0.9577465057373047
Recall: 0.9714285731315613
Accuracy: 0.8181818127632141
```

Figure 6: Result on Test data

8 Conclusion

This project successfully implemented a CNN for multi-class image classification. The model demonstrated strong performance (almost 82 % accuracy)

across the seven categories, highlighting the efficacy of CNNs in extracting and classifying features from images.
Future work includes improving data diversity, exploring more advanced CNN architectures like ResNet or EfficientNet

References

[1] Pavan Sanagapati. *Images Dataset*. 2021. Available at: `https://www.kaggle.com/datasets/pavansanagapati/images-dataset`.

[2] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly Media, 2019.