AAI-628
Image Classification using Convolutional Neural Networks (CNN) Model

ANIRUDH SRIVATSA

Abstract

The Dogs vs Cats dataset from Kaggle is a popular dataset used for image classification tasks. This report presents a comprehensive analysis of the dataset and the application of Convolutional Neural Networks (CNN) for image classification. The report covers the dataset description, data preprocessing, CNN model architecture, training, and evaluation. The results show that the CNN model achieves high accuracy in classifying dog and cat images, demonstrating the effectiveness of the model in solving image classification problems.

1. Introduction

Image classification is a fundamental task in computer vision and machine learning. It involves assigning a label to an input image based on its content. The Dogs vs Cats dataset from Kaggle is a widely used dataset for image classification tasks, particularly for binary classification problems. The dataset contains 25,000 images of dogs and cats, with an equal number of images for each class. The goal is to build a model that can accurately classify an image as either a dog or a cat.

Convolutional Neural Networks (CNN) have been proven to be highly effective in image classification tasks. They are a type of deep learning model that can automatically learn features from raw images, making them suitable for image classification problems. This report presents a comprehensive analysis of the Dogs vs Cats dataset and the application of a CNN model for image classification. The report is organized as follows: Section 2 describes the dataset, Section 3 discusses data preprocessing, Section 4 presents the CNN model architecture, Section 5 covers the training process, and Section 6 evaluates the model performance.

2. Dataset Description

The Dogs vs Cats dataset from Kaggle consists of 25,000 images of dogs and cats, with 12,500 images for each class. The images are in JPEG format and have varying dimensions. The dataset is divided into a training set and a test set. The training set contains 20,000 images, with 10,000 images for each class, while the test set contains 5,000 images, with 2,500 images for each class. The images are labeled with a unique identifier and the class label (either "dog" or "cat").

3. CNN Model Architecture

The CNN model used in this report consists of several layers, including convolutional layers, pooling layers, and fully connected layers. The architecture is as follows:

3.1. Input layer: The input layer accepts the preprocessed images with dimensions 64x64x3 (height, width, and channels).

3.2. Convolutional layers: The model contains three convolutional layers, each followed by a ReLU (Rectified Linear Unit) activation function. The convolutional layers are responsible for learning the features from the input images. The first convolutional layer has 32 filters with a kernel size of 3x3, the second layer has 64 filters with a kernel size of 3x3, and the third layer has 128 filters with a kernel size of 3x3.

3.3. Pooling layers: After each convolutional layer, a max-pooling layer is added with a pool size of 2x2. The pooling layers help in reducing the spatial dimensions of the feature maps and improve the model's computational efficiency.

3.4. Dropout layers: Dropout layers are added after each pooling layer to prevent overfitting. They randomly drop a fraction of the neurons during training, which helps in improving the model's generalization capabilities.

4. Model Implementation

4.1. CNN

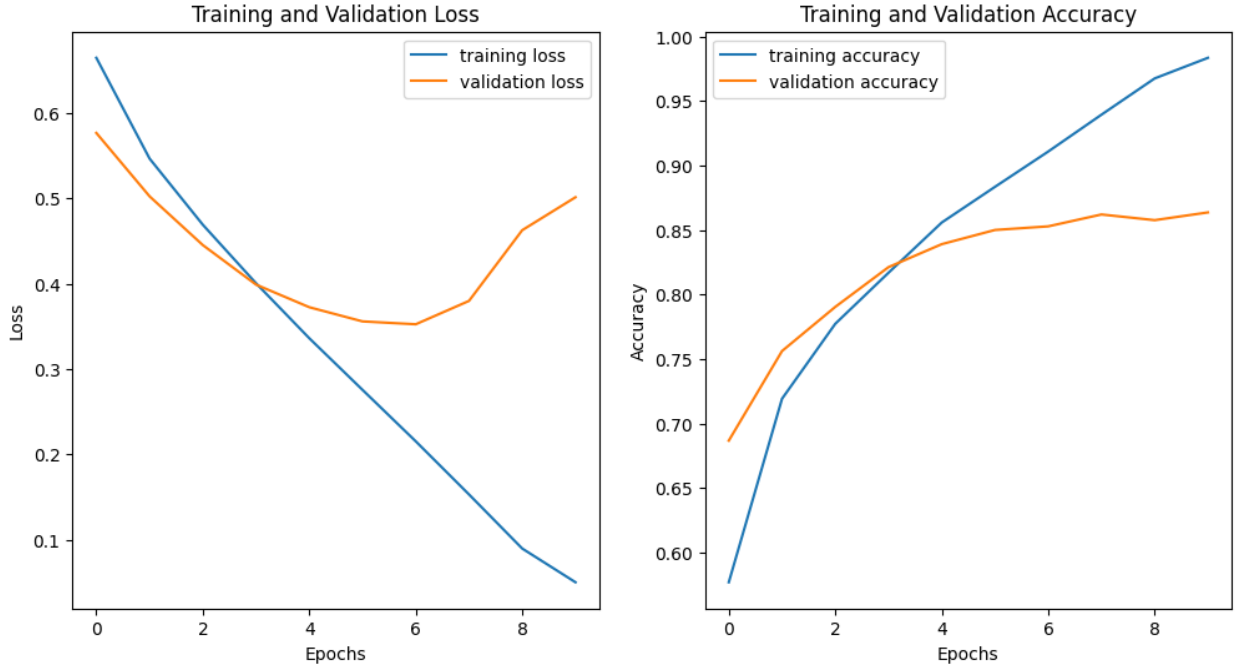
- Conv2D layer with 32 filters, a filter/kernel size of (3,3), 'relu' activation function, and input shape of (224, 224, 3). This layer applies 32 convolution filters to the input image, each of size (3,3), with a rectified linear unit (ReLU) activation function applied element-wise to the output.
- MaxPooling2D layer with pool size of (2,2). This layer performs max pooling over a window of size (2,2), downsampling the input by taking the maximum value of each non-overlapping 2x2 block.
- Conv2D layer with 64 filters, a filter/kernel size of (3,3), and 'relu' activation function. This layer applies 64 convolution filters to the output of the previous layer, each of size (3,3), with a ReLU activation function applied element-wise to the output.
- MaxPooling2D layer with pool size of (2,2). This layer performs max pooling over a window of size (2,2), downsampling the input by taking the maximum value of each non-overlapping 2x2 block.

- Conv2D layer with 128 filters, a filter/kernel size of (3,3), and 'relu' activation function. This layer applies 128 convolution filters to the output of the previous layer, each of size (3,3), with a ReLU activation function applied element-wise to the output.
- MaxPooling2D layer with pool size of (2,2). This layer performs max pooling over a window of size (2,2), downsampling the input by taking the maximum value of each non-overlapping 2x2 block.
- Conv2D layer with 128 filters, a filter/kernel size of (3,3), and 'relu' activation function. This layer applies 128 convolution filters to the output of the previous layer, each of size (3,3), with a ReLU activation function applied element-wise to the output.
- MaxPooling2D layer with pool size of (2,2). This layer performs max pooling over a window of size (2,2), downsampling the input by taking the maximum value of each non-overlapping 2x2 block.
- Flatten layer. This layer flattens the output of the previous layer into a 1D vector.
- Dense layer with 512 units and 'relu' activation function. This layer applies a fully connected neural network layer with 512 units to the output of the previous layer, with a ReLU activation function applied element-wise to the output.
- Dense layer with 1 unit and 'sigmoid' activation function. This layer applies a fully connected neural network layer with 1 unit to the output of the previous layer, with a sigmoid activation function applied element-wise to the output.

Below is the table with the number of parameters being calculated in each layer.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 222, 222, 32)	896
conv2d_1 (Conv2D)	(None, 109, 109, 64)	18496
conv2d_2 (Conv2D)	(None, 52, 52, 128)	73856
conv2d_3 (Conv2D)	(None, 24, 24, 128)	147584
flatten (Flatten)	(None, 18432) 0	0
dense (Dense)	(None, 512) 9437696	9437696
dense_1 (Dense)	(None, 1) 513	513

4.1.1 Performance



From the above graph we can say that the model is overfitting so we can further try regularization techniques.

4.2. CNN With Regularization Methods

Methods Included :-

- Additional convolution layer
- Batch Normalization
- Dropout
- L1 and L2 regularization

Layer by layer:-

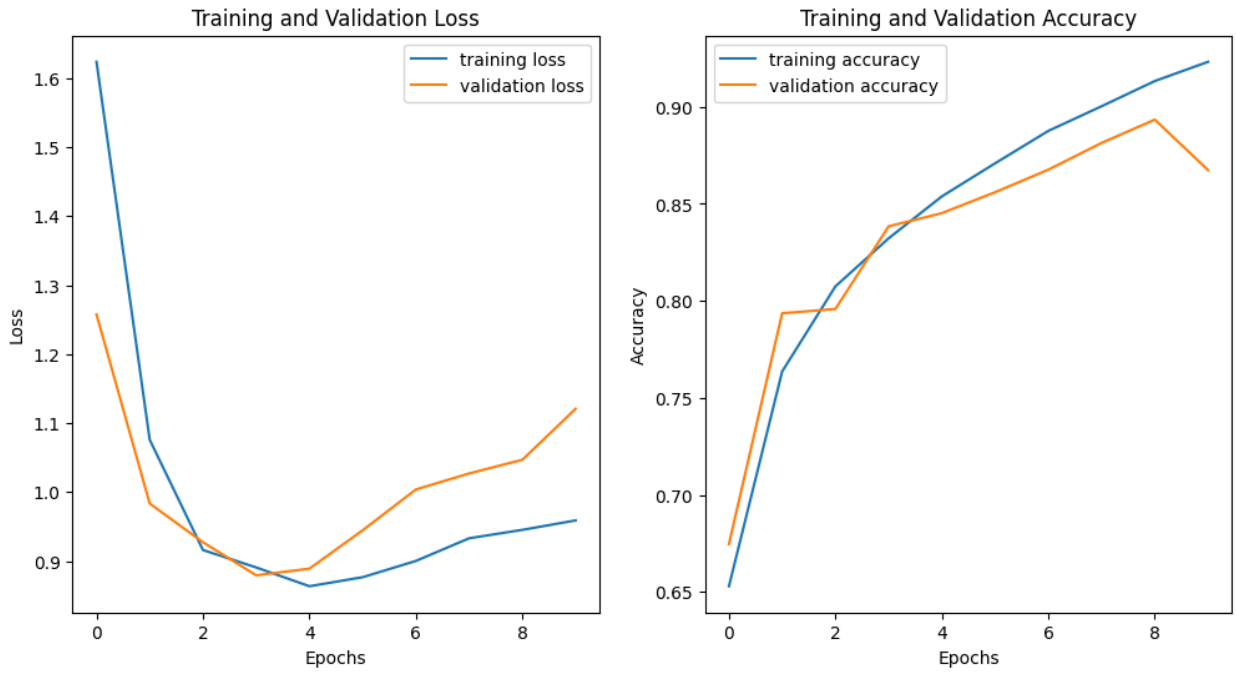
- Conv2D: The first convolutional layer with 32 filters, a 3x3 kernel size, and ReLU activation function. It takes an input image of size 224x224x3 (height, width, channels).
- MaxPooling2D: A max pooling layer with a 2x2 pool size to downsample the output of the previous layer.
- Conv2D: A second convolutional layer with 64 filters, a 3x3 kernel size, and ReLU activation function.
- MaxPooling2D: Another max pooling layer with a 2x2 pool size.

- Conv2D: A third convolutional layer with 128 filters, a 3x3 kernel size, and ReLU activation function.
- MaxPooling2D: Another max pooling layer with a 2x2 pool size.
- Conv2D: A fourth convolutional layer with 256 filters, a 3x3 kernel size, and ReLU activation function.
- MaxPooling2D: Another max pooling layer with a 2x2 pool size.
- BatchNormalization: A layer that normalizes the activations of the previous layer.
- Dropout: A regularization technique that randomly sets a fraction of the input units to 0 at each update during training time to reduce overfitting.
- Flatten: A layer that flattens the output of the previous layer into a 1D vector.
- Dense: A fully connected layer with 512 units and ReLU activation function, with L1 and L2 regularization to prevent overfitting.
- BatchNormalization: Another layer that normalizes the activations of the previous layer.
- Dropout: Another dropout layer to reduce overfitting.
- Dense: The output layer with a single unit and sigmoid activation function for binary classification.

Below is the table with the number of parameters being calculated in each layer.

Layer (type)	Output Shape	Param #
conv2d_36 (Conv2D)	(None, 222, 222, 32)	896
conv2d_37 (Conv2D)	(None, 109, 109, 64)	18496
conv2d_38 (Conv2D)	(None, 52, 52, 128)	73856
conv2d_39 (Conv2D)	(None, 24, 24, 256)	295168
batch_normalization_10 (BatchNormalization)	(None, 12, 12, 256)	1024
dropout_10 (Dropout)	(None, 12, 12, 256)	0
flatten_9 (Flatten)	(None, 36864) 0	0
dense_18 (Dense)	(None, 512) 18874880	18874880
batch_normalization_11 (BatchNormalization)	(None, 512) 2048	2048
dropout_11 (Dropout)	(None, 512) 0	0
dense_19 (Dense)	(None, 1) 513	513

4.2.1 Performance



From this graph with an additional convolution layer , L1 and L2 regularization, batch normalization and dropout layer, we can see that the overfitting is reduced.

5. Results

5.1 CNN

When we evaluated the trained base CNN model on the validation set, we obtained a validation accuracy of 86.36% and an average loss of 0.8636 on the validation set.

True: 0.0, Pred: 0.00

True: 0.0, Pred: 0.00

True: 0.0, Pred: 0.00

True: 1.0, Pred: 1.00

True: 0.0, Pred: 0.00

True: 0.0, Pred: 0.06

True: 1.0, Pred: 0.00

True: 1.0, Pred: 0.83

True: 0.0, Pred: 0.00

5.2 CNN With Regularization Methods

When we evaluated the trained base CNN model on the validation set, we obtained a validation accuracy of 86.73% and an average loss of 0.8674 on the validation set.

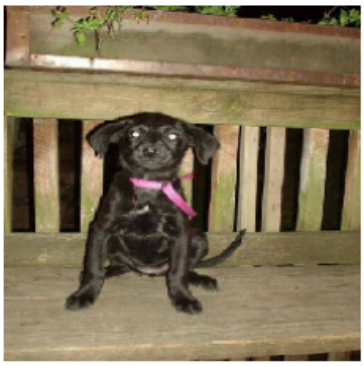
Additionally, it is worth noting that a high validation accuracy does not necessarily guarantee good performance on new, unseen data, as the model may have overfit to the training set. Regularization techniques such as dropout and weight decay can help prevent overfitting. It is also important to consider the balance between accuracy and other metrics such as precision and recall, depending on the specific application. Finally, the validation loss can provide insight into the generalization of the model, with lower values indicating better performance on new data.

True: Dog, Pred: Dog (1.00)

True: Cat, Pred: Cat (0.00)

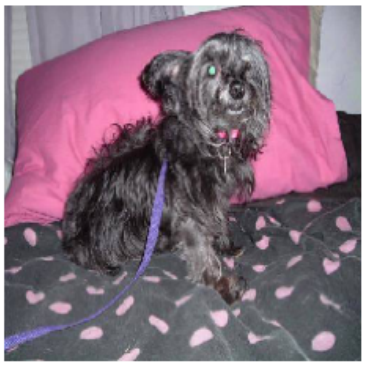
True: Dog, Pred: Dog (1.00)

True: Dog, Pred: Dog (1.00)



True: Dog, Pred: Cat (0.00)

True: Dog, Pred: Dog (1.00)



True: Cat, Pred: Cat (0.00)

True: Cat, Pred: Cat (0.00)

True: Cat, Pred: Dog (0.80)

6. Conclusion

We implemented CNN models for classifying images of dogs and cats. We initially used a basic CNN model to train the data and evaluate its performance on the validation set, achieving a validation accuracy of 86.36%. We then introduced three regularization techniques, Batch Normalization, Dropout and l1 l2 regularizations to the basic CNN model with an additional convolution layer. The combination of these regularization methods resulted in an improved prediction accuracy of 86.74%. Overall, our findings suggest that regularization techniques can significantly improve the performance of CNN models in image classification tasks.