Image Classification using Convolutional Neural Networks

Project Report

1. Introduction:

Image classification is a common task in computer vision, and Convolutional Neural Networks (CNNs) have shown remarkable success in solving such problems. This project focuses on developing a CNN model to classify images of cats and dogs.

2. Dataset:

The dataset used for this project consists of images of cats and dogs. The dataset is divided into a training set and a test set. The training set contains 40 images, with 20 images of cats and 20 images of dogs. The test set contains 20 images, with an equal number of cats and dogs.

3. Model Architecture:

The CNN model used for image classification comprises several layers, each serving a specific purpose:

- Input Layer: This layer receives input images with dimensions of 256x256 pixels and 3 color channels (RGB).

- Convolutional Layer 1: This layer consists of 32 filters with a kernel size of 5x5 and uses the ReLU activation function. It also applies L2 regularization to reduce overfitting.

- Batch Normalization Layer 1: Normalizes the outputs of the previous layer to stabilize training.

- Max Pooling Layer 1: Performs max pooling with a pool size of 2x2 to downsample the feature maps.

- Dropout Layer 1: Applies dropout with a rate of 0.4 to mitigate overfitting.

```
[ ]  # first convolutional layer
     classifier.add(Conv2D(32, 5, 5, input_shape = (256, 256, 3), activation = 'relu',kernel_regularizer=l2(l2=0.01)))
     classifier.add(BatchNormalization())

[ ]  # first pooling layer
     classifier.add(MaxPooling2D(pool_size=(2,2)))
     classifier.add(Dropout(0.4))
```

- Convolutional Layer 2: This layer includes 64 filters with a kernel size of 5x5 and uses the ReLU activation function. It also applies L2 regularization.

- Batch Normalization Layer 2: Normalizes the outputs of the previous layer.

- Max Pooling Layer 2: Performs max pooling with a pool size of 2x2.

- Dropout Layer 2: Applies dropout with a rate of 0.4.

```
     # second convolutional layer
     classifier.add(Conv2D(64, 5,5 ,activation = 'relu', kernel_regularizer= l2(l2=0.01)))
     classifier.add(BatchNormalization())

[ ]  # second pooling layer
     classifier.add(MaxPooling2D(pool_size=(2,2)))
     classifier.add(Dropout(0.4))
```

- Flattening Layer: Converts the 2D feature maps into a 1D feature vector.

- Fully Connected Layer 1: This layer has 32 units and uses the ReLU activation function.

- Dropout Layer 3: Applies dropout with a rate of 0.4.

- Output Layer: Consists of a single unit with sigmoid activation, representing the binary classification task (cat or dog).

```
# Full connections
classifier.add(Dense(32, activation='relu'))
classifier.add(Dropout(0.4))
classifier.add(Dense(1, activation='sigmoid'))

[ ] classifier.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

4. Model Training:

The model is trained using the training set and evaluated on the test set. The following steps are performed during training:

- Data Augmentation: The training images are augmented using the ImageDataGenerator class from Keras. The augmentation techniques include rescaling, random shearing, random zooming, and horizontal flipping.

```
# generate more images
from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip =True)

[ ] test_datagen = ImageDataGenerator(rescale = 1./255)
```

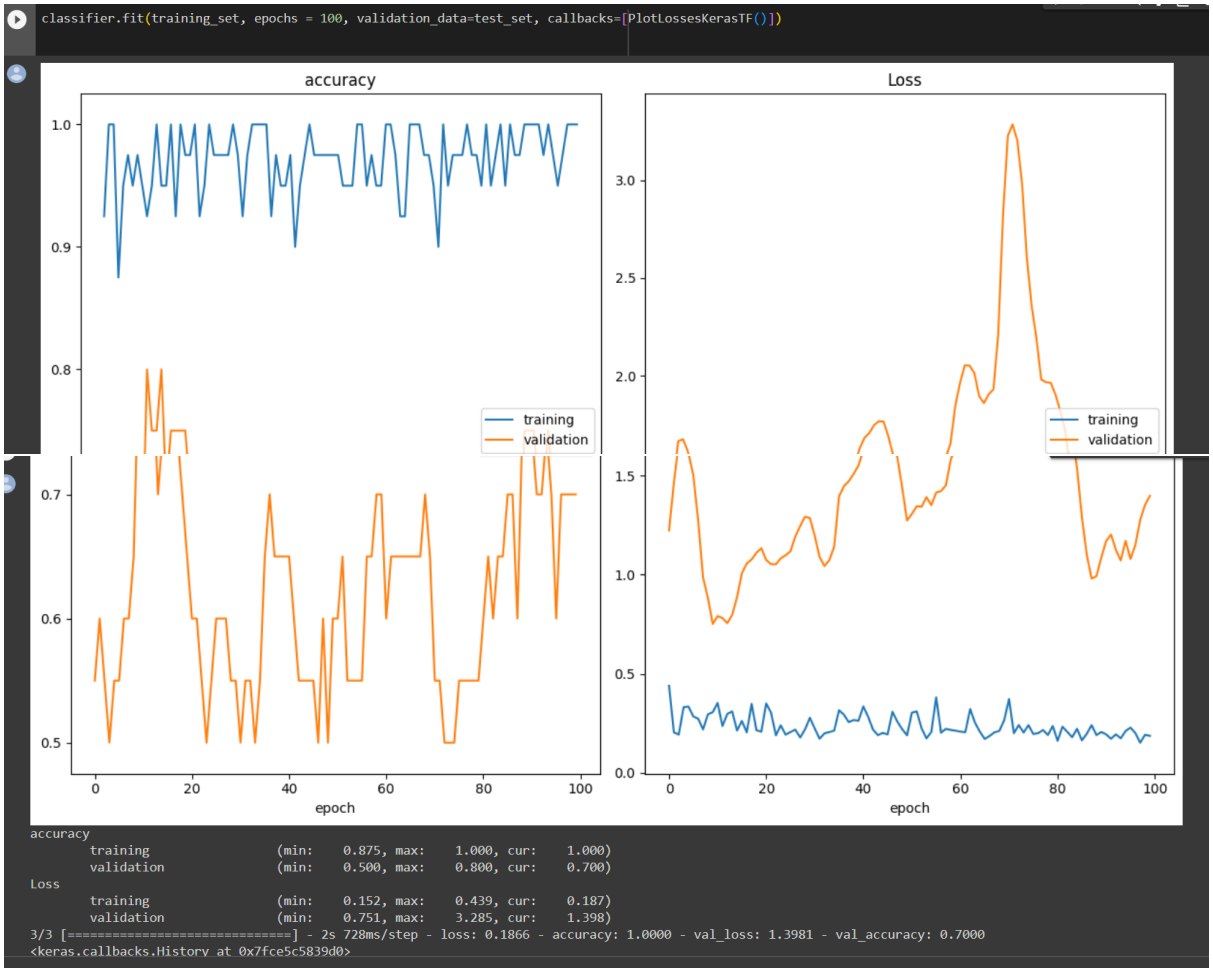
- Compilation: The model is compiled with the Adam optimizer, binary cross-entropy loss function, and accuracy as the evaluation metric.

- Training: The model is trained for multiple epochs (100, 200, and 300) using the fit() function. The training progress is monitored using the PlotLossesKerasTF callback, which provides real-time visualization of the loss and accuracy metrics during training.

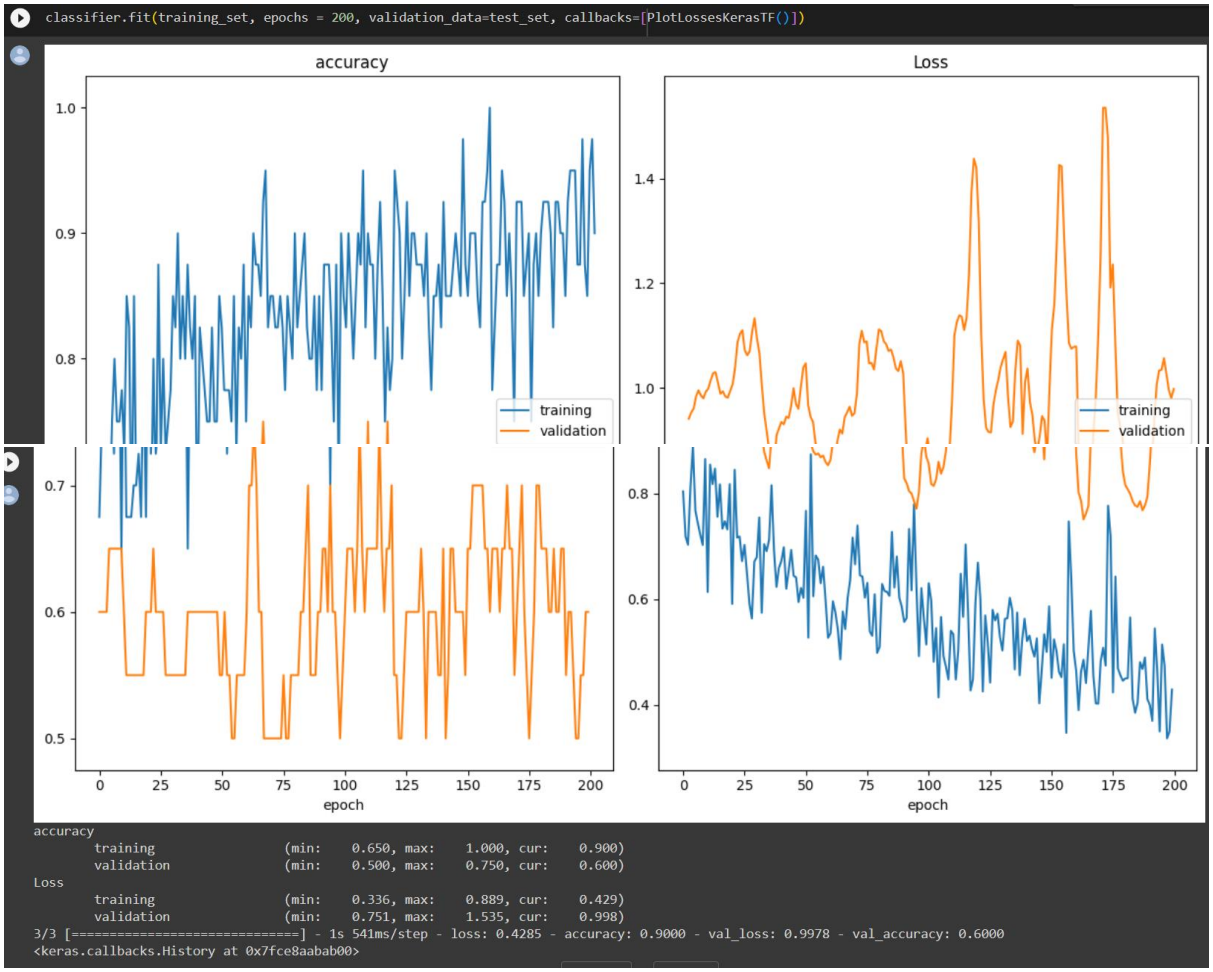
5. Results:

The model's performance is evaluated based on accuracy and loss metrics on both the training and test sets. Here are the key results obtained during training:

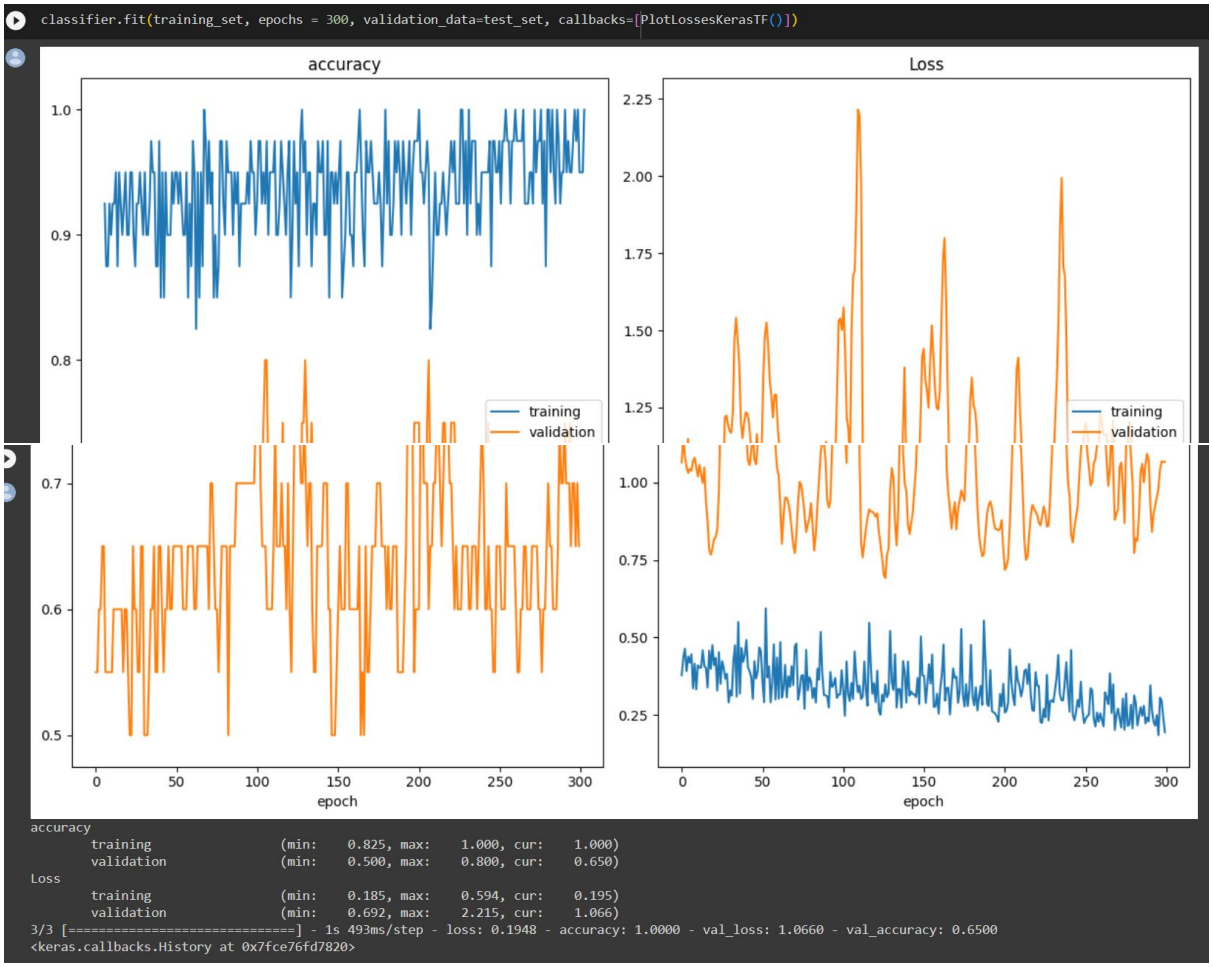
- After 100 epochs: The model achieved a training accuracy of 100% and a validation accuracy of 70%.



- After 200 epochs: The model achieved a training accuracy of 90% and a validation accuracy of 60%.



- After 300 epochs: The model achieved a training accuracy of 100% and a validation accuracy of 65%.



6. Predictions:

After training, the model is used to make predictions on individual images. The model calculates the probability of an image containing a cat or a dog. If the probability exceeds 0.5, the image is classified as a cat; otherwise, it is classified as a dog.

7. Conclusion:

In this project, a CNN model was developed for image classification of cats and dogs. The model showed promising results in differentiating between the two classes. However, further improvements can be made by adjusting the model architecture, increasing the size of the dataset, or fine-tuning hyperparameters. Overall, this project demonstrates the effectiveness of CNNs in image classification tasks.