classification-using-cnn-keras

June 30, 2024

1 Dogs vs. Cats Image Classification using CNN Keras

1.1 1. Introduction

- This project demonstrates the use of Convolutional Neural Networks (CNNs) to classify images of dogs and cats. The dataset used is the "Dogs vs. Cats" dataset from Kaggle. The model is built using Keras with TensorFlow as the backend.

1.2 2. Prerequisites

- Python 3.x
- TensorFlow
- Keras
- Kaggle API
- OpenCV
- Matplotlib

1.3 3. Steps Involved

1.3.1 Step 1: Setting Up Kaggle API

- To download the dataset from Kaggle, you need to set up the Kaggle API. Place your `kaggle.json` file, which contains your Kaggle credentials, in the appropriate directory. This step involves configuring your environment to authenticate with the Kaggle API and download datasets directly from Kaggle.

1.3.2 Step 2: Downloading the Dataset

- Use the Kaggle API to download the "Dogs vs. Cats" dataset. This involves using Kaggle's command-line interface to fetch the dataset from the specified URL.

1.3.3 Step 3: Extracting the Dataset

- Unzip the downloaded dataset to extract its contents. The dataset typically includes a large number of images organized into directories.

1.3.4 Step 4: Data Preparation

1. **Loading Images**: Use Keras utilities to load and preprocess the images from the dataset directory. This includes specifying the directory, batch size, and image dimensions.

2. **Normalization**: Normalize the pixel values of the images to a range between 0 and 1 to improve the training performance and convergence of the neural network.

1.3.5 Step 5: Model Definition

1. **Model Architecture**: Define the architecture of the CNN using Keras' Sequential API. The model consists of convolutional layers, max-pooling layers, a flatten layer, and dense (fully connected) layers.
    - **Convolutional Layers**: Extract features from the input images using filters.
    - **Max-Pooling Layers**: Reduce the spatial dimensions of the feature maps and reduce overfitting.
    - **Flatten Layer**: Convert the 2D feature maps to a 1D feature vector.
    - **Dense Layers**: Perform classification based on the extracted features.
2. **Activation Functions**: Use ReLU (Rectified Linear Unit) activation function for the convolutional and dense layers to introduce non-linearity into the model.
3. **Output Layer**: Use a sigmoid activation function in the output layer for binary classification (dog vs. cat).

1.3.6 Step 6: Model Compilation

Compile the model by specifying the optimizer, loss function, and evaluation metrics. - **Optimizer**: Use the Adam optimizer, which adjusts the learning rate during training. - **Loss Function**: Use binary crossentropy as the loss function, suitable for binary classification tasks. - **Metrics**: Track accuracy during training and validation.

1.3.7 Step 7: Model Training

- Train the model using the training dataset and validate it using the validation dataset. This involves specifying the number of epochs and the datasets for training and validation.

1.3.8 Step 8: Model Evaluation

- Evaluate the model's performance on the validation dataset by calculating the loss and accuracy.

1.3.9 Step 9: Visualizing Training Results

1. **Accuracy**: Plot the training and validation accuracy over the epochs to visualize the model's learning process.
2. **Loss**: Plot the training and validation loss over the epochs to monitor the model's performance and detect overfitting.

1.3.10 Step 10: Making Predictions

1. **Preprocess Test Images**: Read and preprocess test images (resize and reshape) to match the input format expected by the model.
2. **Predict Classes**: Use the trained model to predict the class (dog or cat) of the test images.
3. **Visualization**: Display the test images and the predicted labels.

1.4 4. Key Concepts

- **Convolutional Neural Networks (CNNs)**: A type of deep learning model specifically designed for image data. CNNs are effective at automatically and adaptively learning spatial hierarchies of features.
- **Image Normalization**: Scaling pixel values to a range of 0 to 1 to improve model performance.
- **Activation Functions**: Functions like ReLU introduce non-linearity, helping the network to learn complex patterns.
- **Pooling Layers**: Reduce the dimensionality of feature maps, which helps in reducing computational load and controlling overfitting.
- **Flatten Layer**: Converts 2D feature maps into a 1D vector for input to dense layers.
- **Dense Layers**: Fully connected layers that interpret the features extracted by convolutional layers to make the final prediction.
- **Dropout**: A regularization technique to prevent overfitting by randomly setting a fraction of input units to zero at each update during training.
- **Model Compilation**: Process of configuring the model with an optimizer, loss function, and evaluation metrics.
- **Training and Validation**: Training the model on a dataset and evaluating its performance on a separate validation set to monitor for overfitting.

1.5 5. Technical Words Definitions

- **Convolutional Layer**: A layer in a CNN that applies convolution operations to the input, using filters to extract features such as edges, textures, and patterns from the images.
- **Filter (Kernel)**: A small matrix used in convolutional layers to detect specific features in the input image. It slides over the image to produce a feature map.
- **Feature Map**: The output of a convolutional layer after applying filters to the input image. It highlights the presence of specific features.
- **Max-Pooling Layer**: A layer that reduces the spatial dimensions (width and height) of the feature maps by taking the maximum value in a small region, which helps to down-sample and reduce computation.
- **ReLU (Rectified Linear Unit)**: An activation function that outputs the input directly if it is positive; otherwise, it outputs zero. It introduces non-linearity to the model.
- **Sigmoid Activation Function**: An activation function that maps input values to a range between 0 and 1, often used in binary classification tasks to represent probabilities.
- **Optimizer**: An algorithm that adjusts the weights of the neural network during training to minimize the loss function. Adam is a popular optimizer that adapts the learning rate based on past gradients.
- **Loss Function**: A function that measures how well the model's predictions match the actual labels. Binary crossentropy is used for binary classification tasks.
- **Epoch**: One complete pass through the entire training dataset. Training for multiple epochs helps the model to learn better.
- **Batch Size**: The number of training examples used in one iteration of model training. Smaller batch sizes lead to more updates and potential faster convergence.
- **Accuracy**: A metric that measures the proportion of correct predictions made by the model.
- **Overfitting**: A situation where the model performs well on the training data but poorly on unseen data. Regularization techniques like dropout help to mitigate overfitting.

- **Regularization**: Techniques used to prevent overfitting by adding constraints or penalties to the model.

1.6 6. Training and Validation Accuracy

- During the training process, the model's performance on the training and validation datasets is monitored. The following results were obtained over 10 epochs:

- **Epoch 1**:
  - Training Accuracy: 65.82%
  - Validation Accuracy: 74.34%
- **Epoch 2**:
  - Training Accuracy: 76.34%
  - Validation Accuracy: 81.73%
- **Epoch 3**:
  - Training Accuracy: 81.91%
  - Validation Accuracy: 85.45%
- **Epoch 4**:
  - Training Accuracy: 87.66%
  - Validation Accuracy: 89.37%
- **Epoch 5**:
  - Training Accuracy: 93.34%
  - Validation Accuracy: 92.40%
- **Epoch 6**:
  - Training Accuracy: 96.46%
  - Validation Accuracy: 92.88%
- **Epoch 7**:
  - Training Accuracy: 97.55%
  - Validation Accuracy: 93.52%
- **Epoch 8**:
  - Training Accuracy: 98.22%
  - Validation Accuracy: 95.86%
- **Epoch 9**:
  - Training Accuracy: 98.72%
  - Validation Accuracy: 96.66%
- **Epoch 10**:
  - Training Accuracy: 98.83%
  - Validation Accuracy: 96.71%

- These results show that the model improved its performance over time, with both training and validation accuracy increasing across epochs.

1.7 7. Conclusion

- This project provides a comprehensive example of how to build, train, and evaluate a Convolutional Neural Network for image classification using Keras and TensorFlow.

[1]:
```
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
```

```
cp: cannot stat 'kaggle.json': No such file or directory
```

[2]:
```
!kaggle datasets download -d salader/dogs-vs-cats
```

```
Dataset URL: https://www.kaggle.com/datasets/salader/dogs-vs-cats
License(s): unknown
Downloading dogs-vs-cats.zip to /content
100% 1.06G/1.06G [00:14<00:00, 107MB/s]
100% 1.06G/1.06G [00:14<00:00, 77.9MB/s]
```

[3]:
```
import zipfile
zip_ref = zipfile.ZipFile('/content/dogs-vs-cats.zip','r')
zip_ref.extractall('/content')
zip_ref.close()
```

[4]:
```
import tensorflow as tf
from tensorflow import keras
from keras import Sequential
from keras.layers import Dense,Conv2D,MaxPooling2D,Flatten
```

[5]:
```
# generators
train_ds = keras.utils.image_dataset_from_directory(
    directory = '/content/train',
    labels ="inferred",
    label_mode ="int",
    batch_size =32,
    image_size = (256,256)
)

validation_ds = keras.utils.image_dataset_from_directory(
    directory = '/content/train',
    labels ="inferred",
    label_mode ="int",
    batch_size =32,
    image_size = (256,256)
)
```

```
Found 20000 files belonging to 2 classes.
Found 20000 files belonging to 2 classes.
```

[6]:
```
# Normalize
def process(image,label):
  image = tf.cast(image/255. ,tf.float32)
  return image,label

train_ds = train_ds.map(process)
validation_ds = validation_ds.map(process)
```

[7]:
```
# Create CNN Model

model = Sequential()

model.
  ↪add(Conv2D(32,kernel_size=(3,3),padding='valid',activation='relu',input_shape=(256,256,3)))
model.add(MaxPooling2D(pool_size=(2,2),strides=2,padding='valid'))

model.add(Conv2D(64,kernel_size=(3,3),padding='valid',activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2),strides=2,padding='valid'))

model.add(Conv2D(128,kernel_size=(3,3),padding='valid',activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2),strides=2,padding='valid'))

model.add(Flatten())

model.add(Dense(128,activation='relu'))
model.add(Dense(64,activation='relu'))
model.add(Dense(1,activation='sigmoid'))
```

[8]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 254, 254, 32)      896

 max_pooling2d (MaxPooling2  (None, 127, 127, 32)      0
 D)

 conv2d_1 (Conv2D)           (None, 125, 125, 64)      18496

 max_pooling2d_1 (MaxPoolin  (None, 62, 62, 64)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 60, 60, 128)       73856
```

```
 max_pooling2d_2 (MaxPoolin  (None, 30, 30, 128)       0
 g2D)

 flatten (Flatten)           (None, 115200)            0

 dense (Dense)               (None, 128)               14745728

 dense_1 (Dense)             (None, 64)                8256

 dense_2 (Dense)             (None, 1)                 65

=================================================================
Total params: 14847297 (56.64 MB)
Trainable params: 14847297 (56.64 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[9]:
```
model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
```

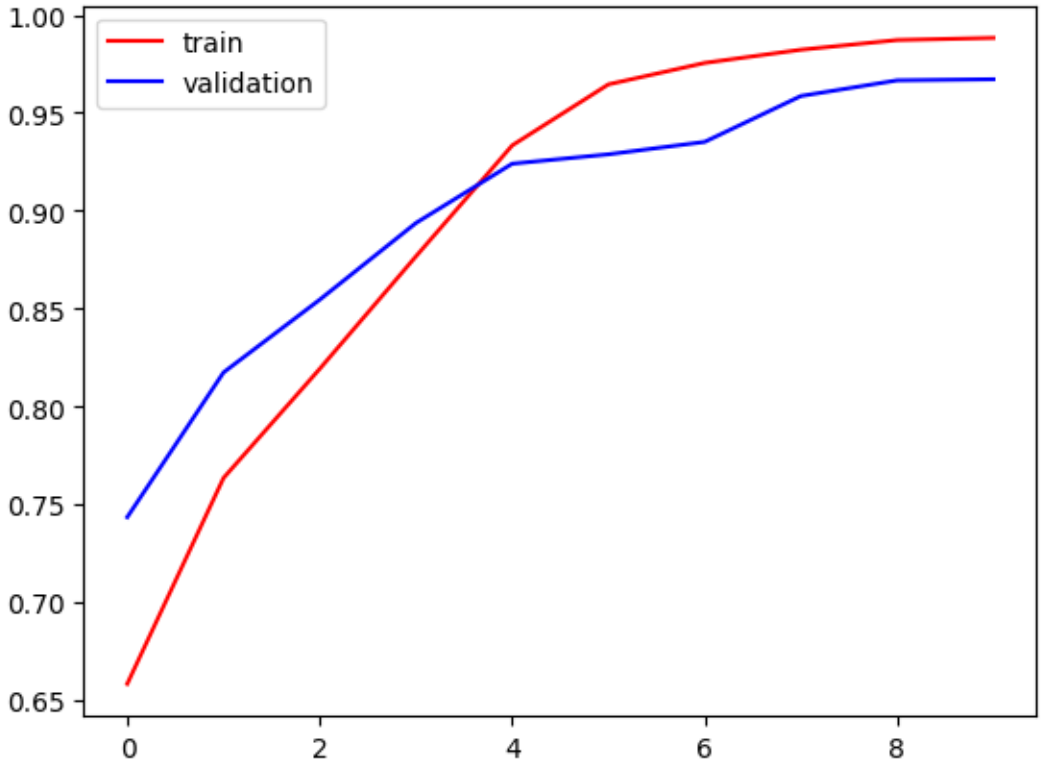
[10]:
```
history = model.fit(train_ds,epochs=10,validation_data=validation_ds)
```

```
Epoch 1/10
625/625 [==============================] - 94s 138ms/step - loss: 0.6089 -
accuracy: 0.6582 - val_loss: 0.5143 - val_accuracy: 0.7434
Epoch 2/10
625/625 [==============================] - 74s 118ms/step - loss: 0.4876 -
accuracy: 0.7634 - val_loss: 0.4030 - val_accuracy: 0.8173
Epoch 3/10
625/625 [==============================] - 70s 112ms/step - loss: 0.3950 -
accuracy: 0.8191 - val_loss: 0.3297 - val_accuracy: 0.8545
Epoch 4/10
625/625 [==============================] - 84s 134ms/step - loss: 0.2836 -
accuracy: 0.8766 - val_loss: 0.2486 - val_accuracy: 0.8937
Epoch 5/10
625/625 [==============================] - 84s 134ms/step - loss: 0.1674 -
accuracy: 0.9334 - val_loss: 0.2028 - val_accuracy: 0.9240
Epoch 6/10
625/625 [==============================] - 72s 114ms/step - loss: 0.0962 -
accuracy: 0.9646 - val_loss: 0.2314 - val_accuracy: 0.9288
Epoch 7/10
625/625 [==============================] - 84s 134ms/step - loss: 0.0700 -
accuracy: 0.9755 - val_loss: 0.2419 - val_accuracy: 0.9352
Epoch 8/10
625/625 [==============================] - 85s 134ms/step - loss: 0.0551 -
accuracy: 0.9822 - val_loss: 0.1200 - val_accuracy: 0.9586
Epoch 9/10
625/625 [==============================] - 72s 115ms/step - loss: 0.0355 -
```

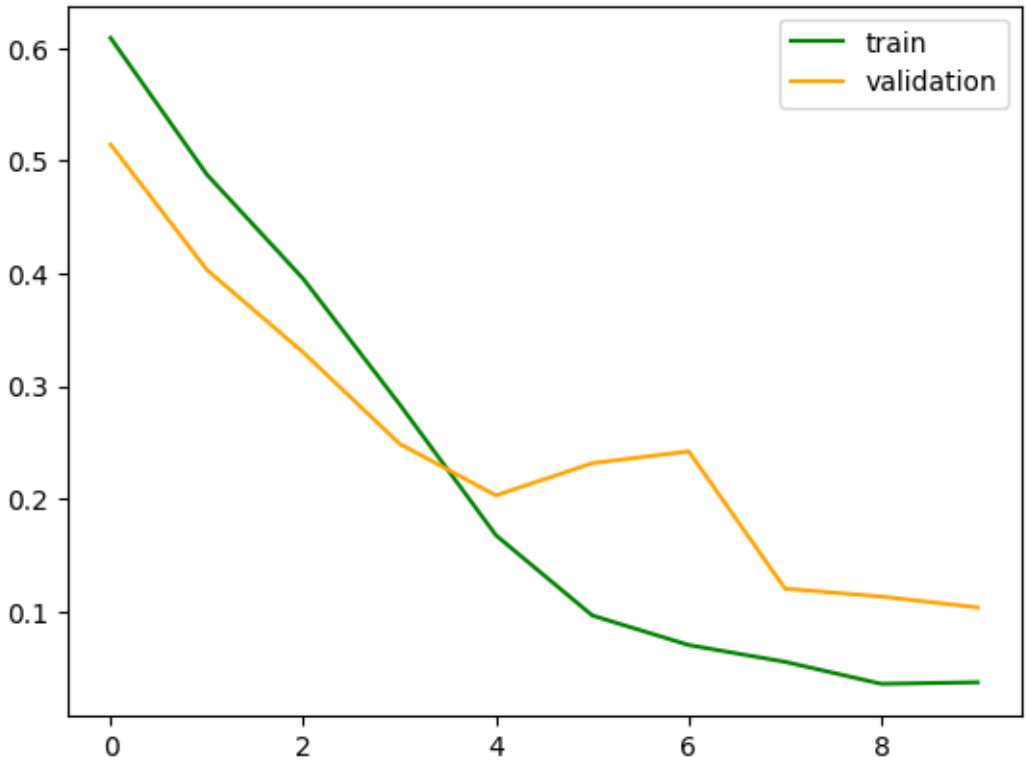
```
accuracy: 0.9872 - val_loss: 0.1130 - val_accuracy: 0.9666
Epoch 10/10
625/625 [==============================] - 72s 115ms/step - loss: 0.0369 -
accuracy: 0.9883 - val_loss: 0.1034 - val_accuracy: 0.9671
```

[11]:
```
import matplotlib.pyplot as plt

plt.plot(history.history['accuracy'],color='red',label='train')
plt.plot(history.history['val_accuracy'],color='blue',label='validation')
plt.legend()
plt.show()
```



[13]:
```
plt.plot(history.history['loss'],color='green',label='train')
plt.plot(history.history['val_loss'],color='orange',label='validation')
plt.legend()
plt.show()
```

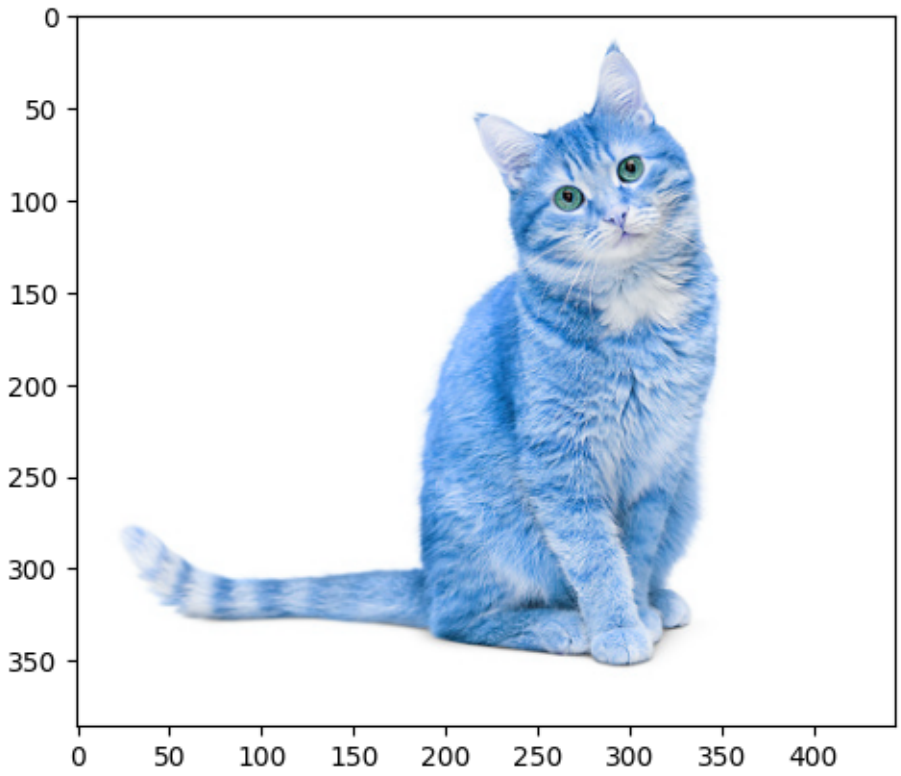


[14]:
```
import cv2
```

[15]:
```
test_img = cv2.imread('/content/cat.jpg')
```

[16]:
```
plt.imshow(test_img)
```

[16]: <matplotlib.image.AxesImage at 0x7c42e7575240>



[17]:
```
test_img.shape
```

[17]: (386, 445, 3)

[18]:
```
test_img = cv2.resize(test_img,(256,256))
```

[19]:
```
test_input = test_img.reshape((1,256,256,3))
```

[20]:
```
model.predict(test_input)
```

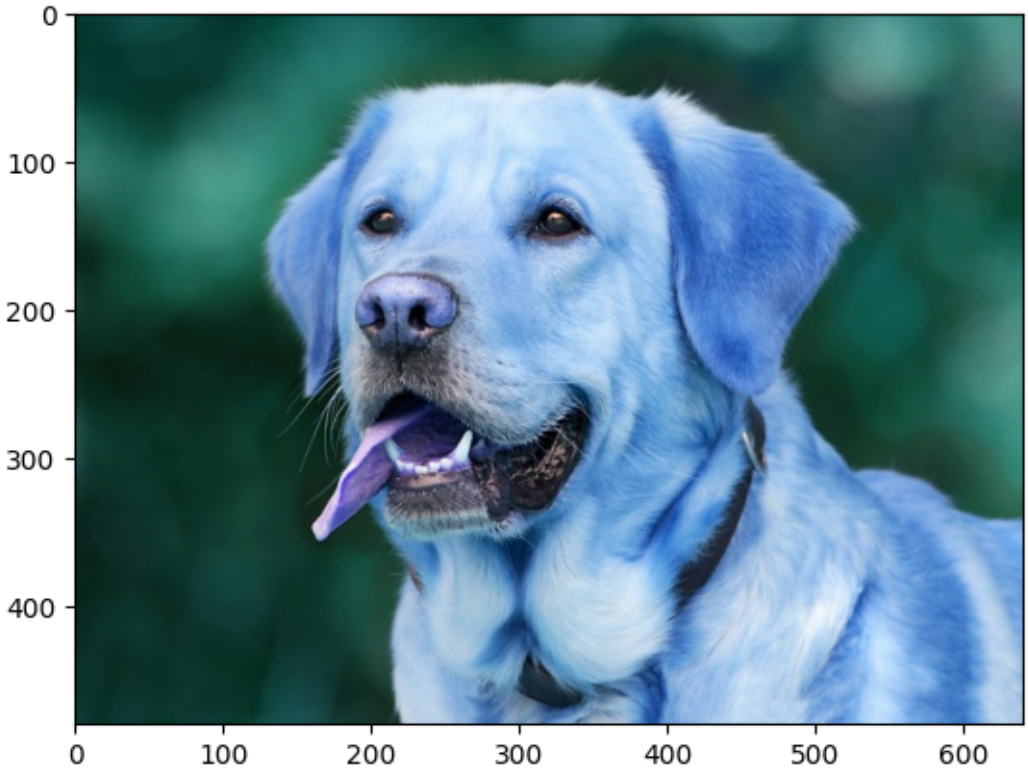
```
1/1 [==============================] - 0s 481ms/step
```

[20]: array([[0.]], dtype=float32)

[21]:
```
test_img = cv2.imread('/content/dog.jpg')
```

[22]:
```
plt.imshow(test_img)
```

[22]: <matplotlib.image.AxesImage at 0x7c42e595afe0>



[23]:
```
test_img.shape
```

[23]:
```
(480, 640, 3)
```

[24]:
```
test_img = cv2.resize(test_img,(256,256))
```

[25]:
```
test_input = test_img.reshape((1,256,256,3))
```

[26]:
```
model.predict(test_input)
```

```
1/1 [==============================] - 0s 29ms/step
```

[26]:
```
array([[1.]], dtype=float32)
```

Thank you !