Binary Image Classification for Pandas and Bears Utilizing Deep Learning and SVM

Eswar Hemant Majeti

G233R756

School of Computing, Wichita State University

CS 898BA: Image Analysis and Computer Vision

Cody Farlow

December 15, 2023

Abstract

In the context of Wildlife conservation and research the need for accurate and efficient identification of pandas and bears plays a vital role. Image classification is a technique where a model is developed to distinguish between these species, that helps wildlife monitoring and protection. This paper presents how pandas and bears are distinguished using binary image classification. The process involves training a Convolutional Neural Network (CNN) model, then the trained model output is given to SVM where feature extraction and classification of these images is performed. The performance metrics, including accuracy, F1-score, precision, and recall, are evaluated to assess the effectiveness of the model

Introduction

Modern methods are used in the fields of wildlife study and conservation to address issues related to protecting endangered animals. They operate at the intersection of ecological preservation and technological advancement. Image classification appears to be a crucial tool in this rapidly developing field that enables unparalleled animal species surveillance and identification. Large numbers of the species that are the focal point of this exploration remain as a cherished memory to them for pandas and bears in light of their natural importance and weakness.

The fascinating mega fauna of pandas and bears shows how biological systems must maintain a fine balance. Since they are important indices of biodiversity, their conservation is essential to preserving both the particular habitats and the overall health and resilience of ecosystems. But accurate species identification and monitoring—which, when done by hand, take a lot of time and effort—are crucial to conservation projects' success.

With the introduction of Convolutional Neural Networks (CNNs) for image classification as part of machine learning has the potential to completely alter current practices. For a specific grouping of panda and bear images, this paper intends to address the connection between innovation and creature insurance. On successful completion of this project, the conservationists and researchers will be able to track and manage these rare species easily, thereby preserving biodiversity.

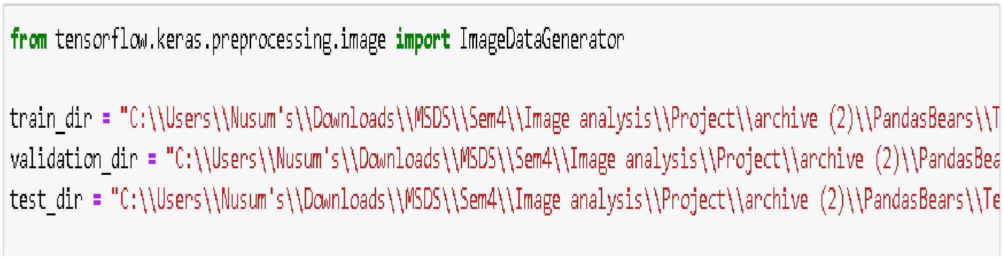
Beyond image classification of these images, the ultimate goal is to improve our understanding of migration patterns, population dynamics, and individual behaviors. This project aims to provide ecologists, researchers, and wildlife managers with an important tool for making decisions that promote the long-term survival of bear and panda populations by employing state-of-the-art photo classification algorithms.

In the following sections, the methodology for developing the binary image classification model, the architecture of the CNN, training processes, and results will be described.

Methodology

In Methodology binary image classification is performed to distinguish between pandas and bears within a dataset containing 600 images which are divided into 500 for training, 80 for validation, and 20 for testing. The dataset has variations in physical characteristics of these animals, captured under diverse settings, lighting conditions, and poses. Additionally, these images captured may vary in image quality, occlusions, and the potential presence of cubs.

A two-step approach is adopted to overcome these problems, combining the power of deep learning and traditional machine learning through Support Vector Machines (SVMs).



```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_dir = "C:\\Users\\Nusum's\\Downloads\\MSDS\\Sem4\\Image analysis\\Project\\archive (2)\\PandasBears\\T
validation_dir = "C:\\Users\\Nusum's\\Downloads\\MSDS\\Sem4\\Image analysis\\Project\\archive (2)\\PandasBea
test_dir = "C:\\Users\\Nusum's\\Downloads\\MSDS\\Sem4\\Image analysis\\Project\\archive (2)\\PandasBears\\Te
```

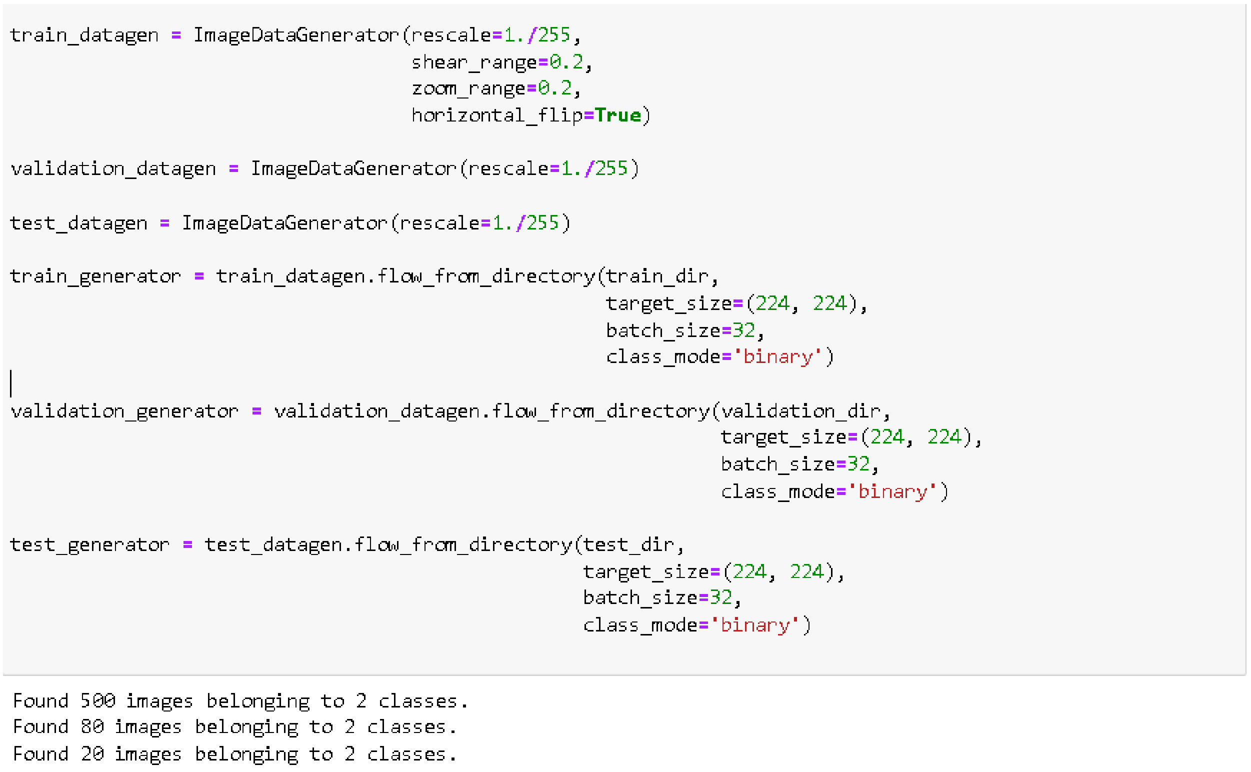
Figure 1: Define paths for training, validation, and test datasets.

1. **Deep Learning Model**

- A Convolutional Neural Network (CNN) is designed to automatically learn hierarchical representations from the input images. CNNs are particularly well-suited for image classification tasks, as they excel at capturing complex patterns and features.

- The CNN has convolutional layers for feature extraction, max pooling for downsampling, and activation functions (ReLU and Sigmoid) for non-linearity.

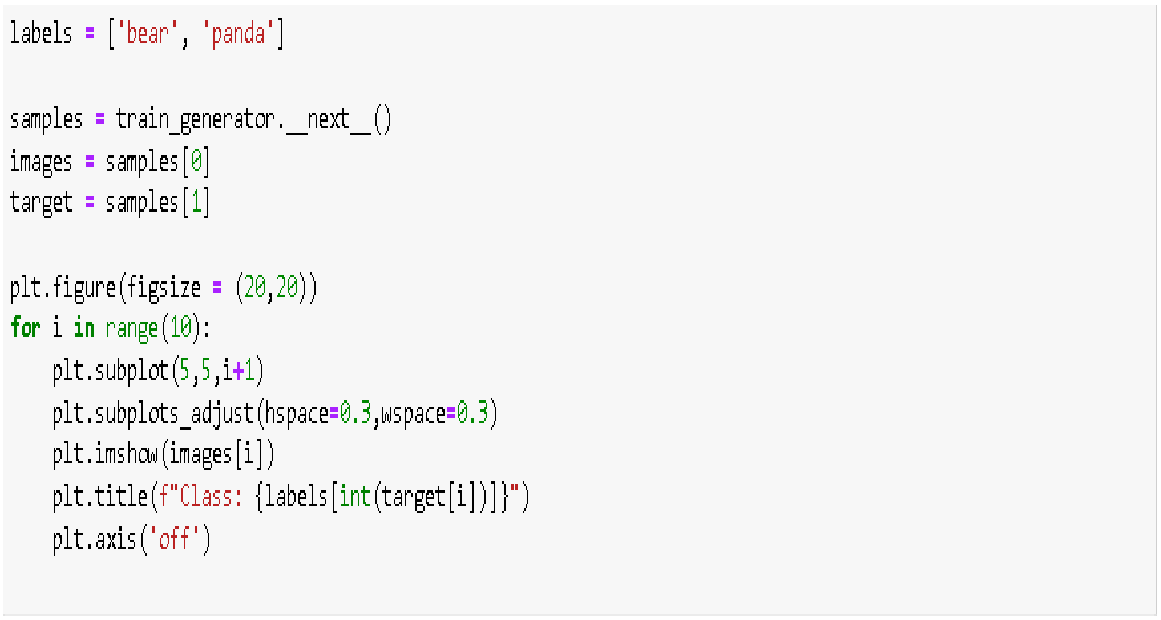
- The model is trained with a dataset of 500 images, 80 images are used for validation over multiple epochs.



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

validation_datagen = ImageDataGenerator(rescale=1./255)

test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(train_dir,
                                                    target_size=(224, 224),
                                                    batch_size=32,
                                                    class_mode='binary')

validation_generator = validation_datagen.flow_from_directory(validation_dir,
                                                              target_size=(224, 224),
                                                              batch_size=32,
                                                              class_mode='binary')

test_generator = test_datagen.flow_from_directory(test_dir,
                                                  target_size=(224, 224),
                                                  batch_size=32,
                                                  class_mode='binary')
```

```
Found 500 images belonging to 2 classes.
Found 80 images belonging to 2 classes.
Found 20 images belonging to 2 classes.
```

Figure 2: Create data generators for training, validation, and test sets



```
labels = ['bear', 'panda']

samples = train_generator.__next__()
images = samples[0]
target = samples[1]

plt.figure(figsize = (20,20))
for i in range(10):
    plt.subplot(5,5,i+1)
    plt.subplots_adjust(hspace=0.3,wspace=0.3)
    plt.imshow(images[i])
    plt.title(f"Class: {labels[int(target[i])]}")
    plt.axis('off')
```

Figure 3: Display images from training dataset

Figure 4: Sample images

CNN Model Architecture:

1. The CNN model is designed using of Convolutional layers and max pooling layers to calculate the maximum value. RELU and Sigmoid are used as activation functions.
2. To create a single, lengthy continuous linear vector from all the pooled feature maps' resulting 2-Dimensional arrays a Flatten layer is used. To categorize the image, the flattened matrix is given as input to the fully linked layer.
3. Each neuron in the straightforward layer of neurons known as the "dense layer", this layer receives input from every neuron in the layer. Images are classified using the dense layer based.
4. The loss function, learning rate, optimizer, and other metrics are determined using Compile function.
5. Training and validation of the model are done by the inputs given to FIT_Generator.

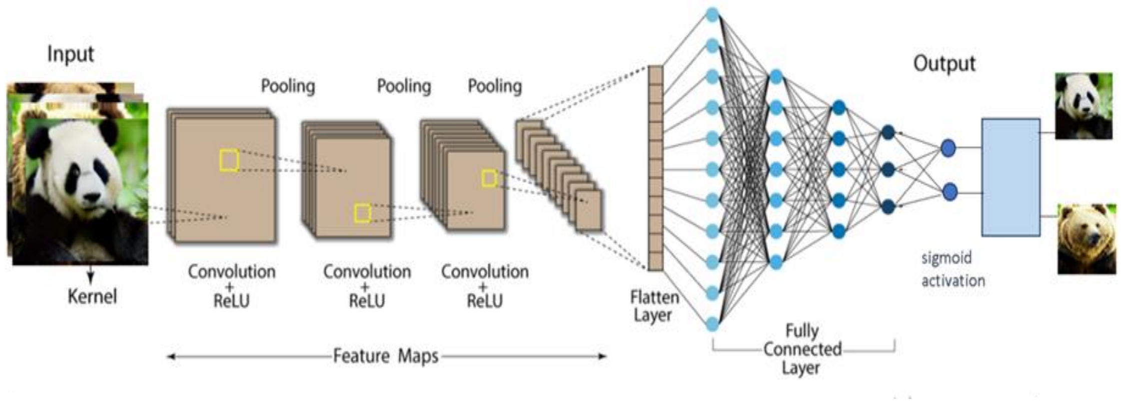
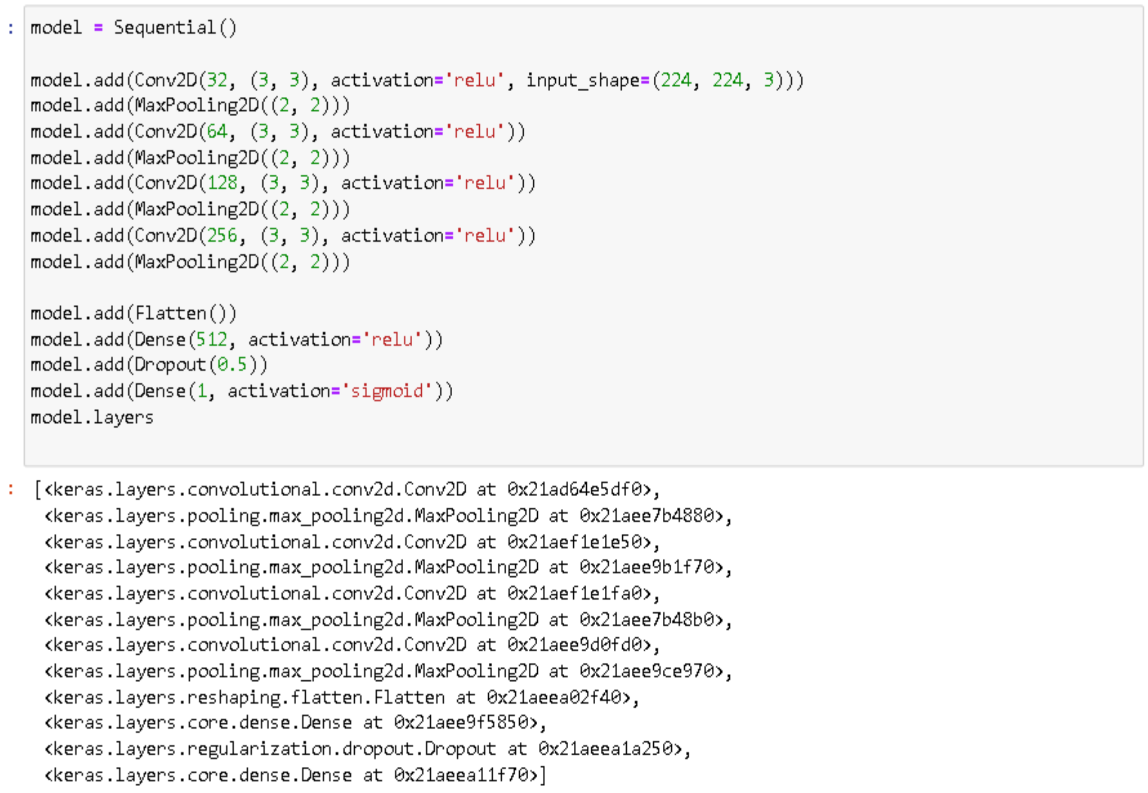


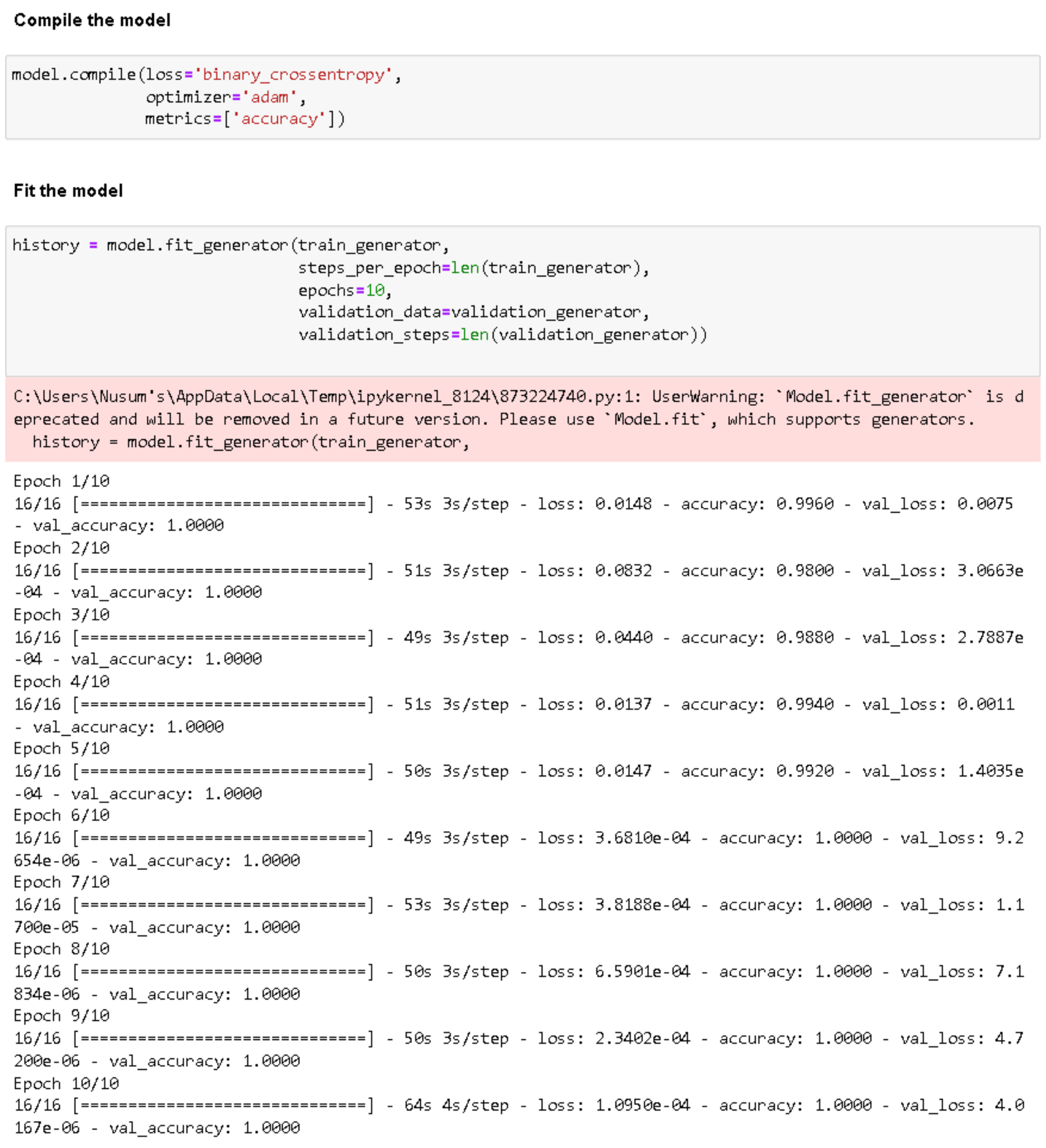
Figure 5: CNN Architecture



```
: model = Sequential()

  model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)))
  model.add(MaxPooling2D((2, 2)))
  model.add(Conv2D(64, (3, 3), activation='relu'))
  model.add(MaxPooling2D((2, 2)))
  model.add(Conv2D(128, (3, 3), activation='relu'))
  model.add(MaxPooling2D((2, 2)))
  model.add(Conv2D(256, (3, 3), activation='relu'))
  model.add(MaxPooling2D((2, 2)))

  model.add(Flatten())
  model.add(Dense(512, activation='relu'))
  model.add(Dropout(0.5))
  model.add(Dense(1, activation='sigmoid'))
  model.layers
```

```
: [<keras.layers.convolutional.conv2d.Conv2D at 0x21ad64e5df0>,
   <keras.layers.pooling.max_pooling2d.MaxPooling2D at 0x21aee7b4880>,
   <keras.layers.convolutional.conv2d.Conv2D at 0x21aef1e1e50>,
   <keras.layers.pooling.max_pooling2d.MaxPooling2D at 0x21aee9b1f70>,
   <keras.layers.convolutional.conv2d.Conv2D at 0x21aef1e1fa0>,
   <keras.layers.pooling.max_pooling2d.MaxPooling2D at 0x21aee7b48b0>,
   <keras.layers.convolutional.conv2d.Conv2D at 0x21aee9d0fd0>,
   <keras.layers.pooling.max_pooling2d.MaxPooling2D at 0x21aee9ce970>,
   <keras.layers.reshaping.flatten.Flatten at 0x21aeea02f40>,
   <keras.layers.core.dense.Dense at 0x21aee9f5850>,
   <keras.layers.regularization.dropout.Dropout at 0x21aeea1a250>,
   <keras.layers.core.dense.Dense at 0x21aeea11f70>]
```

Figure 6: A sequential model with Convolutional and Pooling layers.



Compile the model

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

Fit the model

```
history = model.fit_generator(train_generator,
                              steps_per_epoch=len(train_generator),
                              epochs=10,
                              validation_data=validation_generator,
                              validation_steps=len(validation_generator))
```

```
C:\Users\Nusum's\AppData\Local\Temp\ipykernel_8124\873224740.py:1: UserWarning: `Model.fit_generator` is d
eprecated and will be removed in a future version. Please use `Model.fit`, which supports generators.
  history = model.fit_generator(train_generator,
```

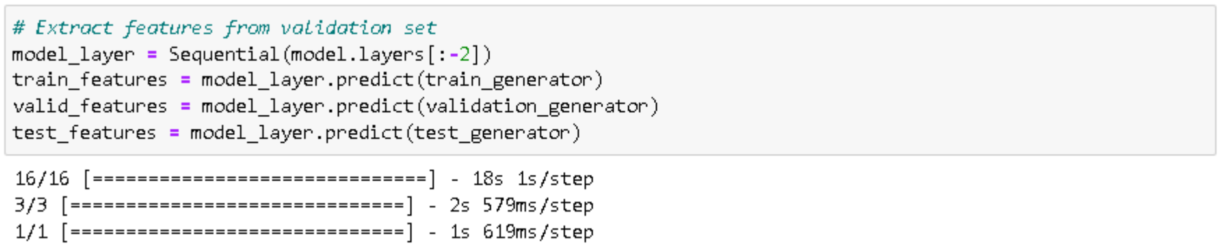
```
Epoch 1/10
16/16 [==============================] - 53s 3s/step - loss: 0.0148 - accuracy: 0.9960 - val_loss: 0.0075
- val_accuracy: 1.0000
Epoch 2/10
16/16 [==============================] - 51s 3s/step - loss: 0.0832 - accuracy: 0.9800 - val_loss: 3.0663e
-04 - val_accuracy: 1.0000
Epoch 3/10
16/16 [==============================] - 49s 3s/step - loss: 0.0440 - accuracy: 0.9880 - val_loss: 2.7887e
-04 - val_accuracy: 1.0000
Epoch 4/10
16/16 [==============================] - 51s 3s/step - loss: 0.0137 - accuracy: 0.9940 - val_loss: 0.0011
- val_accuracy: 1.0000
Epoch 5/10
16/16 [==============================] - 50s 3s/step - loss: 0.0147 - accuracy: 0.9920 - val_loss: 1.4035e
-04 - val_accuracy: 1.0000
Epoch 6/10
16/16 [==============================] - 49s 3s/step - loss: 3.6810e-04 - accuracy: 1.0000 - val_loss: 9.2
654e-06 - val_accuracy: 1.0000
Epoch 7/10
16/16 [==============================] - 53s 3s/step - loss: 3.8188e-04 - accuracy: 1.0000 - val_loss: 1.1
700e-05 - val_accuracy: 1.0000
Epoch 8/10
16/16 [==============================] - 50s 3s/step - loss: 6.5901e-04 - accuracy: 1.0000 - val_loss: 7.1
834e-06 - val_accuracy: 1.0000
Epoch 9/10
16/16 [==============================] - 50s 3s/step - loss: 2.3402e-04 - accuracy: 1.0000 - val_loss: 4.7
200e-06 - val_accuracy: 1.0000
Epoch 10/10
16/16 [==============================] - 64s 4s/step - loss: 1.0950e-04 - accuracy: 1.0000 - val_loss: 4.0
167e-06 - val_accuracy: 1.0000
```

Figure 7: Train the model using the training and validation generators

For training and validating the model, ten epochs, or sixteen steps, with around 50 ms for each time step, were utilized. For both training and validation, the accuracy was 1. The model's competence in picture categorization was demonstrated by the validation loss, which was approximately 4% while the training loss stayed around 3%. After the model fitting process, the predict technique was used to feed the data into the convolutional neural network's output layer in order to obtain features for the validation and test datasets.

2. **Feature Extraction and SVM Classification:**

- After training CNN model, features are extracted from the validation and test datasets.
- These features are then given as input to a Support Vector Machine (SVM) classifier.
- To distinguish between pandas and bears based on the learned features, the SVM model is configured as a linear classifier.



```
# Extract features from validation set
model_layer = Sequential(model.layers[:-2])
train_features = model_layer.predict(train_generator)
valid_features = model_layer.predict(validation_generator)
test_features = model_layer.predict(test_generator)
```

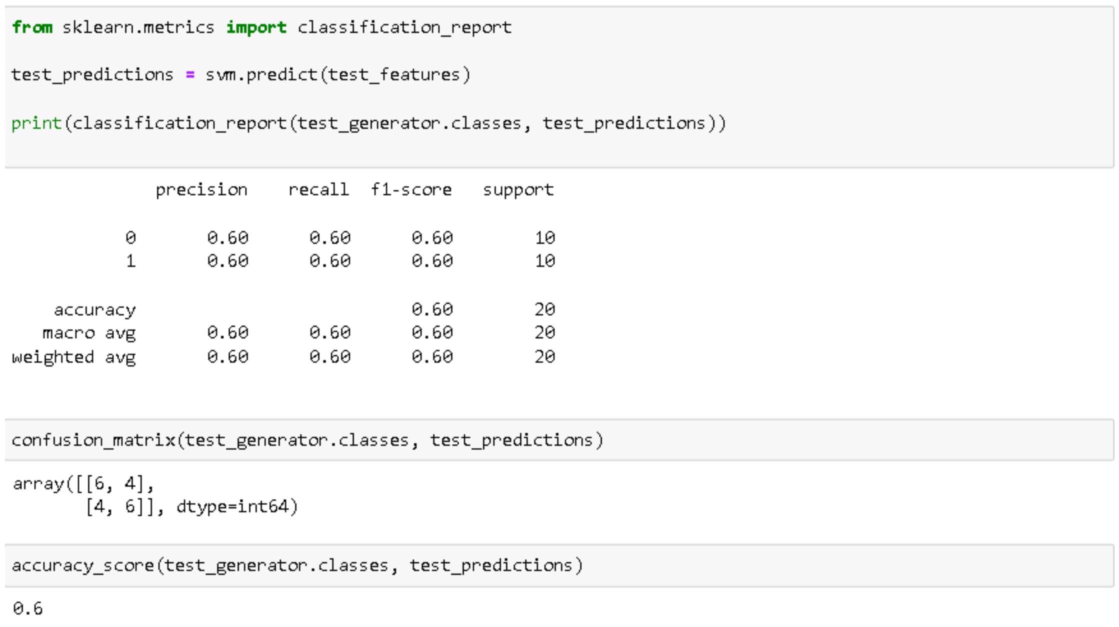
```
16/16 [==============================] - 18s 1s/step
3/3 [==============================] - 2s 579ms/step
1/1 [==============================] - 1s 619ms/step
```

Figure 8: Extract features from the second last dense layer for the training, validation, and test sets.

Evaluation:

Once the model is trained and the features are extracted, the test data is used by the SVM classifier to generate a comprehensive classification report. This report includes metrics such as accuracy, precision, recall, and F1-score, providing insights into the model's performance on distinguishing between pandas and bears.

The model is trained for multiple epochs using different dataset, and its performance is evaluated using accuracy, precision, recall, and F1-score. The goal is to achieve a robust and accurate classification of images depicting pandas and bears.



```
from sklearn.metrics import classification_report

test_predictions = svm.predict(test_features)

print(classification_report(test_generator.classes, test_predictions))
```

```
              precision    recall  f1-score   support

           0       0.60      0.60      0.60        10
           1       0.60      0.60      0.60        10

    accuracy                           0.60        20
   macro avg       0.60      0.60      0.60        20
weighted avg       0.60      0.60      0.60        20
```

```
confusion_matrix(test_generator.classes, test_predictions)
```

```
array([[6, 4],
       [4, 6]], dtype=int64)
```

```
accuracy_score(test_generator.classes, test_predictions)
```

```
0.6
```

Figure 9: Model evaluation

Results and Discussion:

The results show good accuracy during training and validation phases. The model is then used to classify new images, demonstrating its potential use in real-world scenarios.

The model's classification report indicates that the accuracy is 60% and the corresponding F1-scores and precision are 60% for both classes. Given that the model can only classify photographs with a low degree of accuracy, this implies unsatisfactory performance. Recall rates of roughly 60% for each class indicate a passable level of proficiency in identifying photos belonging to that class, or true positive cases. In conclusion, there are several reasons why the binary image classification model might not work effectively, including a small dataset, inadequate training time, or inappropriate selection of learning rates and epochs. Expanding the dataset and carrying out additional training epochs for the model could enhance its performance and produce superior outcomes.

References:

- (Sharma, 2019), Vaibhav Sharma, 2019: A Deep Learning Model to Perform Binary Classification. https://www.pluralsight.com/guides/deep-learning-model-perform-binary-classification
- (Evol, 2022), Ecol Evol, 2022: Giant panda age recognition based on a facial image deep learning system. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9719823/
- (Brownlee, 2022), Jason Brownlee, 2022: Binary Classification Tutorial with the Keras Deep Learning Library. https://machinelearningmastery.com/binary-classification-tutorial-with-the-keras-deep-learning-library/
- (Hongnian Wang), Hongnian Wang, Han Su, Peng Chen, Rong Hou, Zhihe Zhang, Weiyi Xie: Learning Deep Features for Giant Panda Gender Classification using Face. https://openaccess.thecvf.com/content_ICCVW_2019/papers/CVWC/Wang_Learning_Deep_Features_for_Giant_Panda_Gender_Classification_using_Face_ICCVW_2019_paper.pdf
- (Pranjal Swarup, 2021), Pranjal Swarup, Peng Chen, Rong Hou, Pinjia Que, Peng Liu, Adams Wai Kin Kong: Giant panda behaviour recognition using images. https://www.sciencedirect.com/science/article/pii/S2351989421000603