Computer science and artificial intelligence, Helwan university

Selected cs2

Number of member:7

202000351	**زياد أحمد محمد رفعت**
202000925	معاذ ناصر عبدالعزيز العزب
202000975	نادر سعيد جلال السيد
202001000	نور الدين ايمن عيسي سليمان
202000634	فاطمه سعيد فؤاد محمد
202000408	سهيله سيد البس موافي
202000229	تقي اسامه حسن

Animal Image Classifier Based on Convolutional Neural Network

Data set link:

https://www.kaggle.com/datasets/alessiocorrado99/animals10

paper link:

Animal Image Classifier Based on Convolutional Neural Network (shs-conferences.org)

Description

This project aims to build a deep learning model using TensorFlow and Keras for image classification into different categories. The data is loaded using Keras' ImageDataGenerator class and is split into training, validation, and testing sets. The model is built using Convolutional Neural Networks (CNNs) with Batch Normalization and Dropout layers to prevent overfitting. The model is compiled with categorical cross-entropy loss and RMSprop optimizer and trained using the fit() method. The model performance is evaluated on the test set using the evaluate() method. A new image is loaded, preprocessed, and the model is used to make a prediction on the image's class label.

CNN

is one of the main categories to do images recognition, images classifications, objects detections, recognition faces

library used in code

- Os: is a built-in Python library that provides an interface for interacting with the operating system, and is used in many tasks related to managing files and directories. In the mentioned code, `os` was used to determine the file paths and directories used in loading the image data and splitting it into training, validation, and testing sets. For example, `os.listdir()` was used to get a list of file and directory names in a particular path, and `os.path.join()` was used to concatenate file paths and directories. `os.makedirs()` was also used to create a new directory.

- Numpy: is a built-in Python library used for numerical computation, mathematical operations, and reshaping data. In the mentioned code, `numpy` was used to reshape the image data to fit the CNN model.`numpy.load()` was used to load the image data from file paths, and then `numpy.array()` was used to convert the image data into a 2D numpy array of images. When converting the image data to the CNN model, `numpy.expand_dims()` was used to add a new dimension to the array to convert it into a 3D array of images.

- Matplotlib.pyplot : is a Python library used for creating plots and visualizations. In the mentioned code, `matplotlib.pyplot` was used to create a plot showing the accuracy of the model during training.`matplotlib.pyplot.plot()` was used to create a plot of the model's accuracy between the training and validation sets, where the x-axis was set to the number of training epochs and the y-axis was set to accuracy. `matplotlib.pyplot.xlabel()`, `matplotlib.pyplot.ylabel()`, and `matplotlib.pyplot.title()` were also used to add labels to the x-axis, y-axis, and title of the plot.

- Seaborn : was used to create a bar plot showing the distribution of the number of images in each class.`sns.barplot()` was used to create the bar plot, where the x-axis was set to the names of the classes and the y-axis was set to the number of images in each class. `plt.figure(figsize=(10,5))` was used to set the size of the figure, `plt.grid()` was used to add a grid to the plot, `plt.axhline(np.mean(class_sizes), color='black', linestyle=':', label="Average number of images per class")` was used to add a horizontal line representing the average number of images in each class, and `plt.legend()` was used to add a legend to the plot.

- ImageDataGenerator: was used to load and transform images for training and evaluation of the model. Several options were specified in `ImageDataGenerator` to determine how the images are transformed. Here are the options used: `rescale=1./255`: This option is used to convert pixel values to a range from 0 to 1 by reducing pixel values by a factor of 1/255. `horizontal_flip=True`: This option is used to randomly flip images horizontally.`vertical_flip=True`: This option is used to randomly flip images vertically. `rotation_range=20`: This option is used to randomly rotate images by an angle up to 20 degrees.`zoom_range=0.2`: This option is used to randomly zoom in or out of images by up to 20%. `width_shift_range=0.2`: This option is used to randomly shift images horizontally by 20% of the image width. `height_shift_range=0.2`: This option is used to randomly shift images vertically by 20% of the image height.The batch size (`batch_size`) was also specified, and the paths (`directory`) to the training, testing, and validation data directories were set.

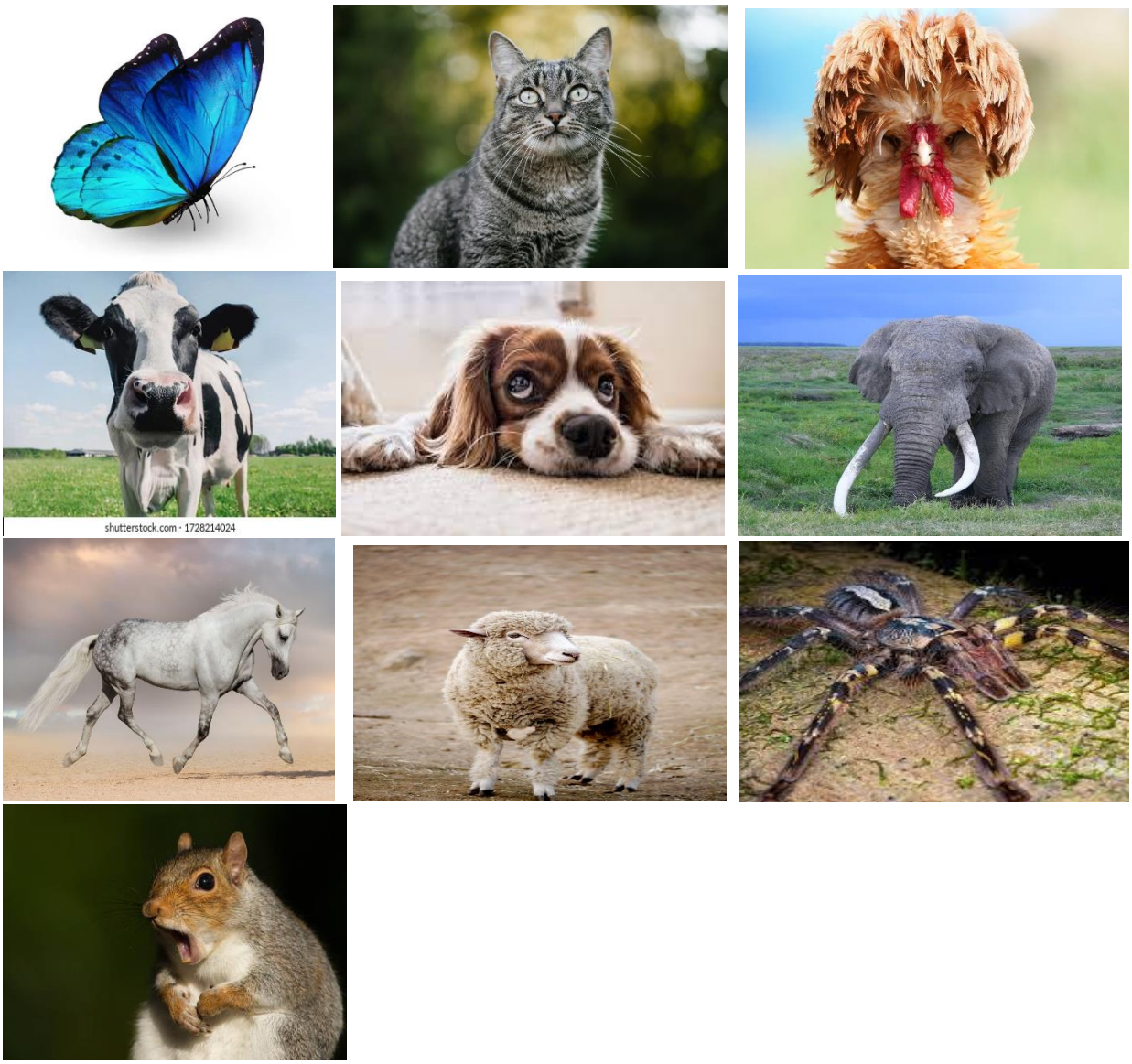
- Flow_from_directory: was used to load images from directories and transform them to a format compatible with the model. Several options were specified in `flow_from_directory` to determine how the images are transformed. Here are the options used: `directory`: The path to the directory containing the images. `target_size`: The size of the images after transformation. `batch_size`: The batch size for loading the images. `class_mode`: The type of data being targeted (classification or regression). `shuffle`: The shuffling status for the images.`color_mode`: The color mode for the images (grayscale or RGB).`flow_from_directory` was used to load the images and transform them to a format compatible with the model, where the images are automatically loaded from the directories and transformed to images of the shape and size specified in `target_size`. `class_mode` was specified as `"categorical"` to handle images with multiple classes.

- Tensorflow : was used to train a machine learning model for image classification and evaluate its performance on the test dataset.`tf.keras.preprocessing.image.ImageDataGenerator()` was used to load and transform the images into a format compatible with the model. `rescale=1./255` was specified to convert the pixel values to a range from 0 to 1. `tf.keras.models.Sequential()` was used to define the structure of the model, which includes multiple layers of a neural network. Various layers such as `tf.keras.layers.Conv2D()`, `tf.keras.layers.MaxPooling2D()`, `tf.keras.layers.Dense()`, `tf.keras.layers.Dropout()`, and `tf.keras.layers.BatchNormalization()` were used to build the neural network.`model.compile()` was used to specify the loss function, optimizer, and metrics used in training the model.`model.fit()` was used to train the model using the training data and evaluate its performance using the validation data.`model.evaluate()` was used to evaluate the performance of the model using the test data.

- Train_test_split: was used to split the data into training and testing sets. The data (`features` and `labels`), the size of the test set (`test_size`), and the random seed (`random_state`) were specified. The data was split into training and testing sets based on a specified ratio.

Class and image

- Number of Classes: 10
- ['Butterfly', 'Cat', 'Chicken', 'Cow', 'Dog', 'Elephant', 'Horse', 'Sheep', 'Spider', 'Squirrel']
- Class Distribution: [599, 710, 679, 648, 619, 516, 695, 319, 422, 550]
- Target size =80*80
- Image of class respectively:



RMSprop

- RMSprop is a commonly used optimization algorithm in deep learning. This algorithm aims to improve the process of updating the weights of the model based on the root mean square of the squared gradient values, in order to update the weights and limit the speed of their update.
- The RMSprop algorithm relies on the following control parameters:
- - Learning rate: which is the value assigned to update the weights of the model.
- - Decay factor: which is a factor that helps control the rate of decay, and can be used to reduce the impact of old root mean square values on the new weight values.
- The RMSprop algorithm is applied to update the weights of the model as follows:
- - The squared gradient values for each weight are calculated using the previous root mean square value and the current gradient.
- - The weight values are updated using a new equation that depends on the root mean square of the gradient, the learning rate, and the decay factor.
- Using the RMSprop algorithm in deep learning is considered a good and effective option, as it helps improve the prediction accuracy and update the weights of the model efficiently.
- To use the RMSprop algorithm in TensorFlow, a new instance of the optimizer should be created and passed as a parameter to the compile() function, which is used to define and prepare the model for training. The learning rate and decay factor can be set as parameters for the optimizer, as shown in the following example:
- model.compile(optimizer=tf.keras.optimizers.RMSprop(learning_rate=0.001, decay=1e-6),
  loss='categorical_crossentropy',
  metrics=['accuracy'])
- The optimizer algorithm, loss function, and evaluation metrics are defined in the compile() function. This optimizer instance can be used in the following steps to update the model's weights, as shown in the following example:
- grads = tape.gradient(loss, model.trainable_variables)
- optimizer.apply_gradients(zip(grads, model.trainable_variables))
- The GradientTape is used to calculate the gradient through the partials of the loss function, and then the optimizer updates the model's weights based on this gradient.
- Using the RMSprop algorithm in TensorFlow is an effective option for improving prediction accuracy and updating the model's weights efficiently.

Relu

In this code, different activation functions are used in the layers of the model. `relu` activation function is used in the first layers of the model (Conv2D layers) to improve performance and speed up the training process. `softmax` activation function is used in the last layer of the model (Dense layer) to convert the results into a probability distribution representing the probability of each class.`relu` is a common activation function used in artificial neural network layers, as it provides acceleration in the training process and improves performance. `softmax` is typically used in the output layer to convert the results into a probability distribution representing the probability of each class.

Parameter

The "parameter" used in this code is `target_size` in `ImageDataGenerator`. This parameter is used to specify the target size of the image during training and testing. When creating a data generator using `ImageDataGenerator`, `target_size` is passed as a parameter, which means that all images will be resized to this size before being passed to the model.

In this code, `target_size` is set to (65, 65), which means that all images will be resized to a size of 65x65 pixels. This parameter is passed to `train_data`, `val_data`, and `test_data` when creating the data generators.

Training	Validation	Testing
60%	20%	20%

Cycle:

No.of.cycle	ms/step	loss	accuracy	value loss	value accuracy	Epoch 108/108
1	91s 812	8.7394	0.1283	2.5657	0.1657	2/60
2	51s 468	2.3853	0.1810	2.3558	0.1622	3/60
3	51s 472	2.3110	0.1988	2.6015	0.1367	4/60
4	53s 487	2.2828	0.2110	2.2860	0.2097	5/60
5	53s 487	2.2715	0.2133	2.2588	0.2063	6/60
6	51s 469	2.2569	0.2177	2.2643	0.2236	7/60
7	51s 469	2.2506	0.2261	2.2915	01970	8/60
8	52s 477	2.2420	0.2372	2.2691	0.2121	9/60
9	52s 482	2.2326	0.2401	2.2430	0.2109	10/60
10	51s 470	2.2211	0.2410	2.2908	0.2039	11/60

Accuracy	Loss function
50.8%	1.69167160987854

Data distribution with plot