GitHub page: https://github.com/ARAlifio/use_case_test

1. Dataset Loading and Preprocessing

The first step in the image classification task involved loading the dataset and preparing it for the model. The dataset comprised images divided into predefined categories, each representing a distinct class for classification.

- **Dataset Directory Structure**: Images were organised in separate directories according to their class labels. This structure allows for easy loading of images with corresponding labels during training.
- **Image Resizing and Normalisation**: Each image was resized to a fixed resolution to ensure consistency in input dimensions across the dataset. Normalisation was performed by scaling pixel values to the range [0, 1], which helps accelerate training and improves model performance by avoiding large weight updates.
- **Data Augmentation**: To reduce overfitting and improve model generalisation, data augmentation techniques such as random rotations, zooms, and horizontal flips were applied to the training images. This process artificially increased the diversity of the training dataset, ensuring the model learns more robust features.

2. Model Architecture Design

A Convolutional Neural Network (CNN) architecture was chosen due to its efficacy in image classification tasks. CNNs are particularly well-suited for detecting hierarchical spatial patterns in images, such as edges, textures, and objects. Between redefining layers and retraining, these parts took the longest.

- **Convolutional Layers**: The initial layers of the network were designed to extract low-level features like edges and textures. Successive convolutional layers allowed the model to learn more complex patterns, such as shapes and objects.
- **Pooling Layers**: Max-pooling was applied after each convolutional block to reduce the spatial dimensions of the feature maps. This down-sampling step helped decrease the computational complexity while retaining the most important information.
- **Fully Connected Layers**: After the feature extraction layers, fully connected layers were added to map the learned feature representations to the classification labels. The final layer applied a sigmoid or softmax activation, depending on whether it was a binary or multi-class classification task.

3. Model Compilation

Once the architecture was defined, the model was compiled to prepare it for training. This step involved specifying the following key components:

- **Loss Function**: Categorical cross-entropy for multi-class classification was selected to quantify the error between the predicted labels and the true labels.

- **Optimizer**: The Adam optimizer was chosen for its efficiency and adaptive learning rate, allowing the model to converge faster and more reliably during training.
- **Evaluation Metrics**: Accuracy was used as the primary metric to monitor the model's performance during training and validation.

4. Model Training

The model was trained using the preprocessed dataset over a set number of epochs. During this stage:

- **Training Generator**: The training data was fed into the model in batches using a generator that yielded augmented images in real-time. This approach avoided loading the entire dataset into memory, making the process more efficient for large datasets.
- **Validation Process**: A separate validation set, which was not used in training, was employed to evaluate the model's performance after each epoch. This allowed for real-time monitoring of the model's generalisation to unseen data.

5. Evaluation and Testing

After training, the model was evaluated on a test set consisting of images that had not been used during training or validation. This final evaluation step provided an unbiased estimate of the model's performance on unseen data. The prediction class is shown in the notebook.