Journal Collabits
Vol. 3 No. 2 May 2026 : 96-104
E-ISSN : 3046-6709, P-ISSN : 3062-8601
https://publikasi.mercubuana.ac.id/index.php/collabits

Using Tensorflow for Clean and Messy Room Image Classification with Python

Gilas Adi Saputra[1*], Damar Pratama Ristadias Hariyanto[2], Roy Mubarak[3]
[1,2,3] Informatics Engineering Study Program, Universitas Mercu Buana, Indonesia

*Corresponding Author: 241522010009@student.mercubuana.ac.id

Abstract - Image classification is a fundamental computer vision task that can support automated visual monitoring in domestic, educational, and workplace environments. This study develops a transparent baseline pipeline using TensorFlow 2.x, Keras, and Python to distinguish clean and messy room images. The dataset contains 192 training images, with 96 images in each class, and 20 validation images, with 10 images in each class. All images are resized to 150 x 150 pixels and normalized to a 0-1 range. Rotation, horizontal flipping, and shearing are applied only to the training data, while validation images are normalized without random transformation. The sequential convolutional neural network contains four convolution-pooling blocks, a fully connected layer, and a sigmoid output for binary classification. Qualitative testing with two external images produced labels that were consistent with visual observation: the cluttered room was classified as messy and the organized room as clean. These demonstrations confirm that the pipeline operates from image input to class prediction, but they do not establish broad generalization or perfect accuracy. The main contribution is a reproducible small-data workflow that documents dataset distribution, preprocessing, augmentation, model parameters, validation procedures, and prediction thresholds. The study is limited by the small validation set, the absence of a large independent test set, and the lack of direct comparison with pretrained models. Future studies should evaluate transfer learning, larger datasets, repeated trials, and metrics such as precision, recall, F1-score, and confusion matrices.

Keywords:
TensorFlow;
Convolutional Neural Network;
Image Classification;
Python;
Data Augmentation;

Article History:
Received: 03-04-2025
Revised: 27-04-2026
Accepted: 10-05-2026

Article DOI: 10.22441/collabits.v3i2.27274

INTRODUCTION

Artificial intelligence has expanded the use of computer vision in activities that require visual pattern recognition. Image classification assigns an image to a predefined class by learning patterns of color, texture, shape, and object arrangement from labeled data. The task is applied in product inspection, object recognition, environmental monitoring, medical imaging, and camera-based space management.

TensorFlow provides an integrated ecosystem for model construction, training, evaluation, and deployment. In TensorFlow 2.x, Keras serves as a high-level interface for building convolutional neural networks (CNNs) through sequential or functional layers. TensorFlow documentation also identifies data augmentation, regularization, and transfer learning as important strategies for improving the generalization of image classification models [1]-[3].

Recent image classification research has followed two broad directions. The first improves accuracy and training efficiency through architectures such as EfficientNetV2 and ResNet-RS [4], [5]. The second emphasizes computational efficiency for mobile and edge devices, as demonstrated by MobileNetV4 [7]. ConvNeXt further shows that convolutional architectures remain competitive when their blocks and training procedures are modernized [6]. However, these models are typically evaluated on large datasets and computational resources that are not always available in small-scale studies.

A practical gap therefore remains in the documentation of a simple, transparent, and reproducible CNN workflow for binary classification with limited data. Small implementation studies often explain the programming steps but omit dataset distribution, validation procedures, prediction thresholds, and the distinction between a working demonstration and evidence of broad model generalization. These omissions make replication difficult and may encourage performance claims that are stronger than the available evidence.

This study addresses that gap by presenting an end-to-end baseline for classifying clean and messy room images in Google Colab. The novelty is practical and methodological rather than architectural: the study separates training and validation treatments, documents the dataset and training configuration, explains the CNN pipeline, and applies a probability threshold that can be replicated. The room domain is also relevant because class differences depend not on one isolated object, but on the density, order, and spatial arrangement of several objects.

The study answers two questions: how can TensorFlow and Python be implemented to build a binary classifier for clean and messy room images, and what can be concluded from the resulting tests when the dataset is relatively small? Accordingly, the objective is to describe data preparation, model construction, training, validation, and prediction while positioning the findings proportionally against current developments in image classification.

LITERATURE REVIEW

TensorFlow 2.x and Keras

TensorFlow is a machine learning platform that supports tensor computation, automatic differentiation, model optimization, and execution on CPUs, GPUs, and specialized accelerators. The integration of Keras into TensorFlow 2.x simplifies model construction through the Sequential and Functional APIs. This study uses the Sequential API because data move linearly from the image input through feature extraction, feature summarization, and binary classification. The approach is appropriate for a baseline model that prioritizes readability and replication [1].

Convolutional Neural Network

A CNN learns image representations through convolution operations that slide kernels across the spatial dimensions of an image. Early layers commonly capture simple patterns such as edges and color transitions, whereas deeper layers combine these patterns into more complex shapes and object arrangements. Rectified linear unit activation introduces nonlinearity, and max pooling reduces spatial dimensions and computational demand. After the features are flattened, a dense layer with sigmoid activation produces a probability for binary classification. Although modern architectures are more efficient, a sequential CNN remains useful as a baseline for examining the relationship between preprocessing, feature extraction, and classification decisions [5], [6].

Data Augmentation and Small Datasets

Small datasets increase the risk that a model will memorize training examples and fail to recognize new variation. Data augmentation reduces this risk by creating realistic variations through rotation, flipping, shifting, zooming, or shearing. Random transformations should be applied only to the training data because the validation set must represent unseen examples without stochastic modification other than normalization. TensorFlow further emphasizes that augmentation must preserve class semantics; therefore, transformations that are too extreme should be avoided [2]. Liu et al. similarly identify limited data as a central challenge in modern visual models and show the need for carefully designed training strategies [8].

ResNet, EfficientNet, MobileNet, and Transfer Learning

ResNet uses residual connections to facilitate the optimization of deep networks. ResNet-RS demonstrates that training procedures and scaling strategies can be as important as architectural changes [5]. EfficientNetV2 combines architecture search with progressive learning to improve parameter efficiency and training speed [4]. MobileNetV4 develops efficient building blocks for mobile and edge devices [7]. These model families are commonly available with pretrained weights and can therefore be used through transfer learning. For a small dataset, transfer learning may provide better generalization because general visual features have already been learned from a substantially larger dataset [3].

Research Position and Novelty

Unlike studies that pursue state-of-the-art performance, this research treats the model as an implementation baseline. The network is trained from scratch with a sequential CNN so that the complete feature-learning process remains visible and easy to modify. The clean-versus-messy room domain is contextually useful because classification depends on clutter density, object placement, and spatial order rather than on the presence of a single object. The study therefore illustrates both the value of a transparent pipeline and the challenges created by subjective labels and varied room backgrounds.

Conceptual Framework

The conceptual framework connects the input data, preprocessing stages, model, and study output. Images are divided into training and validation sets. Augmentation is applied only to training images, while both groups are normalized. The CNN learns features from the training data, and the validation data are used to monitor generalization during training. The trained model then produces a probability that is converted into a clean or messy class label, as illustrated in Figure 1.

Figure 1. Conceptual framework and image classification pipeline

METHODOLOGY

Research Design and Experimental Environment

The study uses a computational experimental design to build and test a binary image classification model. The implementation is conducted in Python on Google Colab using TensorFlow 2.x and Keras. To improve reproducibility, the workflow sets a random seed of 42 for Python, NumPy, and TensorFlow. All images are processed in RGB mode with an input size of 150 x 150 pixels.

Dataset and Data Distribution

The Messy vs Clean Room dataset is obtained from the Dicoding asset repository and is organized into training and validation directories. Table 1 presents the data distribution. The training set contains 192 balanced images, while the validation set contains 20 images. Two external images, one messy and one clean, are used only for a qualitative prediction demonstration and are not included in model training.

Table 1. Dataset distribution

Data Group	Clean	Messy	Total	Purpose
Training	96	96	192	Model weight learning
Validation	10	10	20	Generalization monitoring
External test	1	1	2	Prediction demonstration

Preprocessing and Data Augmentation

Pixel values are normalized from the 0-255 range to 0-1 using a rescaling factor of 1/255. Training images receive random rotation up to 20 degrees, horizontal flipping, and shearing of 0.2 with nearest-neighbor filling. Validation images receive normalization only. This separation prevents random alteration of validation examples and supports a more consistent evaluation procedure. The implemented configuration is shown in Figure 2.

Figure 2. Training augmentation and validation normalization configuration

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(
                    rescale=1./255,
                    rotation_range=20,
                    horizontal_flip=True,
                    shear_range = 0.2,
                    fill_mode = 'nearest')

validation_datagen = ImageDataGenerator(
                    rescale=1./255)
```

Model Architecture

The model is constructed with the Sequential API and four Conv2D layers. The number of filters increases from 32 to 64, 128, and 512 to capture progressively more complex features. Each convolutional layer is followed by MaxPooling2D to reduce spatial dimensions. The final feature map is flattened and passed to a dense layer with 512 units and a one-neuron sigmoid output. The architecture contains 13,529,665 trainable parameters. This parameter count is large relative to the dataset and therefore represents a potential source of overfitting.

Figure 3. Sequential CNN architecture used in the study

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(150, 150, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(512, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

Table 2. Model structure and parameter count

Layer	Output Size	Parameters
Conv2D, 32 filters, 3 x 3	148 x 148 x 32	896
MaxPooling2D	74 x 74 x 32	0
Conv2D, 64 filters, 3 x 3	72 x 72 x 64	18,496
MaxPooling2D	36 x 36 x 64	0
Conv2D, 128 filters, 3 x 3	34 x 34 x 128	73,856
MaxPooling2D	17 x 17 x 128	0
Conv2D, 512 filters, 3 x 3	15 x 15 x 512	590,336
MaxPooling2D	7 x 7 x 512	0
Flatten	25,088	0
Dense, 512 units	512	12,845,568
Dense, 1 sigmoid unit	1	513
Total	-	13,529,665

Training Parameters and Validation Procedure

The model is compiled with binary cross-entropy because the task contains two classes, the Adam optimizer, and accuracy as the monitoring metric. The batch size is 4. The original experiment specifies 25 steps per epoch, a maximum of 250 epochs, and 5 validation steps. Training batches are shuffled, whereas validation order is retained. Validation is performed at every epoch to observe the difference between training and validation behavior. Table 3 summarizes the experimental configuration.

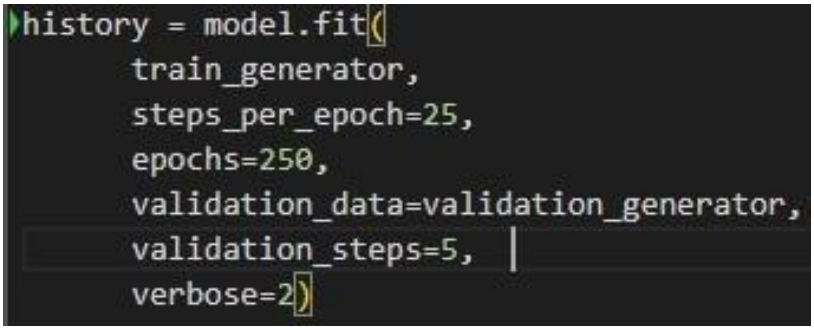
Table 3. Training and validation configuration

Component	Configuration
Language and platform	Python on Google Colab
Framework	TensorFlow 2.x and Keras
Input size	150 x 150 x 3
Batch size	4
Loss function	Binary cross-entropy
Optimizer	Adam
Primary metric	Accuracy
Steps per epoch	25
Maximum epochs	250
Validation steps	5
Random seed	42
Prediction threshold	0.5

Figure 4. Model compilation configuration

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

Figure 5. Model training configuration



Prediction Procedure

A test image is loaded, resized to 150 x 150 pixels, converted into an array, and normalized to the same scale used during training. The model produces a continuous probability between 0 and 1. In the revised prediction procedure, the class label is determined with a threshold of 0.5 rather than by comparing the probability with an exact value of 1. This correction is necessary because model.predict returns a probability, not a discrete class.

```
probability = float(model.predict(images, verbose=0)[0][0])
if probability >= 0.5:
    label = 'messy'
else:
    label = 'clean'
print(f'probability={probability:.4f}, class={label}')
```

Evaluation Criteria

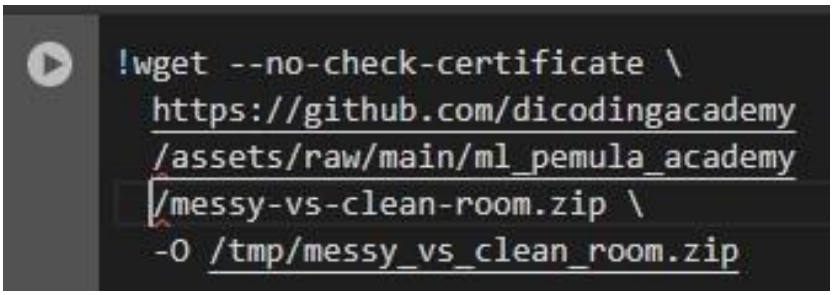
Evaluation is conducted at two levels. First, validation accuracy is monitored during training to examine whether learning remains consistent outside the training batches. Second, two external images are assessed qualitatively by comparing the model label with the visible condition of each room. Because the validation and external test sets are very small, the results are not used to claim broad generalization. A stronger evaluation should include an independent test set, a confusion matrix, precision, recall, F1-score, repeated trials, and confidence intervals.

RESULTS AND DISCUSSION

Implementation of the Classification Pipeline

The complete workflow, including dataset download, directory extraction, generator construction, CNN definition, training, and prediction, can be executed in a single Google Colab notebook. Separate clean and messy directories allow flow_from_directory to assign binary labels automatically. Normalization stabilizes the input scale, while augmentation adds training variation without changing the intended class. Figure 6 shows the dataset retrieval step used in the notebook.

Figure 6. Dataset retrieval in Google Colab



External Image Test Results

The first external image shows a room with numerous unarranged objects, crowded surfaces, and high visual density. The model assigns the messy label, as shown in Figure 7. The second image presents a room with a more organized bed and furniture layout, and the model assigns the clean label, as shown in Figure 8. Both labels are consistent with direct visual observation, which indicates that the pipeline can complete the intended process from image input to class output.

Figure 7. Prediction result for a messy room image

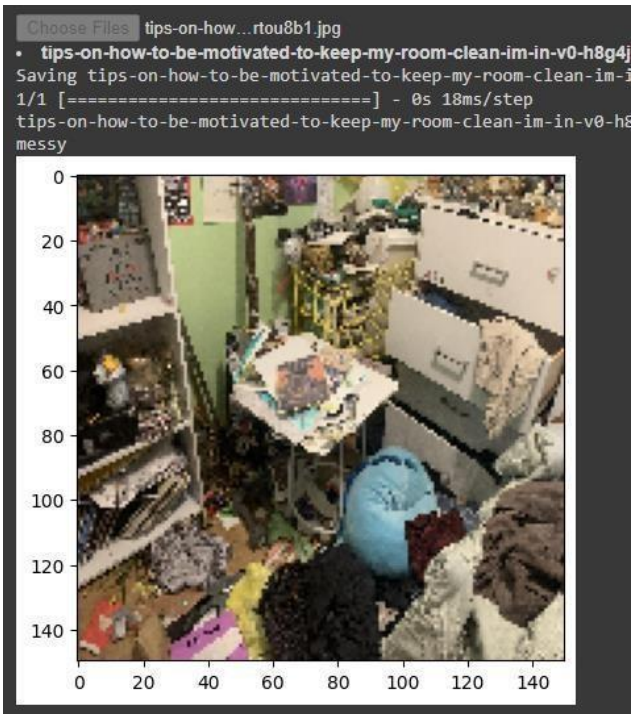
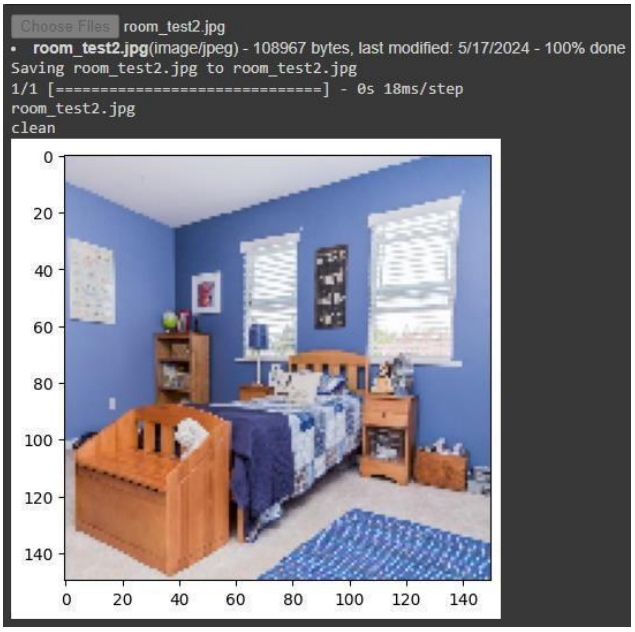


Figure 8. Prediction result for a clean room image



The two successful examples should not be interpreted as 100 percent accuracy. They demonstrate only that the implemented pipeline operates end to end. A reliable performance estimate requires a larger independent test set that represents bedroom, living room, workplace, and kitchen scenes, together with variation in lighting, camera angle, background, and clutter level.

Comparison with Previous Studies and Modern Architectures

The results show that a simple CNN can serve as a baseline for learning visual room order. Nevertheless, the model contains approximately 13.5 million parameters but is trained on only 192 images. This imbalance between model capacity and data volume creates a substantial overfitting risk, so agreement on two external images is insufficient evidence of generalization. This interpretation is consistent with research emphasizing that training procedures, regularization, and model scaling are as important as network depth [5].

EfficientNetV2 jointly targets accuracy, training speed, and parameter efficiency through training-aware architecture search and progressive learning [4], whereas the present model uses a conventional convolutional stack. The current contribution is therefore not performance superiority but an implementation baseline whose stages and parameters can be inspected and modified.

The model is also not optimized for mobile or edge inference as MobileNetV4 is [7]. For the small dataset used here, pretrained ResNet, EfficientNet, or MobileNet models may be more appropriate because the basic visual features do not need to be learned entirely from 192 images [3]. ConvNeXt and ResNet-RS further demonstrate that modernized convolutional blocks, regularization, and training strategies can produce strong baselines [5], [6]. A logical next experiment is therefore a controlled comparison between the CNN trained from scratch and pretrained models using the same data split, metrics, and computational environment.

Scientific and Practical Contributions

Scientifically, the study presents a proportionate interpretation of CNN use on a small dataset. It does not claim a new architecture or state-of-the-art performance; instead, it identifies methodological details that determine experimental reliability, including data distribution, separation of augmentation and validation, parameter documentation, and probability thresholding. Practically, the notebook can serve as an initial prototype for other visual classification tasks, such as packaging separation, workplace cleanliness checks, or room inspection, once the dataset and labeling rules are adjusted.

Research Limitations

1. The training set contains only 192 images and the validation set only 20 images, so environmental variation is not represented adequately.
2. Only two external images are used; the resulting predictions are demonstrative rather than population-level evidence.
3. Clean and messy labels involve subjective judgment, and differences among annotators may reduce class consistency.
4. The model contains approximately 13.5 million parameters, creating a high overfitting risk for the available dataset.
5. The study does not report a confusion matrix, precision, recall, F1-score, confidence intervals, or repeated trials with different seeds.
6. No direct benchmark is conducted against transfer learning, EfficientNetV2, ResNet, or MobileNet.
7. The minor TensorFlow version and detailed Colab runtime specifications are not recorded, so environmental differences may affect reproduction.

CONCLUSION

This study develops an end-to-end pipeline for classifying clean and messy room images using TensorFlow 2.x, Keras, and Python in Google Colab. The dataset is balanced with 192 training images and 20 validation images. Normalization and augmentation are applied to training data, while validation data are normalized without random transformation. The sequential CNN consists of four convolution-pooling blocks, a dense layer, and a sigmoid output. Qualitative testing on two external images produces labels that are consistent with their visible room conditions. The contribution lies in a transparent and reproducible baseline rather than a new architecture or a state-of-the-art performance claim. Because the validation and test samples remain limited, the results cannot establish broad model generalization. Future research should use larger and more diverse datasets, an independent test set, repeated experiments, and evaluation metrics including precision, recall, F1-score, and confusion matrices. Controlled comparison with pretrained EfficientNetV2, ResNet, and MobileNet models is also needed to evaluate the trade-off among accuracy, parameter efficiency, training cost, and inference time.

REFERENCES

[1] TensorFlow, "Image classification," TensorFlow Core Tutorials, updated Apr. 3, 2024.

[2] TensorFlow, "Data augmentation," TensorFlow Core Tutorials, updated Jul. 19, 2024.

[3] TensorFlow, "Transfer learning and fine-tuning," TensorFlow Core Tutorials, updated Aug. 16, 2024.

[4] M. Tan and Q. V. Le, "EfficientNetV2: Smaller models and faster training," in Proceedings of the 38th International Conference on Machine Learning, vol. 139, pp. 10096-10106, 2021.

[5] I. Bello, W. Fedus, X. Du, E. D. Cubuk, A. Srinivas, T.-Y. Lin, J. Shlens, and B. Zoph, "Revisiting ResNets: Improved training and scaling strategies," Advances in Neural Information Processing Systems, vol. 34, pp. 22614-22627, 2021.

[6] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, "A ConvNet for the 2020s," in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 11976-11986, 2022, doi: 10.1109/CVPR52688.2022.01167.

[7] D. Qin et al., "MobileNetV4: Universal models for the mobile ecosystem," in Computer Vision - ECCV 2024, Lecture Notes in Computer Science, vol. 15098, pp. 78-96, 2024, doi: 10.1007/978-3-031-73661-2_5.

[8] Y. Liu, E. Sangineto, W. Bi, N. Sebe, B. Lepri, and M. De Nadai, "Efficient training of visual transformers with small datasets," Advances in Neural Information Processing Systems, vol. 34, pp. 23818-23830, 2021.

[9] TensorFlow, "Image classification with Model Garden," TensorFlow Core, updated Oct. 17, 2023.

[10] G. Ding, "Messy vs Clean Room," Kaggle Dataset, 2019.