

Few-Shot Image Classification of Generation 3 Pokémons using MobileNetV2 and Transfer Learning

Richard Bryan Antonius
School of Computer Science
Bina Nusantara University Jakarta, Indonesia 11480
richard.antonius@binus.ac.id

Abstract— The classification of image categories, such as distinguishing between 135 unique Pokémons from Generation 3, presents a significant challenge in computer vision, particularly when training data is scarce. While deep learning models typically require thousands of images per class to achieve high accuracy, collecting such large datasets for specific domains is often unfeasible. To address this "few-shot" learning problem, this study explores the application of Transfer Learning using the MobileNetV2 architecture. We constructed a custom dataset of approximately 4,500 images (averaging ~35 images per class) using web scraping techniques. To mitigate the risk of overfitting inherent in small datasets, we implemented aggressive data augmentation strategies, including random rotation, horizontal flipping, and color jittering. The model was trained over 30 epochs using the Adam optimizer. Experimental results demonstrate that despite the limited data, the system achieved a Validation Accuracy of 72.9% and a Top-3 Accuracy of 83.5%, significantly outperforming random baseline probability. The final model was deployed as a real-time web application using Streamlit, demonstrating that lightweight convolutional neural networks can effectively solve complex classification tasks on consumer-grade hardware with limited data.

Keywords— Computer Vision, Transfer Learning, MobileNetV2, Few-Shot Learning, Image Classification, Pokémons.

I. Introduction

The Pokémon IP is one of the most beloved pieces of media curated in the last 30 years of the digital era with over 1,000 unique creatures designed. Its 3rd generation which was released way back in the 21st of November 2002 is one such of the release waves of new added Pokémon. This generation introduced 135 distinct Pokémon which for us human fans may be easily distinguishable from one another. It is a whole different ball park when a machine tries to distinguish them from one another. For a machine this represents a difficult large-scale classification problem due to the sheer amount of classes, the model must learn to map visual features to 135 different labels.

For machines thousands upon thousands images are needed to help the machine to recognize each distinct Pokémon with high accuracy. Large dataset for a specific subject have a clear weakness when the amount of variation of the subject is limited. The problem that arises is the “Few-Shot” problem which mainly is a problem of data scarcity. Commonly used models such as ResNet and VGG16 trains on the ImageNet dataset which is a large database designed for use in visual object recognition research. This database consists of millions of images whereas in our case the amount of data per class that we were able to collect is approximately 30-45 images. Training a deep neural network from scratch on such a small dataset typically leads to severe overfitting, where the model memorizes the available training data but fails when trying to classify new images.

To overcome the limitations of a small dataset and limited compute resources, this study utilizes Transfer Learning. Instead of training a model from scratch, we use a pre-existing model which is MobileNetV2, a lightweight Convolutional Neural Network(CNN) pre-trained on the massive ImageNet dataset. By freezing the feature extraction layers of MobileNetV2, we reused the learned ability to detect edges, shapes, and textures then retrain the final classification layer to recognize 135 different classes that we have. And to add we employ data augmentation techniques

such as random rotation, random horizontal flip, and color jitter to make it so the model sees a new image on each iteration.

The main objectives of this project are:

1. To curate and clean a dataset of Generation 3 Pokéémon images using automated web scraping.
2. To fine-tune a MobileNetV2 model to achieve a Top-3 Accuracy of greater than 80%.
3. To analyze the impact of data augmentation on reducing overfitting in few-shot scenarios.
4. To deploy the trained model as an interactive, user-friendly web application using Streamlit that allows real-time classification.

This project demonstrates the ease of Artificial Intelligence utilization. It proves that high accuracy image recognition tools can be built for niche hobbies without access to many high end resources that the Artificial Intelligence industry standard has. The methodology used here can be adapted for other real-world applications with limited data, such as organizing small retail inventories or categorizing personal collections.

II. Literature Review

A. Convolutional Neural Network

Convolutional Neural Network are one type of Artificial Neural Network which mimics the biological nervous system to be used for complex pattern recognition, prediction, and classification tasks. Convolutional Neural Network is typically used for pattern recognition within images which allows to encode specific features found inside images to reduce the parameters required to set up the model[1]. Standard Artificial Neural Networks are built of fully connected layers where every neuron connects to every neuron in the subsequent layer. While effective for simple tasks it struggles with images because of the sheer computational resources needed. For example a 255*255

pixel color image would require over 65000 weights for a single neuron in the first hidden layer which makes the model computationally expensive and prone to overfitting. While having data dimensionality issues the pixels in the images no matter how far are treated as being close together because standard Artificial Neural Network doesn't consider the input topology[2].

To solve these limitations, Convolutional Neural Network uses three main architectural ideas: local connections, shared weights, and pooling. These components enable the CNN to detect local motifs such as edges regardless of their position in the image, significantly reducing the number of parameters required[2]. A typical CNN is composed of a sequence of layers:

1. **The Convolutional Layer** is the heart of CNN. Instead of connecting to every input pixel, neurons in this layer connect to a local region known as the receptive field. The layer uses a set of learnable kernels that moves around the image computing a dot product every time it moves. All units in the same feature map share the same parameter, allowing the network to detect the same feature anywhere on the image such as edges. While the output is determined by the number of filters/depth and the stride.
2. **Batch Normalization** to address the issue of Internal Covariate Shift and accelerate training, Batch Normalization is applied to the network's activations. When applied to Convolutional Neural Networks, the standard batch normalization formulation is modified to respect the spatial structure of the data.

Placement and bias handling in a convolutional layer, the batch normalization transform is inserted immediately before the non-linearity (such as ReLU). The normalization is applied to the output of the convolution operation, Wu , effectively replacing the standard layer equation $z=g(Wu+b)$ with $z=g(BN(Wu))$. Because the batch normalization transform includes a learned mean subtraction, the

original bias parameter b of the layer becomes redundant and can be ignored[3].

3. **Activation Function (ReLU)** Following the convolutional operation, the output is passed through a non-linear activation function. Modern convolutional neural Networks typically use the Rectified Linear Unit (ReLU), defined as $f(x)=\max(0,x)$. ReLU is preferred over traditional non-linearities like tanh or sigmoid because it significantly accelerates convergence during training. Deep networks with ReLUs can train several times faster than equivalent networks using tanh units[2].
4. **Pooling Layers** are incorporated between convolutional layers to reduce spatial dimensionality. The most common used method is max pooling which outputs the maximum value within a local area typically a $2*2$ area. This reduction lightens the load on computation and introduces invariance or small shifts in the input.
5. **Fully-Connected Layers** after several convolutional and pooling stages, the high-level reasoning is performed by fully connected layers, identical to those in traditional ANNs. These layers take the flattened output of the previous layers and generate the final class scores for classification. To prevent overfitting in these dense layers, techniques such as "Dropout" (setting the output of random neurons to zero during training) are often employed[4].

B. Transfer Learning

In traditional machine learning, models are trained and tested under the assumption that the data distributions for both tasks are the same. However, in many real-world scenarios, collecting sufficient high-quality labeled data for a specific target domain is expensive or impossible. Transfer learning addresses this by extracting knowledge from one or more source tasks and applying it to a related target task, thereby reducing the need for extensive data labeling and re-calibration. This ability to recognize and apply previously learned skills to

novel situations parallels human learning, where skills like recognizing apples can facilitate recognizing pears[5].

Transfer learning techniques can be categorized based on the relationship between the source and target domains and tasks. A "domain" consists of a feature space and a marginal probability distribution, while a "task" consists of a label space and an objective predictive function. Based on these definitions, transfer learning is divided into three main settings:

1. **Inductive Transfer Learning:** The target task is different from the source task, regardless of whether the domains are the same. This setting requires some labeled data in the target domain to induce the predictive model. It is commonly used in classification and regression tasks.
2. **Transductive Transfer Learning:** The source and target tasks are the same, but the domains are different. In this setting, no labeled data is available in the target domain, but unlabeled data is available at training time. This relates closely to domain adaptation and sample selection bias.
3. **Unsupervised Transfer Learning:** The target task is different from but related to the source task, and no labeled data is available in either domain. This setting focuses on unsupervised tasks like clustering and dimensionality reduction[5].

In the context of deep neural networks, transfer learning relies on the observation that the initial layers of a network trained on natural images learn general, task-independent features such as Gabor filters and color blobs, which are applicable across various datasets. As information propagates through the network, these features transition from general to highly specific by the final layer, which is tailored to the distinct classes of the original task. Consequently, the transferability of features varies by layer, with performance degradation caused by two distinct issues: the over-specialization of higher-layer neurons to the source task and optimization difficulties caused by disrupting co-adapted neurons in neighboring layers. While transferability decreases as the semantic distance between the source and target tasks increases, transferring features

remains superior to using random weights. Notably, initializing a network with transferred features and fine-tuning them produces a boost in generalization performance that persists even after extensive training on the new target dataset[6].

C. MobileNetV2

MobileNetV2 is a neural network architecture specifically tailored for mobile and resource-constrained environments. It builds upon the efficiency of MobileNetV1 by introducing two novel architectural features: Linear Bottlenecks and Inverted Residuals. These innovations allow the model to significantly reduce memory footprint and operation count while retaining state-of-the-art accuracy[8].

MobileNetV1 & V2 is distinctly recognized by its Depthwise Separable Convolution originally introduced in MobileNetV1 to reduce the computational cost of standard convolutions. By splitting the convolution into 2 one being depthwise convolution which applies a single filter to each input channel for spatial filtering and the other pointwise convolution applying a 1*1 convolution to linearly combine the output of the depthwise layer. This factorization drastically reduces the computation load. For example, using 3*3 depthwise separable convolutions reduce computation needs by 8 to 9 times compared to standard convolutions with only a minimal reduction in accuracy[7].

MobileNetV2 addresses a critical inefficiency in deep neural networks concerning the information loss caused by non-linear activation functions like ReLU in low-dimensional spaces. While ReLU is effective in high-dimensional activation tensors, it inevitably destroys information when applied to low-dimensional tensors by collapsing all negative values to zero. Experimental evidence suggests that if the manifold of interest remains non-zero volume after a ReLU transformation, it corresponds merely to a linear transformation, limiting the network's expressive power. To prevent this, MobileNetV2 removes the non-linearity from the output of the bottleneck layer, replacing it with a linear transformation that projects features back to a

low-dimensional representation. This design choice is crucial because it prevents non-linearities from destroying too much information in the compressed bottleneck state, preserving the model's representational power[8].

The defining architectural feature of MobileNetV2 is the "Inverted Residual" block, which fundamentally flips the design of traditional residual blocks found in networks like ResNet. A traditional residual block connects layers with a high number of channels, following a "wide → narrow → wide" structure where the input is compressed, processed, and then expanded[9]. In contrast, MobileNetV2 uses an inverted "narrow → wide → narrow" pattern: the low-dimensional input is first expanded to a high dimension using a 1×1 convolution, filtered with a lightweight depthwise convolution, and then projected back down to a low dimension via a linear convolution. Crucially, the residual shortcut connections are placed directly between the thin bottleneck layers rather than the expanded layers. This design is significantly more memory efficient for mobile applications because it allows the model to never fully materialize large intermediate tensors during inference, thereby reducing the need for main memory access on embedded hardware[8].

D. Data Augmentation

Deep learning models, such as deep convolutional neural networks, need large scale datasets to avoid overfitting and achieve high performance. However, obtaining enough labeled data of real-world objects is often limited. Data augmentation helps solve the problem of limited data by artificially increasing the quantity and diversity of the training data. The most commonly used data augmentation techniques are direct manipulation on input data whether by rotation, cropping, flipping, translation, color adjustment, noise injection, and sharpening. While these techniques are easy to apply we need to keep in mind to avoid issues such as the padding effect, where the image is moved out of frame.

Other than standard manipulations other techniques have been developed such as image erasing to improve model robustness by randomly

deleting or masking regions of an image and image mixing which blends two or more images or their sub-regions to create new training samples to use. All these techniques help diversify the training data to make sure the model doesn't just memorize specific distinct features of the target[10].

III. Methodology

A. Data Collection

To train the model, a custom labelled dataset was needed with focus on the third generation of Pokémon consisting of 135 unique species of Pokémon. As there was no pre-existing public dataset that fits our needs for this research we decided to curate a new dataset from scratch.

We collected the data with the help of the library bing-image-downloader, an open source Python library designed to scrape image results from the Bing search engine. This automated approach allowed for the efficient collection of diverse images representing the target classes.

With the help of the Python library we were able to web scrape around 30-45 images of 135 different Pokémon in the 3rd generation. Although the data collection was automated, there was some manual verification involved in the process to make sure different species of Pokémon, people that are mingling with the Pokémon, and different forms such as unofficial fanmade forms are deleted from each class.

B. Data Pre-Processing

Having a small dataset can lead to severe problems such as overfitting, and the inability to recognize even the slightest change that's why we employ a series of pre-processing and data augmentation techniques. Using the library torchvision.transforms to the raw input images.

For the pre-processing resizing is first on the chopping block, all input images were resized to 256 pixels along the shorter dimension to ensure a consistent scale. Then we used center cropping of a 224 * 224 from the resized

image to focus on the central subject and last we converted the pixel values to tensor values 0-1 and normalized using the standard ImageNet mean ($\mu=[0.485, 0.456, 0.406]$) and standard deviation ($\sigma=[0.229, 0.224, 0.225]$).

Data augmentation techniques included for the training input are random rotation where the images were rotated within -30° - 30° to make the model able to recognize different object orientation. Random horizontal flip with a probability of 0.5 was also used as a way to address lateral symmetries in the data. And last but not least color jittering where random perturbations were applied to the brightness, contrast, saturation, and hue (each with a factor of 0.2) to simulate varying lighting conditions and camera characteristics.

C. Model Architecture

Using MobileNetV2 architecture as our base model that was designed to be lightweight and efficient and can run on datasets that were small compared to the ones they were trained on. Instead of re-training the model with random weights from scratch we utilised transfer learning. We loaded the model with pre-trained weights from the ImageNet dataset (ImageNet1K_V2). Through this step the model already has basic knowledge on detecting basic features such as edges, textures, and shapes before even seeing a single Pokémon image.

To use this pre-trained model for our needs the first step in order is to freeze the feature extractor layer. This ensures the model doesn't change the weights it has learned from the ImageNet dataset. The next modification we will do is for the final layer to classify our dataset. MobileNetV2 is trained with the intention to output 1,000 different classes while our dataset only has 135 different classes. To address that we replace the final layer with a fully connected linear layer that has exactly 135 output neurons. The output from this new layer is then passed through a Cross Entropy Loss function, which compares the model's prediction with the actual label, and the weights of this new layer are optimized using the Adam optimizer.

D. Experimental Setup

Experiments and model training was done using the Python programming language version 3.13.7. To build and train the neural network, PyTorch deep learning framework was used along with Torchvision for loading the pre-trained models and conducting data augmentations. Other libraries were also used such as Scikit-learn to calculate performance metrics such as precision, recall, f1 and the confusion matrix. Also used was PIL(Python Imaging Library) to handle image file loading.

The experiments were run on a computer equipped with NVIDIA GeForce RTX 3060 GPU with 12 GB of VRAM. We utilize the availability of the GPU by checking for CUDA. By using the GPU we can do computation which significantly accelerates the training process.

For the training configurations, we trained the model on 30 epochs. The batch size while training was 32 images at a time to manage memory usage. The learning rate was set to 0.001 which controls how much the model updates its weights after each batch. Adam optimizer was used to adjust weights during training because of speed and stability in comparison to standard gradient descent. And cross entropy loss was used to calculate error between the model's predictions and the actual labels.

E. Training and Testing Strategies

To train and evaluate our model, we split our dataset of over 4,500 images into 2 groups: 80% used for training the model to recognize each Pokémon and 20% was used as a validation set to test how well it performs when seeing new data. The training is set to run for 30 epochs with a batch size of 32 images, using a learning rate of 0.001 to update weights after each batch. For the optimization we used Adam as it handles sparse gradients well and converges faster, while cross entropy loss was used to determine the validity of the model's prediction based on the actual label. To get more insight we also calculated precision, recall, f1 score, and confusion matrix each time the epoch achieved a new record for accuracy.

F. Deployment

To make our 3rd generation Pokémon image classifier accessible to the general public we deployed the model using Streamlit, a Python library designed to build interactive web applications. The weights we got from the trained model are loaded into a MobileNetV2 model. Then we proceed to set up a simple interface where the user is able to input an image of a Pokémon(JPG, JPEG or PNG format). After the image is uploaded the system will resize the image to 224*224 pixels and normalizing to match the images we used for training. The model then analyzes the image and predicts the class of the input image, the app will then proceed to show the top 3 highest expected classes. To deploy we push all of our necessary code and model to a git repository then connect to the streamlit web service.

IV. Results

A. Performance Metrics

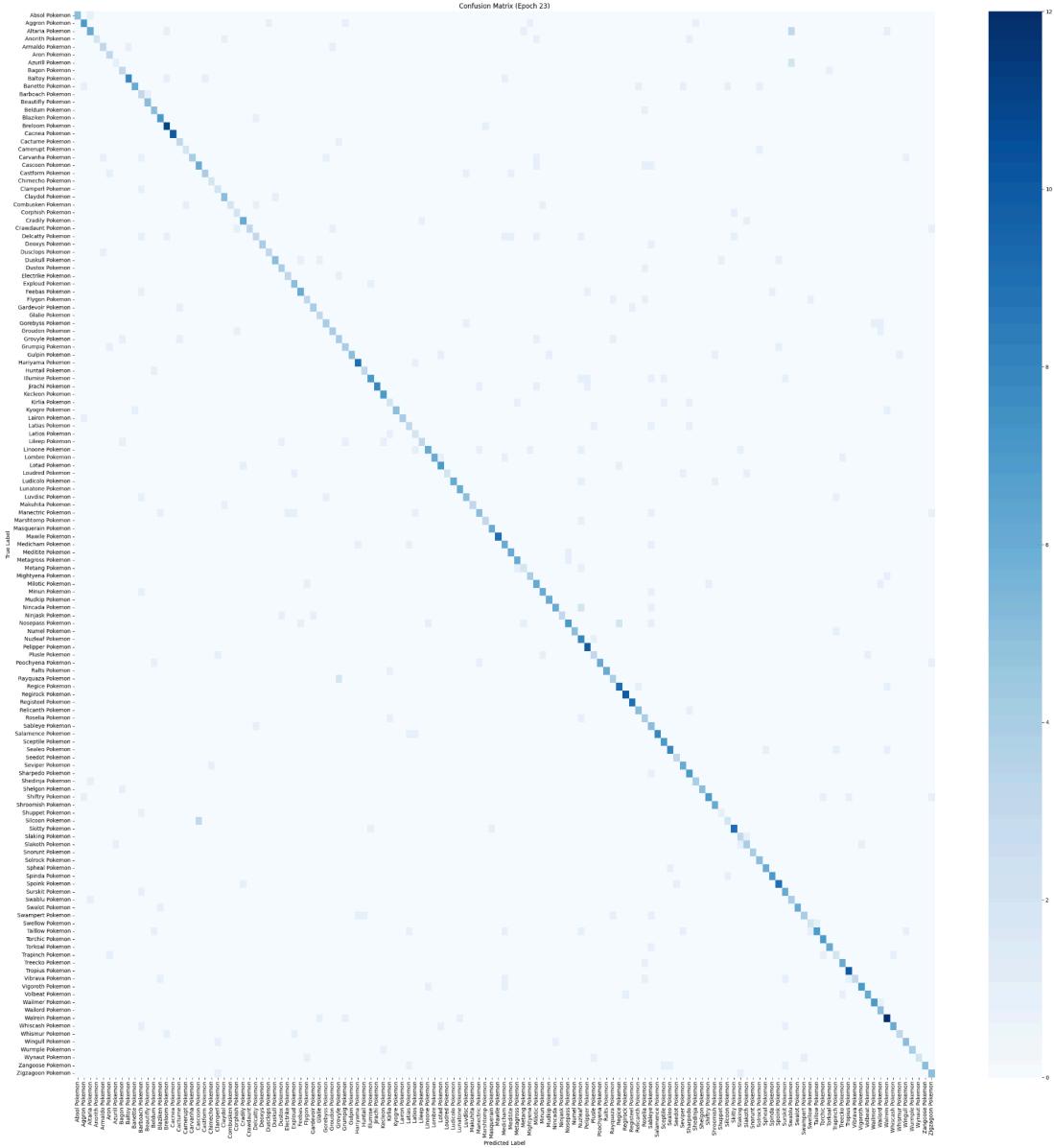
The MobileNetV2 model was trained on the 3rd Generation Pokémon and was evaluated on splitted data from the original dataset. Key performance are summarised in the table

Metric	Score
Accuracy	72.90%
Precision	0.78
Recall	0.73
F1-Score	0.73

B. Confusion Matrix

The following confusion matrix visualizes the model's classification validity across 135 classes. For diagonal elements it represents correct

predictions while off diagonal elements represent wrong predictions.



C. Discussion

Overall Accuracy and robustness

The model that was trained showed respectable efforts given such a small dataset with only basic data augmentations, with a 72.9% score on top 1 accuracy and 83.5% on top 3 accuracy. Having a weighted precision higher than the recall means when the model predicts a certain Pokémon, it is often correct but it occasionally misses identifying specific classes entirely. This confirms the effectiveness of transfer learning as the model was capable

enough to adapt from the ImageNet dataset until it was able to recognize specific Pokémon.

Class Specific Performance

The model showed a difference in ability to distinguish different species of Pokémon. While some have extremely high performance achieving perfect f1 scores of 1 which may stem from distinct color palettes or specific features that separates them from the other species. Examples of these high performers are Beautifly, Cacnea, Crawdaunt, and Feebas. While others have poor performance achieving f1 scores nearing zero or actually zero. This may occur because of its similarity to other evolutionary stages or insufficient unique features that define the species.

V. Conclusion

Through this study we successfully demonstrated that transfer learning with the MobileNetV2 architecture is a viable and effective solution for classifying our dataset of 135 distinct species of Pokémon. Even when having a small dataset of around 4,500 images by reusing pre-trained weights and applying data augmentation techniques to prevent overfitting, the model achieved an accuracy of 72.9% and a top 3 accuracy of 83.5%. These results confirm that a lightweight neural network is able to distinguish complex visual features without requiring massive datasets that is synonymous with deep learning. Which implies it can be used for other fields that may have problems in collecting a large amount of data. However the analysis revealed a glaring weakness where the performance gap between classes with distinct feature and those with high similarity with others. Future work should focus on targeted data collection for underperforming classes and further hyperparameter tuning to enhance the model's sensitivity to subtle details.

References

- [1] K. O’Shea and R. Nash, “An introduction to convolutional neural networks,” *arXiv preprint arXiv:1511.08458*, 2015.

- [2] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015, doi: 10.1038/nature14539.
- [3] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in Proc. 32nd Int. Conf. Mach. Learn., Lille, France, 2015, pp. 448–456.
- [4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, vol. 25, 2012, pp. 1097–1105.
- [5] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 10, pp. 1345–1359, Oct. 2010.
- [6] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" in *Advances in Neural Information Processing Systems*, vol. 27, 2014.
- [7] A. G. Howard *et al.*, "MobileNets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.
- [8] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Salt Lake City, UT, USA, 2018, pp. 4510–4520.
- [9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.
- [10] S. Yang, W. Xiao, M. Zhang, S. Guo, J. Zhao, and F. Shen, "Image data augmentation for deep learning: A survey," *arXiv preprint arXiv:2204.08610*, 2023.