CIS-5603

Temple University
Fall 2023
Artificial Intelligence

Photo Culling using Machine Learning

Sayantan Kundu - TUP47447

Abstract

Brief Overview of the Project

This project delves into the realm of photo culling, a vital process in digital imagery for eliminating unsatisfactory images such as those that are blurred, overexposed, or underexposed. Traditionally a labor-intensive task, the advent of machine learning (ML) offers a promising avenue for automation, enhancing both efficiency and effectiveness. The project harnesses ML to transform the photo culling process, demonstrating its potential to significantly reduce manual effort in digital image processing.

Key Objectives

The primary goal of this initiative was to create an end-to-end application utilizing a Kaggle-sourced dataset of human portraits. This dataset was meticulously transformed into a custom dataset, labeled with specific photographic flaws, to serve as the cornerstone for ML training. The project emphasized the application of transfer learning to fine-tune advanced neural network models such as VGG16, EfficientNetB0, and ResNet50. The focus was on their capability to classify images accurately according to the custom labels. A significant aspect of this project was also the development of a user-friendly interface (UI) application, designed to employ the most effective model for real-time, automated photo culling on user-uploaded images.

Summary of Findings

The project yielded impressive results, with the ResNet50 model standing out for its exceptional performance. It achieved a remarkable accuracy of 99% on a dataset comprising only 1700 images. This high level of accuracy in identifying quality-related issues in images marks a significant advancement in automated photo-culling technology. When deployed through the developed UI on custom images, the model maintained its high performance, effectively demonstrating its practical utility in real-world situations. This breakthrough not only underscores the efficacy of ML in automating the photo culling process but also paves the way for future innovations in digital image management.

Introduction

Background of the Project

Photo culling stands as a pivotal yet daunting task in the world of photography, where professionals and hobbyists alike spend considerable time sifting through vast collections of images. This process involves meticulously reviewing each photograph to identify and eliminate those that do not meet certain quality standards, such as being blurred, poorly lit, or incorrectly composed. In the digital age, where the volume of photos captured is exponentially higher than in the era of film, the task of photo culling has become increasingly overwhelming. Photographers often find themselves spending more time on this laborious task than on the creative aspects of their work. The necessity for a streamlined and efficient culling process is more pressing than ever, as the digital revolution continues to flood our world with images.

Problem Statement

The challenge addressed by this project lies in the time-consuming and subjective nature of manual photo culling. The traditional process is not only labor-intensive but also prone to inconsistency due to the variability in human judgment. As photographers grapple with ever-growing image libraries, the need for an automated, reliable, and efficient method of photo culling is evident. This project seeks to address these challenges by harnessing the power of

machine learning to automate the photo culling process, thus reducing the time burden on photographers and increasing the consistency of outcomes.

Importance of Photo Culling in Digital Imagery

In the realm of digital imagery, photo culling is a crucial step in the workflow of photographers and image editors. It is the gatekeeper of image quality, ensuring that only the best photographs make it to the final album, publication, or digital gallery. Efficient photo culling not only enhances the overall quality of visual content but also streamlines the workflow, allowing photographers to focus more on creative pursuits rather than the monotonous task of sorting through thousands of images. In industries where image quality is paramount, such as in media, advertising, and photography, effective photo culling can significantly impact the final output's success and appeal.

Objectives of the Project

The primary objective of this project is to revolutionize the process of photo culling by employing artificial intelligence (AI). By leveraging machine learning algorithms, the project aims to automate the culling process, making it faster, more accurate, and less dependent on human intervention. The goal is to develop a system that can accurately identify and categorize images based on common quality issues, thereby assisting photographers in quickly filtering out undesirable photos from their collections. In essence, this project seeks to blend the precision and speed of AI with the nuanced needs of photo culling, providing a modern solution to a long-standing challenge in the field of digital imagery.

Literature Review

Resources Looked into for this project

1. PhotoML: Photo culling with Machine Learning
2. The Role of AI in Photo Culling and Editing: Revolutionizing Every Photographer's Workflow
3. Camera Futura uses ML.NET to automate photo culling and organizing.

Overview of Existing Methods and Technologies in Photo Culling and Machine Learning

Photo culling, a vital step in a photography workflow, involves selecting the best images from a shoot, often a time-consuming and subjective task. Recent developments in machine learning (ML) have opened avenues for automating this process. ML-based solutions like Camera Futura's Futura Photo software utilize algorithms to automate culling and organizing photos, thereby addressing the challenges of time and subjectivity inherent in manual culling. This software focuses on enthusiast photographers and employs ML.NET, a framework allowing seamless integration with existing .NET applications, to power its machine learning features.

Discussion of Similar Works or Studies

In the field of automated photo culling, one notable study, PhotoML, demonstrates the application of ML in culling photos based on technical quality. This project mirrors the broader trend in the photography industry, where ML is increasingly being utilized to address the repetitive and tedious aspects of photo management, especially culling and organizing.

Relevance of These Works to Your Project

Both the Camera Futura case and the PhotoML study provide valuable insights into how ML can be effectively utilized in photo culling, highlighting the potential for significant time savings and improved consistency in results. These examples are particularly relevant to our project as they showcase successful implementations of ML in addressing the same problem our project aims to solve. The use of ML.NET in Camera Futura's solution is also noteworthy, as it demonstrates the feasibility of integrating machine learning capabilities into existing software frameworks, a consideration crucial for our project's development pathway.

Methodology

1. **Dataset Preparation**

a. Source of the Base Dataset

The base dataset for this project was sourced from Kaggle, specifically the Human Faces (Object Detection) dataset available at Kaggle's Human Faces Dataset. This dataset comprises a comprehensive collection of human portrait images, which serve as the foundational data for the project's photo culling model. It provides a diverse range of human faces, essential for developing a robust and versatile culling system.

b. Steps for Creating a Custom Dataset for Culling

The custom dataset for culling was crafted using a Python script that employs several libraries and techniques to simulate common photographic flaws:

Libraries Used: The script uses OS for directory operations, cv2 (OpenCV) for image processing, NumPy for numerical operations, and CSV for data storage.

Image Transformations:

Blurring: Applied Gaussian blur with varying intensities to simulate blurred images.

Camera Shake: Added a shake effect using random displacements and image remapping to mimic camera movement.

X-axis Distortion: Distorted images along the X-axis to varying degrees to represent lens or perspective distortions.

Exposure Adjustments: Modified image exposure to create underexposed and overexposed variants, simulating common exposure issues.

Image Sampling: The script randomly selected a subset of images (120 in this case) from the input directory for transformation. This approach ensures diversity and reduces the computational load.

Saving Transformed Images: Each transformed image was saved in corresponding folders (like 'blurred', 'camera_shake', etc.), creating a structured dataset.

Data Recording: The paths of transformed images along with their labels (indicating the type of transformation) were stored in a CSV file. This file serves as a reference point for model training, linking each image to its respective label.

Duplicate Handling and Cleaning: The script also manages duplicates and removes unselected images from the original dataset to maintain dataset integrity and relevance.

This meticulous and systematic approach to dataset creation ensures that the machine learning model is trained on a wide range of image qualities, enhancing its ability to accurately perform photo culling.

c. Preprocessing and Data Cleaning Techniques

The preprocessing and data-cleaning stage of the project involved several key steps:

Data Loading and Preparation:

The dataset was loaded using pandas.
The dataset was divided into training, validation, and test sets, with a standard split ratio.

Image Resizing and Batch Processing:

Images were resized to a uniform size (224x224 pixels) for consistency.
Batch processing was set with a batch size of 32, optimizing the training process for efficiency.

Visualization:

A function was created to display random images from the dataset along with their labels. This visual inspection ensures diversity and correctness in the dataset. As shown in Figure 1.

Data Augmentation:

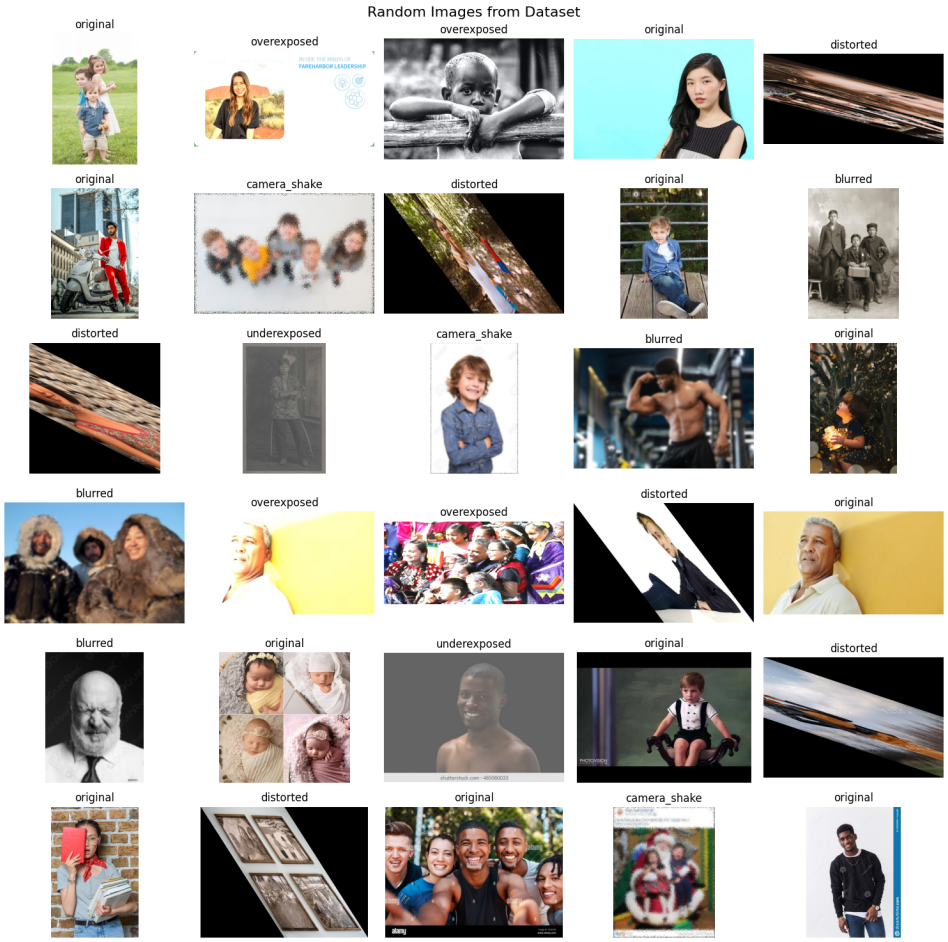
For the training set, augmentation techniques like shearing, zooming, and horizontal flipping were applied. These augmentations help the model generalize better by introducing variations in the training data.

Generators for Data Flow:

ImageDataGenerator was used for rescaling and augmenting the images. Separate generators were created for the training, validation, and test sets to streamline the flow of data during model training.

These steps ensure that the data is well-prepared, augmented, and fed into the machine learning model in an efficient and structured manner, thereby enhancing the model's training process and performance.

Figure 1: Sample Image Visualization

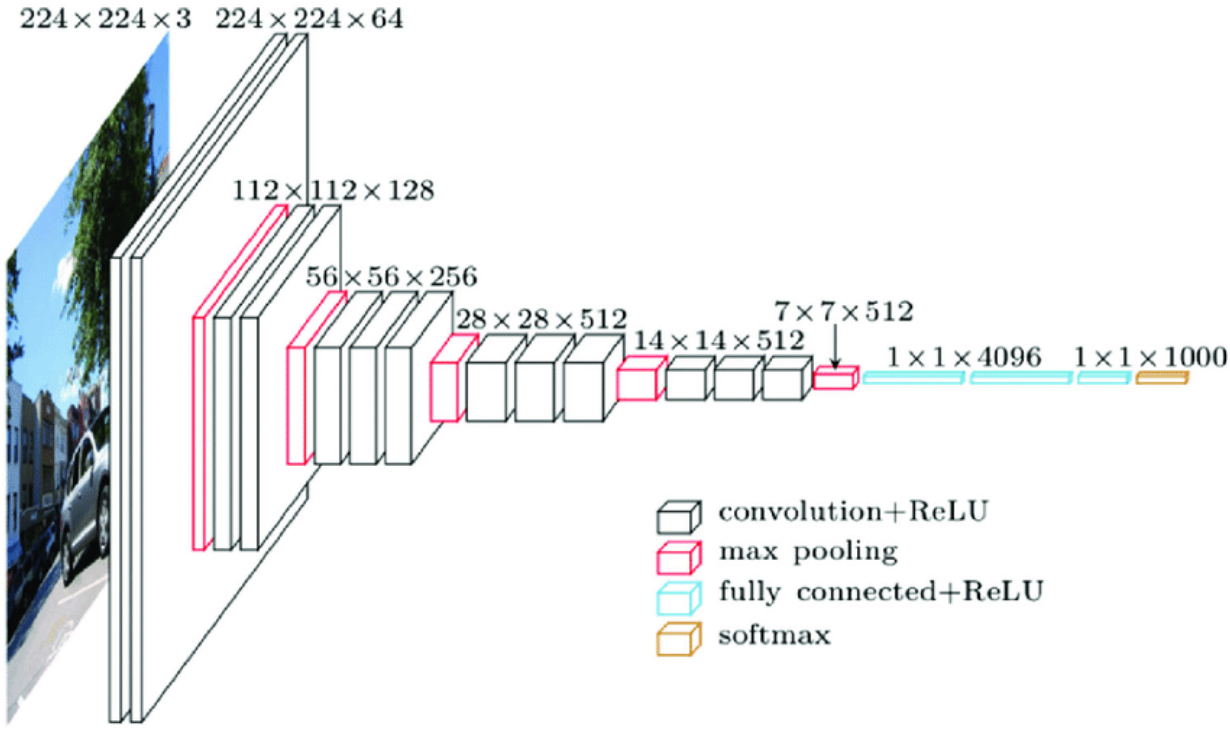


2. Model Development and Testing

a. Description of the Models Used (VGG16, EfficientNetB0, ResNet50)

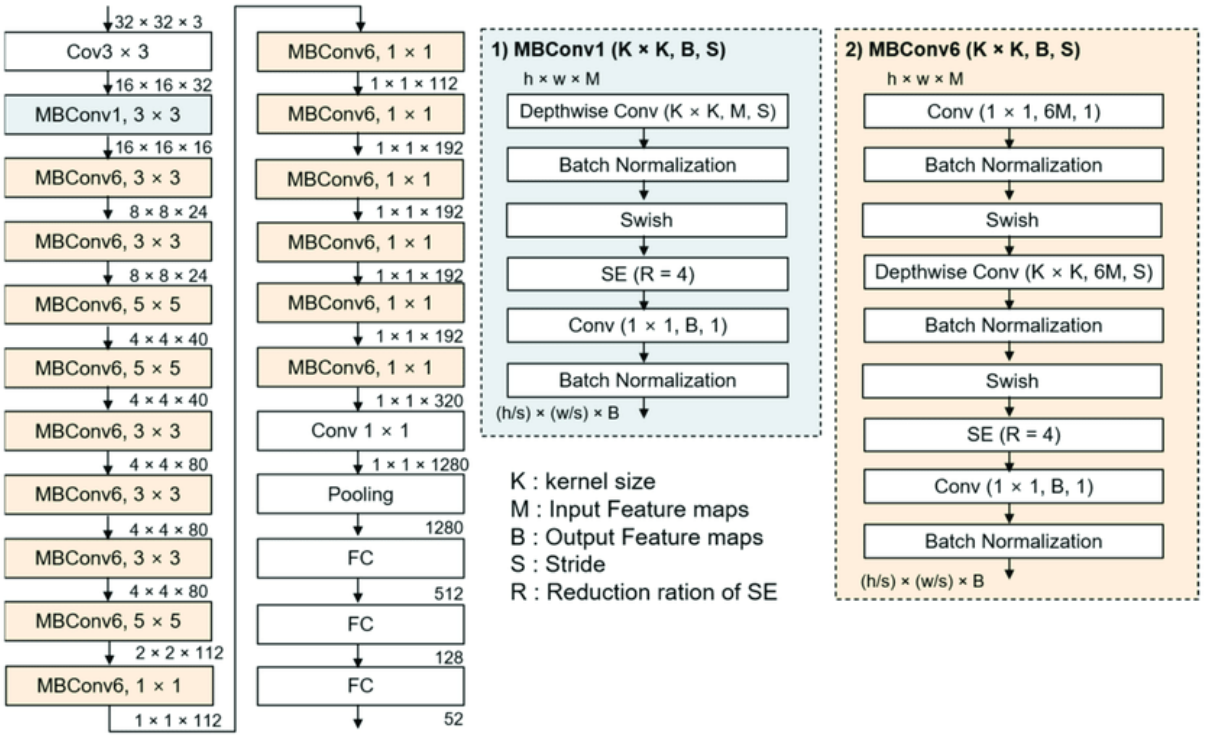
1. VGG16: Developed by the Visual Graphics Group at Oxford, VGG16 is a deep convolutional neural network known for its simplicity and depth. It has 16 layers and is known for its excellent performance on image recognition tasks. The architecture diagram is shown in Figure 2. (sourced from - https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.researchgate.net%2Ffigure%2FAn-overview-of-the-VGG-16-model-architecture-this-model-uses-simple-convolutional-blocks_fig2_328966158&psig=AOvVaw1e2eeK8gJLyvuJ2DTUeGRG&ust=1702687878075000&source=images&cd=vfe&ved=0CBIQjRxqFwoTCIiVo-mckIMDFQAAAAAdAAAAABAI)

Figure 2: VGG16 Model Architecture:



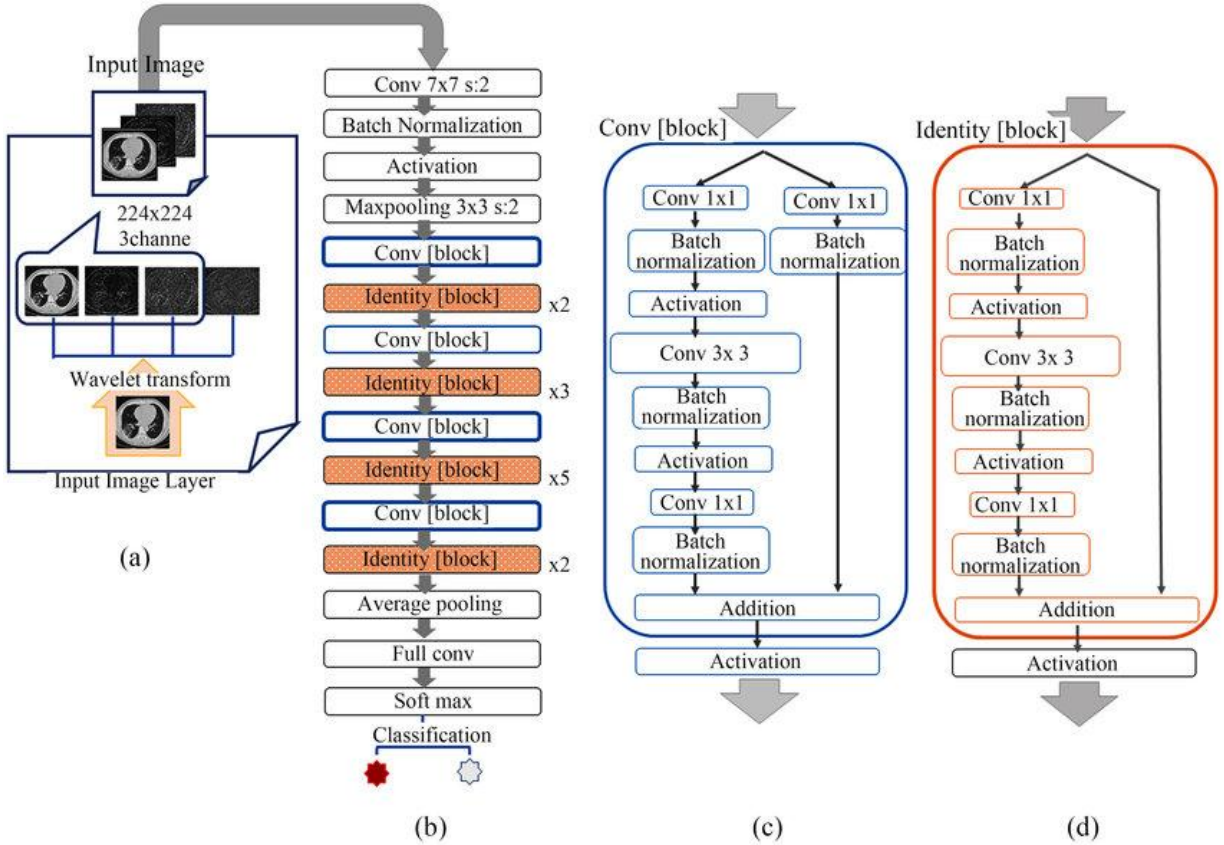
2. EfficientNetB0: EfficientNetB0, part of the EfficientNet family, represents a breakthrough in scaling CNNs. Developed by Google researchers, it systematically scales network width, depth, and resolution with a set of fixed scaling coefficients. EfficientNetB0, as the baseline model, achieves remarkable efficiency and accuracy due to this balanced scaling, making it superior for tasks that require both high performance and computational efficiency. The architecture diagram is shown in Figure 3. (sourced from - https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.researchgate.net%2Ffigure%2FThe-structure-of-an-EfficientNetB0-model-with-the-internal-structure-of-MBConv1-and_fig2_351057828&psig=AOvVaw0b7O3sEqnnr5sSX5vhuwnT&ust=1702688132134000&source=images&cd=vfe&ved=0CBIQjRxqFwoTCIjl0NydkIMDFQAAAAAdAAAAABAE)

Figure 3: EfficientNetB0 Model Architecture:



3. ResNet50: ResNet50, a key model in the ResNet family, stands out with its deep 50-layer architecture. Its primary innovation is the introduction of 'skip connections' or residual connections, allowing layers to skip one or more layers. This design counters the vanishing gradient problem in deep networks, maintaining performance even as depth increases. ResNet50's architecture makes it adept at handling complex image recognition tasks, and its success has led to wide adoption in various deep-learning applications. The architecture diagram is shown in Figure 4. (sourced from - https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.researchgate.net%2Ffigure%2FOutline-of-ResNet-50-architecture-a-A-3-channel-image-input-layer-The-LL-LH-and-HH_fig3_343233188&psig=AOvVaw02MOkaFKG7lgN7JlwUo55z&ust=1702688218622000&source=images&cd=vfe&ved=0CBIQjRxqFwoTCOilrYeekIMDFQAAAAAdAAAAABAE)

Figure 4: ResNet50 Model Architecture:



The table below outlines the key similarities and differences between these models:

Feature	VGG16	EfficientNetB0	ResNet50
Depth	16 layers	Scalable (B0 baseline)	50 layers
Key Characteristics	Deep with sequential layers	Balanced scaling of depth, width, resolution	Skip connections for deeper networks
Use Case	Image recognition	Efficient performance and scalability	Deep learning with reduced gradient problems

b. Hyperparameter Tuning and Optimization Strategies

Hyperparameter tuning involves adjusting parameters like learning rate, number of layers, and activation functions to optimize model performance. Strategies like grid search or random search can be used to experiment with different combinations. For fine-tuning, parts of pre-trained models are often re-trained on new data to adapt to specific tasks, which is crucial for achieving high accuracy in tasks like image culling.

Model Compilation: Each model is compiled using categorical cross-entropy loss and Adam optimizer. The learning rate is set at 1e-4, which is a crucial hyperparameter for training efficiency.

Model Training: Models are trained using the fit method on the augmented training data, with epochs and validation data specified. This process iteratively updates model weights to minimize loss and improve accuracy.

Model Evaluation: After training, models are evaluated on a test dataset to obtain test accuracy, providing an unbiased performance metric.

Experiments with VGG16: Various VGG16 configurations were tested, including base models, increased dense layers, additional convolutional layers, and transfer learning without freezing layers. Each experiment was logged for comparison.

Loss Visualization: Training and validation loss curves for each experiment were plotted to track overfitting or underfitting and to understand the model's learning pattern.

Experiments with EfficientNet and ResNet: Similar strategies were applied to EfficientNetB0 and ResNet50, including modifications to dense layers, increased dropout, additional layers, and fine-tuning.

Fine-Tuning: For EfficientNet and ResNet, some experiments involved fine-tuning by making deeper layers trainable, which adapts the model more closely to the specific dataset.

c. Training Process and Validation Methods

The models can be trained on the custom dataset by feeding them the augmented image data, using techniques like batch normalization, dropout, and appropriate activation functions to enhance learning. The training involves monitoring loss and accuracy metrics, adjusting learning rates, and using callbacks for optimal performance. Validation methods include using a separate dataset to evaluate the model and visualizing loss and accuracy curves to understand model performance over training epochs. This helps in identifying overfitting or underfitting and in making necessary adjustments.

Training: The models are trained using the custom dataset, augmented with various transformations. The training process involves iterating through epochs while monitoring loss and accuracy.

Validation: Validation data is used alongside training to assess model performance and generalization capability.

Loss and Accuracy Tracking: Loss and accuracy curves for both training and validation phases are plotted. This visualization helps in identifying trends like overfitting or underperformance and informs adjustments in model architecture or training strategy.

Saving Experiments: Details of each experiment, including model configurations and test accuracies, are stored for analysis and comparison, aiding in the selection of the best-performing model.

3. Prototype User Interface

a. UI Design and Development:

The UI, designed using Flask, a Python web framework, offers a clean and interactive user experience. The Flask application handles the backend processes, including loading the machine learning model and handling image uploads. The HTML and CSS, along with Bootstrap, are utilized for the front end, creating a responsive and aesthetically pleasing interface. This combination ensures the UI is adaptable to various devices and screen sizes. The design emphasizes user-friendliness, with a focus on simplicity and efficient navigation, allowing users to easily upload and view images. Figure 5 shows the UI prototype window with custom images.

b. Integration with Machine Learning Model:

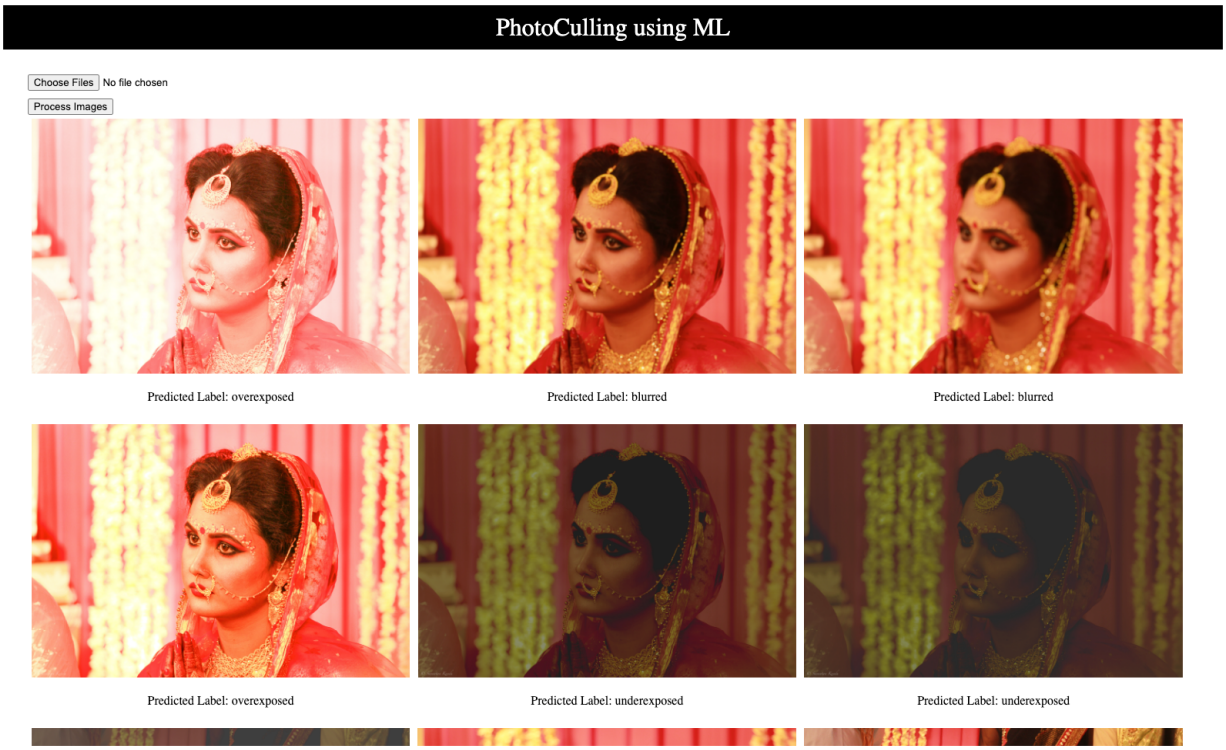
In the integration with the machine learning model, the Flask application loads the pre-trained ResNet50 model (resnet_exp4.h5) and class labels (class_labels.pkl). The code includes a function to preprocess uploaded images, resizing them to 224x224 pixels and normalizing their pixel values. This preprocessing ensures the images are compatible with the model's input specifications. The application then uses the model to predict labels for these images, employing the predict function on the processed image arrays. The predicted labels are derived from the model's output, indicating the

classifications of the uploaded images. This integration demonstrates the practical application of the ResNet50 model in a user-friendly web application.

c. Features and Functionalities:

The UI's features and functionalities include a multi-image upload feature, enabling users to upload several images simultaneously. Once uploaded, the images are displayed on the UI alongside their predicted labels, as determined by the machine learning model. This functionality enhances the user experience, making the application a practical tool for tasks such as photo culling. Users can easily view and interpret the model's predictions, streamlining the process of categorizing and managing large sets of images.

Figure 5: UI Prototype with Custom Images:



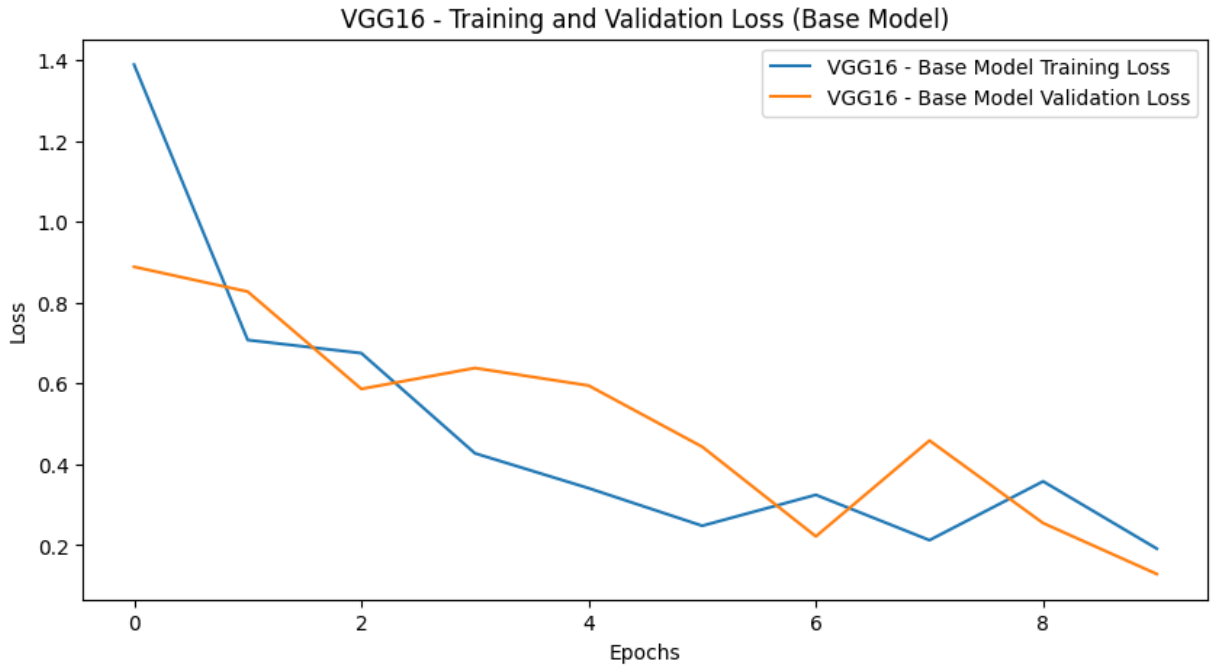
Results and Discussion

1. Performance evaluation of the models

a. VGG16 Model Experiments:

Base Model Experiment: Started with the original VGG16 architecture, modified by removing the top layers and adding a new set of layers tailored for the specific classification task. This included a flattening layer, a dense layer with 256 neurons, a dropout layer for regularization to prevent overfitting, and a final softmax layer for outputting the class probabilities. Figure 6 will show how the model behaves as a base experiment.

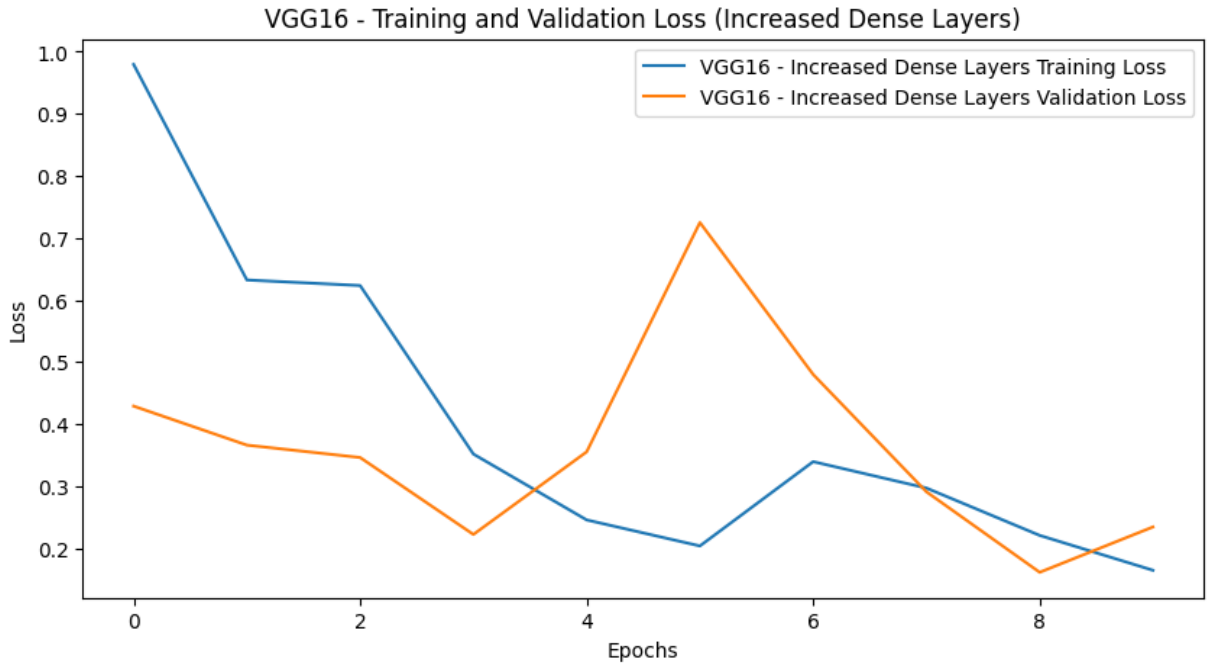
Figure 6: VGG16 Base Model Experiment:



Increased Dense Layers: The complexity was increased from the base model by adding more dense layers, which aimed to enhance the model's ability to learn more complex features. However, this also increased the risk of overfitting due to

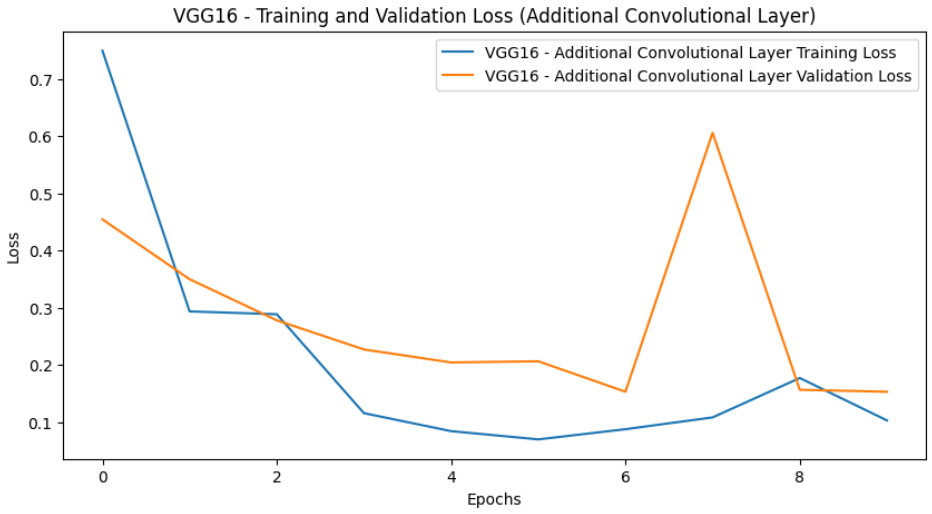
the greater number of trainable parameters. Figure 7 will show how the model behaves with an increased dense layer in the VGG16 experiment.

Figure 7: VGG16 Increased Dense Layer Experiment:



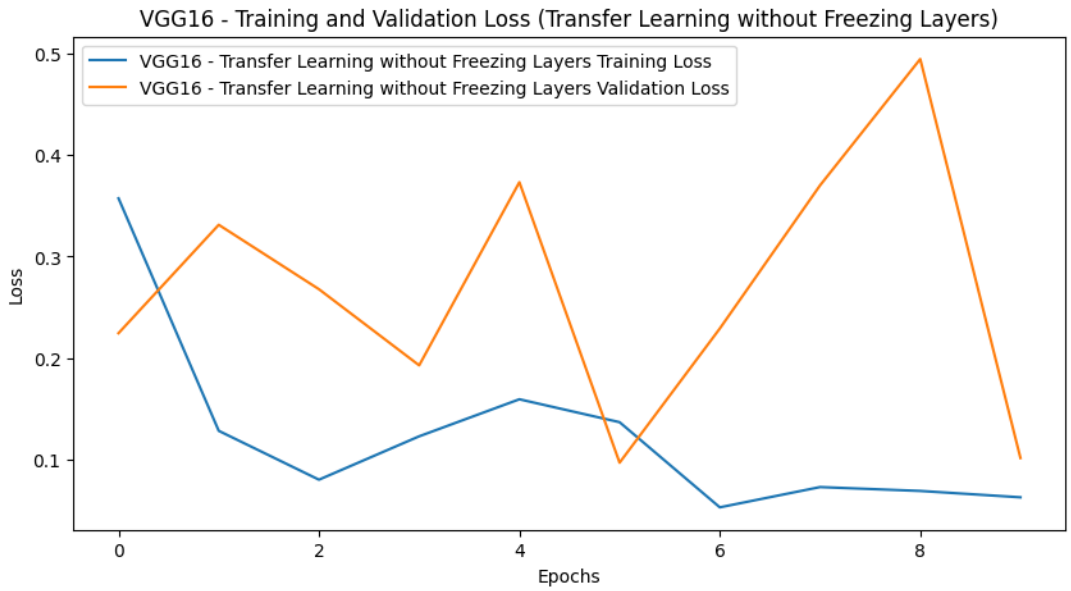
Additional Convolutional Layer: A convolutional layer was added to the base model to improve the feature extraction capabilities. This experiment was designed to determine whether additional convolutional processing would result in better performance on the image classification task. Figure 8 will show how the model behaves with an additional convolutional layer in the VGG16 experiment.

Figure 8: VGG16 Additional Convolutional Layer Experiment:



Transfer Learning without Freezing Layers: Instead of freezing the pre-trained layers of the model, they were left trainable to adapt more closely to the new dataset. This approach can sometimes yield better performance as it allows the model to fine-tune the pre-trained weights on the new data, but it also requires careful tuning to avoid overfitting. Figure 9 will show how the model behaves with transfer learning in the VGG16 experiment.

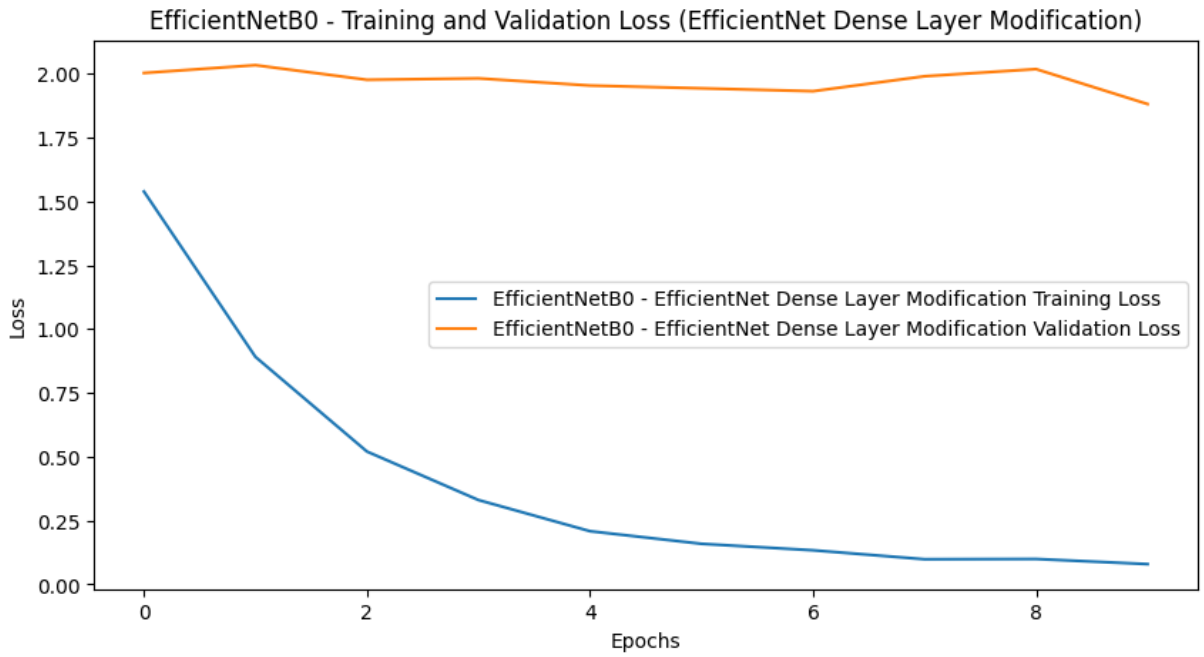
Figure 9: VGG16 Transfer Learning Experiment:



b. EfficientNetB0 Model Experiments:

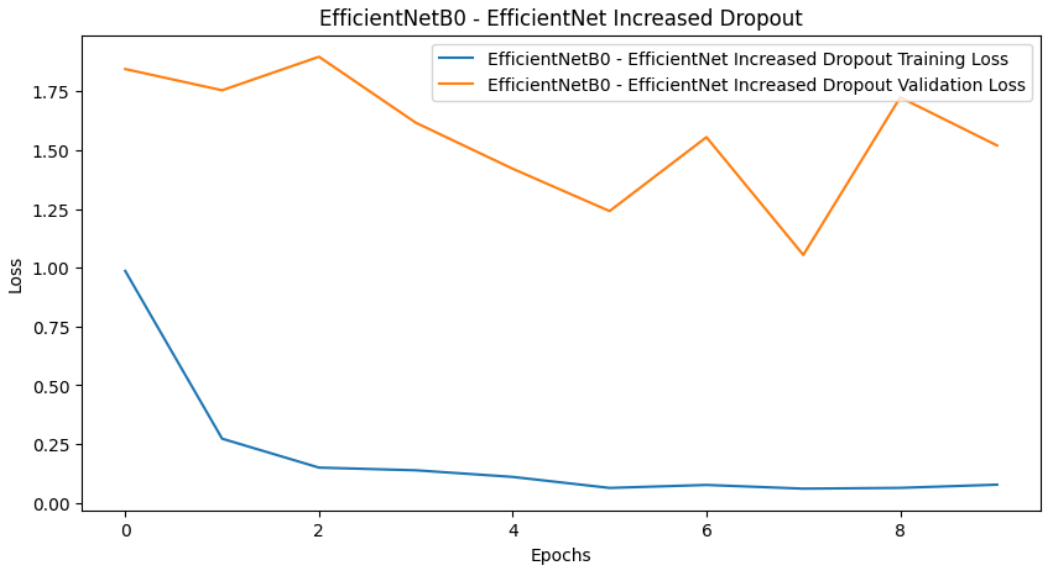
Dense Layer Modification: The experiment began with incorporating a global average pooling layer to the base EfficientNetB0 model, ensuring that the subsequent dense layer with 256 neurons had a manageable input size. The purpose was to capture the essence of the features extracted by the EfficientNet architecture. Figure 10 will show how the model behaves with dense layer modification in the EfficientNetB0 experiment.

Figure 10: EfficientNetB0 Dense Layer Modification Experiment:



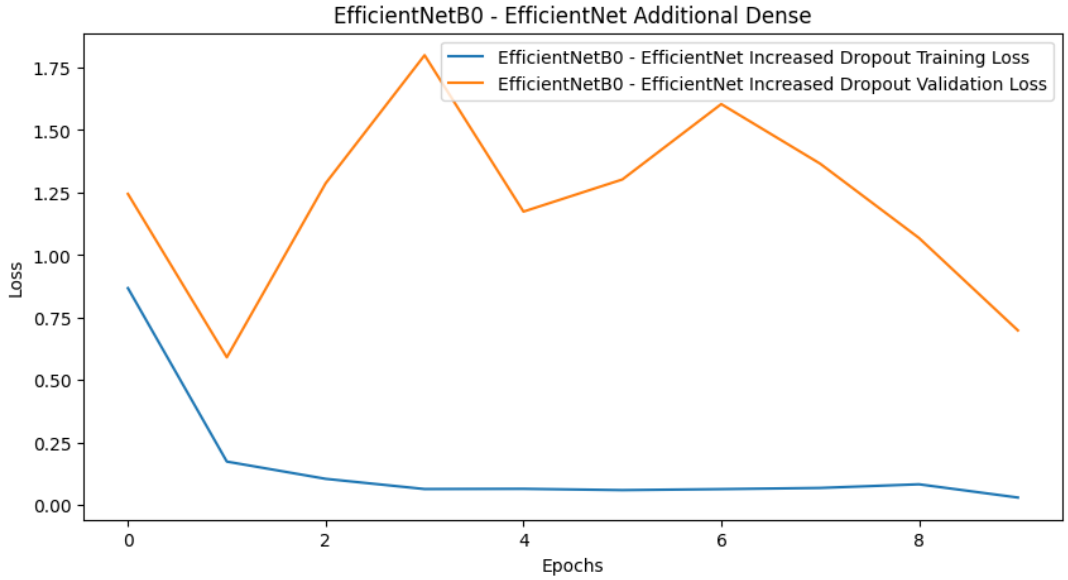
Increased Dropout: To enhance the model's generalization, the dropout rate was elevated. This experiment tested the hypothesis that a higher dropout would lead to better generalization by forcing the network to learn more robust features that do not rely on the presence of specific neurons. Figure 11 will show how the model behaves with increased dropout in the EfficientNetB0 experiment.

Figure 11: EfficientNetB0 Increased Dropout Experiment:



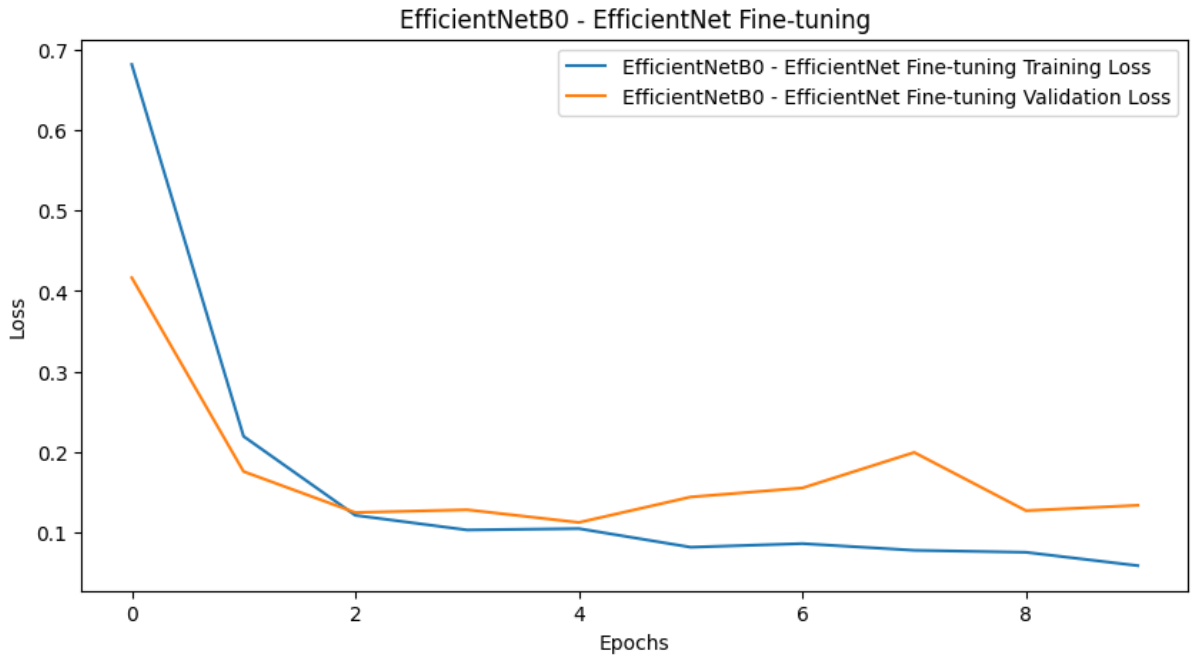
Additional Dense Layer: By introducing an additional dense layer with 512 neurons, the model's capacity for feature interpretation was extended. This layer aimed to provide a deeper level of abstraction of the features, potentially capturing more complex relationships in the data. Figure 12 will show how the model behaves with additional dense layers in the EfficientNetB0 experiment.

Figure 12: EfficientNetB0 Additional Dense Layer Experiment:



Fine-tuning: The most advanced of the experiments, fine-tuning involved strategically allowing the latter half of the EfficientNetB0 layers to adjust their weights during training. This selective fine-tuning sought to strike a balance between leveraging the pre-trained knowledge embedded in the model and adapting to the nuances of the new dataset. Figure 13 will show how the model behaves with fine-tuning the EfficientNetB0 experiment.

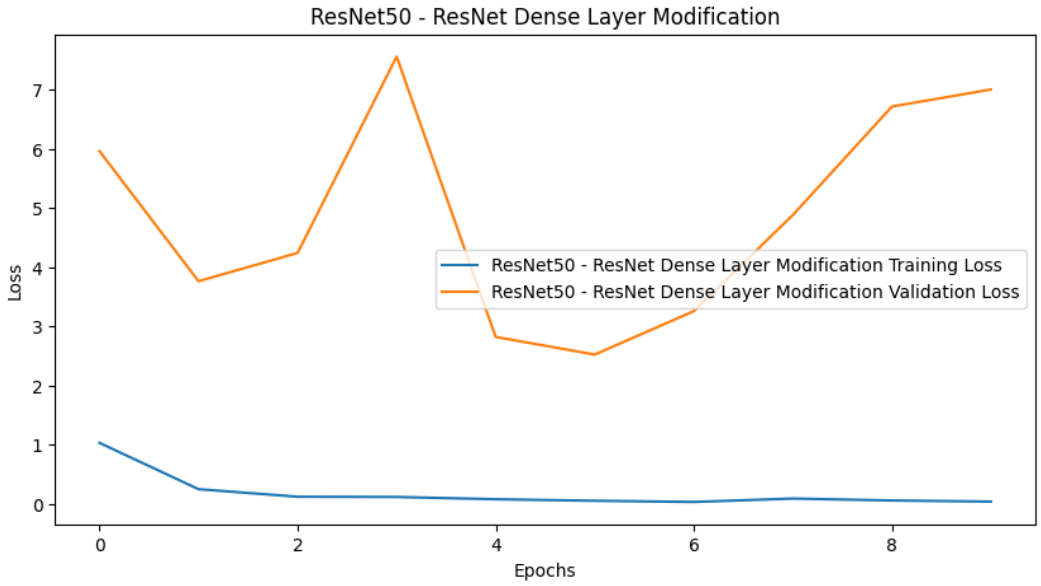
Figure 13: EfficientNetB0 Fine-tuning Experiment:



c. ResNet50 Model Experiments:

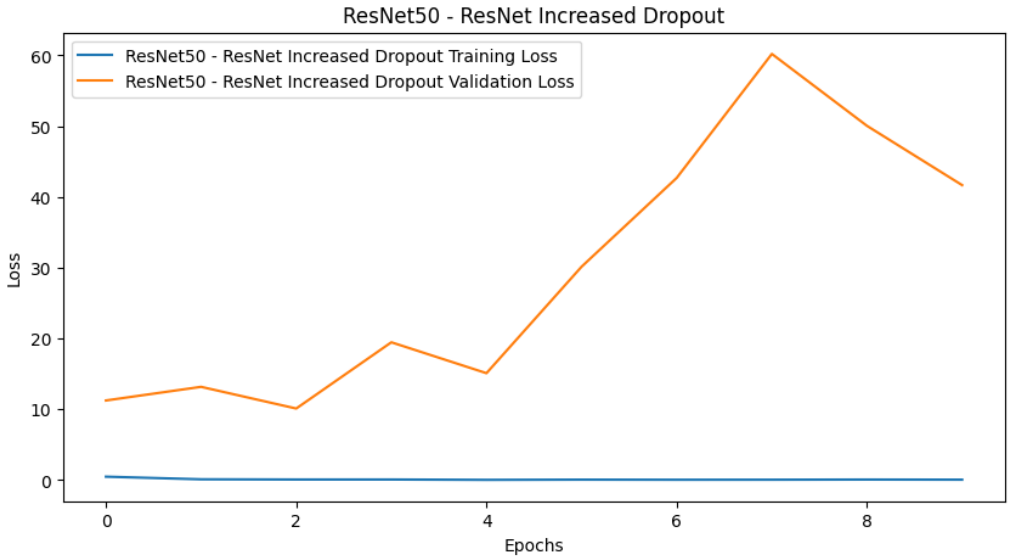
Dense Layer Modification: The initial experiment begins with a foundational approach, leveraging the ResNet50 model's powerful feature extraction capabilities pre-trained on ImageNet, and tailoring it to a new classification task by appending a series of layers designed to process the extracted features for a six-class output. Figure 14 will show how the model behaves with dense layer modification in the ResNet50 model experiment.

Figure 14: ResNet50 Dense Layer Modification Experiment:



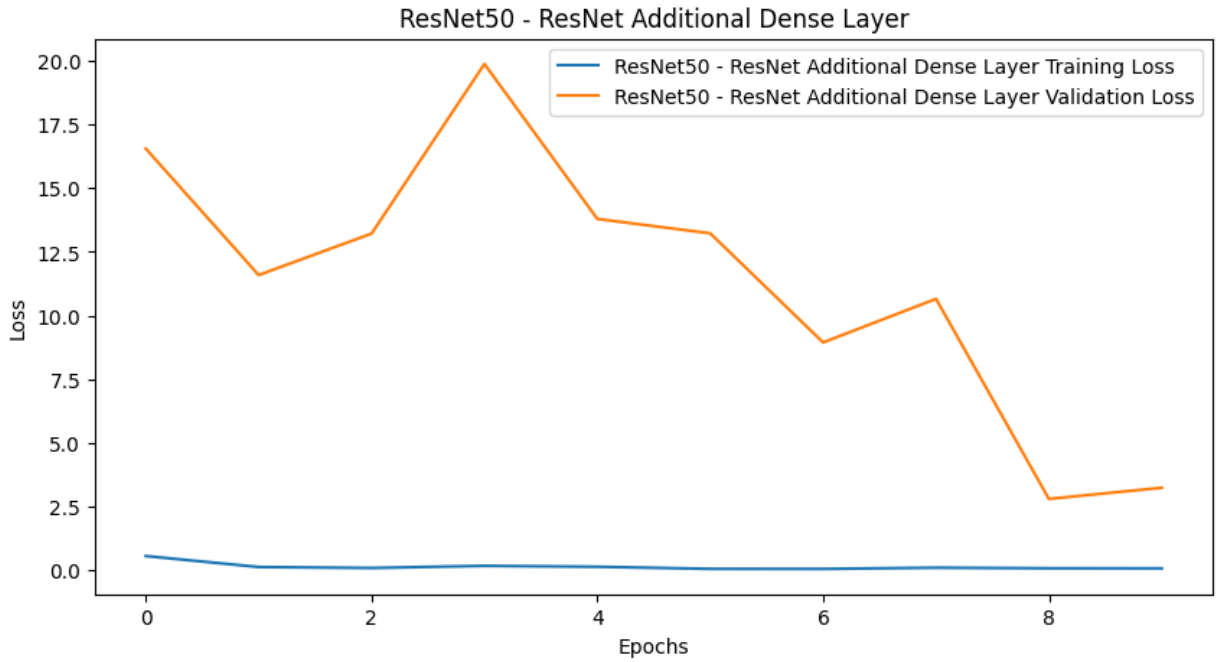
Increased Dropout: In the second experiment, the dropout rate is increased to combat overfitting, a strategy predicated on the hypothesis that by randomly deactivating neurons during training, the model is forced to learn more robust features. Figure 15 will show how the model behaves with increased dropout in the ResNet50 model experiment.

Figure 15: ResNet50 Increased Dropout Experiment:



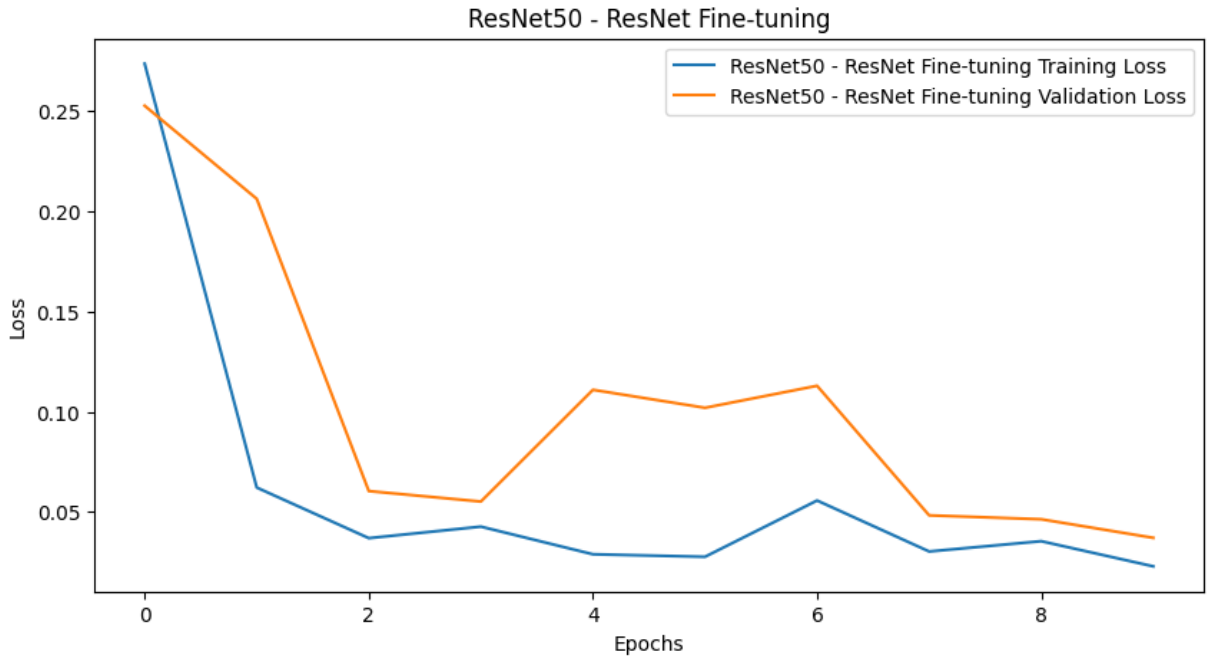
Additional Dense Layer: The third experiment introduces an additional dense layer with a significant increase in neurons, testing whether this added complexity enables the model to capture a richer representation of the data, thus enhancing its ability to distinguish between more subtle variations within the classes. Figure 16 will show how the model behaves with an additional dense layer in the ResNet50 model experiment.

Figure 16: ResNet50 Additional Dense Layer Experiment:



Fine-tuning: The fourth and most intricate experiment employs a fine-tuning strategy. By making the layers trainable halfway through the network, the model can fine-tune the pre-learned features to the specific dataset, potentially leading to an increase in accuracy and a better generalization to new images. Figure 17 will show how the model behaves with fine-tuning the ResNet50 model experiment.

Figure 17: ResNet50 Fine-tuning Experiment:



2. Comparative analysis of VGG16, EfficientNetB0, and ResNet50

Table-1 shown below will show details of the Experiment with the Test Accuracy:

Sl No.	Experiment	Test Accuracy	Model Variable Name
0	VGG16 - Base Model	0.949405	vgg16_base_model
1	VGG16 - Increased Dense Layers	0.913690	vgg16_exp2

2	VGG16 - Additional Convolutional Layer	0.925595	vgg16_exp3
3	VGG16 - Transfer Learning without Freezing Layers	0.961310	vgg16_exp4
4	EfficientNetB0 - EfficientNet Dense Layer Modification	0.369048	efficientnet_exp1
5	EfficientNetB0 - EfficientNet Increased Dropout	0.497024	efficientnet_exp2
6	EfficientNetB0 - EfficientNet Additional Dense Layer	0.782738	efficientnet_exp3
7	EfficientNetB0 - EfficientNet Fine-tuning	0.949405	efficientnet_exp4
8	ResNet50 - ResNet Dense Layer Modification	0.142857	resnet_exp1
9	ResNet50 - ResNet Increased Dropout	0.136905	resnet_exp2
10	ResNet50 - ResNet Additional Dense Layer	0.619048	resnet_exp3
11	ResNet50 - ResNet Fine-tuning	0.991071	resnet_exp4

The **VGG16** model's performance across different experimental conditions shows varied behavior in training and validation loss curves. Initially, the base model demonstrates a steady decrease in both training and validation loss, suggesting good learning with generalization. However, with increased dense layers, there's a spike in validation loss, indicating potential overfitting where the model learns the training data too well but fails to generalize. Adding an additional convolutional layer seems to stabilize learning initially, but an eventual increase in validation loss again points toward overfitting. Lastly, the experiment with transfer learning without freezing layers exhibits fluctuating losses, which could imply that the model is too complex for the data size or that it requires further fine-tuning of hyperparameters.

The **EfficientNetB0** loss curves indicate various outcomes from the experiments. The first experiment shows a consistent decrease in training loss, suggesting that the model is learning effectively, but the flat validation loss implies a plateau in learning from the validation set. In the second experiment, the fluctuating validation loss might suggest that the model struggles with generalization, possibly due to the increased dropout rate. The third chart again shows volatility in validation performance, raising questions about the model's stability. Finally, the fourth experiment presents a more stable decrease in both training and validation loss, indicating a balance between learning and generalization, which may be attributed to the fine-tuning process.

The **ResNet50** loss curves for the experiments reveal a compelling story about the model's learning process. Initially, with the Dense Layer Modification experiment, we observe a descending training loss indicating learning progress. However, the validation loss shows fluctuations, suggesting that the model may not be generalizing well. Increased Dropout introduces more significant variability, particularly in the validation phase, which could be due to the model's inability to establish stable patterns within the data. The Additional Dense Layer's effect is somewhat mixed, with a general trend of improvement but still some variability. Lastly, the Fine-tuning experiment shows a promising reduction in loss, with both the training and validation losses decreasing, indicating an effective learning and generalization balance.

The final experiment with ResNet50, which incorporated fine-tuning, achieved the best results, likely due to a combination of factors. Fine-tuning allows the pre-trained layers of

the network, already skilled at extracting general features from the ImageNet dataset, to adjust to the specifics of the new task. This process can capture more relevant and subtle features important for the specific dataset used. Moreover, fine-tuning only a portion of the layers prevents overfitting, as it retains some of the model's original, generalized knowledge. This approach finds a balance between retaining learned patterns and adapting to new data, resulting in a model that performs well on the test set, as reflected in the highest test accuracy among all experiments.

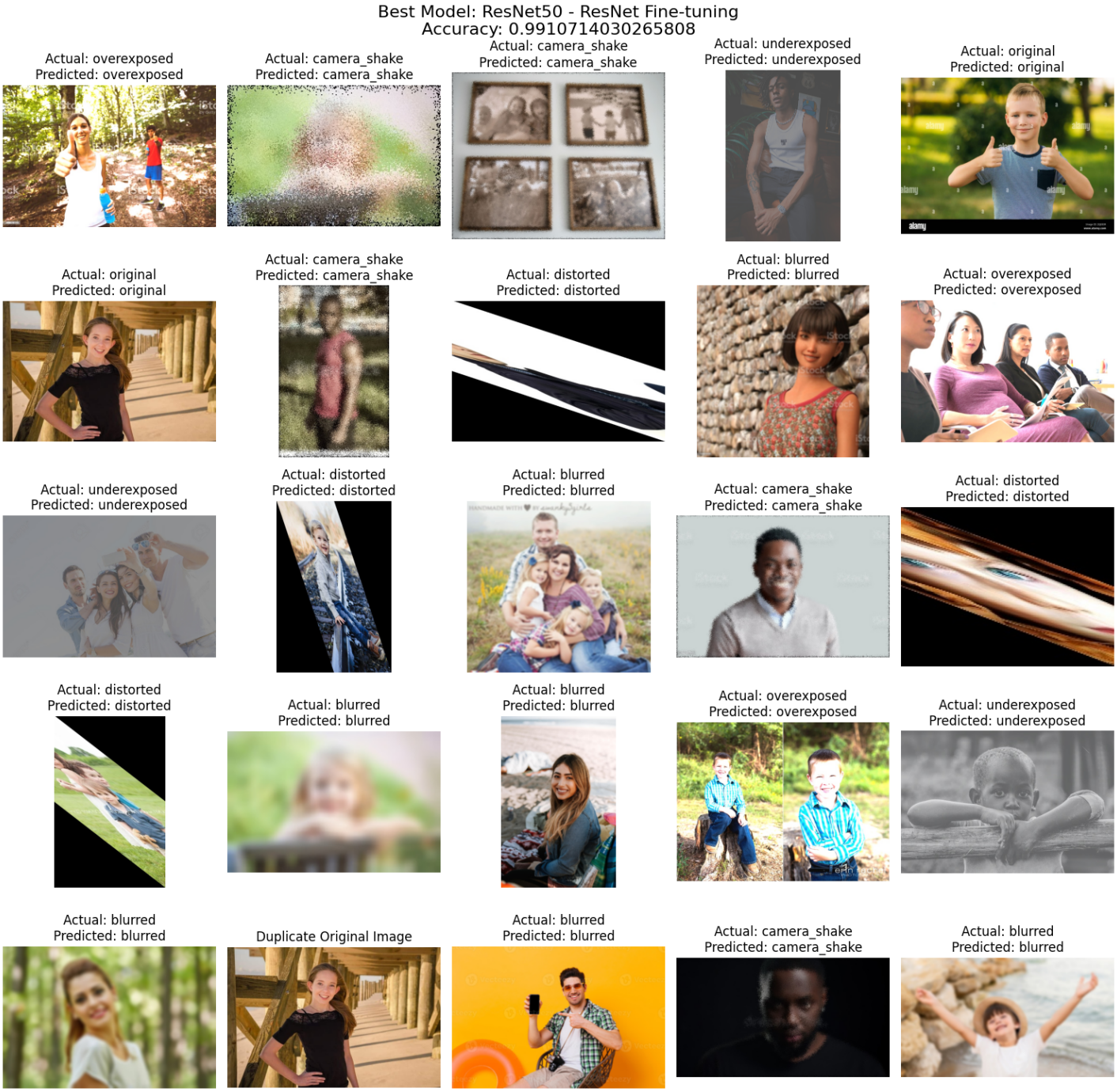
3. Insights from the model testing and selection

Testing and selection of the model involved a structured approach, utilizing a dataset to assess each model's performance rigorously. The best model was identified based on test accuracy from a series of experiments logged in a CSV file. For a detailed analysis, a subset of the test dataset was chosen randomly. This subset was used to evaluate the model's predictive capabilities and to visually compare actual versus predicted labels.

The most accurate model, as determined from the experiments, was then loaded, and its performance was further scrutinized on a random sample of images. Unique hashes of the original images were created to check for duplicates, ensuring that the model's performance was evaluated on distinct data points. The results were plotted to provide a visual representation of the model's prediction against the actual labels, offering insights into its accuracy and potential areas for improvement. This process exemplifies the rigorous testing needed to select the most effective model for practical application.

Figure 18 shows what the best model ie, the ResNet50 model which has been fine-tuned with the custom dataset is performing on an unseen random sampled dataset of 75 images out of which again 15 images are sampled each time to see how the model is performing to predict on the image.

Figure 18: Random Sampled Image Test on Best Model:



4. **Functionality of the prototype UI**

   The prototype UI, realized through Flask, is a web-based interface that interacts with a pre-trained ResNet50 model. Users can upload images directly on the platform, triggering a sequence of backend operations: the images are decoded, preprocessed to conform to the model's input size, and normalized to match the training conditions. The ResNet50 model then processes the images, predicting their categories. These predictions are encoded and sent back to the UI, where users can view both the uploaded images and their corresponding labels. This setup offers a practical and user-friendly application for image classification tasks, enabling real-time predictions without the need for manual coding or command-line interactions.

Conclusion

The project's foray into leveraging machine learning for photo culling culminated in a series of experiments with various architectures, each with its strengths. The ResNet50 model, after fine-tuning, stood out with exemplary performance, achieving near-perfect accuracy. This high level of precision from ResNet50's fine-tuning underscores the potential for machine learning to significantly enhance the photo selection process, traditionally a time-consuming task requiring substantial manual effort.

In the broader context of digital photography and media management, the implications of these findings are profound. The integration of a machine learning model like the fine-tuned ResNet50 into the workflow of photo culling could revolutionize the process, drastically reducing the time and resources required for manual curation. Such technology can assist photographers and organizations in managing and utilizing visual assets more effectively, allowing for rapid sorting and selection based on quality and content criteria predefined in the training of the model.

Despite the successes, the study acknowledges limitations, including the potential for overfitting and the dependency on substantial and diverse datasets for training. The variability in performance across different experiments also suggests room for improvement in model architecture and hyperparameter optimization. Future work could explore advanced regularization techniques, data augmentation strategies, and even the adoption of more complex or novel machine learning models. Additionally, expanding the training dataset and incorporating user feedback loops could further refine the model's accuracy and utility in real-world applications.

Future Work

For further development, enhancing the model's ability to differentiate between shallow depth of field and actual blurriness presents a promising challenge. This would require a more nuanced understanding of focus cues within an image and might involve training on datasets specifically labeled for depth-of-field variations.

Incorporating more realistic camera shake data could improve the model's accuracy in distinguishing between true camera shake and intended motion blur. This could involve gathering a dataset of images taken under varied conditions of movement to better represent real-life scenarios.

The inclusion of eye blink detection and facial expression recognition is an intriguing area of research that could provide additional filters for photo culling. By recognizing subtle facial cues, the system could prioritize images where the subjects have their eyes open and are displaying desirable expressions.

To enhance the user experience, the implementation of a triage system that sorts images into "To be Deleted," "Must Haves," and "Maybes" would be valuable. This system would not only streamline the culling process but also provide a more nuanced categorization of images, which could be achieved by assigning priority labels during the training phase.

Lastly, incorporating compositional quality assessments, such as adherence to the rule of thirds and the golden ratio, would add another layer of sophistication to the model. This might involve more complex pattern recognition and aesthetic evaluation capabilities, potentially drawing from the rich literature in the field of computational creativity and computer vision.

References & Acknowledgments

Acknowledgments

The project's success was significantly bolstered by various resources and guidance. Platforms and tools like TensorFlow and the Intel® Distribution of OpenVINO™ toolkit played a pivotal role in developing and optimizing our convolutional neural network (CNN) models. TensorFlow, in particular, provided an essential framework for building and training our models, especially in the context of image classification tasks, as highlighted in online resources and tutorials from TensorFlow developers and Edureka.

The Intel® oneAPI DL Frame Developer Toolkit and the Intel® Distribution of OpenVINO™ toolkit were instrumental in optimizing the neural networks, reducing model sizes, and improving latency with minimal accuracy degradation. These tools streamlined the designing, training, and validation of neural networks, contributing significantly to the project's progress.

In terms of learning and development, online courses and tutorials such as those offered by DeepLearning.AI on Coursera, Data Flair, DataCamp, and Kaggle provided valuable insights into the intricacies of CNNs and their application in computer vision. These resources were crucial in understanding how to implement and leverage CNNs effectively for our project's objectives, especially in the realm of photo culling and image recognition.

References

- TensorFlow developers' tutorial on building CNNs for image classification: This provided foundational knowledge on CNN architecture and its application in image classification tasks (TensorFlow).

- Edureka's tutorial on CNNs in Python using TensorFlow: Offered a practical guide on developing an image classifier and understanding the architecture behind CNNs (Edureka).

- DeepLearning.AI's TensorFlow Developer Professional Certificate on Coursera: Advanced techniques to improve computer vision models were learned here, including working with real-world images in different shapes and sizes (Coursera).

- Data Flair's tutorial on CNNs: Helped in understanding the broader implications of CNNs in the AI industry and their future shaping of various industries (DataFlair).

- Kaggle's practical course on CNNs: Assisted in learning hands-on implementation of CNNs using Keras, crucial for practical application in our project (Kaggle).

Each of these resources played a crucial role in the successful development and implementation of our project, providing both theoretical knowledge and practical skills essential for navigating the complexities of deep learning and computer vision.