TESTING TRANSFER LEARNING FOR IMAGE CLASIFICATION

by

Abhijith Dameruppala

Abstract

This project investigates the efficiency of transfer learning with datasets in the context of image classification. Utilizing a custom dataset, we investigate the minimal number of image samples required to achieve a predetermined level of accuracy across various pre-trained models: VGG-16, ResNet-50, MobileNet, and EfficientNet. This research not only addresses the practical constraints of data collection in machine learning projects but also explores the adaptability and efficiency of transfer learning across different models. The core methodology of this project involves the careful curation of a unique dataset comprising images of smartphones and remotes, characterized by their diversity in design, orientation, and context. Through this investigation, the project seeks to advance the understanding of transfer learning's limits and possibilities, providing a foundational framework for future research in the field.

Index Terms—**Data Augmentation, Dataset Curation, Deep Learning Models, EfficientNet, Feature Extraction, Image Classification, Model Fine-Tuning, MobileNet, Remote Controls, ResNet50, Smartphones, Transfer Learning, VGG-16.**

I. INTRODUCTION

Imagine trying to explain to a friend over the phone which remote on the coffee table turns on the TV, and which one is your phone lying face down. It sounds simple, but visually, these objects share a lot of similarities—flat, rectangular, and filled with buttons. This everyday confusion sets the stage for our project, diving into the nuanced world of image classification where we tackle distinguishing between smartphones and remote controls using the magic of machine learning.

At the heart of our exploration is a technique known as transfer learning. It's a bit like teaching an experienced chess player how to play checkers. They don't start from scratch; instead, they adapt their rich understanding of board games to master a new game more quickly. Similarly, transfer learning allows us to take a model that's already great at recognizing objects and fine-tune it to become an expert in spotting the differences between our two chosen objects.

But why smartphones and remotes? Apart from them often being mistaken for one another, they represent a broader challenge in machine learning: making accurate calls when data is scarce. Whether it's in healthcare, where each data point is precious, or in environmental science, where rare species are hard to catch on camera, the need to learn a lot from a little is everywhere.

The dataset we built will be used for testing out some of the smartest models such as VGG-16, ResNet-50, MobileNet, and EfficientNet. Through this project, we aim to shed light on the efficiency of transfer learning and how it might be the best in tackling data scarcity across various fields.

The project provides a comprehensive exploration of transfer learning for image classification using a curated dataset of smartphones and remote controls. After a literature review that shows the integration of deep and traditional features for enhanced accuracy, the report details dataset preparation and model selection. It then elaborates on the methodology of applying transfer learning strategies, data augmentation, and model optimization. The results section evaluates model performance across various training sizes, comparing pretrained and baseline models. The paper concludes with ethical considerations, stressing the importance of responsible AI use, and includes a robust set of references for further reading.

II. LITERATURE REVIEW

The study "Bansal, M., Kumar, M., Sachdeva, M. et al. Transfer learning for image classification using VGG19: Caltech-101 image data set. J Ambient Intel Human Computer 14, 3609–3620 (2023). https://doi.org/10.1007/s12652-021-03488-z" utilizes the VGG19 model to extract deep features, which are then combined with features derived from traditional image processing techniques such as SIFT, SURF, ORB, and the Shi-Tomasi corner detector. This hybrid approach allows the model to leverage the strengths of deep learning's hierarchical feature learning and the specific pattern recognition capabilities of handcrafted features. The combination has proven to be more effective than using either approach alone, as evidenced by the high accuracy achieved on the Caltech-101 dataset. This suggests that for projects requiring differentiation among detailed and diverse image categories, integrating deep and traditional features can provide a more nuanced understanding of image content, leading to superior classification results. For our project, this approach could be particularly beneficial. By combining deep features that capture hierarchical patterns with handcrafted features that capture specific textures and shapes, our model could achieve greater accuracy and robustness in distinguishing between smartphones and remotes, which may have subtle visual differences.

Other papers such as " Kim HE, Cosa-Linan A, Santhanam N, Jannesari M, Maros ME, Ganslandt T. Transfer learning for medical image classification: a literature review. BMC Med Imaging. 2022 Apr 13;22(1):69. doi: 10.1186/s12880-022-00793-7. PMID: 35418051; PMCID: PMC9007400. " and " J. Plested and T. Gedeon, "Deep transfer learning for image classification: a survey," 2022, arXiv:2205.09904," focus on how establish deep learning models, particularly Inception and ResNet, which are used in transfer learning settings to address challenges posed by specific datasets, especially in medical imaging and other specialized fields. These models are highlighted for their efficiency as feature extractors, which is crucial when dealing with limited datasets or when computational resources are constrained. The reviews underscore the versatility of transfer learning approaches, such as feature extraction and fine-tuning, to adapt pre-trained networks to new tasks effectively. This adaptability is particularly relevant

when dealing with unique dataset characteristics, such as varying image modalities, subjects, or dataset sizes. For projects like distinguishing between smartphones and remote controls, these insights suggest that starting with a pre-trained model and then tuning it to the specific textures, shapes, and sizes found in the training data can optimize performance, ensuring the model is not only accurate but also efficient in handling real-world variability in the input data. Both reviews highlight the effectiveness of transfer learning in scenarios where data may be limited or highly specialized, like medical imaging. Applying these strategies to our project, using models like ResNet or Inception as base models, could improve feature extraction capabilities significantly. Starting with a pre-trained model and adapting it through fine-tuning could allow your model to better learn the specific features of smartphones and remote controls, leveraging learned knowledge from vast and varied datasets these models were originally trained on. These surveys provide a comprehensive overview of transfer learning applications in image classification, emphasizing the use of deep models for robust feature extraction. For our project, this suggests that employing a strategy of feature extraction followed by fine-tuning would be ideal. This dual approach helps in utilizing pre-trained neural architectures to capture generic features and then tuning them to home in on the specific details of your target classes, which in this case are different electronic devices.

III. METHODOLOGY

DATASET CREATION AND PREPROCESSING:

We gathered a diverse set of images for both smartphones and remote controls. This will include images from online repositories, and open-source datasets, such as Google's Open Images Dataset, Public Domain Pictures, Adobe Stock Free Collection, Wikimedia Commons. Preprocessing the dataset will include standardizing the image sizes, and performing data augmentation (rotation, flipping, scaling) to increase the robustness of the model.

SELECTON OF PRETRAINED MODELS:

Based on the literature review, and the nature of our project, we are considering the following models for their effectiveness in image classification tasks:

1. VGG16
2. ResNet-50
3. EfficientNet
4. MobileNet

VGG16

The VGG16 model was introduced by the Visual Graphics Group from Oxford, which is where it gets its name (VGG). It was a breakthrough model because of its simplicity and depth. The model has 16 layers that have weights; this number includes convolutional layers, fully connected layers, and the output layer.

It uses small 3x3 convolutional filters throughout the architecture and follows each set of convolutional layers with a max-pooling layer. One of the key contributions of the VGG16 model was to show that depth is a critical component for good performance in convolutional neural networks.

Following is the architecture of VGG16 model:

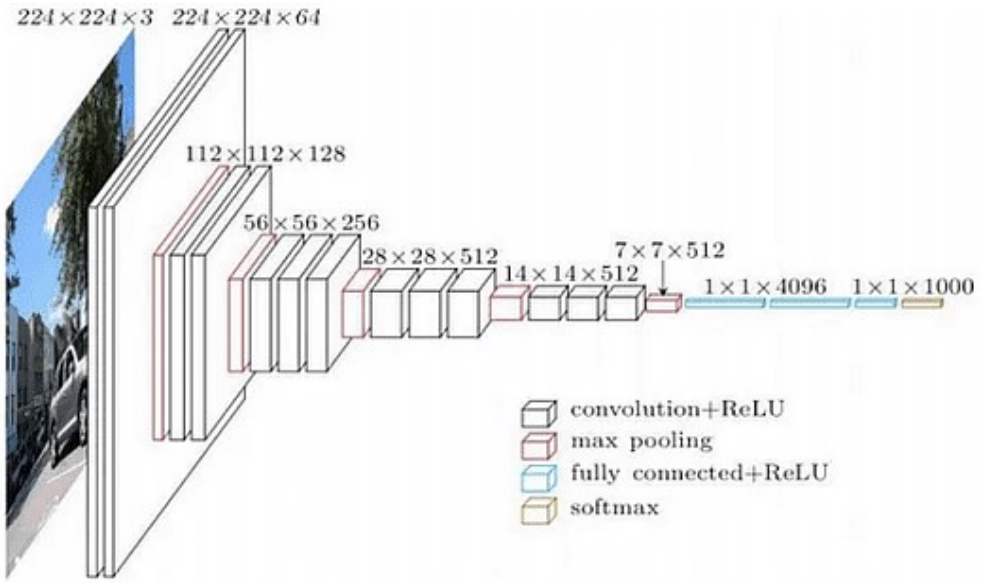


Fig 1: Architecture of VGG16 model

RESNET50:

The ResNet50 model, introduced by Microsoft Research, is part of a larger set of models known as Residual Networks. The "50" refers to the fact that it is a 50-layer deep network. The defining characteristic of ResNet is the introduction of what's called a "skip connection" or "shortcut connection," which allows the output of one layer to bypass intermediate layers and be added directly to the output of another layer further down the network. This helps to address the vanishing gradient problem that can occur with very deep networks by allowing gradients to flow through the network more easily during training. As a result, ResNet makes it feasible to train much deeper networks than was previously practical.

Following is the architecture of RESNET50 model:

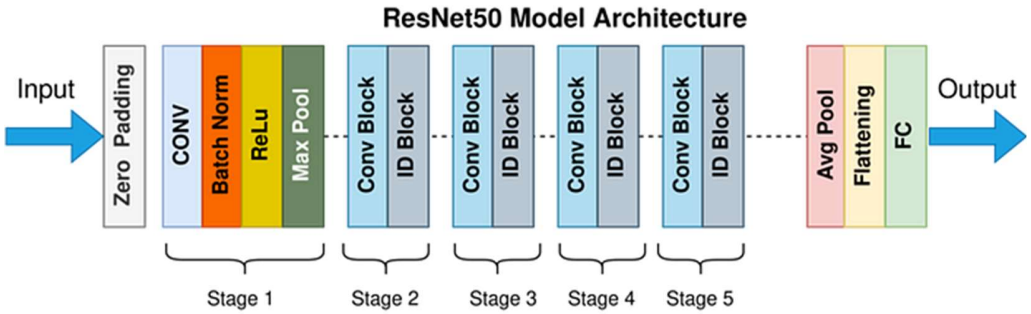


Fig 2: Architecture of Resnet50 model

EfficientNet:

This model is known as the model with the most powerful CNN architecture. Because it uses a technique called compound coefficient to scale up models in a simple but effective manner. Using the scaling method and AutoML, the authors of efficient developed seven models of various dimensions, which surpassed the state-of-the-art accuracy of most convolutional neural networks, and with much better efficiency.

Following is an image that shows the different scaling methods vs compound scaling:

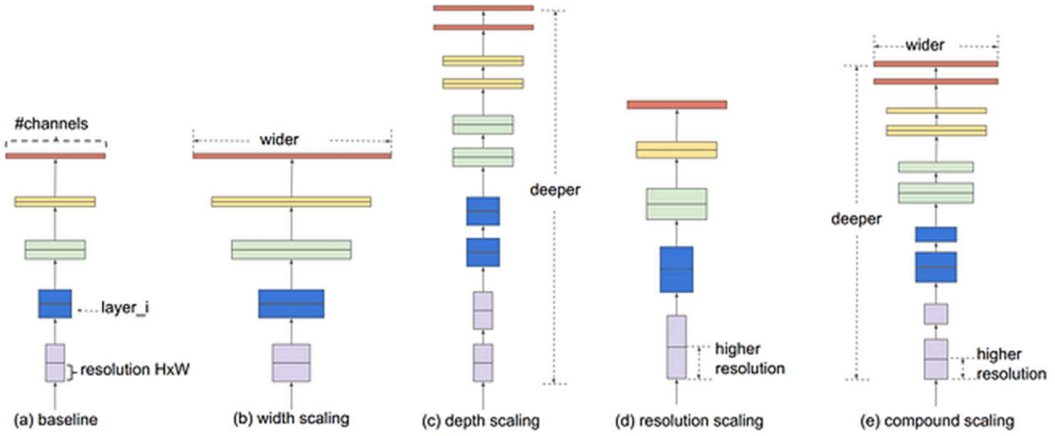


Fig 3: Architecture of EfficientNet model

MobileNet:

MobileNet is TensorFlow's first mobile computer vision model. It uses depthwise separable convolutions to significantly reduce the number of parameters compared to other networks with regular convolutions and the same depth in the nets. This results in lightweight deep neural networks.

Following is the architecture of MobileNetV1 model:

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Fig 4: Architecture of MobileNet model

TRANSFER LEARNING STRATEGY:

- We first initialized a new Sequential model, which (in Keras) allows us to build a model layer by layer in a step-by-step fashion. Each layer has exactly one input tensor and one output tensor.
- We used the convolutional base of the pre-trained models to extract features from our dataset, keeping the model's pre-trained weights frozen. We then flattened the output of the convolutional base into a single long vector.
- We then added a dense fully connected layer with 256 neurons to the network. The activation function used for all the models is “relu” (rectified linear unit), which introduces non-linearity into the model allowing it to learn more complex patterns in the data.
- We then used Dropout, which is a regularization technique used to prevent overfitting in neural networks. This layer randomly sets the output features of 50% (dropout rate = 0.5) of the neurons in the previous layer to zero during training. This makes the model robust.
- And finally, a dense layer with a single neuron is added. This layer uses the “sigmoid” activation function, which is commonly used for binary classification. The sigmoid function outputs a probability indicating how likely is that the input belongs to one class versus another.

MODEL TRAINING:

We split our dataset into training, and validation sets to evaluate the model's performance. The training sets contain several image samples ranging from 20 images to 1200 images. The validation set remains constant size at 300 images for all the models.

EVALUATION AND TESTING:

We evaluate the model's performance on the validation set during training to fine-tune hyperparameters and training configurations. And upon satisfactory validation performance, assess the final model on the test set to reported validation accuracy.

MODEL OPTIMIZATION:

Based on the initial results, we iteratively adjusted the model architecture, training process, or transfer learning strategy to improve performance. This involved experimenting with different models, adjusting the number of layers unfrozen during fine-tuning, or varying the data augmentation techniques used.

IV. Materials And Techniques

ENVIRONMENT SETUP:

To gather dataset from previously mentioned sources, we designed a web scraper, which extracts data from websites. The scraper functions by making HTTP requests to the specified URLs to retrieve web pages and then parse data from the HTML content of those pages. The primary purpose of us using that scraper function is to gather the sources of images from the URLs, and then download the images from the original sources into the user's computer.

Python 3.8 and the following libraries and packages are prerequisites for the web scraper to function:

Table 01: Libraries and packages used for web scraper.

OS	To interact with the operating system for file path operations.
Requests	To make HTTP requests to web pages.
BeautifulSoup	To parse and navigate the HTML tree of the web pages.
Urljoin	To resolve relative URL paths to absolute URLs.
Unquote	To decode URL-encoded data to string characters.

The web scraper's algorithm performs the following steps:

- The function ensures the existence of a directory for saving the downloaded images. If the directory does not exist, it is created.
- Sends an HTTP GET request to the target URL and retrieves the HTML content of the page, which is then parsed to isolate the image elements.
- Searches for <img> tags within the HTML. It employs a loop that continues until it either reaches 100 images or exhausts the list of tags.
- For each image tag found, the script checks for a src attribute and validates that the URL points to a JPG file. This filtration ensures consistency in image format and excludes non-relevant media types.

- Utilizes urljoin to construct the full URL of the image if the src attribute contains a relative path.
- Downloads each image using the resolved URL and writes the binary content to a file within the specified local directory.
- Provides feedback via the console, printing out confirmation messages when images are successfully downloaded or logging errors if the process encounters issues.

An integral part of the scraper's design was ensuring compliance with copyright and usage rights. It is essential to note that all sources for image collection have been carefully selected based on their open-source nature and the explicit permission for images to be used for both commercial and personal purposes. This ensures that the dataset we compile is legally sound for use in developing and training machine learning models, as well as for any potential commercial applications that may arise from this project.

The sources include:

- Google's Open Images Dataset: A collection from Google that is specifically available for academic and research purposes.
- Public Domain Pictures: A repository of images that are explicitly placed in the public domain, allowing free use.
- Adobe Stock Free Collection: Offers a range of images that are royalty-free and available for free.
- Wikimedia Commons: A vast collection of media files that are freely usable and part of the public domain or licensed under free licenses.

To demonstrate the collected images into a visual presentation, we utilized the Pillow library from Python’s PIL (Python Imaging Library). This library generates a collage of images, and it enables a more efficient visual presentation of the dataset.

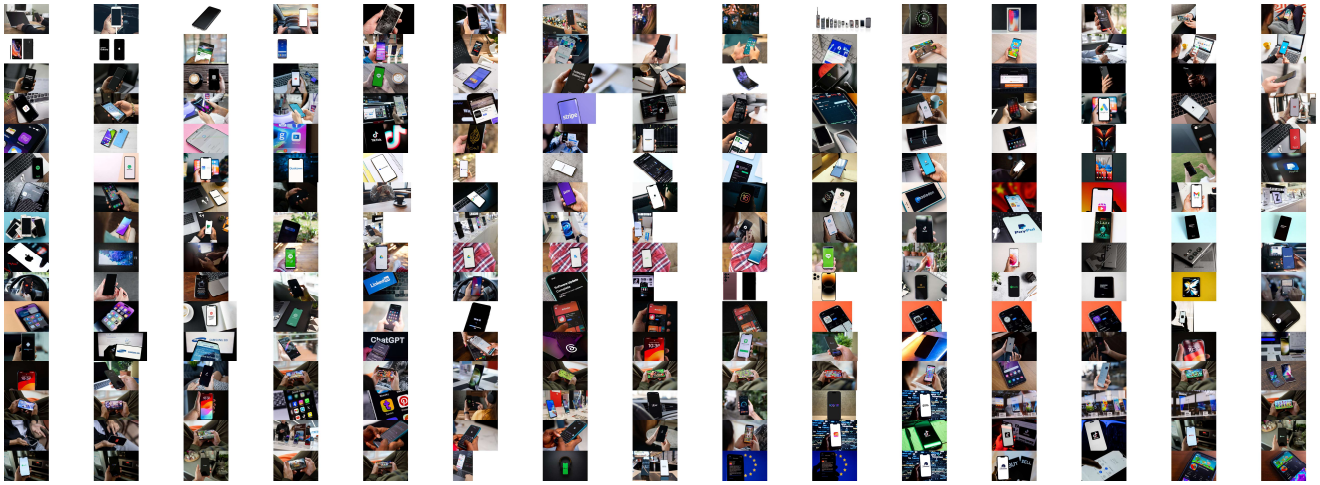


Fig 5: Collage representation of Mobile image samples from the dataset

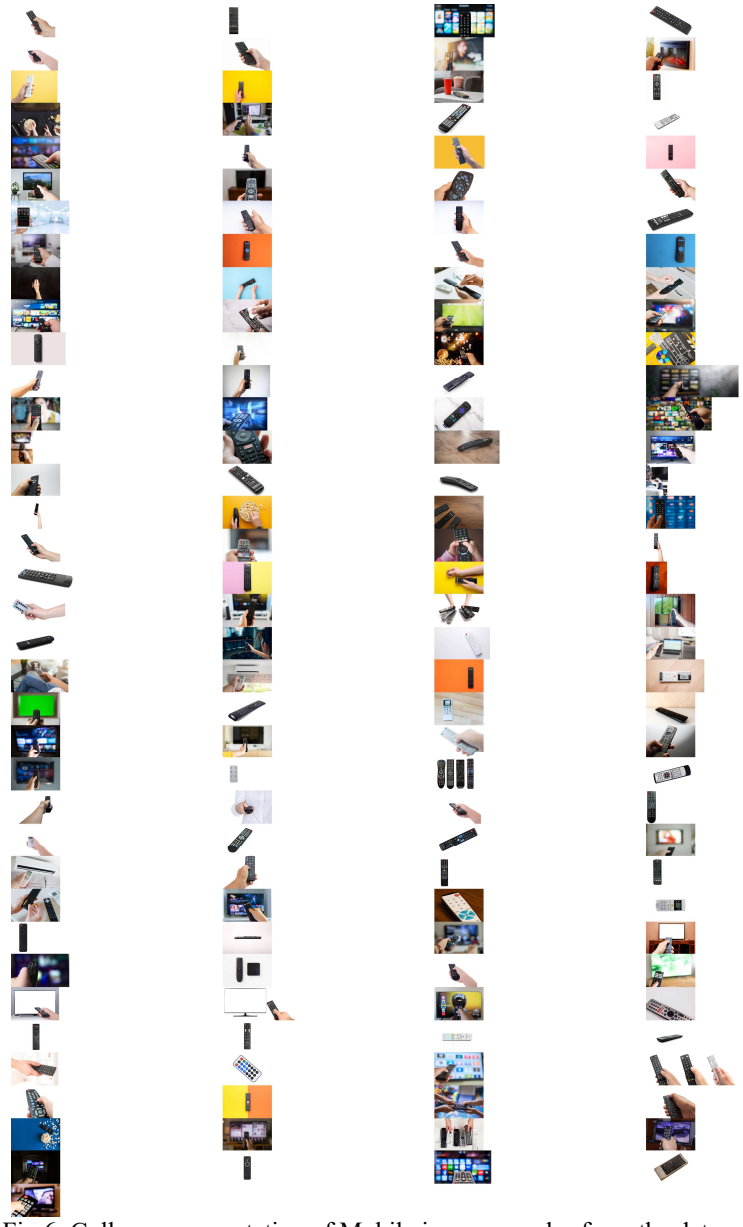


Fig 6: Collage representation of Mobile image samples from the dataset

LABELLING IMAGES

To make accessing and managing the image datasets, we have decided to rename all the images in the datasets, and label them to ensure consistency and make it easier to manipulate the data during training and testing.

Procedure:

- Two separate directories were prepared: one for mobile images and another for remote images.
- A python function is prepared to traverse through each images in these directories and used a loop to rename each image according to its category followed by an integer value. (such as mobile1, mobile2, remote1, remote2 etc.,)
- The renaming was automated to avoid manual errors, and ensure that each image receives a unique identifier, making them easily distinguishable and accessible.

DATA AUGMENTATION

Given the initial dataset's size, data augmentation helps to enhance the model's ability to classify the images. This process artificially increases the size and diversity of our dataset by applying random but realistic transformations to the training images.

Augmentation Techniques:

- Rotation: Images are randomly rotated by up to 40 degrees to simulate different orientations.
- Width and Height Shifts: Images are shifted horizontally and vertically by up to 20% of their dimensions to mimic off-center placements.
- Shear Transformation: Shear mapping was applied to images to simulate a tilting effect that occurs in some real-life scenarios.
- Horizontal Flip: Images are flipped horizontally to duplicate conditions where objects appear in reversed orientations.

IMPLEMENTATION STRATEGY FOR DATA AUGMENTATION

- Configured the “Image Data Generator” class from Keras with the above transformations.
- The generator was set to process images in batches, ensuring that each original image was used once per cycle to generate one new augmented image.
- The augmentation cycle was repeated until the total count of images reached the set target of 1500 images per category. This repetition was controlled by monitoring the output directory until it contained the desired number of images.

To ensure even distribution of augmentation across all images a control condition was incorporated which is to count the total images already generated and to stop further augmentation once the target (which is 1500 images per class) was reached. And the following table shows the number of images before and after augmentation techniques are applied:

Table 2: Change in number of images before and after Augmentation.

	BEFORE AUGMENTATION	AFTER AUGMENTATION
MOBILE IMAGES	240	1500
REMOTE IMAGES	145	1500

The following tables are the samples from both the classes after images have undergone augmentation.

Fig 7: Augmented image samples generated from Mobile images class.

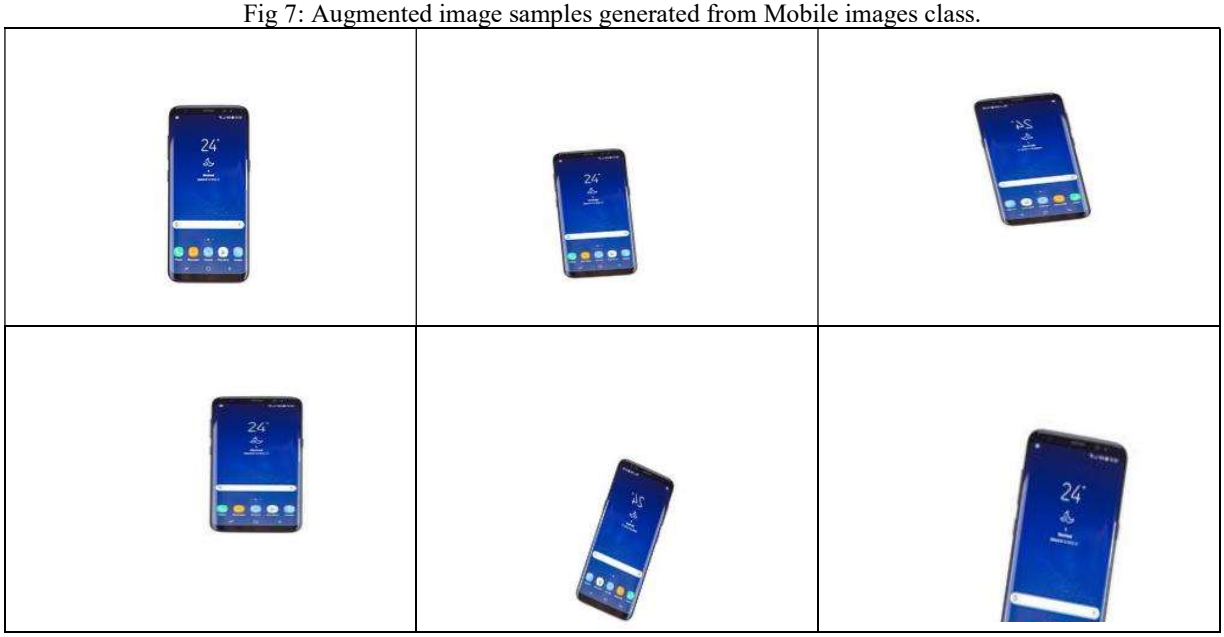
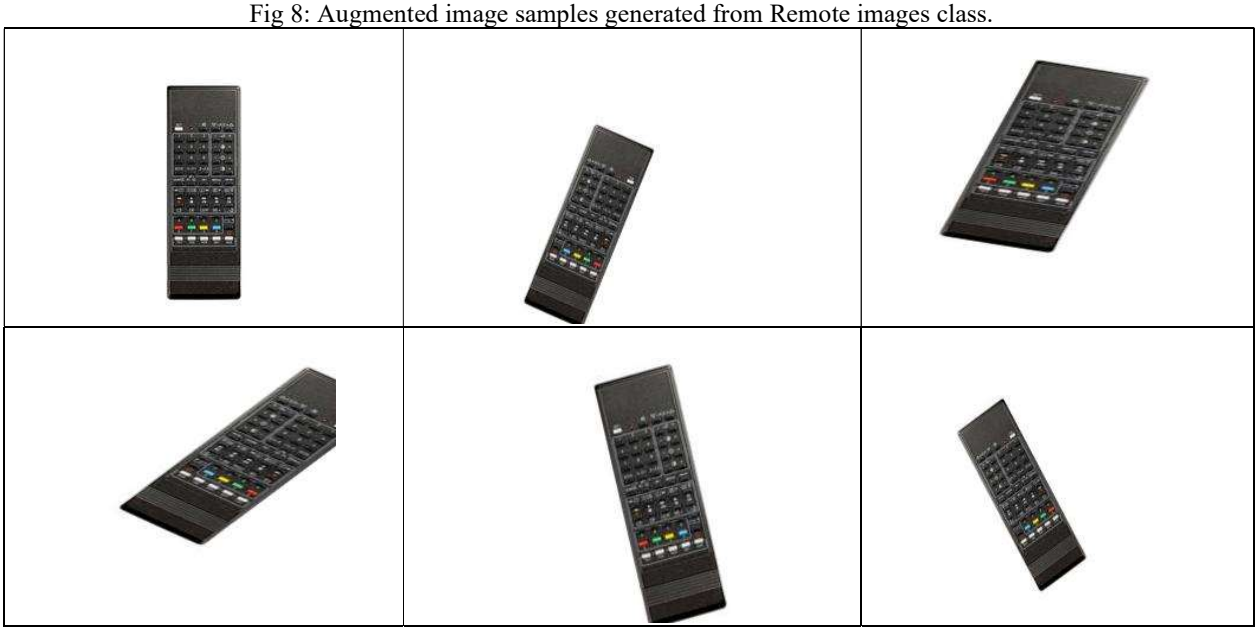


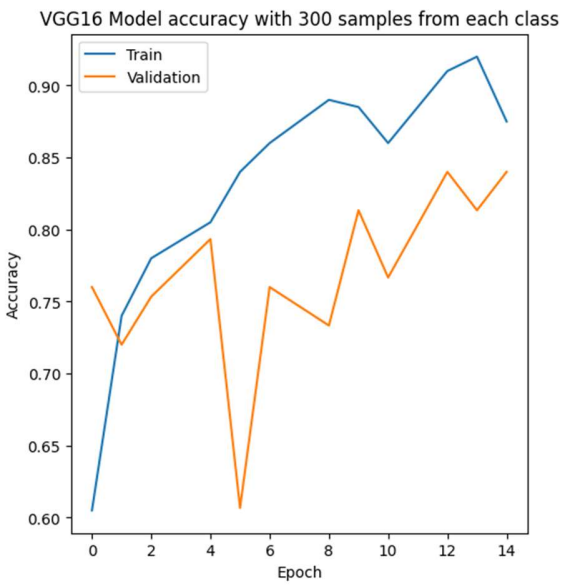
Fig 8: Augmented image samples generated from Remote images class.

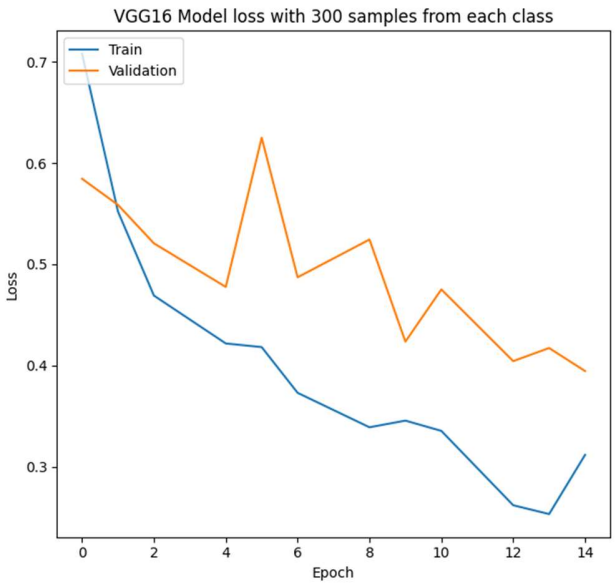


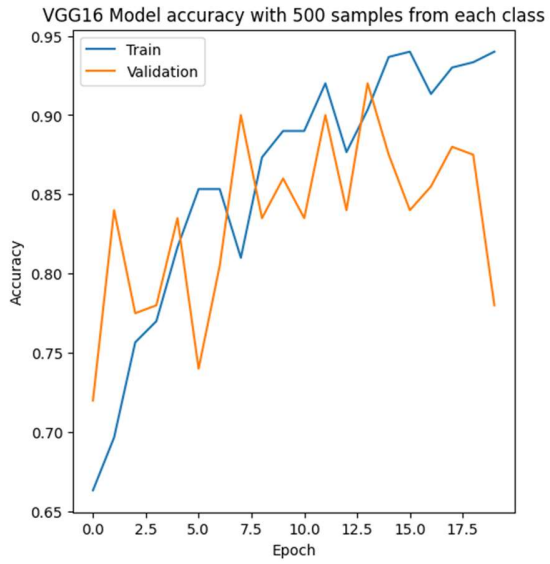
V. IMPLEMENTATION RESULTS

VGG16 RESULTS:

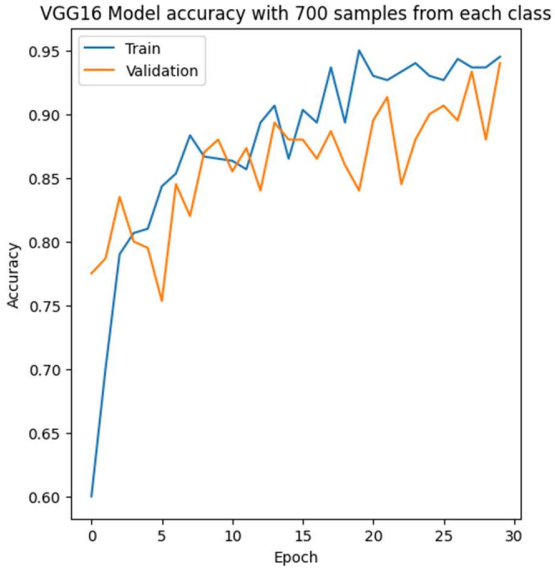
The following are the accuracy and loss plots generated by training with varying sizes of image samples:



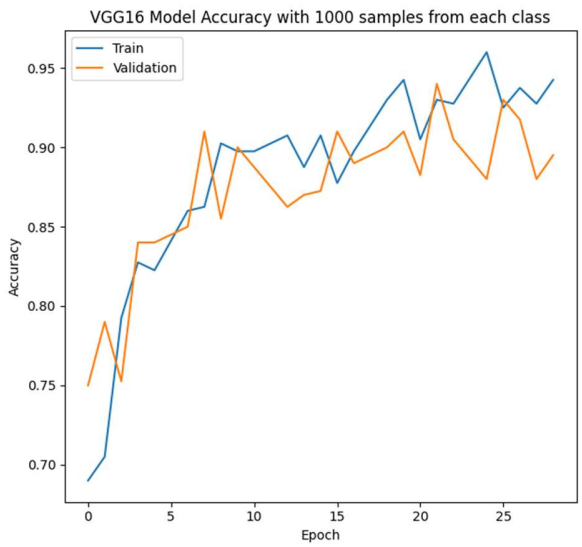



VGG16 Model accuracy with 500 samples from each class
Train
Validation
Accuracy
Epoch


VGG16 Model loss with 500 samples from each class
Train
Validation
Loss
Epoch

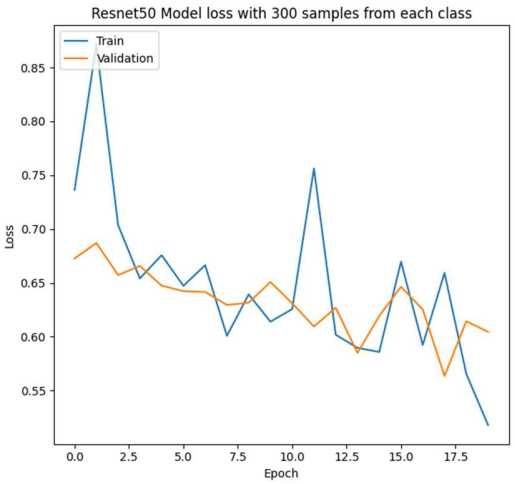

VGG16 Model accuracy with 700 samples from each class
Train
Validation
Accuracy
Epoch

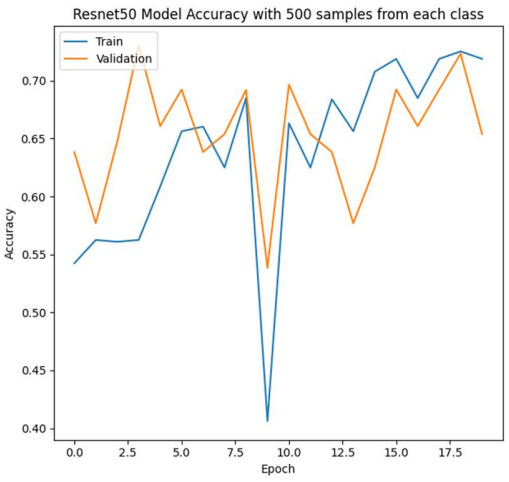

VGG16 Model loss with 700 samples from each class
Train
Validation
Loss
Epoch

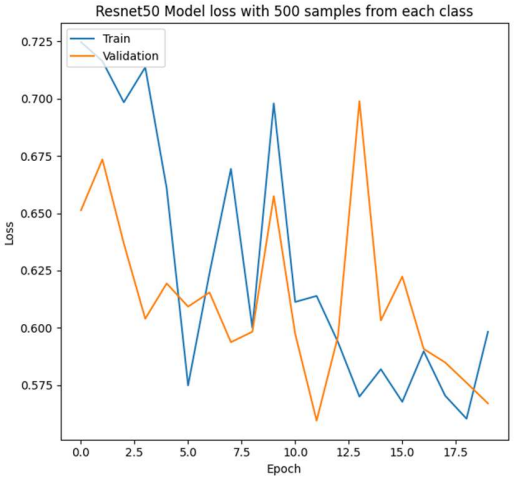

VGG16 Model Accuracy with 1000 samples from each class
Train
Validation
Accuracy
Epoch


VGG16 Model loss with 1000 samples from each class
Train
Validation
Loss
Epoch

RESNET50 RESULTS:







We trained all the models with varying training sizes ranging from 20 samples to 1200 like this: [20, 50, 90, 120, 180, 250, 300, 500, 700, 1000, 1200]. The amount of validation samples is fixed at 300 samples.

Following are the plots of each of the model's performance over the training datasets.

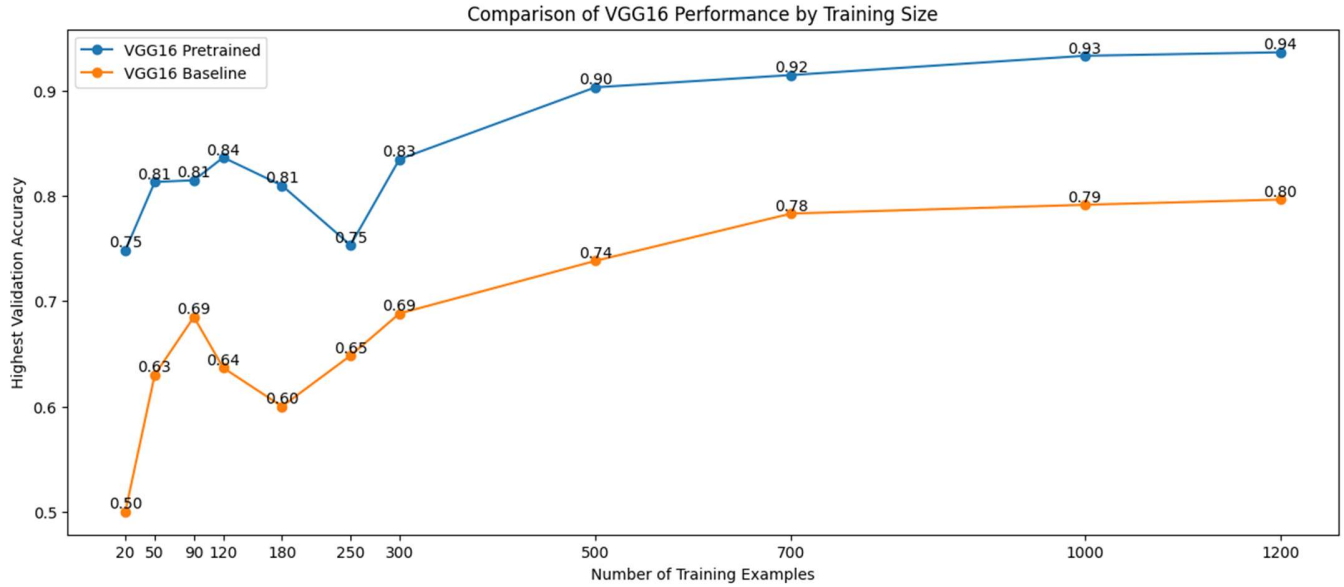


Fig 9: Performance of VGG16 over the training datasets

The following table represents performance of VGG16 model over the training datasets.

Table 03: Performance Results of VGG16 model

Number of training samples	Pretrained model Results	Baseline model Results
20	0.75	0.5
50	0.81	0.63
90	0.81	0.69
120	0.84	0.64
180	0.81	0.60
250	0.75	0.65
300	0.83	0.69
500	0.9	0.74
700	0.92	0.78
1000	0.93	0.79
1200	0.94	0.8

VGG16 shows a strong positive trend in validation accuracy as training size increases. The pretrained model is particularly robust across different training sizes since it starts at a higher accuracy and maintains a lead throughout the training sizes. We also observe a significant increase around the 500 training examples mark for the pretrained model. Baseline performance improves steadily but does not reach the heights of the pretrained model. VGG16 achieved the target accuracy of 90% at 500 training samples. And it took the model took 709 seconds to train.

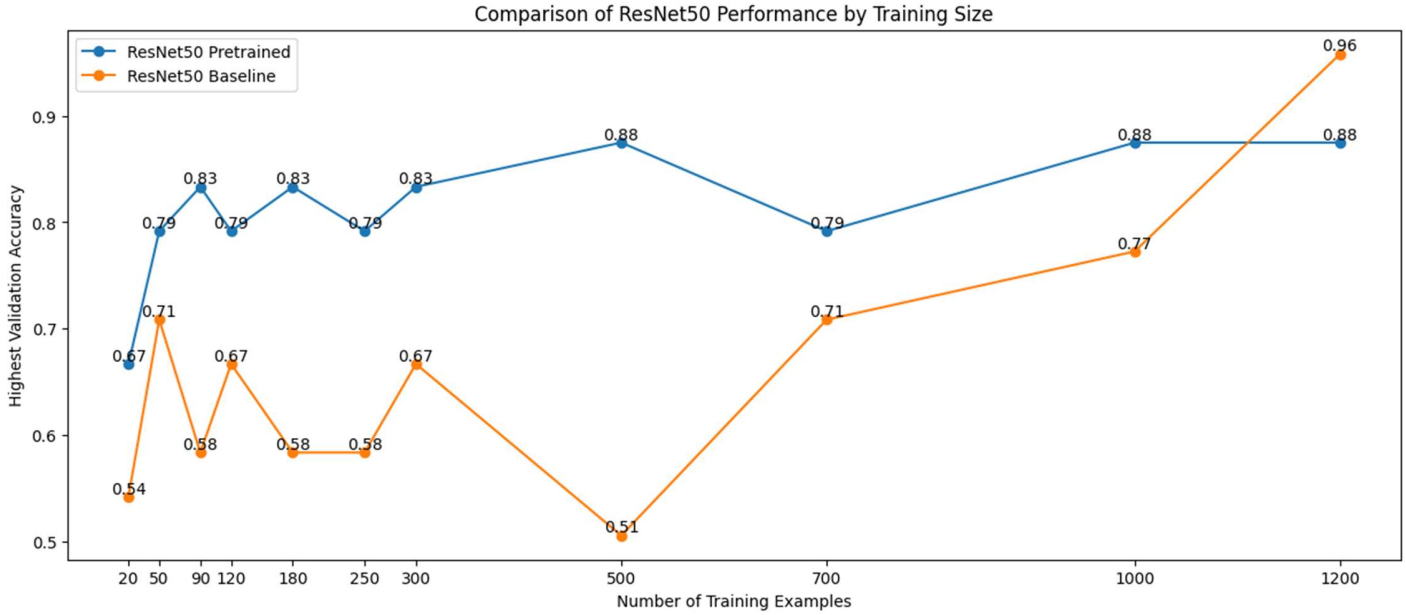


Fig 10: Performance of Resnet50 over the training datasets

The following table represents performance of Resnet50 model over the training datasets.

Table 04: Performance Results of Resnet50model

Number of training samples	Pretrained model Results	Baseline model Results
20	0.67	0.54
50	0.79	0.71
90	0.83	0.58
120	0.79	0.67
180	0.83	0.58
250	0.79	0.58

300	0.83	0.67
500	0.88	0.51
700	0.79	0.71
1000	0.88	0.77
1200	0.88	0.96

Both ResNet50 configurations show an increasing trend in accuracy as training sizes increase. Notably, the pretrained version consistently outperforms the baseline across all training sizes. We observe consistent improvement with an increase in training size for the pretrained model, peaking at 0.96, and the baseline model shows a significant improvement after 500 training examples. Technically we never observe a validation accuracy above 90%, but this model is the most consistent model with validation accuracy. The pretrained model peaked at 88% on 500 samples, but the baseline model crossed the target accuracy on 1200 training samples.

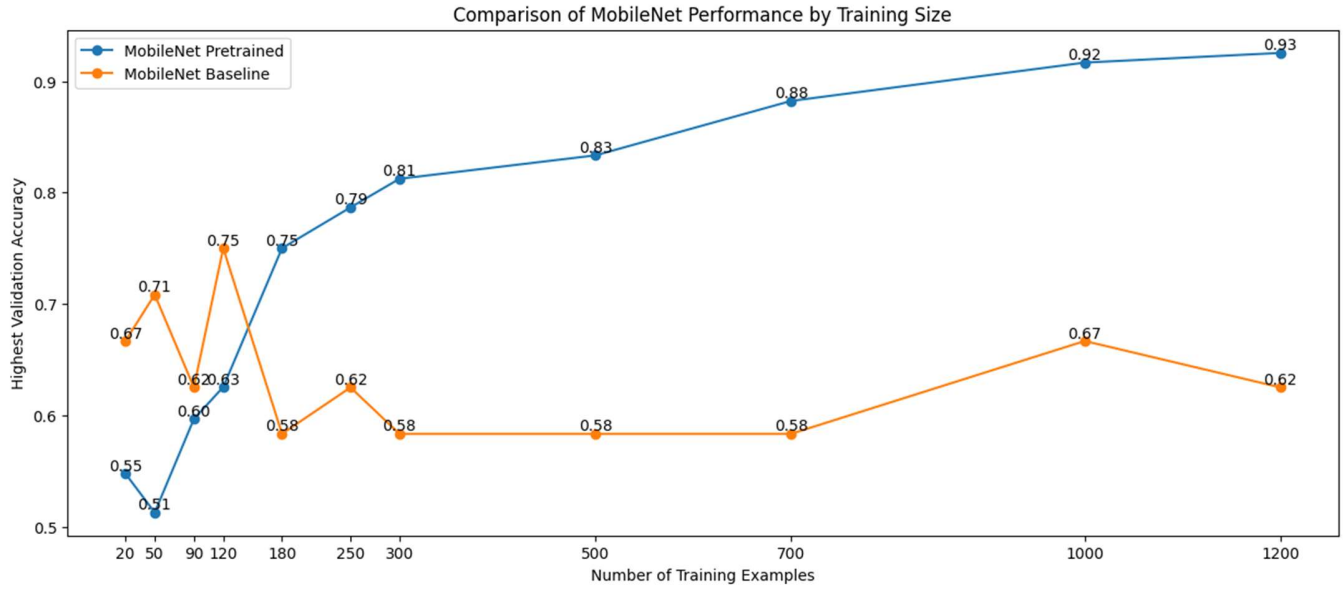


Fig 11: Performance of MobileNet over the training datasets

Table 5: Performance Results of MobileNet model

Number of training samples	Pretrained model Results	Baseline model Results
20	0.75	0.67
50	9.83	0.71
90	0.92	0.62
120	0.92	0.75
180	0.93	0.58
250	0.96	0.62
300	1	0.58
500	1	0.58
700	1	0.58
1000	1	0.67
1200	1	0.62

MobileNet pretrained shows consistent increase in validation accuracy with training sizes. The baseline configuration demonstrates modest improvements as the training size increases but remains significantly lower than the pretrained version. We observe pretrained MobileNet reaches a maximum accuracy of 93% with 1200 training examples. And our target accuracy of 90% is reached with 1000 training samples. Baseline MobileNet shows a peak of 75% accuracy at 120 training examples but then stabilizes around 0.58 before a slight increase at 1000 samples. This model struggled with differentiating from smaller sample sizes, but the model picked up overtime, and achieved 92% accuracy at 1000 training samples. And it took 235 seconds for the model to train (from Table 5) since its architecture is lightweight.

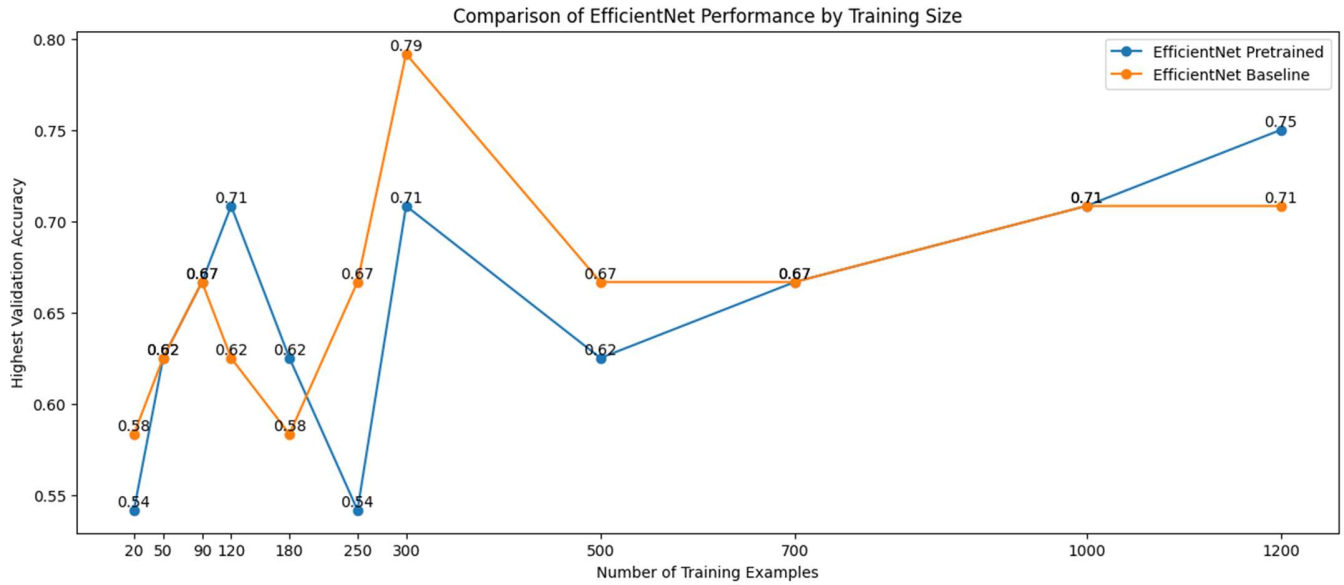


Fig 12: Performance of EfficientNet over the training datasets

The following table represents the performance of the EfficientNet model over the training datasets.

Table 6: Performance Results of EfficientNet model

Number of training samples	Pretrained model Results	Baseline model Results
20	0.54	0.58
50	0.62	0.62
90	0.67	0.67
120	0.71	0.62
180	0.62	0.58
250	0.54	0.67
300	0.71	0.79
500	0.62	0.67
700	0.67	0.67
1000	0.71	0.71
1200	0.75	0.71

The performance of EfficientNet shows notable variability with different training sizes, especially in the pretrained configuration, where the accuracy shows significant drops and spikes as training size increases. The baseline configuration is more consistent but generally lower in accuracy than the pretrained mode. We observed that at lower training sizes (up to 300), the accuracy is generally unstable. And there's a sharp drop and subsequent recovery around the 300 training examples mark. Both the configurations seem to stabilize and converge slightly after 700 training examples, but the pretrained version still performs better. This model is the most inconsistent of all the models, with high fluctuations in validation accuracy. This model never achieved target accuracy, and peaked at 75% with 1200 training samples, and it took 554 seconds to train (from Table 5).

We recorded the times taken for all models to train, along with steps per each epoch. While the number of epochs remain constant for all the models, the steps per epoch are calculated as the ratio between the number of train samples to the batch size. Table 05 contains that data.

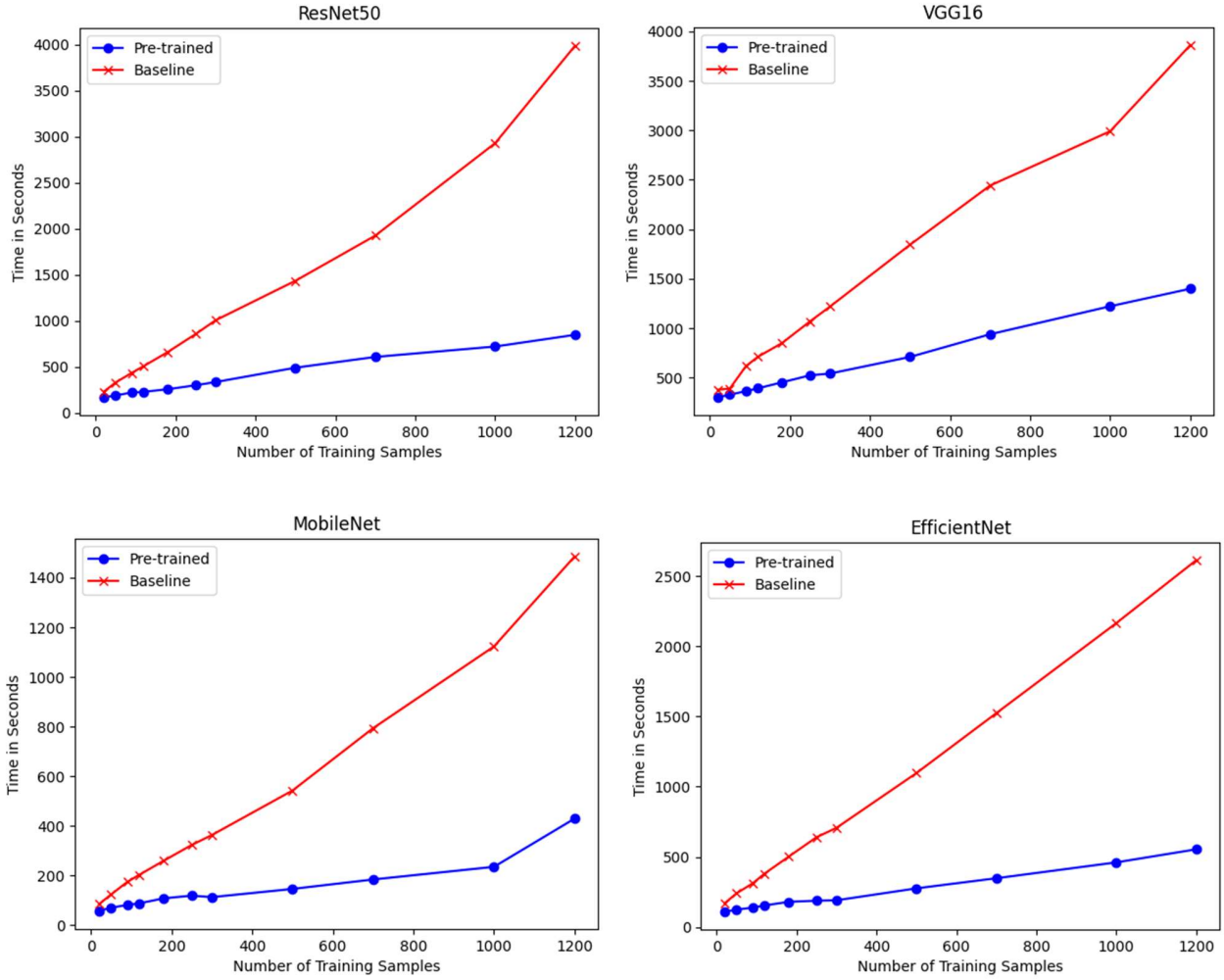
Table 07: Time taken and steps per epoch of all the models.

NO. OF TRAINING SAMPLES	RESNET50			VGG16			MOBILENET			EFFICIENTNET		
	PRE	BASE	STEPS	PRE	BASE	STEPS	PRE	BASE	STEPS	PRE	BASE	STEPS
20	167	228	1	302	378	2	57	85	1	110	169	1
50	188	329	3	328	389	5	71	124	3	124	241	3
90	219	432	5	363	615	9	81	174	5	139	311	5
120	227	506	7	392	710	12	88	203	7	154	381	7
180	255	658	11	452	848	18	108	259	11	180	501	11
250	298	854	15	523	1067	25	119	323	15	188	640	15
300	333	1004	18	541	1217	30	113	363	18	191	707	18
500	489	1433	31	709	1842	50	146	542	31	276	1097	31
700	606	1924	43	938	2440	70	184	793	43	348	1524	43

1000	719	2928	62	1221	2989	100	235	1122	62	461	2165	62
1200	846	3986	75	1398	3859	120	429	1484	75	554	2613	75

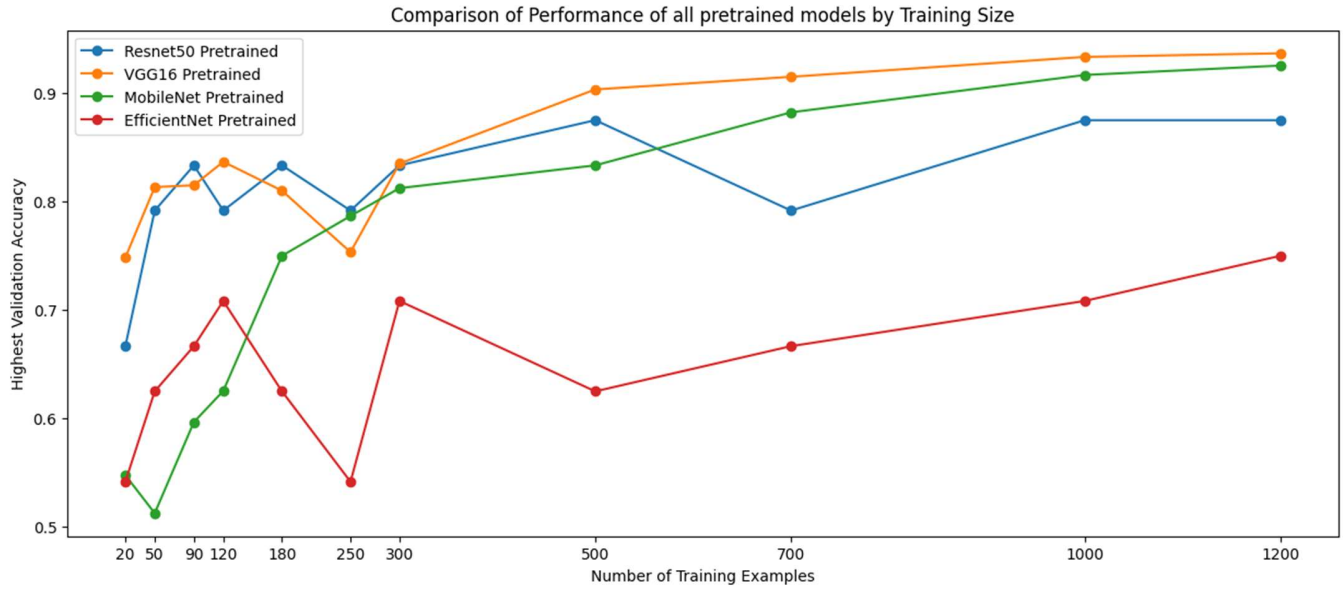
The following plots reflect the time taken for each of the models to run (in seconds) for a different number of training samples.

Fig 13: Plots of time taken for models to run



We consistently observe that Baseline models took considerably more time than the pretrained models. Following is a line plot that includes all the pretrained models plotted with validation accuracy of their best model to the number of training samples.

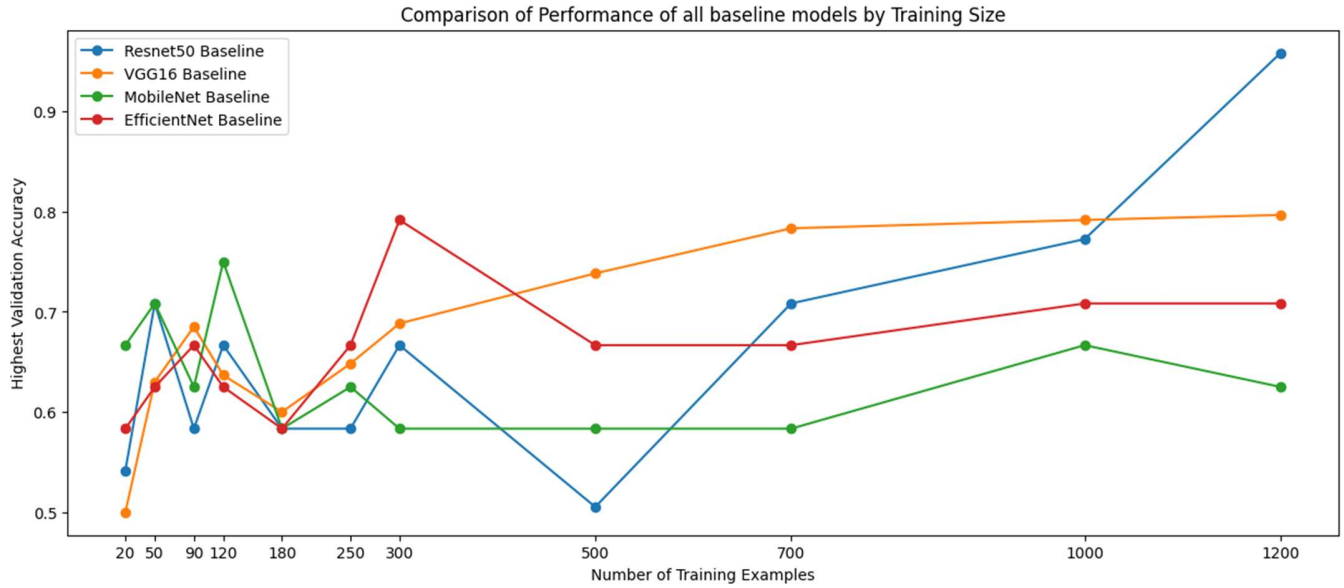
Fig 14: Comparison of all pretrained models



Even though Resnet50 did not reach the target accuracy, it was close at 88%. While the VGG16 took 709 seconds for 90% accuracy, the time taken to train the Resnet50 model is significantly faster at 489 seconds. Of all the pretrained models, VGG16 achieve our target accuracy of 90% the fastest at 500 samples.

The following plot compares all the base line models' best validation accuracy with the sample sizes.

Fig 15: Comparison of all baseline models



Resnet50 is the only model that crossed the target accuracy, at 1200 samples. And it took 3986 seconds to train.

VI. CONCLUSION

Our project, testing transfer learning for image classification using the VGG16, ResNet50, MobileNet, and EfficientNet models on a curated dataset of smartphones and remote controls has provided significant insights. We have successfully trained Resnet50, VGG16, MobileNet, EfficientNet models and documented all the results obtained from them. Of all these models, VGG16, and MobileNet have achieved the target accuracy of 90% at 500, and 1000 training samples, respectively. The minimum number of samples required to achieve 90% accuracy are 500 images. While the VGG16 took 709 seconds for 90% accuracy, the time taken to train the Resnet50 model was significantly faster at 489 seconds and the Resnet50 model peaked at 88%.

VGG16 and MobileNet excelled in achieving our target accuracy of 90% with relatively fewer training samples, highlighting their suitability for tasks requiring high efficiency with limited data. ResNet50, although it did not reach the 90% mark, demonstrated consistent performance across varying training sizes, making it a reliable choice for similar applications. For our dataset, we believe that the EfficientNet model is the least efficient one while VGG16 to be the most consistent model that reached our expectations. Resnet50 is another good choice for classification tasks, as it was the close second model.

VII. ETHICAL CONSIDERATIONS

This project underscores the necessity of ethical considerations in machine learning, particularly concerning data usage and model application. Ensuring the ethical sourcing of training data, respecting privacy, and acknowledging copyright are foundational. The potential for bias in AI systems, particularly in classifications, could possibly be influenced by the training data's diversity. As AI models like the ones tested in our study increasingly find applications in real-world scenarios, ensuring they do not perpetuate or amplify biases is crucial. The interpretability of such models is essential for trust and accountability in automated processes, prompting a need for transparent methodologies and clear communication of model capabilities and limitations to stakeholders.

REFERENCES

[1] "Open Images Dataset V5," Googleapis.com, 2019. [Online]. Available: https://storage.googleapis.com/openimages/web/index.html

[2] Adobe Stock, "Stock Photos, Royalty-Free Images, Graphics, Vectors & Videos," Adobe Stock, 2023. [Online]. Available: https://stock.adobe.com/uk/

[3] "Search Media - Wikimedia Commons," Commons.wikimedia.org. [Online]. Available: https://commons.wikimedia.org/wiki/Main_Page

[4] "Public Domain Pictures – Free Stock Photos," 2024. [Online]. Available: https://publicdomainpictures.net/en/index.php

[5] "Pillow — Pillow (PIL Fork) 6.2.1 Documentation," Readthedocs.io, 2011. [Online]. Available: https://pillow.readthedocs.io/en/stable/

[6] K. Reitz, "Requests: HTTP for Humans™ — Requests 2.27.1 Documentation," [Online]. Available: https://requests.readthedocs.io/en/latest/

[7] "Beautiful Soup Documentation — Beautiful Soup 4.4.0 Documentation," [Online]. Available: https://beautiful-soup-4.readthedocs.io/en/latest/

[8] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," CoRR, abs/1409.1556, 2014.

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770-778, doi: 10.1109/CVPR.2016.90.

[10] C. Shorten and T. M. Khoshgoftaar, "A survey on Image Data Augmentation for Deep Learning," J Big Data, vol. 6, no. 60, 2019. [Online]. Available: https://doi.org/10.1186/s40537-019-0197-0

[11] K. Weiss, T. M. Khoshgoftaar, and D. Wang, "A survey of transfer learning," J Big Data, vol. 3, no. 9, 2016. [Online]. Available: https://doi.org/10.1186/s40537-016-0043-6

[12] "Tf.keras.preprocessing.image.imagedatagenerator," tensorflow V2.16.1, TensorFlow. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator

[13] "Everything You Need to Know about VGG16," Medium, Sept. 23, 2021. [Online]. Available: https://medium.com/@mygreatlearning/everything-you-need-to-know-about-vgg16-7315defb5918

[14] "ResNet-50: The Basics and a Quick Tutorial," Datagen. [Online]. Available: https://www.datagen.tech/guides/computer-vision/resnet-50/

[15] A. Sarkar, "Understanding EfficientNet — the Most Powerful CNN Architecture," Medium, May 8, 2021. [Online]. Available: https://www.arjun-sarkar786.medium.com/understanding-efficientnet-the-most-powerful-cnn-architecture-eaeb40386fad

[16] T. Mostafid, "Overview of VGG16, ResNet50, Xception and MobileNet Neural Networks," Medium, Dec. 12, 2023. [Online]. Available: https://www.medium.com/@t.mostafid/overview-of-vgg16-xception-mobilenet-and-resnet50-neural-networks-c678e0c0ee85

[17] S. Pujara, "Image Classification with MobileNet | Built In," Builtin.com, Jan. 26, 2023. [Online]. Available: https://www.builtin.com/machine-learning/mobilenet

[18] "Papers with Code - EfficientNet Explained," Paperswithcode.com. [Online]. Available: https://paperswithcode.com/method/efficientnet

[19] P. A., Srudeep. "An Overview on MobileNet: An Efficient Mobile Vision CNN." Medium, June 11, 2020. [Online]. Available: https://medium.com/@godeep48/an-overview-on-mobilenet-an-efficient-mobile-vision-cnn-f301141db94d

[20] S. Pujara, "Understanding MobileNet: A Compact Vision Model," Built In, Nov. 15, 2022. [Online]. Available: https://builtin.com/machine-learning/understanding-mobilenet-compact-vision-model

[21] "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," arXiv, June 19, 2019. [Online]. Available: https://arxiv.org/abs/1905.11946