Facial recognition performance comparison of SOTA convolutional neural networks with a small dataset using transfer learning

Juan Lopera

December 11, 2022

Introduction

Computer vision is an increasingly important field in artificial intelligence (AI). The purpose of computer vision is to enable computer systems to see the world through our eyes [7]. A vast amount of what we see consists of visuals with patterns and details far too complex for human vision to perceive. The goal in computer vision is to design and deploy systems that can surpass the limitations of human vision. Computer vision began in the mid 20th century, with one of the earliest records of computer vision coming from the creation of the *perceptron* in 1957 by computer vision pioneer Frank Rosenblatt. Throughout the 1960's, many universities worked endlessly to progress artificial intelligence. In 1966, the *Summer Vision Project* was launched by Seymour Papert and Marvin Minsky with the goal of creating a computer system that could identify objects in images [1]. Thanks to the early works of brilliant engineers and scientists including Larry Roberts, Marvin Minsky, and Kunihiko Fukushima, more modern contributions from entities such as Google and Facebook, and international competitions like the ILSVRC, computer vision has grown to be one of the most progressive and compelling tools AI has to offer the future of humanity.

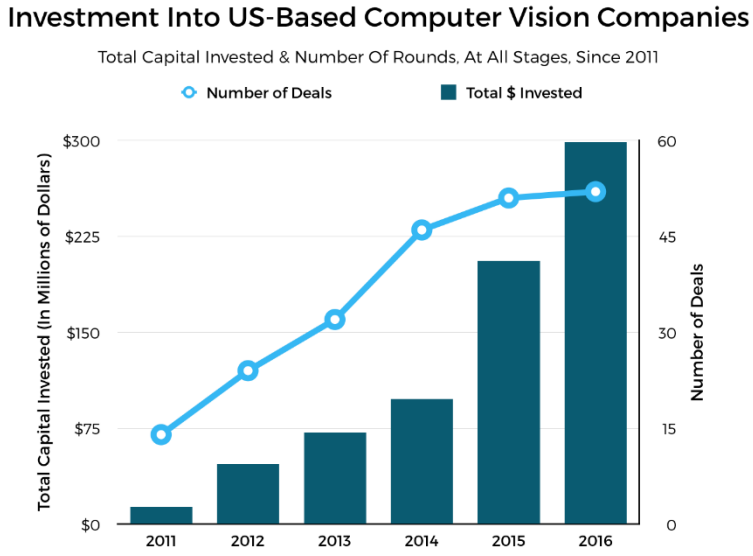

Figure 1. Market growth of computer vision in the last decade in the US. Source: Adapted from [2]

Image classification is one of many tasks in computer vision. The task is to correctly categorize an image input after processing the image's content [7]. For this, we make use of artificial neural networks. Mimicking our understanding of neurons, artificial neural networks are a series of interconnected matrices (layers) designed to identify patterns and relationships between large quantities of data. Deep learning is an improvement from conventional artificial neural networks due to its ability to discover hierarchical feature representations [6]. The name 'deep' learning originates from artificial neural networks that consist of multiple (more than two) layers. Although there are several distinct types of neural networks, convolutional neural networks are the most widely used for image classification. Convolutional Neural Networks (CNNs) are powerful image recognition networks built with convolutional, pooling, and fully connected layers [3]. Keras is a high-level, deep learning API developed by Google for implementing artificial neural networks. Keras offers more than 10 different state-of-the-art (SOTA) CNNs, including VGG16, DenseNet, Inception, and ResNet. Although their differences in structure and the parameter size, the listed CNNs above are powerful modes that can be used in numerous image classification tasks.

Model	Size (MB)	Parameters	Depth
MobileNet	16	4.3M	55
MobileNetV2	14	3.5M	105
DenseNet121	33	8.1M	242
NASNetMobile	23	5.3M	389
EfficientNetB0	29	5.3M	132
EfficientNetB1	31	7.9M	186
EfficientNetB2	36	9.2M	186
EfficientNetB3	48	12.3M	210

Table 1. SOTA models used in this project and their specifications. Source: Adapted from [3]

These are only 8 of the available models in Keras, and the reason I chose these models is because I intend to expand this project into a mobile, real-time system. To do this I plan to work with either a Raspberry Pi or an Arduino, and the processing power of either is significantly lower than that of my MSI NoteBook. Thus, I chose these models due to their small size in terms of memory, parameters, and depth. Now, one thing a neural network needs to perform with acceptable prediction accuracy (>80%), is a large amount of data. These networks learn through a process called back propagation which aims at tweaking the weights in each layer to minimize labelling error, or loss. The network modifies its weights based on information gained from every new image. Therefore, the more images, or data, the network has access to, the more information is available to adjust every weight to achieve minimal loss when labelling images.

However, substantial amounts of data are often not available for many image classification problems. Consequently, having a network that can perform accurately with small datasets is crucial. The aim of this project is to evaluate and compare the performance of several different SOTA CNNs in image classification with a small dataset.

Methods

For this project, I worked on a Windows 11 machine, using Python 3.7 with TensorFlow and Keras 2.7. The hardware specifications for my computer system consist of an MSI GP75 Leopard NoteBook with 16 GB of RAM, an Intel Core i7-10750H CPU and an NVIDIA GeForce RTX 2060 GPU. The SOTA models can be found at https://keras.io/api/applications/. The data consists of 1200 224x224x3 images of 8 of my friends faces, including myself. The images were divided into training, validation, and testing folders. 100 images of each person were added into 8 respective folders within the training folder, 35 images of each person were added into 8 respective folders in the validation folder, and finally, 15 unlabelled images of each person were placed into the testing folder.

When working with limited data, its often a good idea to increase the size of your dataset by augmenting the images. This does not create drastically different images, its merely the process of making minor realistic changes to the data to provide new information for the model to train on and learn from. This is a great way to increase the size of your data, and using Keras, this can easily be done in training using *ImageDataGenerator* from Keras. In this project I fit the models twice. Once without augmenting the data, and once more by including several augmentation parameters as shown in Figure 2. The purpose of this is to observe the difference in the performance of each model.

```
datagen = ImageDataGenerator(
        rotation_range=20,          # rotation
        width_shift_range=0.2,      # horizontal shift
        height_shift_range=0.2,     # vertical shift
        zoom_range=0.2,             # zoom
        horizontal_flip=False,      # horizontal flip
        brightness_range=[0.2,1.2]) # brightness
```

Figure 2. Augmentation parameters using ImageDataGenerator

I used the *ImageDataGenerator* module from the Keras preprocessing library along with the *flow_from_directory* function to feed the input data to each model. The SOTA models I used are designed to classify images from more than 1000 categories, so I needed to modify their outputs to predict on only 8 categories. To do this, I connected a classifier with 8 output neurons to each SOTA model.

This process of using a pretrained model with a custom classifier is called transfer learning. When using transfer learning, there are 4 ways to build and train your full model [4]:

1. Freeze the entire convolutional base model, leaving only the classifier to have trainable parameters
2. Freeze a large section of the convolutional base model, leaving only the classifier and a small part of the convolutional base model to have trainable parameters
3. Freeze a small section of the convolutional base model, leaving the classifier and a large part of the convolutional base model to have trainable parameters
4. Set the entire model, convolutional base model and classifier, to have trainable parameters

In this project I decided to set evaluate all 4 transfer learning modalities. Versions 1 and 4 were set exactly as described above. Versions 2 and 3 were set by freezing the top 2/3 and 1/3 of the convolutional base model layers respectively. For training, the models were fit with images in batches of 5, for 30 epochs, while monitoring the validation accuracy parameter and keeping only the max value. The models were compiled using the Adam optimizer with a learning rate of 0.0001, categorical crossentropy as the loss, and training accuracy as the metric of interest. Finally, for testing, each model predicted labels on 5 random unlabelled images from the test directory. These sets of predictions were automatically repeated 1000 times and the performance was calculated by averaging 1000 scores from the predictions.

Hypotheses

Hypothesis #1: *The largest model in terms of number of parameters will achieve the highest prediction performance*

Hypothesis #2: *Image augmentation will result in an overall increase prediction performance in each model*

Hypothesis #3: *The third version of each of the four transfer learning model versions will have the highest prediction performance*

First, I believe the largest model in terms of parameters will perform best since it has more parameters to adjust when learning from the data. Next, image augmentation increases the size of the dataset, which means the models will have more information to learn from, theoretically leading to better performance. Lastly, I think the third version of each of the four transfer learning models will do best because the pretrained models have already been designed to recognize low-level features, so freezing them and adjusting the remaining parameters will allow the entire model to learn much more efficiently.

Results

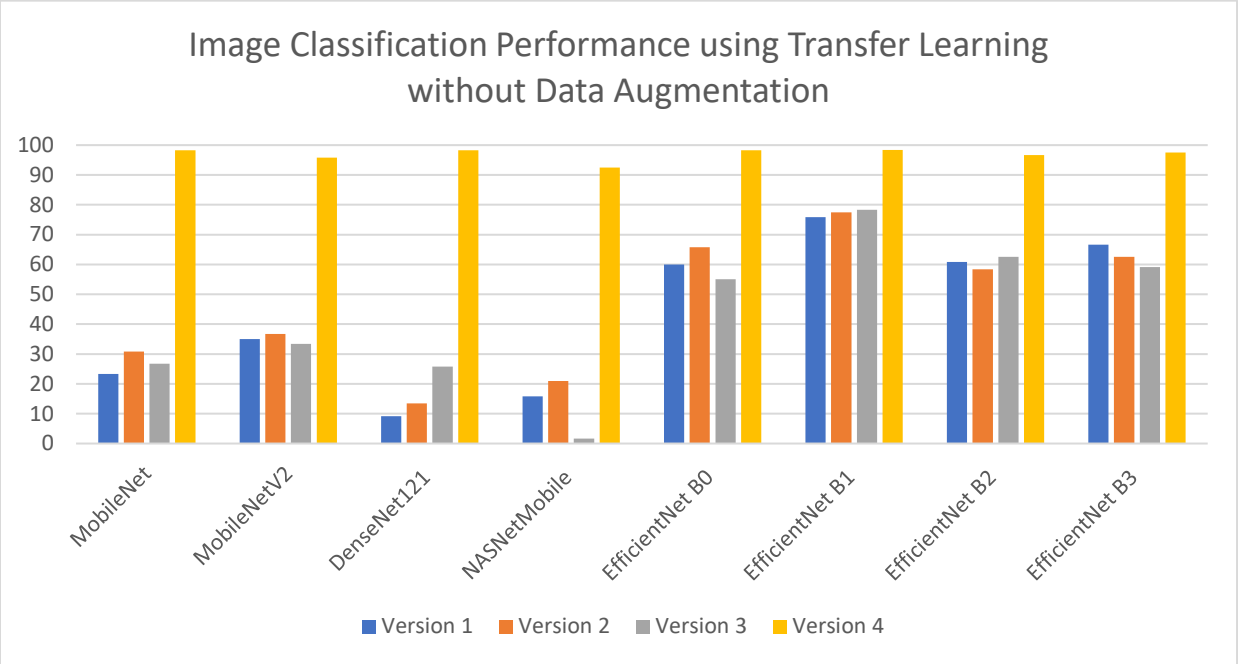

Figure 3. Performance chart of each model version without data augmentation

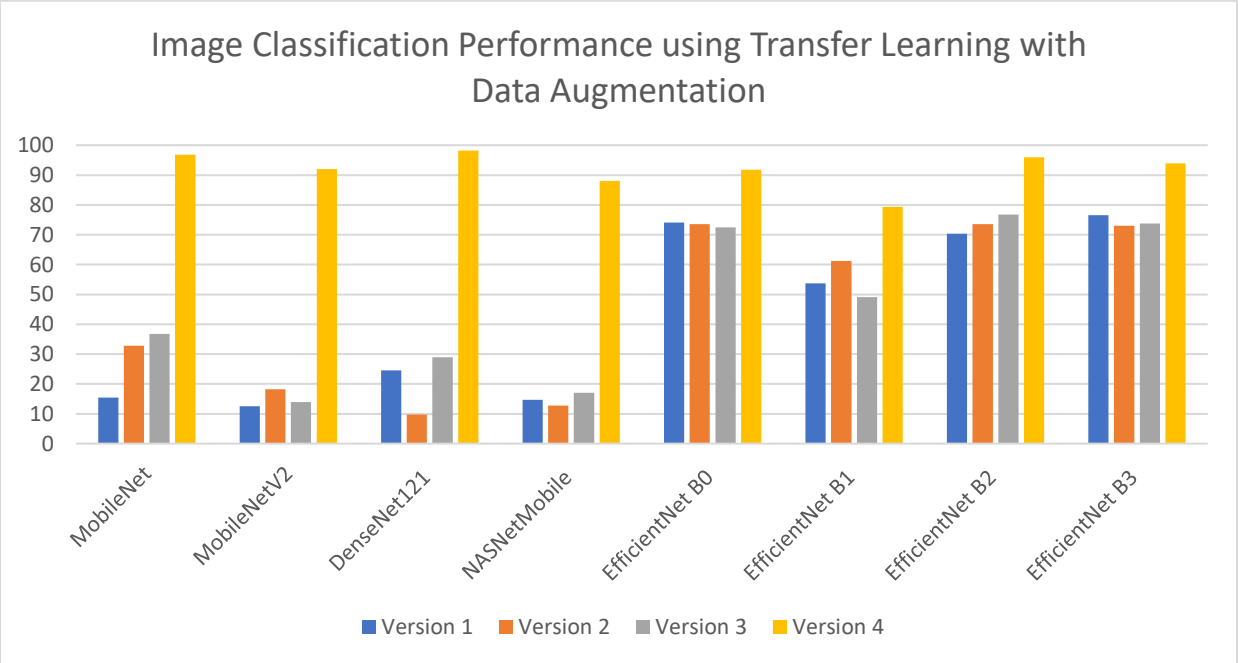

Figure 4. Performance chart of each model version with data augmentation

Model	Accuracy without Data Augmentation (%)	Accuracy with Data Augmentation (%)
MobileNet #1	23.359999999999907	15.460000000000024
MobileNet #2	30.799999999999788	32.759999999999806
MobileNet #3	26.719999999999832	36.71999999999981
MobileNet #4	98.31999999999984	96.83999999999979
MobileNetV2 #1	34.9799999999998	12.480000000000098
MobileNetV2 #2	36.719999999999814	18.23999999999996
MobileNetV2 #3	33.35999999999977	13.920000000000057
MobileNet V2#4	95.79999999999973	92.01999999999967
DenseNet121 #1	9.160000000000053	24.579999999999906
DenseNet121 #2	13.400000000000079	9.780000000000067
DenseNet121 #3	25.79999999999992	28.939999999999817
DenseNet121 #4	98.31999999999985	98.25999999999982
NASNetMobile #1	15.800000000000008	14.680000000000037
NASNetMobile #2	20.95999999999993	12.780000000000099
NASNetMobile #3	1.6399999999999977	17.019999999999996
NASNetMobile #4	92.47999999999959	88.0599999999996
EfficientNet B0 #1	59.96000000000018	74.11999999999985
EfficientNet B0 #2	65.82000000000014	73.53999999999994
EfficientNet B0 #3	55.04000000000008	72.46000000000006
EfficientNet B0 #4	98.31999999999985	91.77999999999962
EfficientNet B1 #1	75.87999999999997	53.660000000000075
EfficientNet B1 #2	77.45999999999978	61.22000000000013
EfficientNet B1 #3	78.35999999999977	49.11999999999992
EfficientNet B1 #4	98.33999999999986	79.31999999999982
EfficientNet B2 #1	60.84000000000016	70.36000000000008
EfficientNet B2 #2	58.36000000000011	73.55999999999999
EfficientNet B2 #3	62.56000000000013	76.79999999999981
EfficientNet B2 #4	96.63999999999973	95.97999999999972
EfficientNet B3 #1	66.68000000000004	76.5799999999998
EfficientNet B3 #2	62.56000000000015	72.97999999999986
EfficientNet B3 #3	59.140000000000136	73.7199999999999
EfficientNet B3 #4	97.49999999999984	93.93999999999964

Table 2. Prediction accuracy results of each version of the model with and without data augmentation

After visualizing the performance data, I see many surprising results. From Figures 3 and 4 we can see significant information about the performance of the models. The best version of almost every model performs slightly better without using data augmentation. For example, the best 3 performing models without using data augmentation are EfficientNet B1#4, EfficientNet B0 #4, and DenseNet 121 #4. These same 3 models go from a 98.34%, 98.32%, and 98.32% to a 79.32%, 91.78%, and 98.26% prediction accuracy respectively when trained on the augmented dataset. The drastic difference in performance between the EfficientNet models and DenseNet 121 when trained on both augmented and non-augmented data is likely due to the way they both extract and process image features. DenseNet 121 seems to be is clearly more robust when provided with new modified data. Another key result is that version #4 of every model consistently outperforms all the other respective versions by at least 20%. All these SOTA models are trained on thousands of images of animals, plants, and common objects. This is a good indication that the layers and weights responsible for extracting low- and high-level features of the pretrained model are not effective at the task of facial recognition. As we can see clearly, when allowing every parameter to be modified during training, each model outperforms significantly.

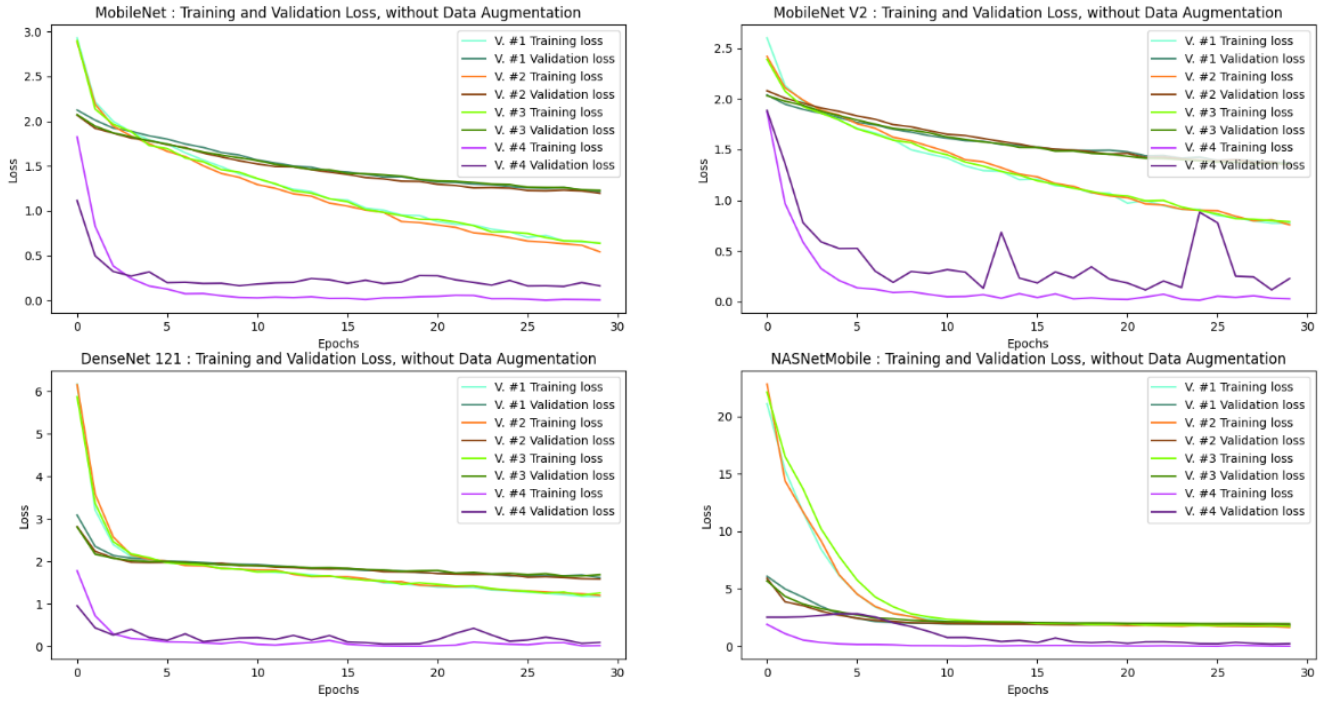

Figure 5. Training and Validation Loss of each version of MobileNet, MobileNet V2, DenseNet 121, and NASNetMobile without Data Augmentation

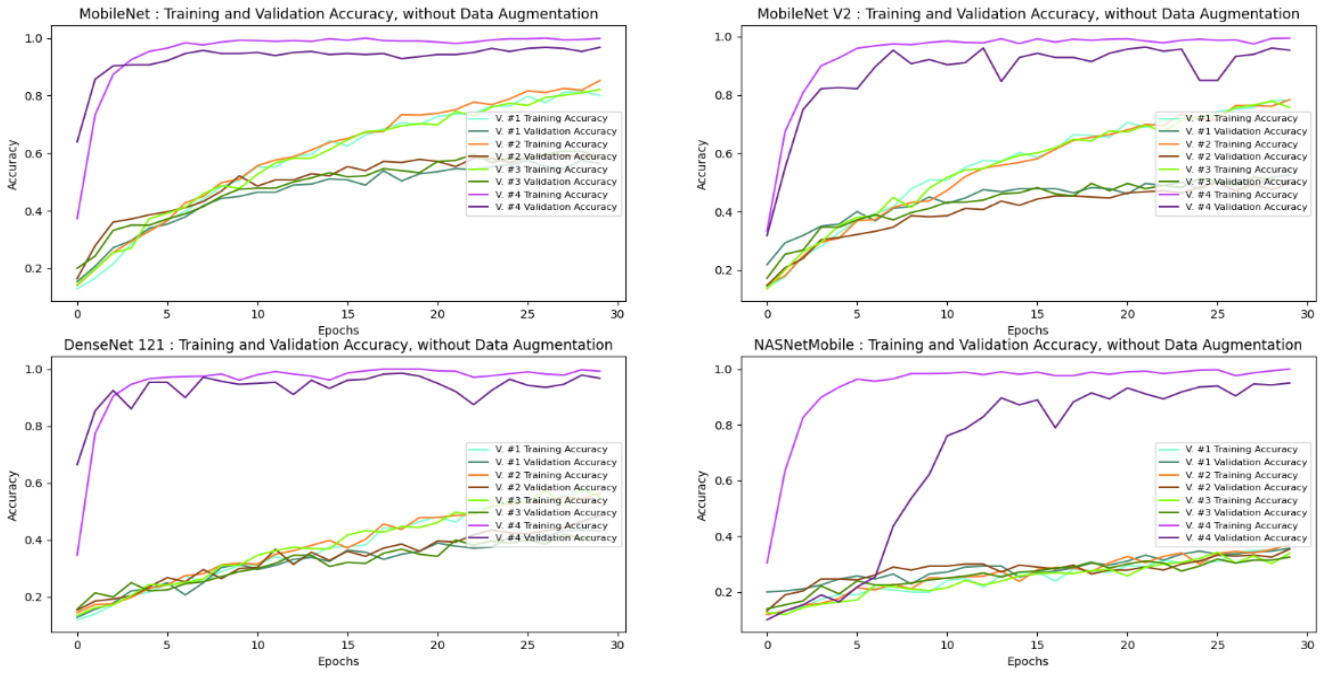

Figure 6. Training and Validation Accuracy of each version of MobileNet, MobileNet V2, DenseNet 121, and NASNetMobile without Data Augmentation

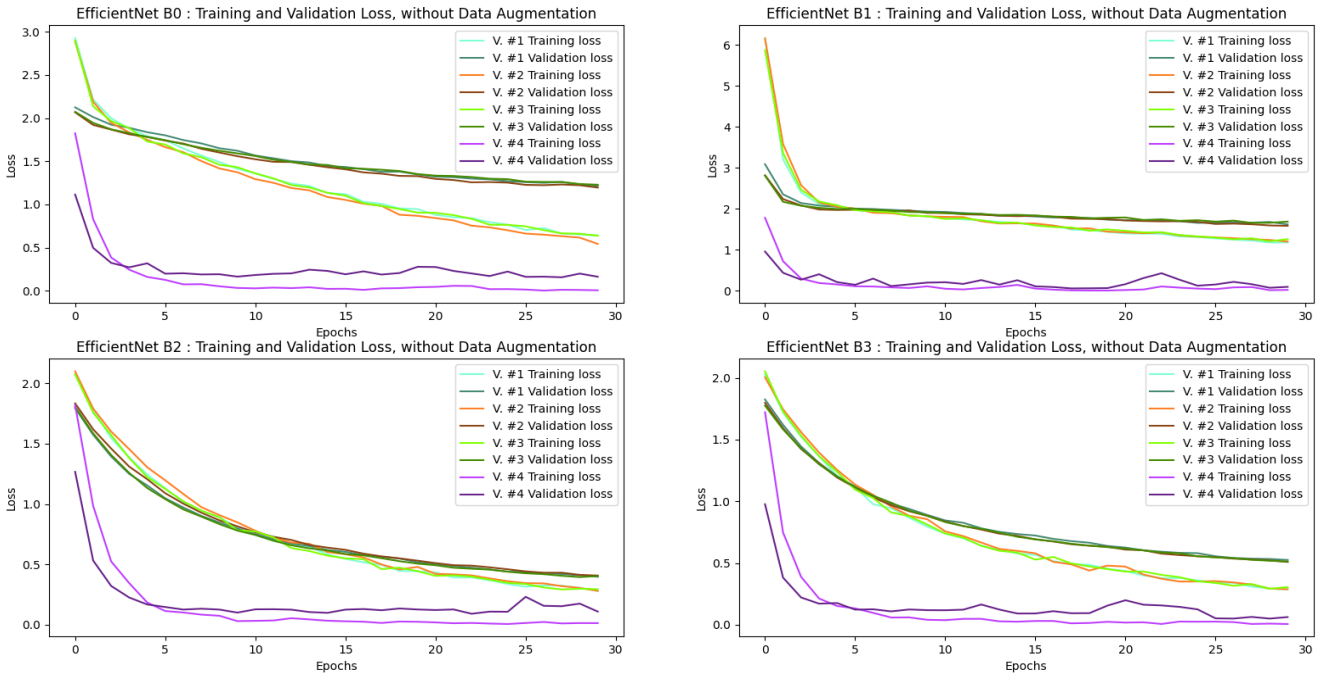

Figure 7. Training and validation loss of each version of EfficientNet B0, B1, B2 and B3 without data augmentation

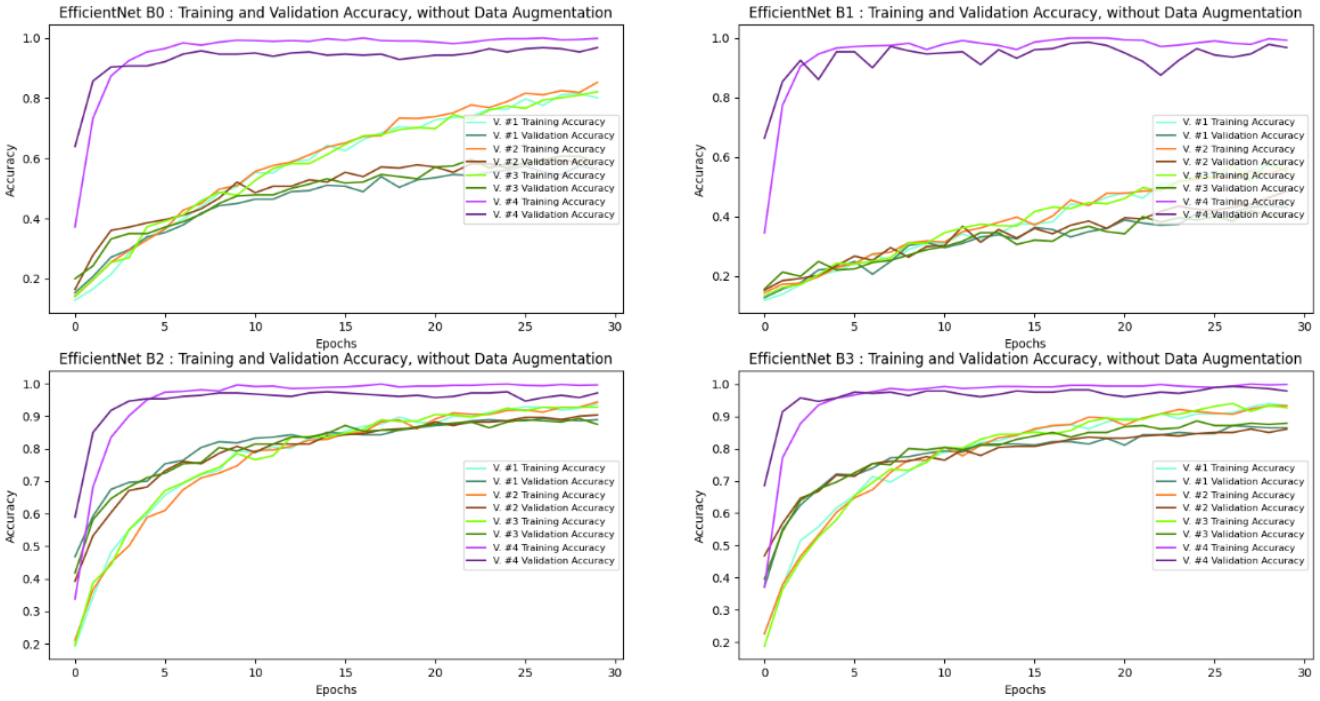

Figure 8. Training and validation accuracy of each version of EfficientNet B0, B1, B2 and B3 without data augmentation

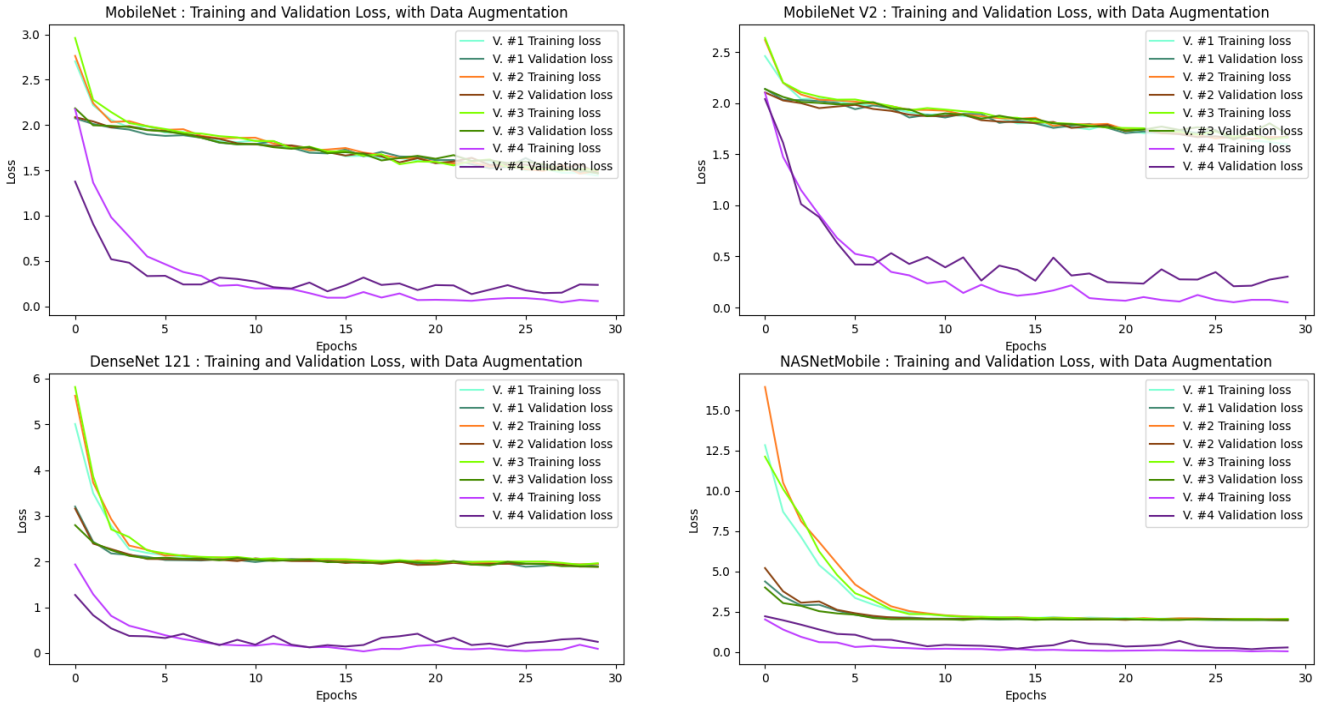

Figure 9. Training and validation loss of each version of MobileNet, MobileNet V2, DenseNet 121, and NASNetMobile with data augmentation

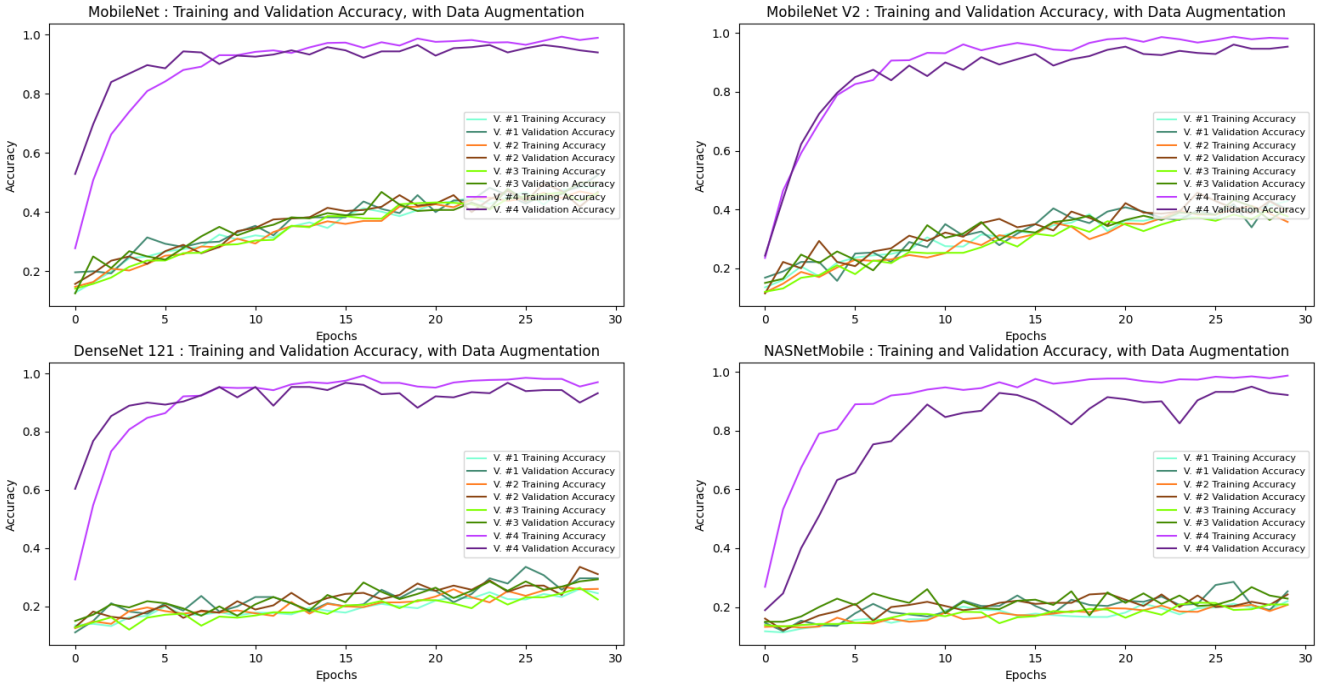

Figure 10. Training and validation accuracy of each version of MobileNet, MobileNet V2, DenseNet 121, and NASNetMobile with data augmentation

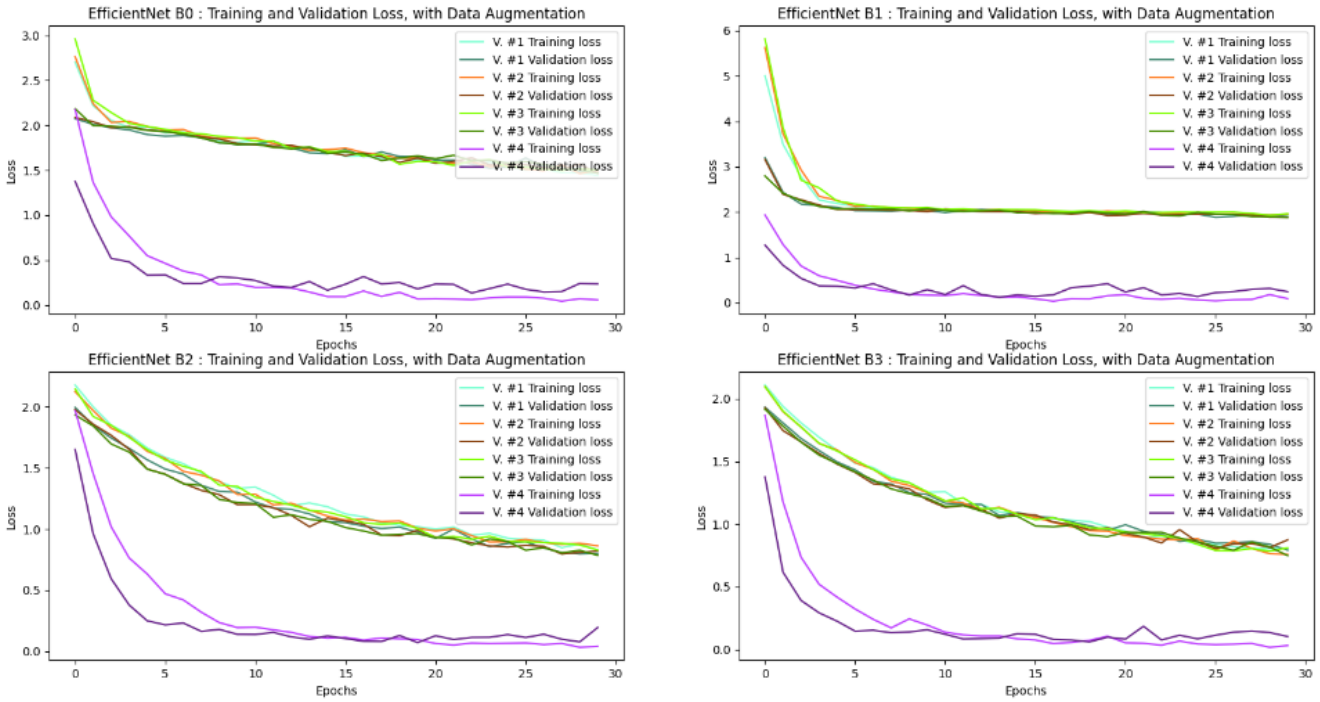

Figure 11. Training and validation loss of each version of EfficientNet B0, B1, B2 and B3 with data augmentation

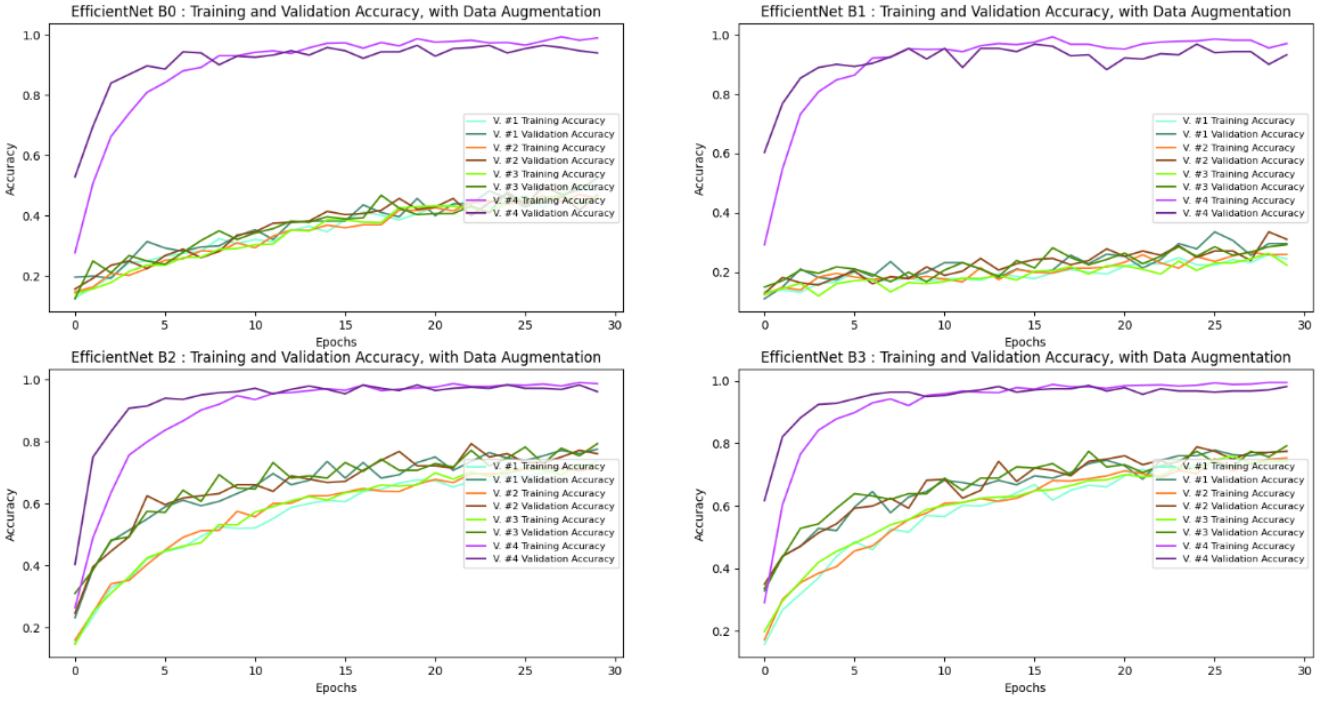

Figure 12. Training and validation accuracy of each version of EfficientNet B0, B1, B2 and B3 with data augmentation

The training data from Figures 5-12 clearly matches the performance results. First, the training and validation loss drops below 0.25 before 5 epochs of training for almost each model when training in the absence of data augmentation. However, when using data augmentation, its clear that most models take longer to approach the same minimum values for training and validation loss. DenseNet 121 does not seem to be affected as much by the data augmentation. As explained before, it seems that due to their architecture, DenseNet 121 models are much more robust at image classification when training on altered data than the other SOTA models used in this project. Second, its clear that version 4 for every model result in the highest accuracy and lowest loss. This is exactly what we saw in the performance data. After carefully observing each models training data, I noticed that NASNetMobile had very strange results. Even version 4 for this model, the validation accuracy achieves its maximum at a much slower rate than the other models. In Figure 6 we can see the validation accuracy for version 4 of NASNetMobile rise to above 80% after 10 epochs. This is very significant because every other model in either Figure 6 or 8 shows their respective validation accuracy values pass 90% in under 5 epochs. This leads me to believe that NASNetMobile might not be suitable for image classification when it comes to facial recognition. Looking back at Table 1, we can see that NASNetMobile has a size of 23 MB, 5.3 million parameters and a depth of 389 layers. Therefore, the reason this specific SOTA model underperforms compared to other, even smaller models like MobileNet, is likely due to the structure of the neural network and how the overall network creates a feature hierarchy from the input for making predictions.

After studying at all the results, I understood that all my hypotheses were, for the most part, incorrect. In my first hypothesis, I stated that the largest model in terms of parameters would have the highest prediction accuracy. Of the 8 chosen SOTA models, EfficientNet B3 is the largest with 12.3 million parameters. Version 4 of EfficientNet B3 peaks with a prediction accuracy of 97.5%. This is very high performance, but it comes in 5th overall, so clearly a higher number of parameters does not directly correlate to higher performance in image classification. In hypothesis 2, I said that image augmentation would lead to increased prediction accuracy in each model, but this clearly isn't the case. From Table 2 we can clearly see that almost every model performs 5% lower when training on the augmented images. Finally for hypothesis 3, I said that the version 3 of each model would perform the best because I believed that the layers loaded with the *ImageNet* weights responsible for extracting low-level features in the pretrained model would accelerate and improve the transfer learning with this small dataset. However, as I explained earlier, because the pretrained models were trained on datasets that consisted of animals, plants, and common objects such as food, household items and cars, I believe that these weights are not effective at extracting facial features. This might explain why version 4, the version where the weights of the entire model are trainable, of almost every model significantly outperforms the other versions.

Conclusions

This study is a precursor of a mobile, real-time system. In this next project I'll be building an autonomous RC car able to recognize litter in its path. The vehicle will need to be able to process and make decisions on live video in real-time. Although many models surpassed my performance expectations, I believe that MobileNet will be my model of choice. With only 4.3M parameters, a depth size of 55 layers, and a max prediction accuracy of 98%, this model is ideal for my next endeavor. I think this project was a success, but of course, there is always room for improvement. One way to improve the results is to use higher quality images. This would provide clearer features in each image, allowing the models to learn more in every step during training, leading to improved weight adjustments and overall higher image classification performance. Another source of improvement for understanding the performance of SOTA models in image classification with small datasets, is to train them for different image classification tasks. A few examples of this include medical imaging, images of rare species, and astronomy. These are real world applications that could significantly benefit from computer vision. Many rare medical conditions have little to no documentation. Creutzfeldt-Jakob disease (CJD) is a rare, degenerative, fatal brain disorder. In the initial stages of the disease, memory may begin to fail, people may present with behavioral changes, lack of coordination, and visual disturbances. As the illness progresses, mental deterioration becomes pronounced and involuntary movements, blindness, weakness of extremities, and coma may occur [5]. For most treatments, early detection leads to the highest chance of successful treatment. Therefore, a deep learning model that could accurately classify patients with the disease at an early stage based on a small set of high-quality MRI images could be extremely beneficial. Another application for computer vision is astronomy; objects fly around the Earth every second without us truly understanding what they are or how they might affect us. In deep space imaging, given the extremely low level of visibility, changes in the movements of objects can be very sudden and unpredictable. These changes are virtually impossible for human vision to immediately identify. Thus, a computer vision system that would constantly observe and report any detected changes in real-time would have significant potential.

In conclusion, I learned many fundamentals of deep learning and computer vision from this project, but I still have many questions. In what applications could these models perform better? How can these models be modified to improve accuracy when working on small datasets? Can a new more powerful, hybrid model be created from combining the architecture of two or more SOTA models? In my next project, I would like to assess these models to classify images in real-time. This makes me wonder which would be ideal for real-time applications. Overall, I've gained valuable knowledge and experience working with Python, TensorFlow, and Keras. I'm very excited to continue to learn and grow within the field of computer vision.

References

[1] G. D. T. Research and G. D. T. Research, “History of computer vision: Timeline,” Verdict, 08-Jul-2020. [Online]. Available: https://www.verdict.co.uk/computer-vision-timeline/. [Accessed: 25-November-2022].

[2] Zbigatron, “The reasons behind the recent growth of Computer Vision,” Zbigatron, 24-Nov-2021. [Online]. Available: https://zbigatron.com/the-reasons-behind-the-recent-growth-of-computer-vision/. [Accessed: 26-November-2022].

[3] K. Team, “Keras Documentation: Keras applications,” Keras. [Online]. Available: https://keras.io/api/applications/. [Accessed: 05-Dec-2022].

[4] R. Godasu, D. Zeng, and K. Sutrave, “Transfer learning in medical image classification: Challenges and opportunities,” CORE. [Online]. Available: https://core.ac.uk/display/326833449?utm_source=pdf&utm_medium=banner&utm_campaign=pdf-decoration-v1. [Accessed: 05-Dec-2022].

[5] S. M. Lee, J. W. Hyeon, S.-J. Kim, H. Kim, R. Noh, S. Kim, Y. S. Lee, and S. Y. Kim, “Sensitivity and specificity evaluation of multiple neurodegenerative proteins for Creutzfeldt-Jakob disease diagnosis using a deep-learning approach,” Prion, 15-Jan-2019. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6650195/. [Accessed: 05-Dec-2022].

[6] D. Shen, G. Wu, and H.-I. Suk, “Deep learning in medical image analysis,” Annual review of biomedical engineering, 21-Jun-2017. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5479722/. [Accessed: 05-Dec-2022].

[7] J. Gao, Y. Yang, P. Lin, and D. S. Park, “Computer vision in healthcare applications,” Journal of healthcare engineering, 04-Mar-2018. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5857319/. [Accessed: 08-Dec-2022].