1 word recognition using VGG 16

1.2 Transfer Learning

Why use transfer learning?

Transfer learning is a machine learning technique where a pre-trained model, which has been trained on a large dataset, is used as a starting point for solving a related but different task. Instead of training a model from scratch, transfer learning allows us to leverage the knowledge and features learned by the pre-trained model on a different task and apply it to our specific problem.

VGG16 (Visual Geometry Group 16) is a popular pre-trained convolutional neural network (CNN) architecture for image classification. The Visual Geometry Group at the University of Oxford developed it. VGG16 consists of 16 layers, including convolutional layers, max-pooling layers, and fully connected layers.

The advantages of using VGG16 in transfer learning include:

Powerful feature extraction: VGG16 has shown excellent performance in extracting meaningful features from images. The network's deep architecture allows it to learn and represent complex patterns and structures present in images effectively.

Generalization capabilities: Due to its architecture, VGG16 has been trained on a large-scale dataset, enabling it to learn a rich set of features that can generalize well to various visual recognition tasks. This makes it a suitable choice for transfer learning across different image classification problems.

Availability of pre-trained weights: Pre-trained weights for VGG16 are readily available in popular deep learning frameworks like Keras and PyTorch. These pre-trained weights save significant training time and computational resources, as you can directly use them as a starting point for your task.

Ease of fine-tuning: VGG16's modular architecture makes it easier to modify and fine-tune for specific tasks. By freezing certain layers and only training the last few layers or adding additional layers on top, you can adapt the pre-trained VGG16 model to your specific problem with relatively fewer training samples.

Wide adoption and community support: VGG16 has been widely used and studied in the deep learning community, resulting in extensive documentation, code examples, and online resources. This availability of knowledge and support makes it easier to implement and troubleshoot when using VGG16 for transfer learning.

1.3 VGG-16 Model

The previous challenges that we mentioned were not the last challenges, but while working on this transfer-learning model, many challenges appeared, so I will present these challenges in the form of scenarios, but first we will talk about the common things in all scenarios

Architecture: VGG16 is known for its simplicity and deep structure, consisting of 16 layers, including convolutional layers, max-pooling layers, and fully connected layers. Here is a summary of the VGG16 architecture:

Input layer: The input to the network is an image of fixed size, typically 224x224 pixels.

Convolutional layers: VGG16 starts with a stack of convolutional layers, each using small 3x3 filters with a stride of 1. The number of filters increases as we go deeper into the network, starting from 64 filters and doubling after each max-pooling layer. These layers are responsible for learning hierarchical features from the input image.

Max-pooling layers: After each stack of convolutional layers, max-pooling layers with 2x2 filters and stride 2 are applied to reduce the spatial dimensions and extract the most prominent features.

Fully connected layers: The last few layers of VGG16 are fully connected layers. These layers take the flattened features from the previous layers and map them to the desired number of output classes. The fully connected layers are followed by a softmax activation to produce class probabilities.

Activation function: Throughout the network, rectified linear units (ReLU) are used as the activation function, which helps introduce non-linearity and allows the model to learn complex relationships in the data.

Dropout: VGG16 also incorporates dropout regularization, specifically after the fully connected layers, to prevent overfitting and improve generalization performance. Dropout randomly sets a fraction of the input units to 0 during training.

The VGG16 architecture is characterized by its deep structure, with multiple stacked convolutional layers and relatively small filter sizes. This design allows the network to learn more complex and abstract features from images. However, the large number of parameters in VGG16 makes it more computationally expensive to train compared to shallower networks.

VGG16 has achieved impressive performance on various image classification benchmarks and has become a popular base model for transfer learning due to its strong feature extraction capabilities and generalization performance.

First scenario

In the beginning, we started training the model with five classes

(Panadol, Ketolac, Cataflam, Catafast, Brufen) written by only three people. The total number of images was 1043 images, divided into approximately 230 images for each drug, taking into account the balance of data and its distribution equally among all classes.

As for the structure, we gave another layer just to be trained on our data.

Then we started training the model, after 500 epochs, The Training accuracy became 99%, training loss = 4.6%, test accuracy = 95.9% and test loss = 9.3%

The challenge in the first scenario is overfitting on the data, and although it is one of the advantages of the VGG16 model to handling the problems of overfitting, it happened and this reason is due to the lack of the data.

Second scenario

In this scenario, we tried to solve the problem of lack of data, so our friends helped us write data with us and increase the number of data.

Therefore, the number of images became 2130, distributed over 5 classes, which means about 430 images for each drug, twice the number of images in the previous scenario, but they were written by 9 people, and we did not change anything in the structure of the model.

But the new challenge in this scenario was the opposite of the challenge in the previous scenario, so under fitting was our problem at this time. Despite the increase in the amount of data, the data accuracy was 79.3%, and testing accuracy was 77%

The last two Classes were always predicted wrong, no matter what the name of the last two Classes was, and no matter how much we tried to replace them

So we started thinking about what might be the cause of this problem, so we started to change the transfer learning model from VGG16 to MobileNet, but there was no significant difference, so the old model was not the obstacle.

Third scenario

On the next one, we thought the hitch was in the architecture, so we changed part of it and added 5 hidden layers, but that change was not right, and instead of increasing efficiency, the training accuracy became 77.8%, and testing accuracy 73.5%

Fourth scenario

Therefore, we returned the structure as it was and made sure that the error would be in the dataset

Nevertheless, in order to verify this, we reduced the number of classes to three classes, so the total number of data became 1276 images written by nine people, they were according to our correct expectation, and the training accuracy became 95.5%, training loss 16.1%, testing accuracy 96.8% and testing loss 14%

In addition, the new challenge became the small number of classes

Fifth scenario

So after we made sure that the obstacle was in the dataset, it was necessary to determine where the error was first in order to correct it, so we divided the data of each of the nine people separately in order to test each of them separately and know which one was wrong

After testing the data, we found the error in one person, and we were not thinking about the criteria for writing the dataset, the error was that one of the colleagues had written the medications with a pencil, and when photographing, the flash was lit, so the pen line was close to the color of white, but it can be distinguished by the human eye, but in the model when converting The image to Gray could not be recognized by the model, and therefore the training was weak

Models	Train Acc	Train Loss	Test Acc	Test Loss	Notes
Model 8	95.9%	17.6%	91.4%	32.5%	best
Model 4	99.4%	2.6%	88.7%	54.1%	OverFit
Model 5	97.8%	8.0%	88.2%	51.5%	
Model 7	97.8%	14.5%	87.3%	56.4%	
Model 1	92.2%	30.8%	86.9%	53.8%	
Model 2	99.0%	4.3%	82.4%	77.7%	OverFit
Model 3	99.7%	1.7%	74.7%	97.0%	OverFit
Model 6	99.7%	2.2%	63.0%	161.8%	OverFit

Data test results

Sixth scenario

After correcting the corrupted data, we have about 2,550 images, out of about 510 images for each drug, distributed unevenly over five chapters and written by nine people. The final output was as follows:

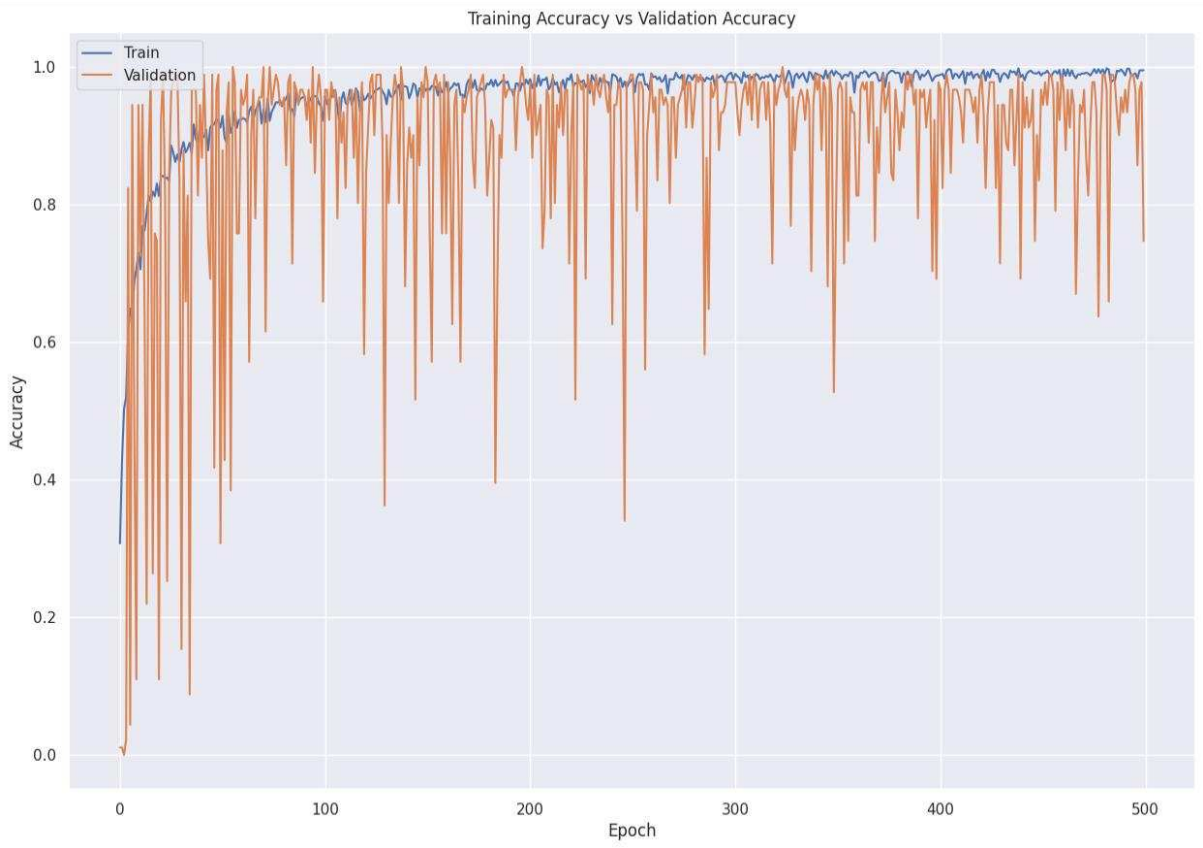
Training Accuracy = 98.6%

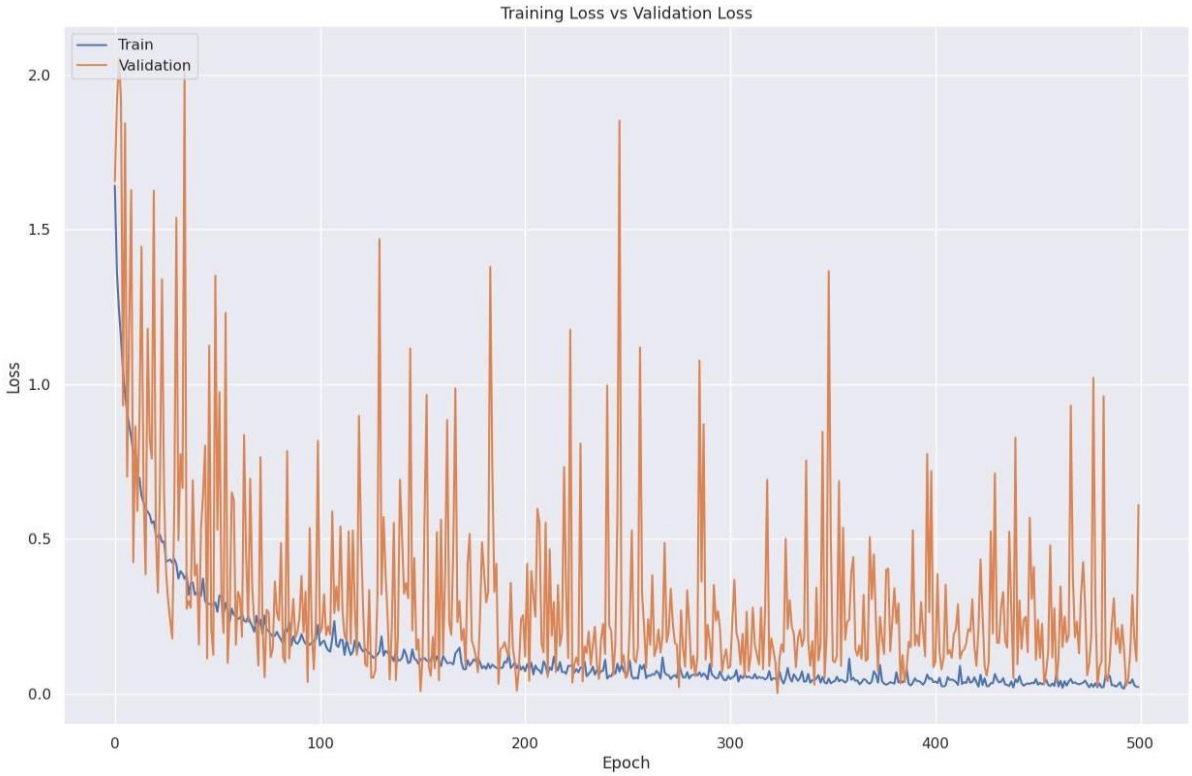
Training loss = 4.8%

Testing Accuracy = 96%

Testing loss = 19%

Noting that about twenty images written in the hands of people who are not among the previous nine people were tested, and the model predicted all of them correctly.





Seven scenario

After we succeeded in predicting the five classes successfully, we both increased the number of classes and wrote 2 additional classes to become 7 classes ('Cataflam', 'Ketolac', 'Brufen', 'Panadol', 'Actos', 'Diclac', 'Insulin'), this time taking into account the standards for writing dataset.

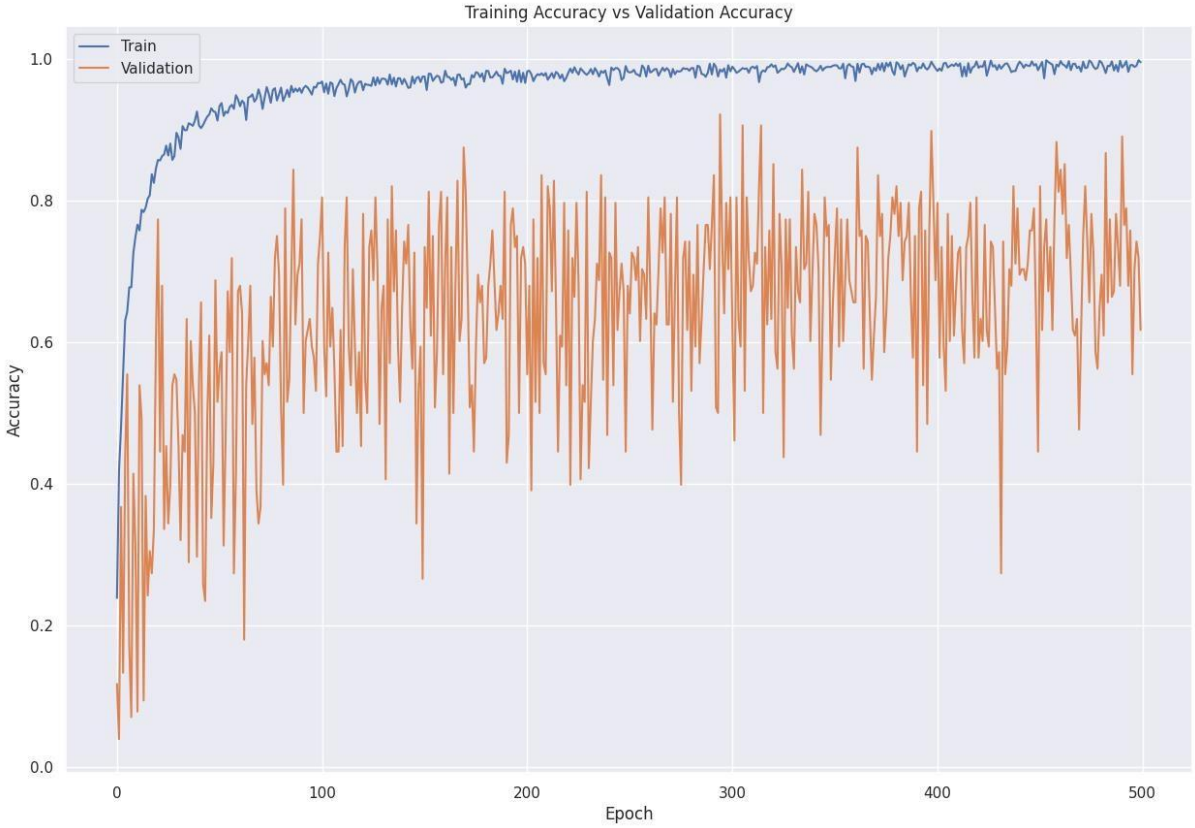
Therefore, the total data became 2861, or 510 images for each drug, as in the previous scenario, written by the same nine people and the same structure, and its efficiency was as follows:

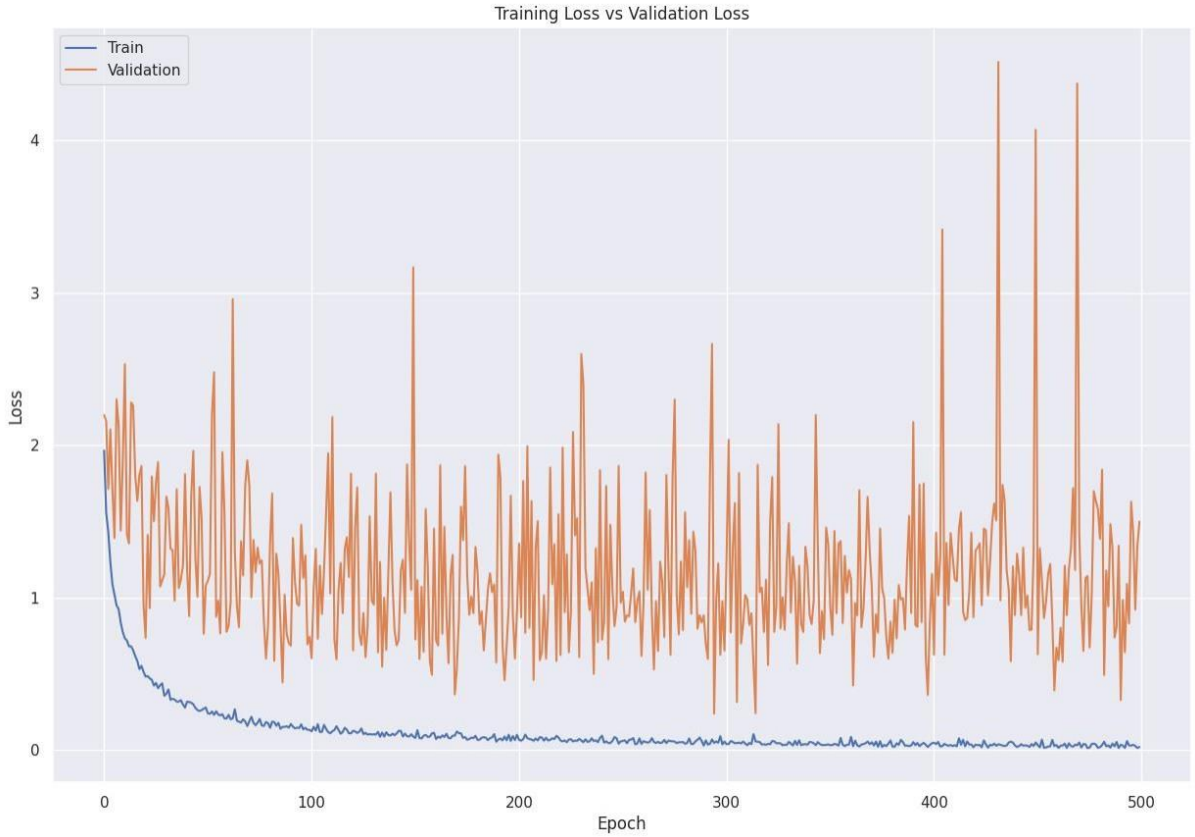
Training accuracy = 97.2%

Training loss = 10.3%

Testing accuracy = 94.7%

Testing loss = 18.5%


Training Accuracy vs Validation Accuracy
Train
Validation
Accuracy
Epoch
0.0
0.2
0.4
0.6
0.8
1.0
0
100
200
300
400
500


Training Loss vs Validation Loss
Train
Validation
Loss
Epoch
0
1
2
3
4
0
100
200
300
400
500