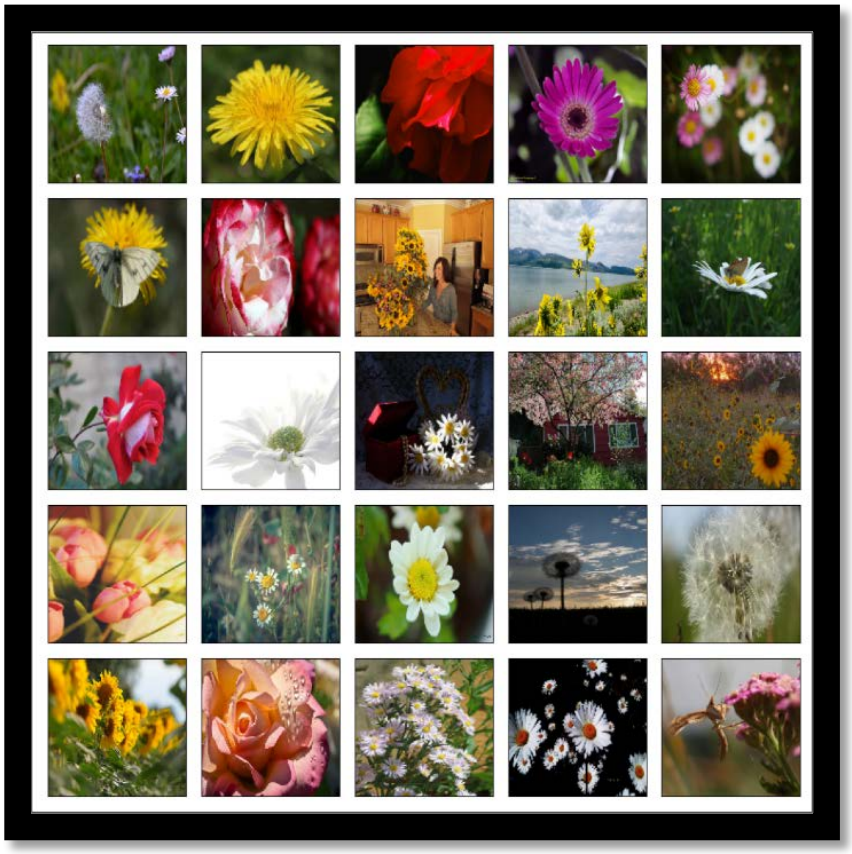
THE GEORGE WASHINGTON UNIVERSITY

WASHINGTON, DC

Data Augmentation and Neural Networks To Improve Image Classification

FINAL PROJECT: MACHINE LEARNING II



Joseph Francis
Srilatha Lakka
Professor Amir Jafari
DATS6203

Table of Contents

1. INTRODUCTION

In machine learning, image classification is a process to analyze the extracted image features and organize them into categories by using neural networks. In recent years, the neural network techniques have improved image classification accuracy quickly, such as AlexNet can achieve 15.3% image classification error rates1. These techniques are practically applied in many fields, such as artificial intelligence, medical diagnostics, E-commerce, gaming and automotive industries. The convolutional neural network (CNN) is one of the most popular deep neural network (DNN) learning algorithms which can perform classification tasks directly from images. CNN models can produce the state-of-the-art classification predictions with directly learned features. A CNN model generally contains more than one convolutional layer for specialized linear operations and includes local or global pooling layers to perform nonlinear down sampling. After learning features from many layers, a fully connected layer outputs the probabilities for each class to be predicted. However, model overfitting and poor performance are common problems in applying neural network techniques.

1.1 Problem Statement

Model overfitting and poor performance are common problems in applying neural network techniques because some of the high frequency features may not be useful in classification. Approaches to bring intra-class differences down and retain sensitivity to the inter-class variations are important to maximize model accuracy and minimize the loss function.

1.2 Project Outline

Data augmentation is a common technique to overcome the lack of large, annotated databases, a usual situation when applying deep learning to medical imaging problems. Nevertheless, there is no consensus on which transformations to apply for a particular field. This project aims at identifying the effect of different transformations on images using deep learning and various pre-trained models such as InceptionV3, Xception and ResNet50 and finally use Generative Adversarial Networks or GAN.

We will first introduce the data, then concepts, algorithms, structures of our various models and their theories. Then we'll explain each step of implementation, the challenges we encountered, and how we arrived in solving. Finally, we will explain the conclusion and the ideas for future improvement.

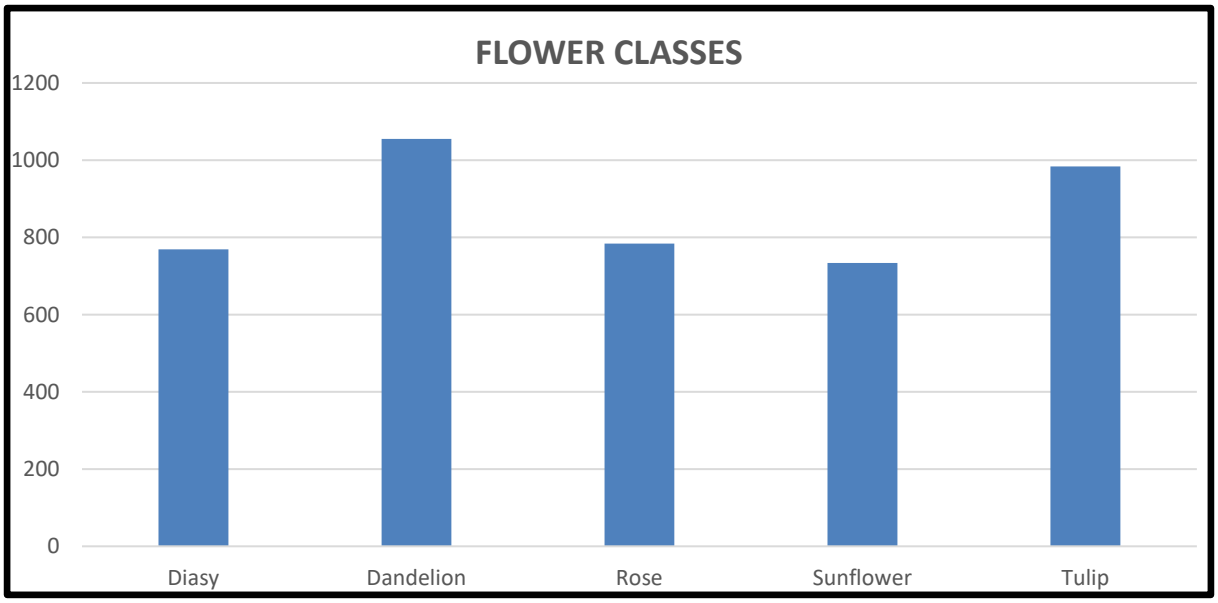
2. DATA SET

We collected data from Kaggle (https://www.kaggle.com/alxmamaev/flowers-recognition). This dataset contains 4242 images of flowers. The data collection is based on the data flicr, google images, and yandex images. This can be used to recognize plants from the photo.

2.1 Overview

The pictures are divided into five classes: daisy, tulip, rose, sunflower, and dandelion. For each class there are about 800 photos. Photos are not high resolution, about 320x240 pixels. Photos are not reduced to a single size; they have different proportions!

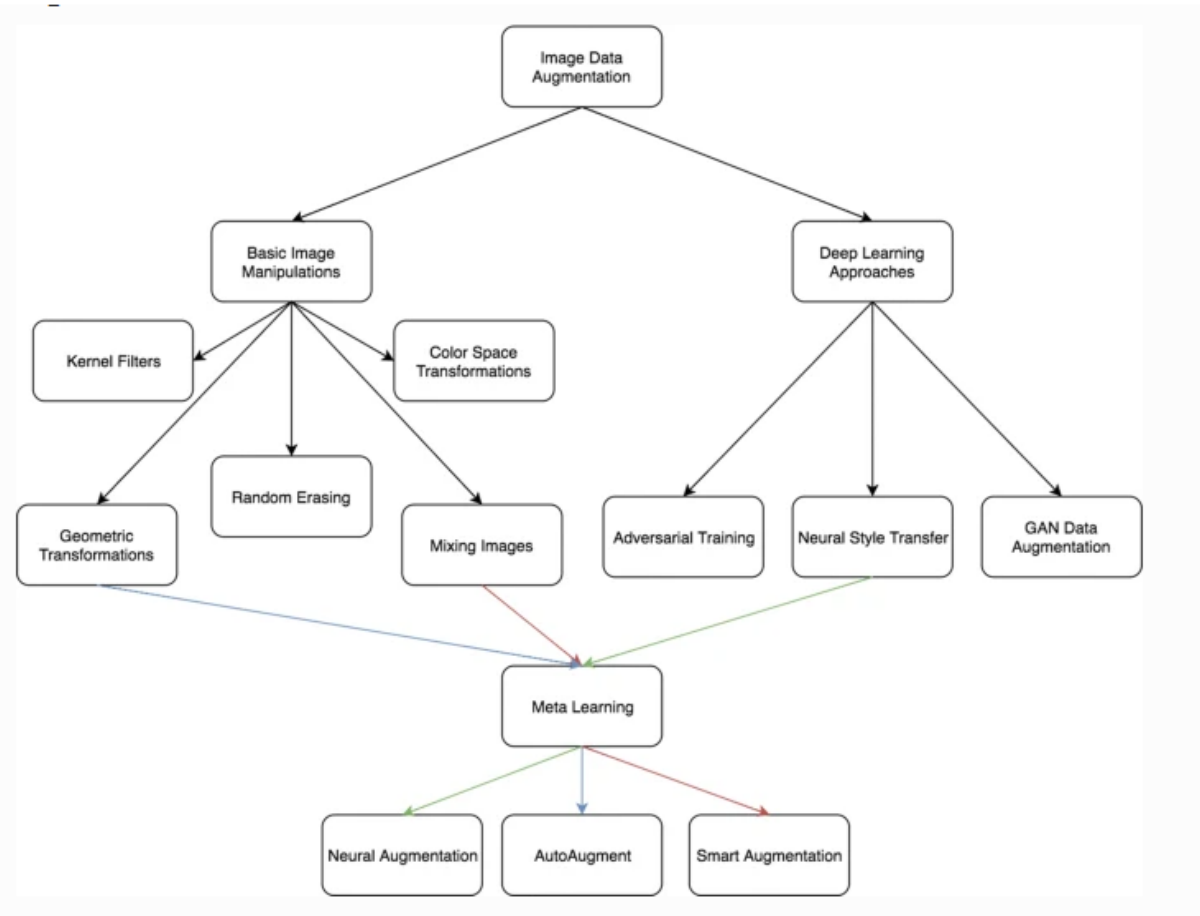
Distribution of Classes



Data Augmentation

Every Frame work as its own augmentation methods or even a whole library. In Keras we have a variety of preprocessing layers that may be used for Data augmentation. Also we may use ImageDataGenerator that generates batches of tensor images with real-time data augmentation. In general all libraries can be used with all frameworks, if you perform augmentation before training the model. Albumentations and transforms are really fast and that is why they are commonly used.

The selection of the most suitable strategy remains a train and error process. Here we have set the fill mode as nearest and applied geometric transformations such as flipping, rotation, translation, cropping and scaling.



Source: https://link.springer.com/article/10.1186/s40537-019-0197-0

3. PROJECT WORKFLOW

In the process workflow, we are going to see how we process the data, apply the models and get the result at the end.

First, in the data block, we load the libraries and the dataset, divide them into three distinct datasets like training, testing and validation datasets. We divide the total 4242 images into 2593 images for training and 860 images for testing and 870 images for validation which is in the ratio of 60% for training, 20 and 20 for testing and validation.

Then in pre-processing, we normalize both training as well as testing data images, and convert them into dimensions that are accepted by Keras. We also convert labels to categorical values.

In Model, we start with the basic CNN model, with three convolutional layers, one max-pooling layer, and one classifier layer for image classification, because they are spatially invariant and great for images.

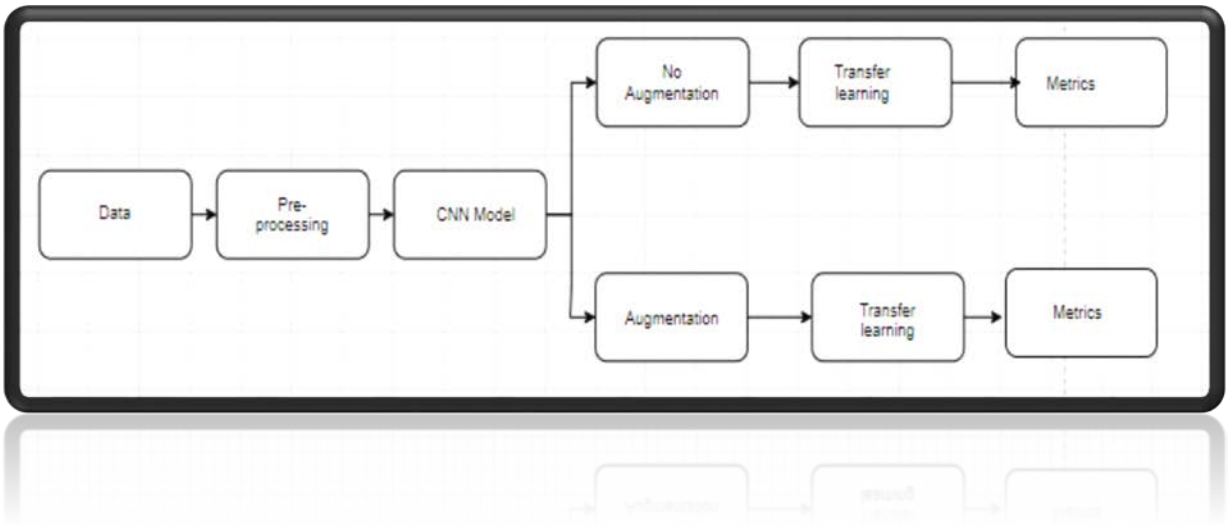
The max pooling layers generally come after the convolutional layers. They shrink the size of the volume by taking the max value within the filter, and hence reduce computation costs as we move through the network.

ReLU's or Rectified Linear Units are used as the activation function, because they help to reduce vanishing gradients, which are an issues with sigmoid and TanH activation layers.

Usually Vanishing gradients are a problem, because they reduce the likelihood that a signal will propagate to the input layer and adjust those weights. Hence the earlier layers will not learn.

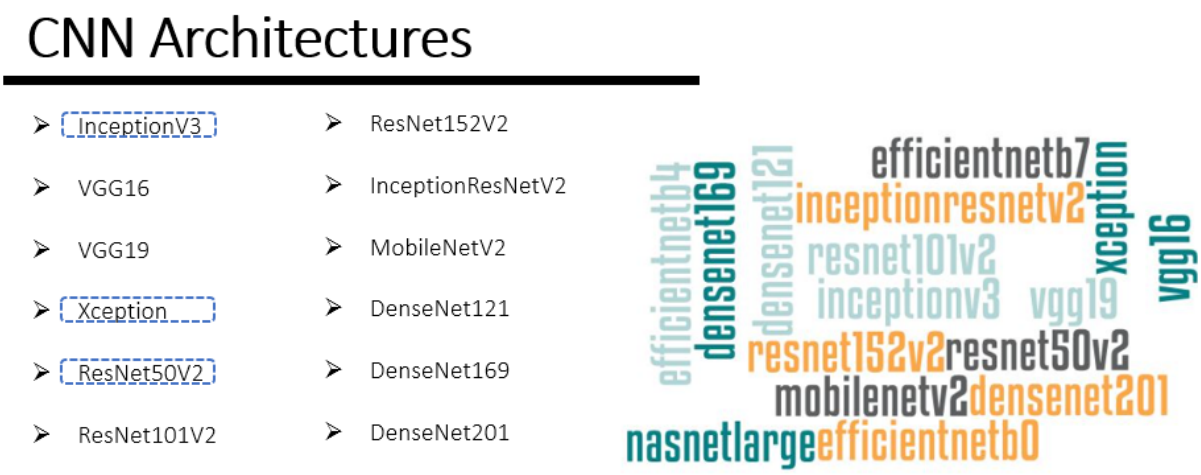
The last layer has a softmax activation with 5 units, to account for the 5 classes of the dataset, which is daisy, rose, tulip, sun flower and dandelion. We will get a probability for each of the classes.

We used Adam optimizer, because it adjusts the learning rate separately per layer, and can be used with weight decay, so that when we get close to convergence, so that the learning can be reduced to get better results. Categorical crossentropy is chosen since this is a multiclass classification problem and set the epoch to 30 to see that the best validation accuracy. Finally we will compile them. After compiling, we fit the training data to the model. Here we see two routes one with data augmentation and one without data augmentation. We developed our first deep learning model without performing augmentation, and computed the accuracy. After that, we ran a similar deep learning model with augmentation, and computed the accuracy. Finally, we will compare the performance of both models and evaluate our models by using metrics such as accuracy, precision, recall, f1-score and MCC which is known as Mathews Correlation Coefficient and mostly used in machine learning.



4. NEURAL NETWORK ARCHITECTURE

CNN Architectures:



When we have a small dataset, the models we build cannot represent patterns well. For example, if you wanted to use the same model we build, to recognize the flowers from the large volume of flower images dataset in the future it may or may not give the same accuracy because of big dataset. That's why it is recommended to use pre-trained models to get better accuracy in prediction.

Usually transfer learning takes place with fairly large models with millions or even hundreds of millions of parameters that have been trained on a gigantic volume of data to generalize.

To train them with our data it is necessary to recover the pre-trained model(s), to freeze the weights of their layers. Leave only the last layer, or the last layers

In this project, we compare different pre-trained models to see which one would be best fit for our study. We used all the pre-trained model mentioned above, but we primarily focused on working in INceptionv3, Xception and Resnet50. To change the pre-trained model easily and quickly, we created a function, that contains the architecture for tuning a pre-trained model on the data and evaluate them with metrics.

4.1.1 Description

Inception v3 is a convolutional neural network for assisting in image analysis and object detection, and got its start as a module for Googlenet. It is the third edition of Google's inception Convolutional Neural Network, originally introduced during the ImageNet Recognition Challenge.

4.1.2 Configuration

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Fig 1: Source: https://cdn.analyticsvidhya.com/wp-content/uploads/2018/10/Screenshot-from-2018-10-16-11-56-41-850x462.png

4.1.3. Architecture

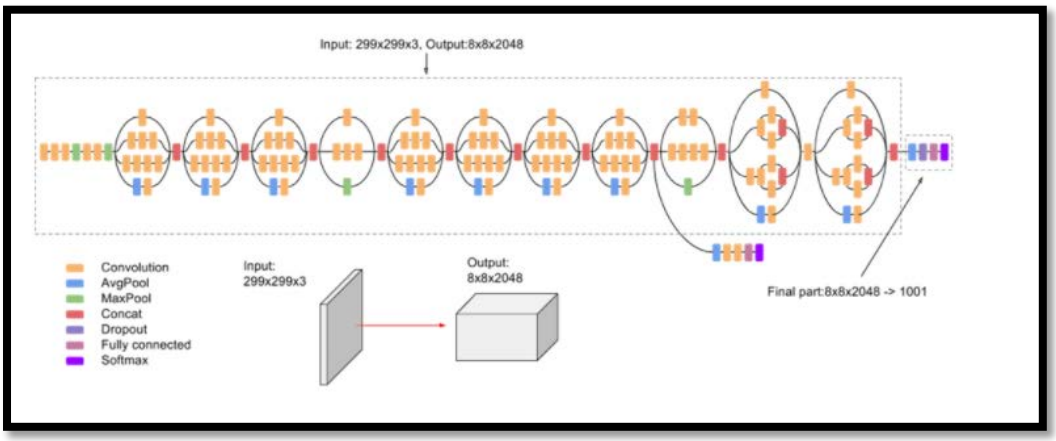


Fig2 Source: https://cloud.google.com/tpu/docs/inception-v3-advanced

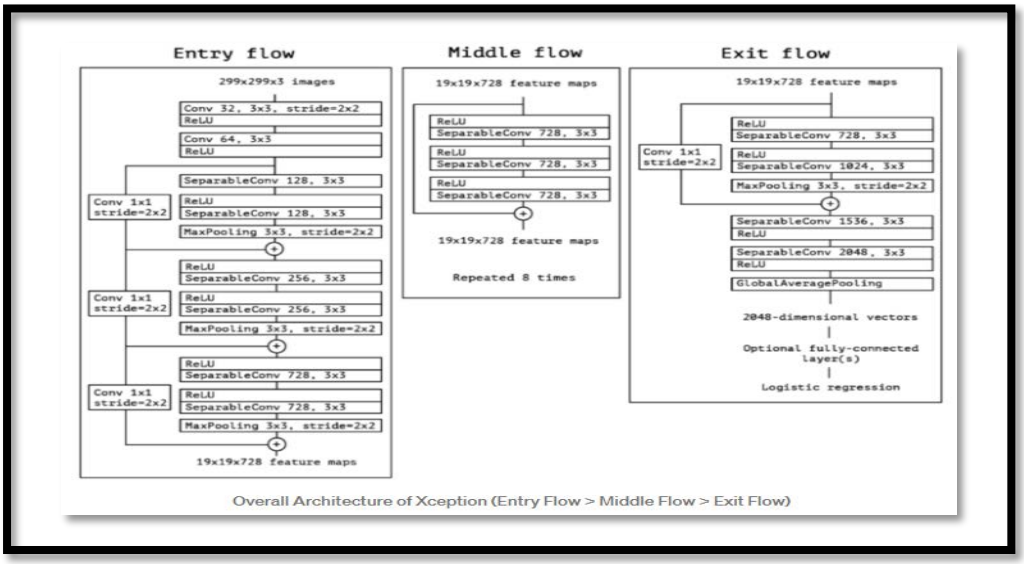
Inception-v3is a convolutional neural network architecture from the Inception family that makes several improvements including using Label Smoothing, Factorized 7 x 7 convolutions, and the use of an auxiliary classifier to propagate label information lower down the network (along with the use of batch normalization for layers in the sidehead)

type	patch size/stride or remarks	input size
conv	3×3/2	299×299×3
conv	3×3/1	149×149×32
conv padded	3×3/1	147×147×32
pool	3×3/2	147×147×64
conv	3×3/1	73×73×64
conv	3×3/2	71×71×80
conv	3×3/1	35×35×192
3×Inception	As in figure 5	35×35×288
5×Inception	As in figure 6	17×17×768
2×Inception	As in figure 7	8×8×1280
pool	8 × 8	8 × 8 × 2048
linear	logits	1 × 1 × 2048
softmax	classifier	1 × 1 × 1000

Fig 2: Source: https://www.geeksforgeeks.org/inception-v2-and-v3-inception-network-versions/

3.2.1 XCEPTION

Xception is an extension of the inception Architecture which replaces the standard Inception modules with depthwise Separable Convolutions.



Fif4source:https://maelfabien.github.io/deeplearning/xception/

As in the figure above, SeparableConv is the modified depthwise separable convolution. We can see that SeparableConvs are treated as Inception Modules and placed throughout.

3.3.1 RESNET50

ResNet is a powerful backbone model that is used very frequently in many computer vision tasks. ResNet50 is a variant of ResNet model which has 48 Convolution layers along with 1 MaxPool and 1 Average Pool layer. It has 3.8 x 10^9 Floating points operations.

ResNet50 Architecture

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Fig 3: Source: https://www.geeksforgeeks.org/inception-v2-and-v3-inception-network-versions/

5. EXPERIMENTAL SETUP

5.1 Pre-Processing

The image dimensions are not uniform, after loading them, they were resized before split to training part and validation part. In this place, we tried 150 x 150.

Classes	Dimension
Diasy	125,134
Dandelion	98,134
Rose	80,150
Sunflower	134,152
Tulip	134,134

5.1.1. Train, Validation Test Split

For training and testing purposes for our model, we have our data broken down into three distinct datasets like training, testing and validation datasets. We divide the total images (4242) into 2593 images for training and 860 images for testing and 870 images for validation which is in the ratio of 60% for training, 20 and 20 for testing and validation. These datasets will consist of the following:

Training set: It's the set of data used to train the model. During each epoch, our model will be trained over and over again on this same data in our training set, and it will continue to learn about the features of this data. Later we can deploy our model, and have it accurately predicted on new data that it's never seen before. It will be making these predictions based on what it's learned about the training data.

Validation set: This is a set of data, separate from the training set that is used to validate our model during training. This validation process helps give information that may assist us with adjusting our hyper parameters. With each epoch during training, the model will be trained on the data in the training set and it will also simultaneously be validated on the data in the validation set. During the training process, the model will be classifying the output for each input in the training set. After this classification occurs, the loss will then be calculated, and the weights in the model will be adjusted. Then, during the next epoch, it will classify the same input again.

Note: The data in the validation set is separate from the data in the training set. When the model is validating on this data, this data does not consist of samples that the model already is familiar with from training to ensure that our model is not overfitting to the data in the training set.

Test set: This is a set of data that is used to test the model after the model has already been trained. The test set is separate from both the training set and validation set. After our model has been trained and validated using our training and validation sets, we will then use our model to predict the output of the data in the test set.

5.2 No Augmentation

We start with the basic CNN model with two use convolutional layers because they are spatially invariant and great for images. This means that no matter where an element appears in an image it can be detected. This is because convent layers use filters which slide over the image/volumes to produce activation maps.

The max pooling layers generally come after the convolutional layers. They shrink the size of the volume by taking the max value within the filter and hence reduce computation costs as we move through the network.

ReLU's or Rectified Linear Units are used as the activation function because they help reduce vanishing gradients which are an issue with sigmoid and TanH activation layers. Vanishing gradients are a problem because they reduce the likelihood that a signal will propagate to the input layer and adjust those weights. Hence the earlier layers will not learn.

The last layer has a softmax activation with 5 units to account for the 5 classes of the dataset. We will get a probability for each of the classes.

We used Adam optimizer because it adjusts the learning rate separately per layer and can be used with weight decay so that when we get close to convergence the learning can be reduced so we can thus get better results. Categorical crossentropy is chosen since this is a multiclass classification problem.

We use the fit generator here because we are getting the data from an ImageDataGenerator. These correspond to generators in python and can produce batches into infinity. Running for 30 epochs we see that the best validation accuracy is about 0.6440.

5.3 Conventional Data Augmentation

Data augmentation techniques are often used to regularize models which work with images in neural networks and other learning algorithms. With the labelled original training dataset, synthetic images can be created by various transformations to the original images.

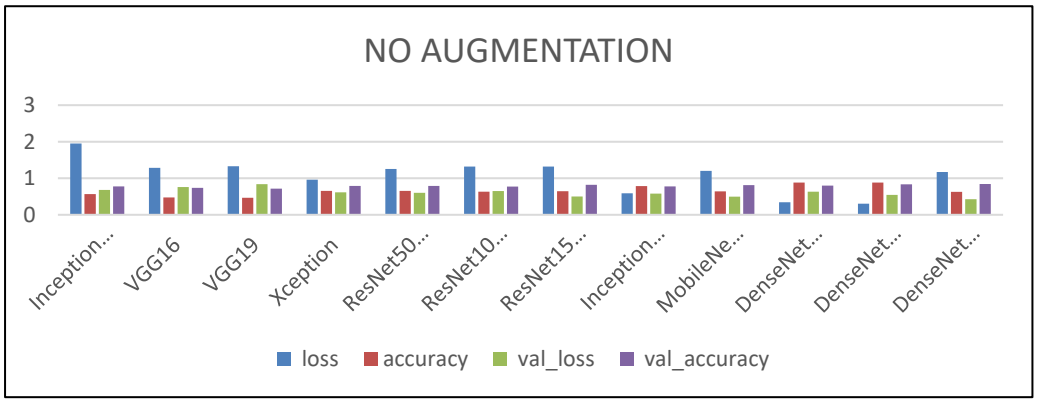
Keras ImageDataGenerators11 is the tool used for generating more training data from the original data to avoid model overfitting. It is conducted online by looping over in small batches during each optimizer iteration. There are some graphic parameters (e.g. rotation, shift, flip, add Gaussian noises) to help generate artificial images. In the above figure some example images generated by using the ImageDataGenerator are shown. Importantly, these images are not included in the original CIFAR-10 image dataset. These generated new image data are mini-batched and discarded after model training.

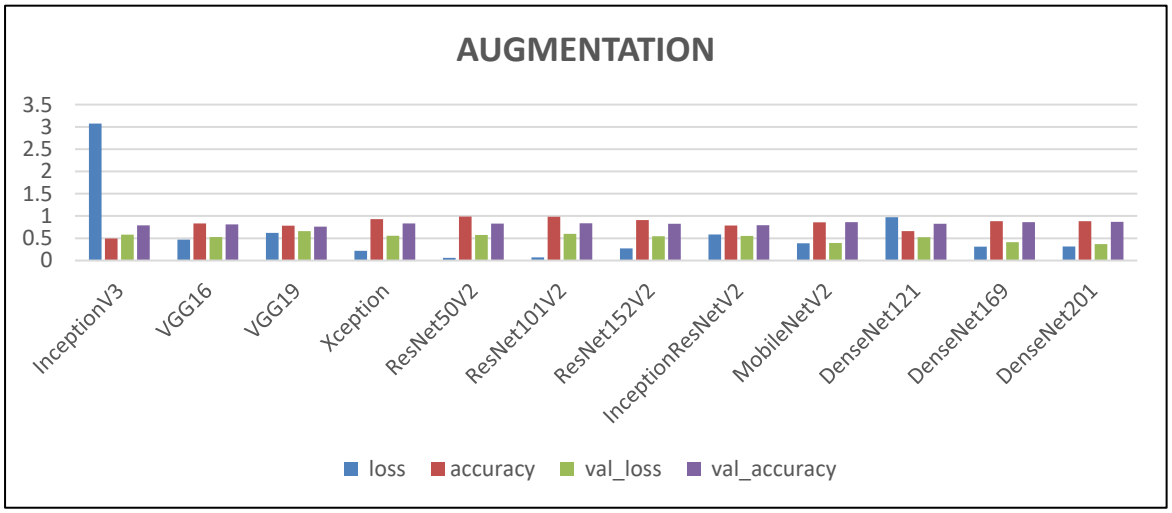
There are various data augmentation techniques: (1) flipping images horizontally or vertically; (2) rotating images at some degrees; (3) rescaling outward or inward; (4) randomly cropping; (5) translating by width and height shifts; (6) whitening, (7) shearing, (8) zooming and (9) adding 10% Gaussian noises to prevent model over-fitting and enhance learning capability

6. RESULTS

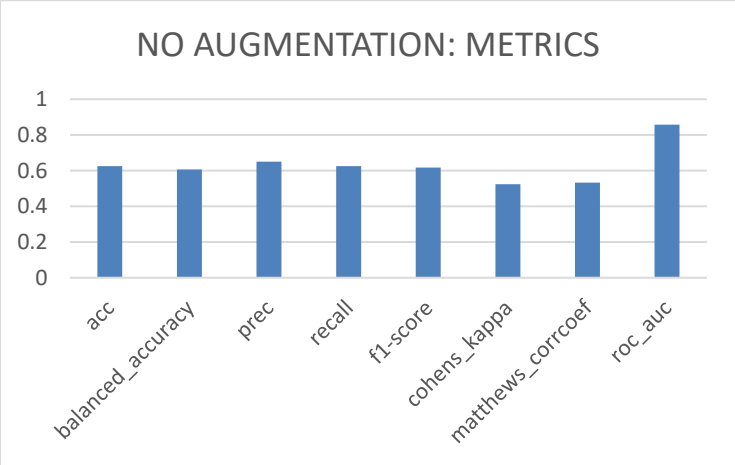
6.1 Accuracy Report

The two bar chart below shows accuracy and loss value for all the pre-trained model we used with and without augmentation. Here we don't see much significant difference between two graphs, no augmentation and with augmentation, because we already had lot of the image in the training images. Data Augmentation techniques can work more effectively, when working with less amount of data or with in balanced dataset. You can see a significant difference in accuracy when you perform data augmentation on a small data set having fewer on training samples.





6.2 Performance Metrics



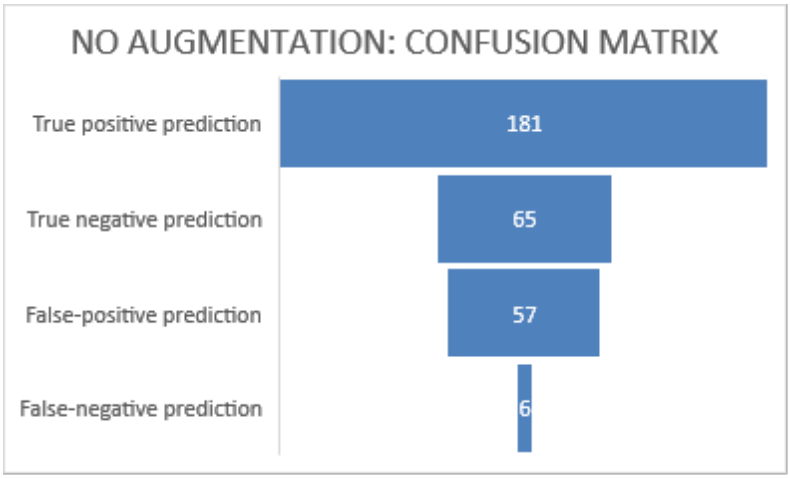
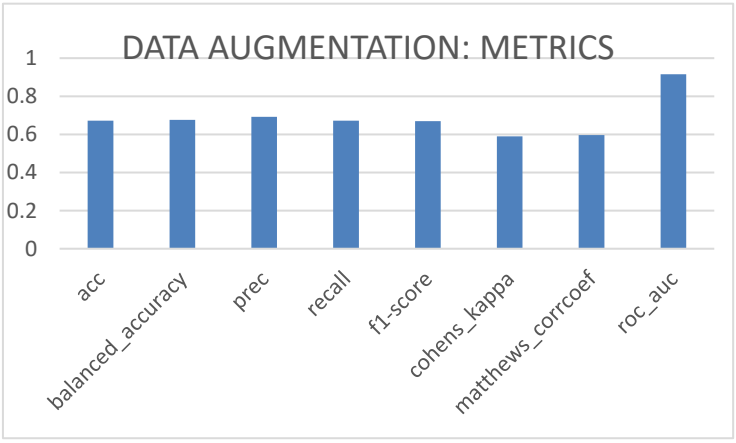


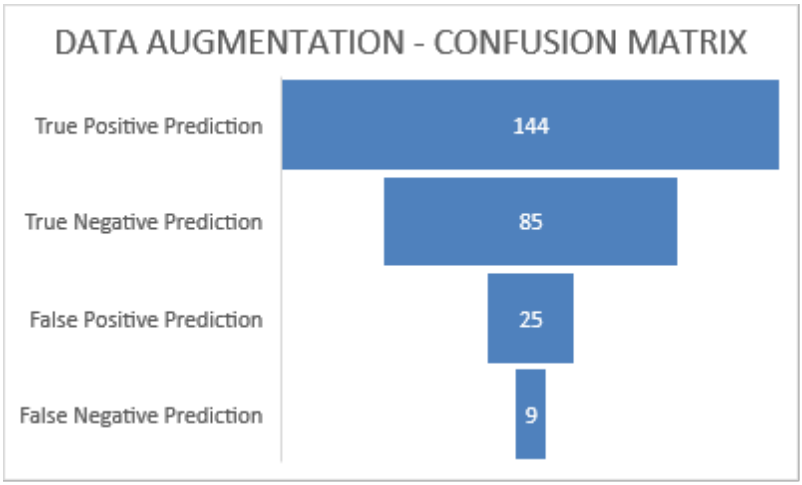
Fig 4: No Augmentation: MetricS

Fig 5: Confusion Matrix

Fig 6: Data Augmentation: Metrics

Fig 7: Data Aug. Confusion Matrix





To evaluate our models, we used metrics such as accuracy, precision, recall, f1-score, Cohen kappa, the one we used in our exam 1 and exam 2. We also used Mathews Correlation Coefficient. The Mathews Correlation Coefficient is used in machine learning to measure the quality of binary two-class classifications

Below bar chart with no augmentation and data augmentation of the pre-trained models with the transfer learning have shown interesting results. No augmentation method do not surpass Data augmentation method results.

5.3. Training & Validation Loss (no augmentation)

From the accuracy and loss curves below, the overfitting is easy to spot. As you can see from the accuracy curve, when training without augmentation, the accuracy on the test set levels are slightly off., while the accuracy on the training set keeps improving. There is a small gap between those two curves, which clearly shows that we are overfitting. This can also be seen when you look at how the loss behaves. Without augmentation, the loss on the training set nicely decreases, but the loss on the test set actually increases. Clearly not ideal!

On the other hand, training with augmentation allows the model to reach a slightly better accuracy of about. There is still a difference between the training and test accuracy, so we are probably still overfitting a bit. But it is a little better of that data augmentation!

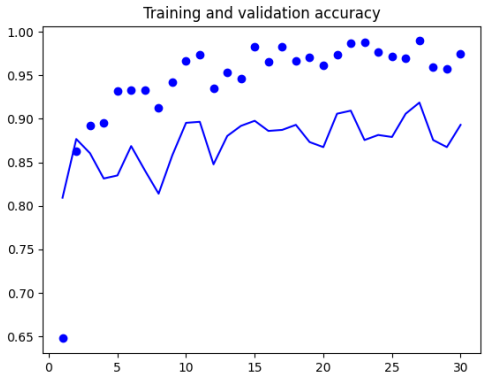


Fig 8: Inception accuracy

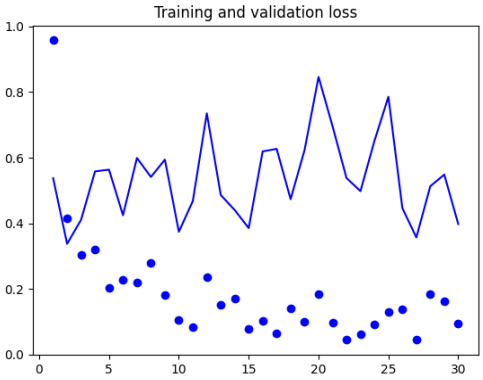


Fig 9: Inception loss

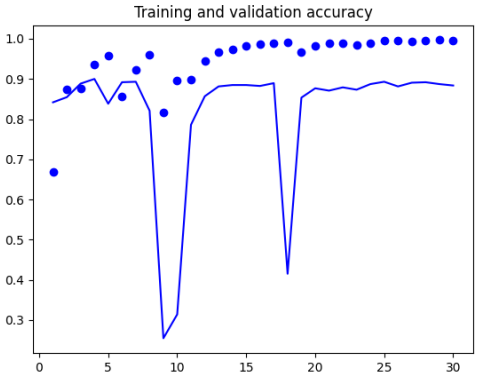


Fig 10: Xception accuracy

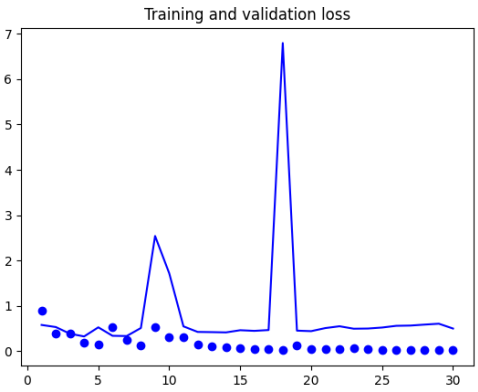


Fig 11: Xception loss

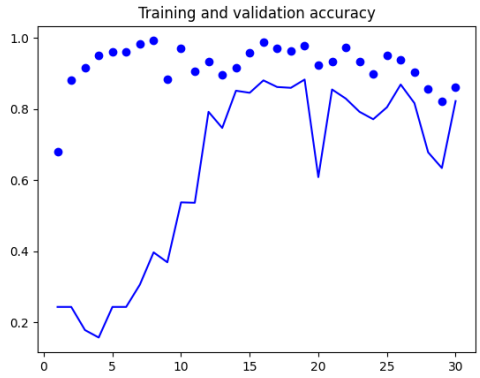


Fig 12: Resnet50 accuracy

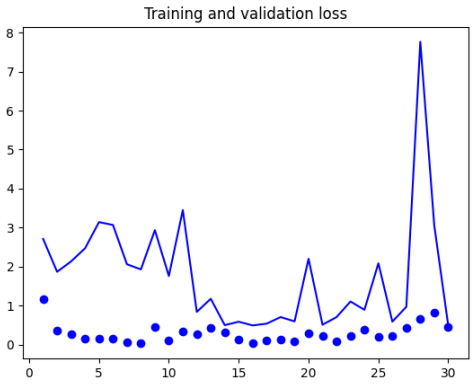


Fig 13: Resnet50 loss

5.4. Training & Validation Loss (with augmentation)

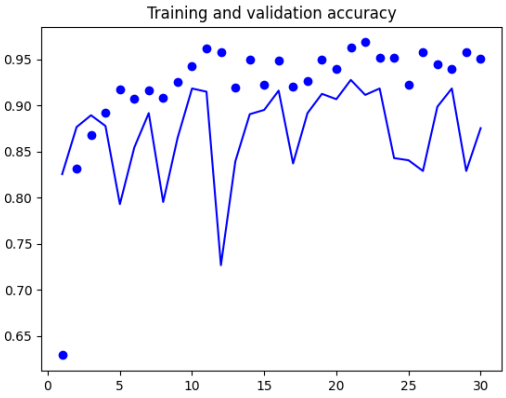


Fig 14: Inception accuracy

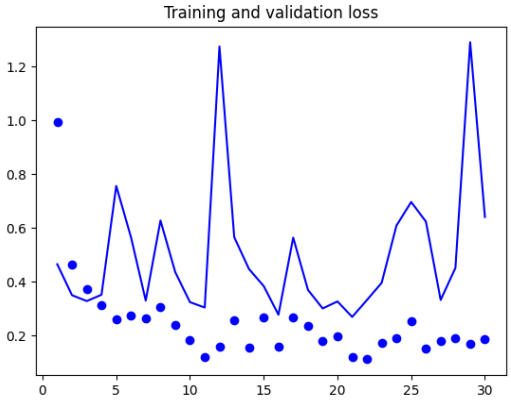


Fig 15: Inception loss

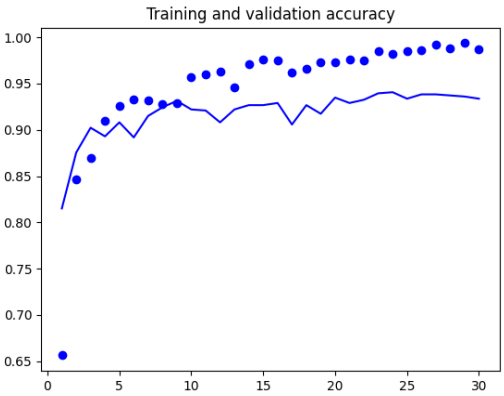


Fig 16: Xception accuracy

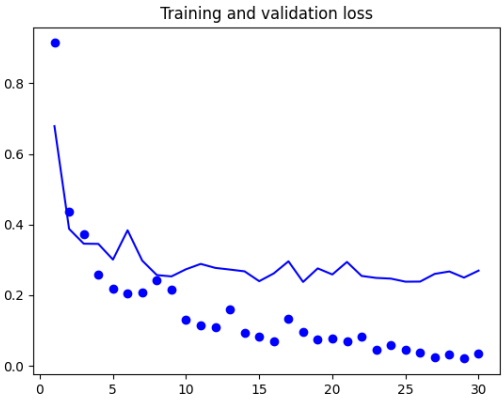


Fig 17: Xception loss

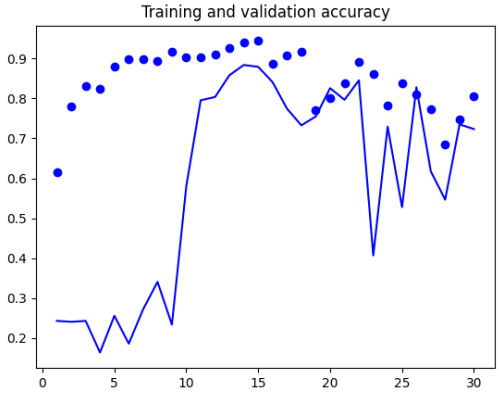


Fig 18: Resnet50 accuracy

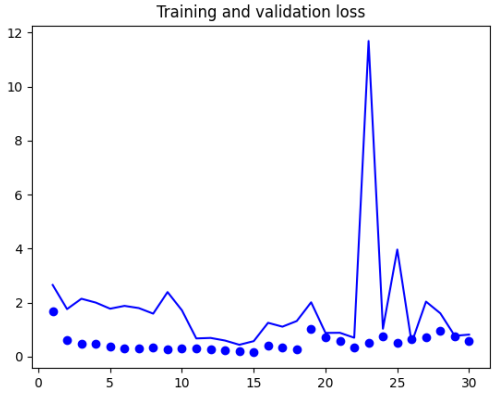


Fig 19: Resnet50 loss

7. KEY INSIGHTS

In this project, we compared twelve different pre-trained models to see which one would be best fit for our study, with data augmentation and without augmentation.

8. CONCLUSION

In this project, we demonstrated the effectiveness of the proposed framework by comparison with state-of-the-art algorithms (pre-trained models) with data augmentation.

The reason for not much significant difference is that, we already had much of the image in the training images. Data Augmentation techniques can work more effectively when working with less amount of data.

It's worth mentioning that despite Data Augmentation is a powerful tool, we should use it carefully. For example, by using too many augmentations in one sequence, we may simply create a new observation that has nothing in common with your original training or testing data.

9. FURTUE PROSPECTS

Automated Data Augmentation and Model patching is becoming a late breaking area. The idea of this project was also to experiment on Neural based transformations with GAN, Generative Adversarial Networks, which provides a new research direction for data augmentation. Training with adversarial samples generated by GANs can improve the generalization ability of deep CNNs, but, we realized that, GANs require considerable time for training and it was difficult to converge.

10. REFERENCES

[1] keras.io

[2] https://keras.io/api/preprocessing/image/

[3] Pratt, L. Y. (1993). "Discriminability-based transfer between neural networks" (PDF). *NIPS Conference: Advances in Neural Information Processing Systems 5*. Morgan Kaufmann Publishers. pp. 204–211.

[4] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". IJCV, 2015.

[5] LeCun, Y. et al., 1998. "Gradient-based learning applied to document recognition." Proceedings of the IEEE, 86(11):2278–2324

[6] Han X., Kashif R., and Roland V., 2017. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms arXiv preprint

[7] Chollet F., 2016. Xception: Deep Learning with Depthwise Separable Convolutions arXiv preprint

[8] Simonyan K. and Zisserman A., 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition arXiv preprint

[9] Szegedy C. et al., 2016. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning arXiv preprint

[10] Sandler M. et al., 2019. MobileNetV2: Inverted Residuals and Linear Bottlenecks arXiv preprint

[11] Zoph B. et al., 2018. Learning Transferable Architectures for Scalable Image Recognition arXiv preprint

[12] He K. et al., 2016. Deep Residual Learning for Image Recognition arXiv preprint

[13] Huang G. et al., 2017. Densely Connected Convolutional Networks arXiv preprint

[14] Szegedy C. et al., 2016. Rethinking the Inception Architecture for Computer Vision arXiv preprint

11. APPENDIX

NO AUGMENTATION				
Model	loss	accuracy	val_loss	val_accuracy
InceptionV3	1.9503	0.5678	0.6798	0.7784
	0.5027	0.8195	0.6003	0.8073
	0.3936	0.8708	0.629	0.8015
VGG16	loss	accuracy	val_loss	val_accuracy
	1.2831	0.4743	0.7613	0.736
	0.7522	0.7135	0.6677	0.7842
	0.64	0.7702	0.5869	0.8112
	0.5402	0.8078	0.6163	0.7958
VGG19	loss	accuracy	val_loss	val_accuracy
	1.3296	0.4666	0.8404	0.7148
	0.8048	0.7174	0.7684	0.7225
	0.7013	0.7381	0.6997	0.7341
	0.6334	0.7755	0.6862	0.7495
	0.574	0.784	0.702	0.7322
Xception	loss	accuracy	val_loss	val_accuracy
	0.9598	0.6534	0.6156	0.7919
	0.3607	0.8701	0.5516	0.817
	0.2674	0.9031	0.5293	0.8189
	0.1686	0.9468	0.5906	0.8189
ResNet50V2	loss	accuracy	val_loss	val_accuracy
	1.2529	0.6549	0.6021	0.7919
	0.293	0.8819	0.5332	0.8073
	0.1452	0.9608	0.6333	0.8054

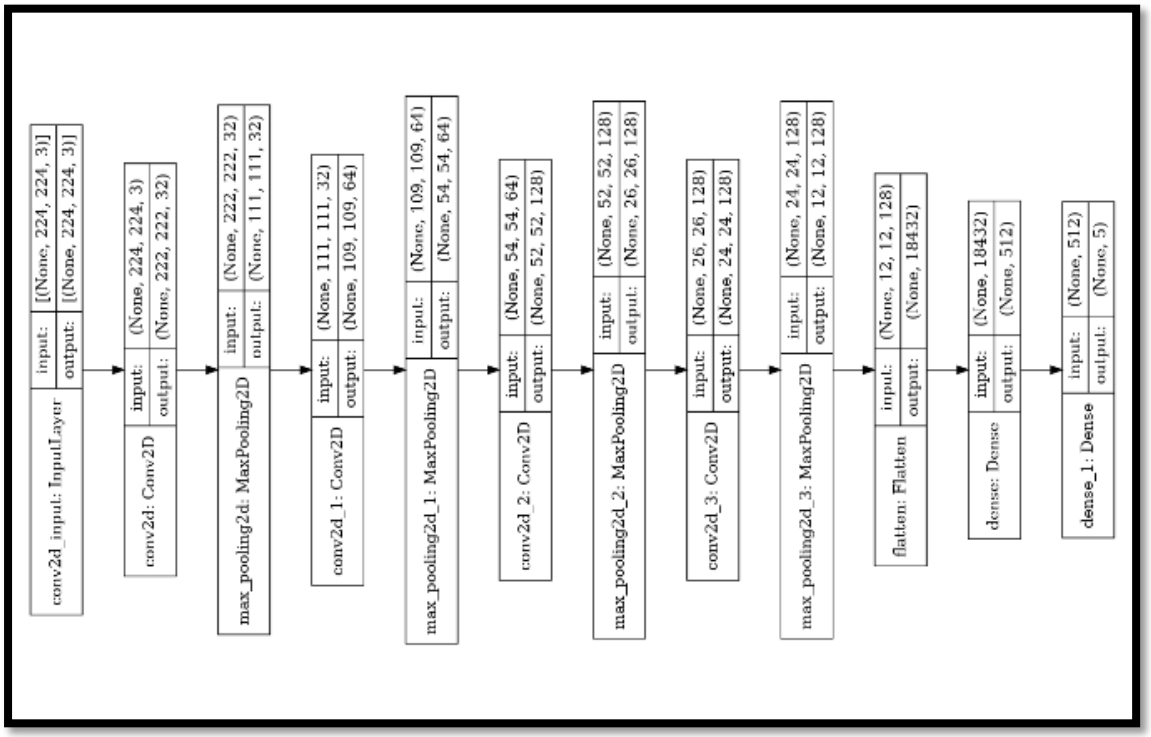
NO AUGMENTATION (continued)				
ResNet101V2	loss	accuracy	val_loss	val_accuracy
	1.3222	0.6308	0.6507	0.7746
	0.2633	0.9128	0.5555	0.8208
	0.1444	0.9515	0.5501	0.8285
	0.0736	0.9818	0.5849	0.8266
ResNet152V2	loss	accuracy	val_loss	val_accuracy
	1.3194	0.6448	0.5005	0.8208
	0.31	0.8891	0.4871	0.8478
	0.1463	0.944	0.5065	0.8382
InceptionResNetV2	loss	accuracy	val_loss	val_accuracy
	2.835	0.5479	0.5429	0.8015
	0.5902	0.7844	0.5806	0.7784
MobileNetV2	loss	accuracy	val_loss	val_accuracy
	1.2011	0.6411	0.4965	0.8131
	0.3558	0.8731	0.4355	0.8324
	0.2167	0.9168	0.42	0.8516
	0.1221	0.9611	0.5025	0.8343
DenseNet121	loss	accuracy	val_loss	val_accuracy
	0.9722	0.6592	0.5225	0.8247
	0.3522	0.8709	0.4753	0.8401
	0.2109	0.9293	0.5041	0.8362
DenseNet169	loss	accuracy	val_loss	val_accuracy
	1.4627	0.6242	0.4403	0.842
	0.302	0.8814	0.5432	0.8343
DenseNet201	loss	accuracy	val_loss	val_accuracy
	1.1724	0.6291	0.4247	0.8439
	0.2727	0.9047	0.4349	0.8439

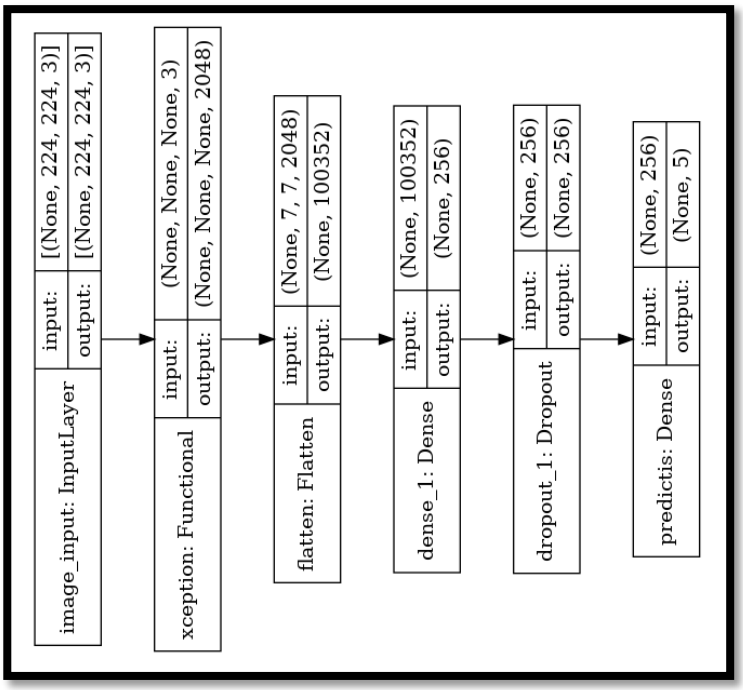
AUGMENTATION				
Model	loss	accuracy	val_loss	val_accuracy
InceptionV3	3.0749	0.4961	0.5809	0.7881
	0.5496	0.7983	0.6107	0.7765
VGG16	loss	accuracy	val_loss	val_accuracy
	1.2717	0.4847	0.7771	0.7245
	0.7468	0.7223	0.715	0.7495
	0.659	0.7545	0.6208	0.7784
	0.5962	0.7912	0.5878	0.8015
	0.5174	0.8169	0.5871	0.8015
	0.4708	0.8346	0.5277	0.8131
	0.4338	0.8418	0.5463	0.8015
VGG19	loss	accuracy	val_loss	val_accuracy
	1.3459	0.4496	0.8686	0.6782
	0.8017	0.7088	0.7699	0.7225
	0.7149	0.7288	0.676	0.763
	0.6208	0.7812	0.6606	0.7611
	0.5716	0.803	0.6869	0.7476
Xception	loss	accuracy	val_loss	val_accuracy
	1.0328	0.6318	0.532	0.7938
	0.348	0.8747	0.5068	0.817
	0.2183	0.9306	0.5568	0.8343
ResNet50V2	loss	accuracy	val_loss	val_accuracy
	1.3953	0.6277	0.5388	0.8015
	0.2702	0.9016	0.5382	0.8227
	0.1537	0.9516	0.5183	0.8227
	0.0612	0.9882	0.5745	0.8285

AUGMENTATION(continued)				
ResNet101V2	loss	accuracy	val_loss	val_accuracy
	1.5403	0.6262	0.5613	0.7996
	0.2981	0.8993	0.5606	0.8073
	0.1673	0.9437	0.5335	0.8343
	0.0707	0.9827	0.5987	0.8362
ResNet152V2	loss	accuracy	val_loss	val_accuracy
	1.3434	0.6293	0.5228	0.817
	0.2743	0.9082	0.5456	0.8247
InceptionResNetV2	loss	accuracy	val_loss	val_accuracy
	3.6756	0.5095	0.6638	0.7399
	0.5842	0.7846	0.5545	0.7919
	0.4591	0.8279	0.6601	0.7803
MobileNetV2	loss	accuracy	val_loss	val_accuracy
	1.2205	0.6387	0.5384	0.8035
	0.3872	0.8593	0.3935	0.8632
	0.2232	0.9271	0.4459	0.8439
DenseNet121	loss	accuracy	val_loss	val_accuracy
	1.3378	0.6222	0.5017	0.817
	0.3421	0.8814	0.6314	0.7996
DenseNet169	loss	accuracy	val_loss	val_accuracy
	1.0067	0.6777	0.5162	0.8324
	0.3129	0.8847	0.4109	0.8632
	0.1709	0.9435	0.5659	0.8208
DenseNet201	loss	accuracy	val_loss	val_accuracy
	1.2237	0.6228	0.4491	0.8362
	0.3169	0.8822	0.3696	0.8671
	0.171	0.9432	0.4239	0.8459

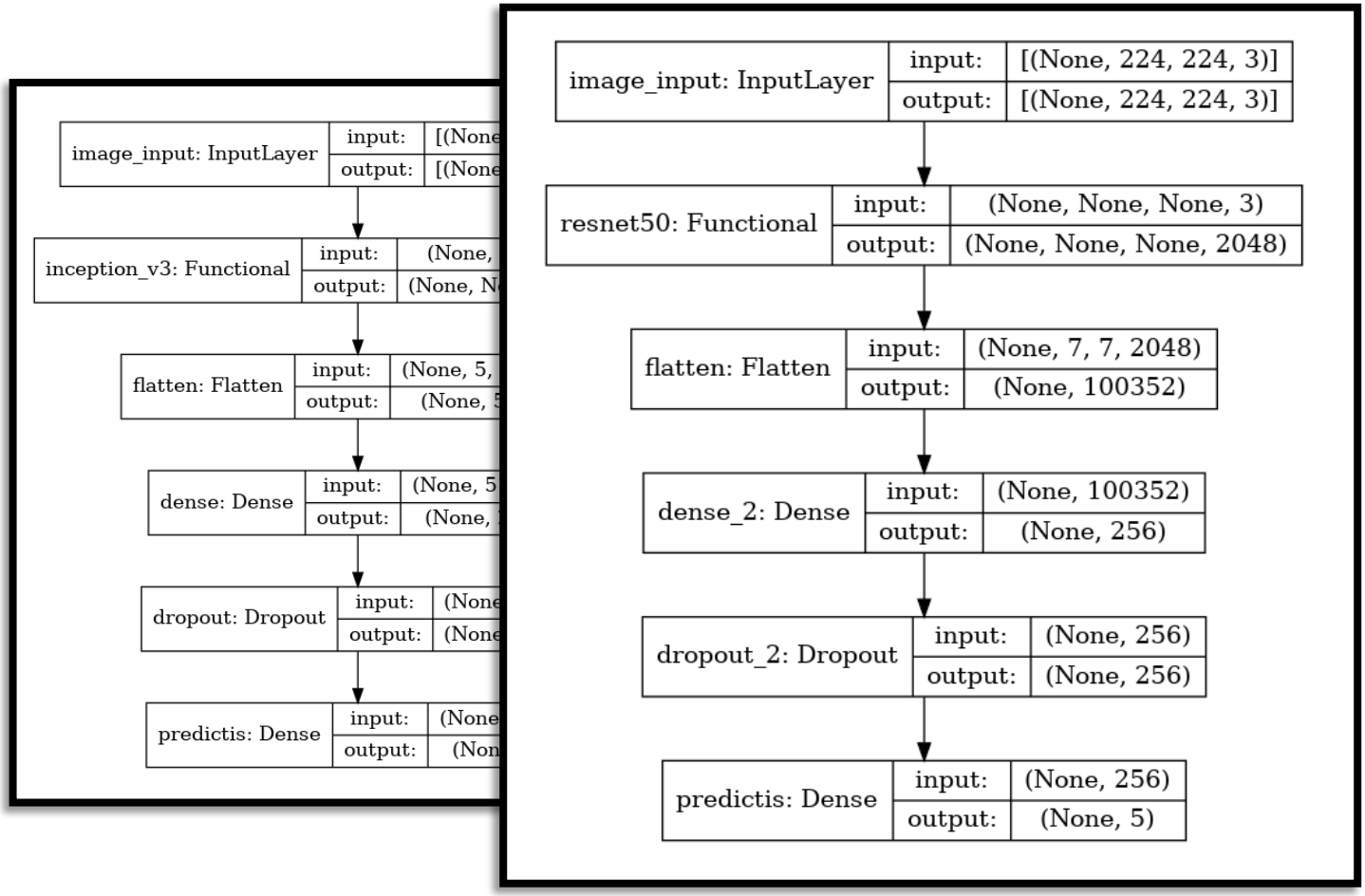
MODEL SUMMARY PLOTS

Model1: Baseline





Model: Xception



Model4: Resnet50