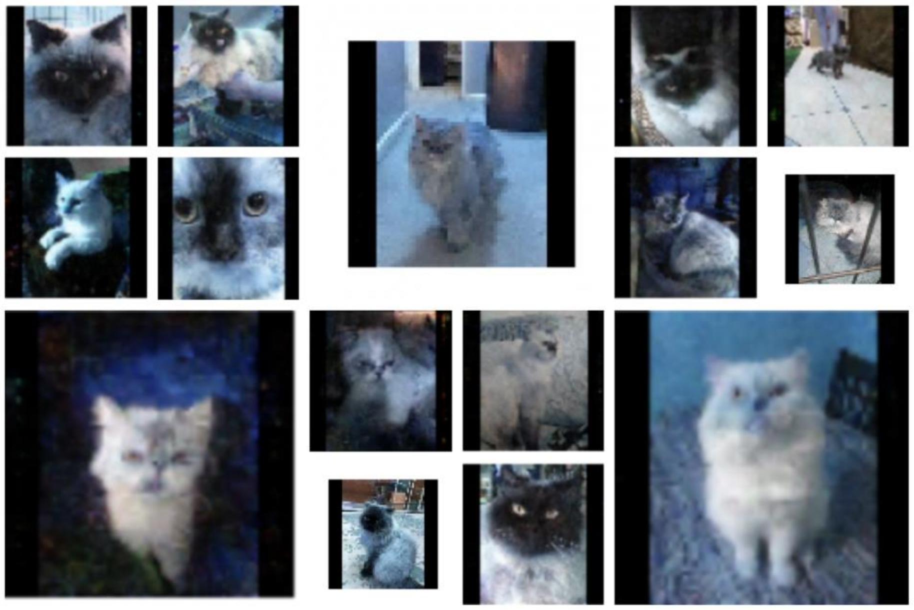
Data Augmentation using GAN to Improve Training of CNN Models

Machine Learning II Project



Dinesh Padmanabhan
Utkarsh Nigam
Professor Amir Jafari
DATS6203

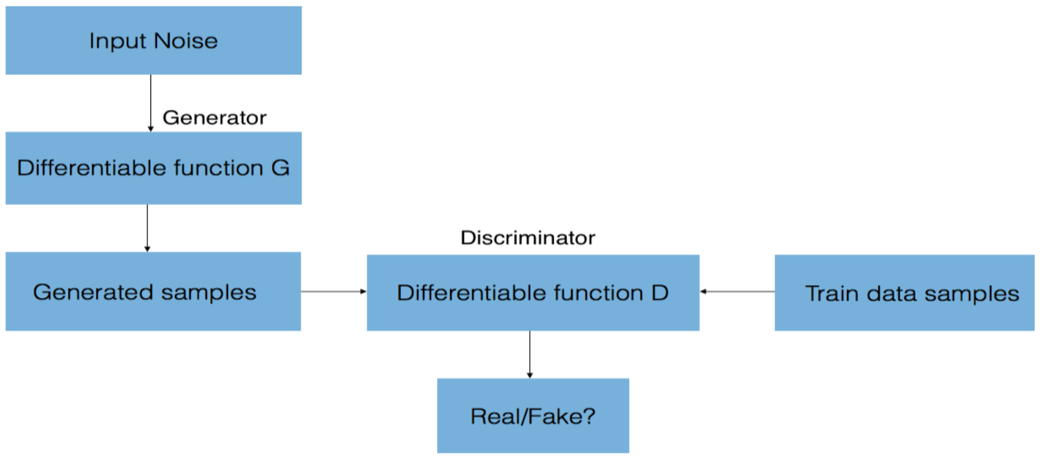
Table of Contents

I. Introduction

When working with machine learning, it is important to have a high-quality dataset to train the algorithm. This means that the data should not only be sufficiently large, to cover as many cases as possible, but also be a good representation of the reality. Unbalanced classes are one of the most frequent struggles when dealing with real data. Is it better to down/up sample, or do nothing at all? Another approach is to generate samples resembling the smallest class. In this project, we are using Generative Adversarial Networks (GANs) to generate samples. Using cats breed, we will determine if a convolutional neural network (CNN) will be trained better with generated samples, or without. In this Project, Generating Adversarial Networks (GANs) were used to try to deal with the previous mentioned case. This will have synthetic data generated by a GAN and then the quality of it will be tested and compared using it to train and test on real data. Finally, this approach will compare its efficiency to other models

1.1 Background of GAN

Of late, generative modelling has seen a rise in popularity. In particular, a relatively recent model called Generative Adversarial Networks or GANs introduced by Ian Goodfellow et al. shows promise in producing realistic samples. The main idea behind GANs can be interpreted as a game between two players- the generator and the discriminator. The generator tries to generate samples that follow the same underlying distribution as the train data. The discriminator tries to distinguish between the samples generated by the generator (fake data), and the actual data from the train set. The goal of the generator is to fool the discriminator by closely approximating the underlying distribution in order to generate samples that are indistinguishable from the actual data. On the other hand, the goal of the discriminator is to identify the fake data from the real data. The discriminator simply has the task of a binary classification problem, where it determines if the data is real or fake.



1.2. Outline of Our Project

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.

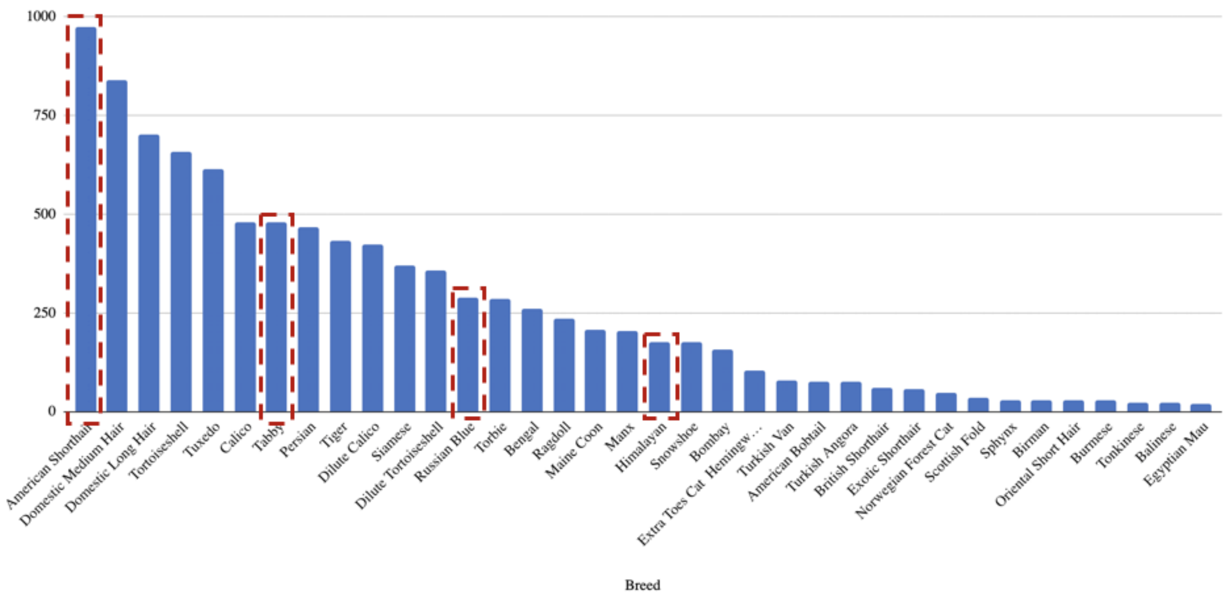
II. Description of the Data Set

2.1. Resource

Our data comes from the Kaggle, since following the steps of Stanford Dogs Dataset, a similar one for cats had to be available. The open source data consists of the image archive. After further study, we believe that the subfolder name of the picture package can fully perform the task of label, so our model only USES the picture compression package

2.2. Overview of the Data Set

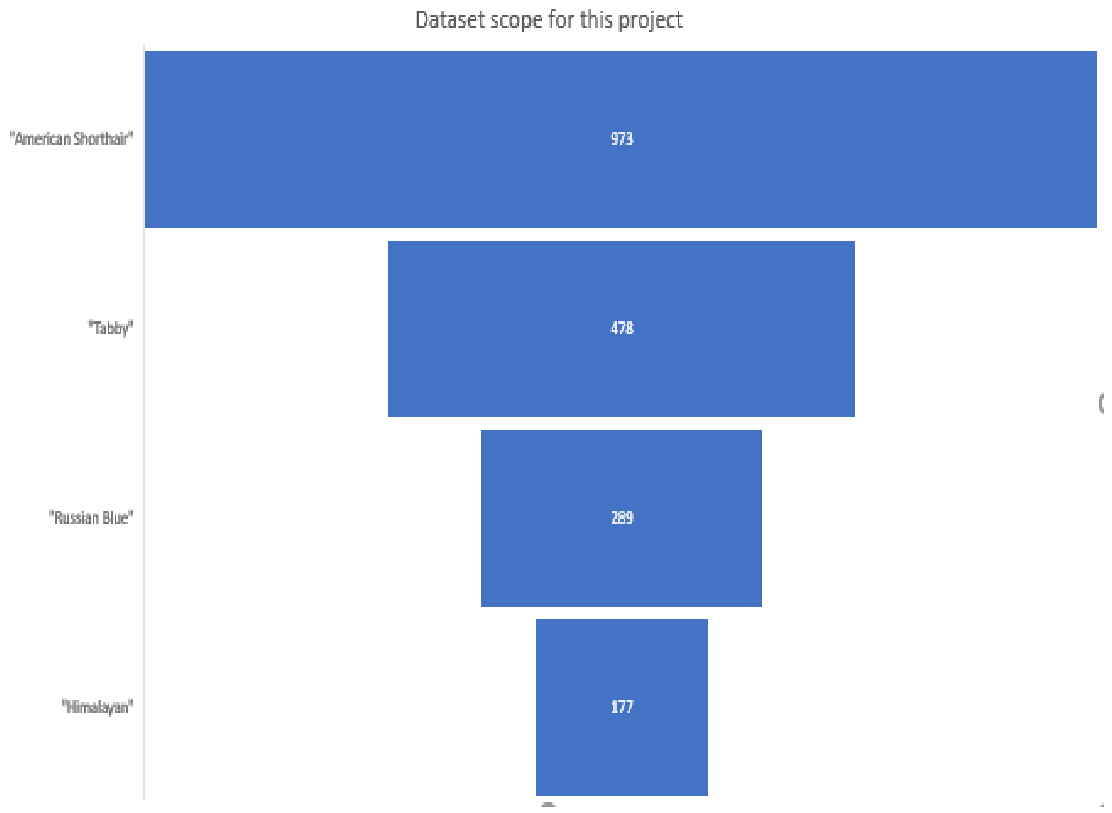
Our dataset contains 67,145 images belonging to 37 cat breeds. All images are colored with RGB channels, but the size and aspect ratio are not fixed, and the proportion of cats (main features) in the picture is skewed. While the amount of data is not too small, there are too many labels, and the data for each category is not significant when evenly distributed. To visualize the data which is skewed below is the bar graph to show the data distribution.



Distribution of Cats breed

2.3. Initial Scope for this project

In this project we are focused to identify only the following four cat breeds "American Shorthair", "Tabby", "Russian Blue" and "Himalayan" by doing so it can be found that this dataset is very imbalanced.



III. Description of Neural Network Architecture

We implemented AlexNet, ResNext, DenseNet and Vgg19 as our main network and tried CGAN for promoting the results.

3.1. ALEXNET

3.1.1. Description

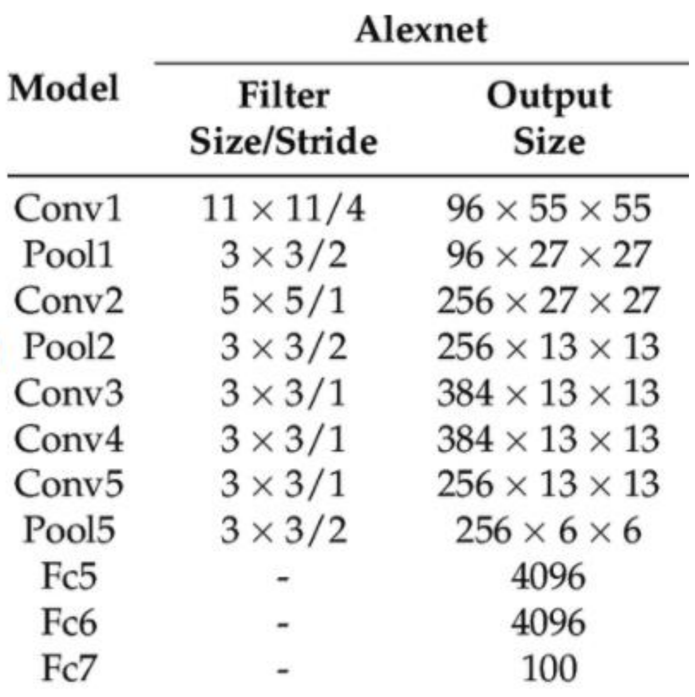
AlexNet is one of the most popular neural network architectures to date. It was proposed by Alex Krizhevsky for the ImageNet Large Scale Visual Recognition Challenge (ILSVRV), and is based on convolutional neural networks. ILSVRV evaluates algorithms for Object Detection and Image Classification. In 2012, Alex Krizhevsky et al. published ImageNet Classification with Deep Convolutional Neural Networks. This is when AlexNet was first heard of.

3.1.2. Configuration

The input dimensions of the network are (256 × 256 × 3), meaning that the input to AlexNet is an RGB (3 channels) image of (256 × 256) pixels.

There are more than 60 million parameters and 650,000 neurons involved in the architecture. To reduce overfitting during the training process, the network uses dropout layers.

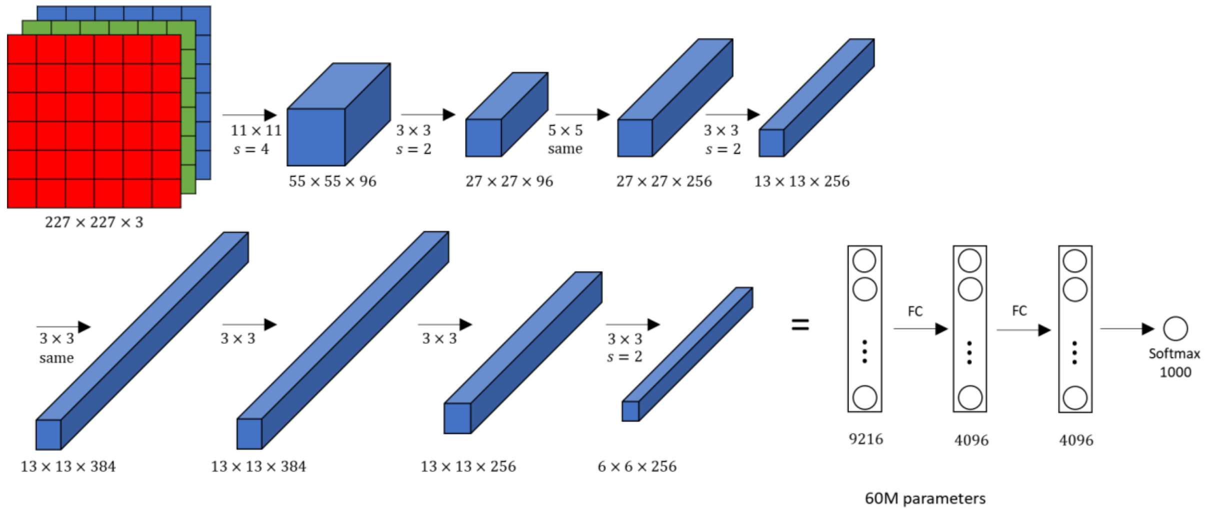
The neurons that are “dropped out” do not contribute to the forward pass and do not participate in backpropagation. These layers are present in the first two fully connected layers.



Model	Alexnet	
	Filter Size/Stride	Output Size
Conv1	$11 \times 11/4$	$96 \times 55 \times 55$
Pool1	$3 \times 3/2$	$96 \times 27 \times 27$
Conv2	$5 \times 5/1$	$256 \times 27 \times 27$
Pool2	$3 \times 3/2$	$256 \times 13 \times 13$
Conv3	$3 \times 3/1$	$384 \times 13 \times 13$
Conv4	$3 \times 3/1$	$384 \times 13 \times 13$
Conv5	$3 \times 3/1$	$256 \times 13 \times 13$
Pool5	$3 \times 3/2$	$256 \times 6 \times 6$
Fc5	-	4096
Fc6	-	4096
Fc7	-	100

3.1.3. Architecture

The architecture is comprised of eight layers in total, out of which the first 5 are convolutional layers and the last 3 are fully-connected. The first two convolutional layers are connected to overlapping max-pooling layers to extract a maximum number of features. The third, fourth, and fifth convolutional layers are directly connected to the fully connected layers. All the outputs of the convolutional and fully connected layers are connected to ReLu non-linear activation function. The final output layer is connected to a softmax activation layer, which produces a distribution of 1000 class labels.



3.2. RESNEXT

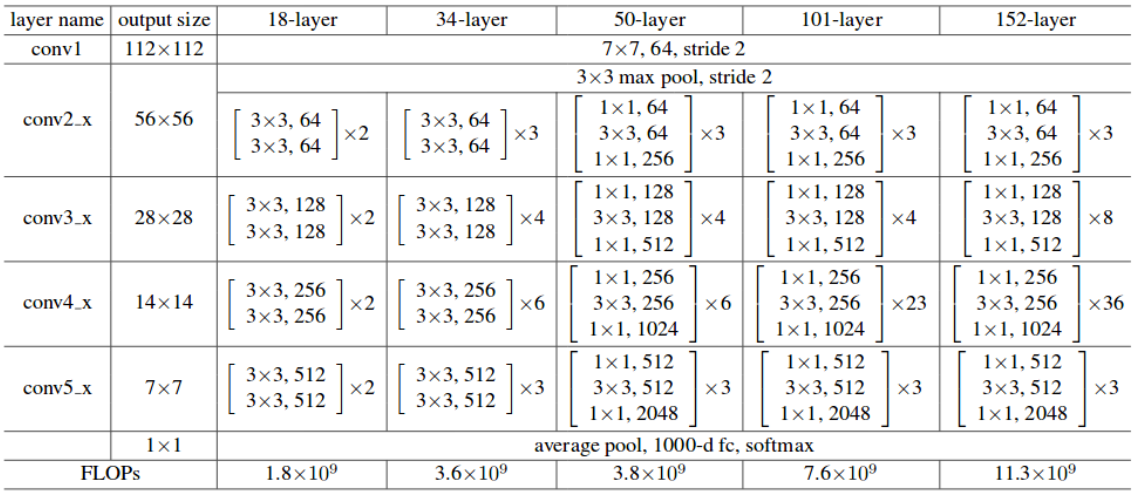
3.2.1. Description

Over the last few years, there have been a series of breakthroughs in the field of Vision. Especially with the introduction of deep Convolutional neural networks, we are getting state of the art results on problems such as image classification and image recognition. So, over the years, researchers tend to make deeper neural networks(adding more layers) to solve such complex tasks and to also improve the classification/recognition accuracy. But, it has been seen that as we go adding on more layers to the neural network, it becomes difficult to train them and the accuracy starts saturating and then degrades also. Here ResNext comes into rescue and helps solve this problem

3.2.2. Configuration

Each of the layers follow the same pattern. They perform 3x3 convolution with a fixed feature map dimension (F) [64, 128, 256, 512] respectively, bypassing the input every 2 convolutions. Furthermore, the width (W) and height (H) dimensions remain constant during the entire layer.

In the table, there is a summary of the output size at every layer and the dimension of the convolutional kernels at every point in the



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

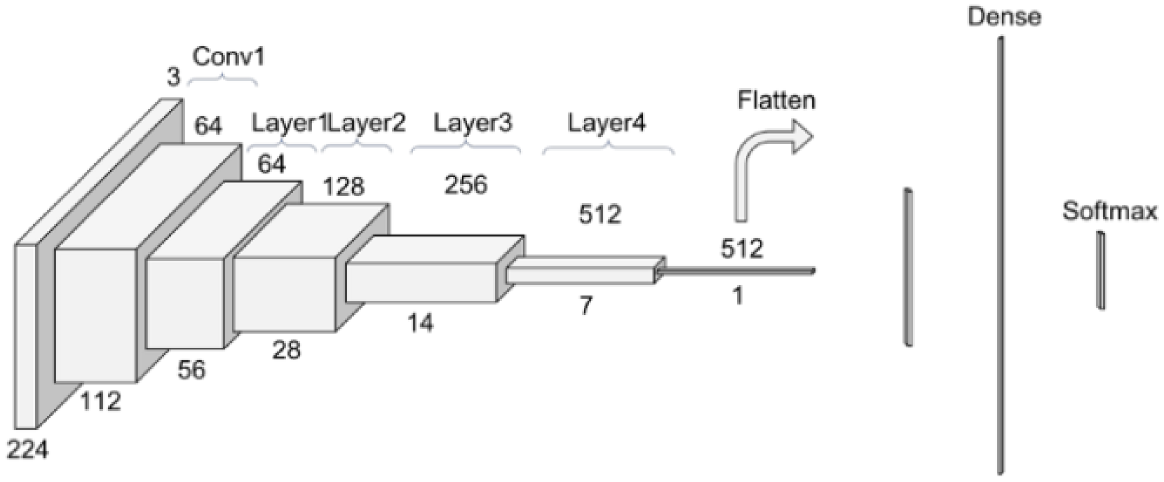
structure.

3.2.3. Architecture

ResNext before entering the common layer behavior is a block — called here Conv1 — consisting on a convolution + batch normalization + max pooling operation. Every layer of a ResNext is composed of several blocks. This is because when ResNexts go deeper, they normally do it by increasing the number of operations within a block, but the number of total layers remains the same — 4. An operation here refers to a convolution a batch normalization and a ReLU activation to an input, except the last operation of a block, that does not have the ReLU. Therefore, in the PyTorch implementation they distinguish between the blocks that includes 2 operations — Basic Block — and the blocks that include 3 operations — Bottleneck Block. Note that normally each of these operations is called layer, but we are using layer already for a group of blocks. We are facing a Basic one now. The input volume is the last output volume from Conv1.

3.3. DENSENET

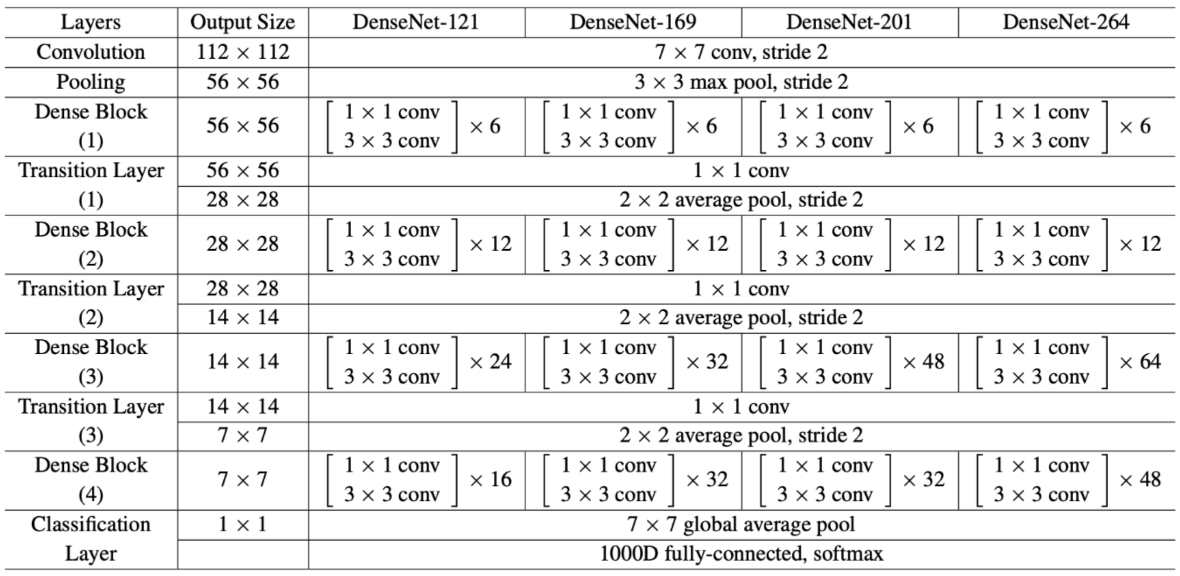
3.3.1. Description



A DenseNet is a type of convolutional neural network that utilises dense connections between layers, through Dense Blocks, where we connect *all layers* (with matching feature-map sizes) directly with each other. To preserve the feed-forward nature, each layer obtains additional inputs from all preceding layers and passes on its own feature-maps to all subsequent layers.

3.3.2. Configuration

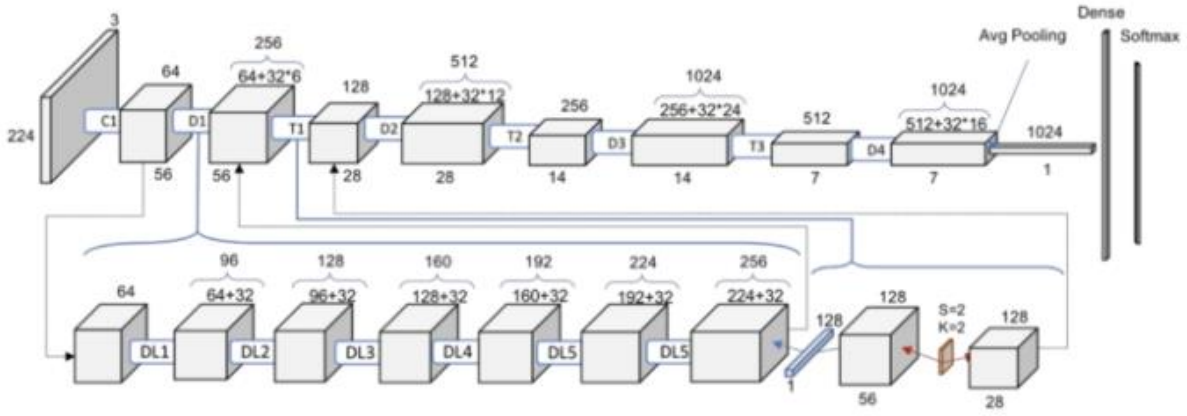
The measures under each volume represent the sizes of the width and depth, whereas the numbers on top represents the feature maps dimension. I included how they are derived to help us understand better incoming step



Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112×112	7×7 conv, stride 2			
Pooling	56×56	3×3 max pool, stride 2			
Dense Block (1)	56×56	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56×56	1×1 conv			
	28×28	2×2 average pool, stride 2			
Dense Block (2)	28×28	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28×28	1×1 conv			
	14×14	2×2 average pool, stride 2			
Dense Block (3)	14×14	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 64$
Transition Layer (3)	14×14	1×1 conv			
	7×7	2×2 average pool, stride 2			
Dense Block (4)	7×7	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$
Classification Layer	1×1	7×7 global average pool			
		1000D fully-connected, softmax			

3.3.3. Architecture

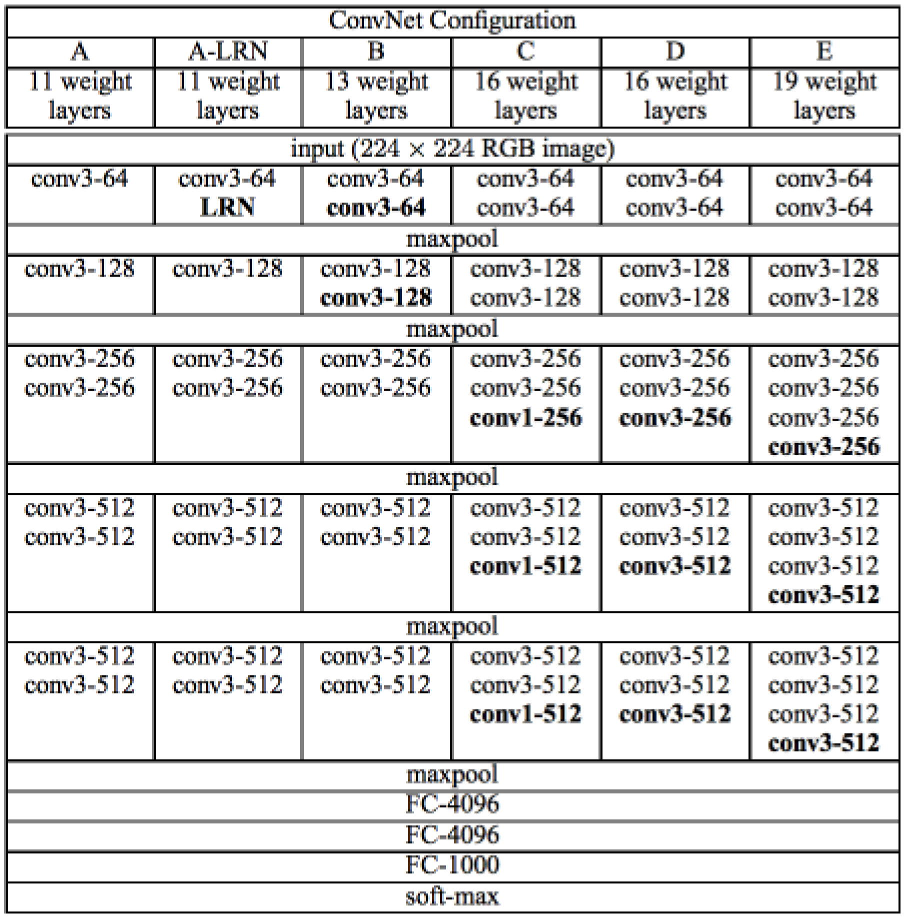
Every layer is adding to the previous volume these 32 new feature maps. This is why we go from 64 to 256 after 6 layers. In addition, Transition Block performs as 1x1 convolution with 128 filters. followed by a 2x2 pooling with a stride of 2, resulting on dividing the size of the volume and the number of feature maps on half.



3.4. VGG19

3.4.1. Description

VGG-19 is a trained Convolutional Neural Network, from **V**isual **G**eometry **G**roup, Department of Engineering Science, University of Oxford. The number 19 stands for the number of layers with trainable weights. 16 Convolutional layers and 3 Fully Connected layers.3.4.2. Configuration

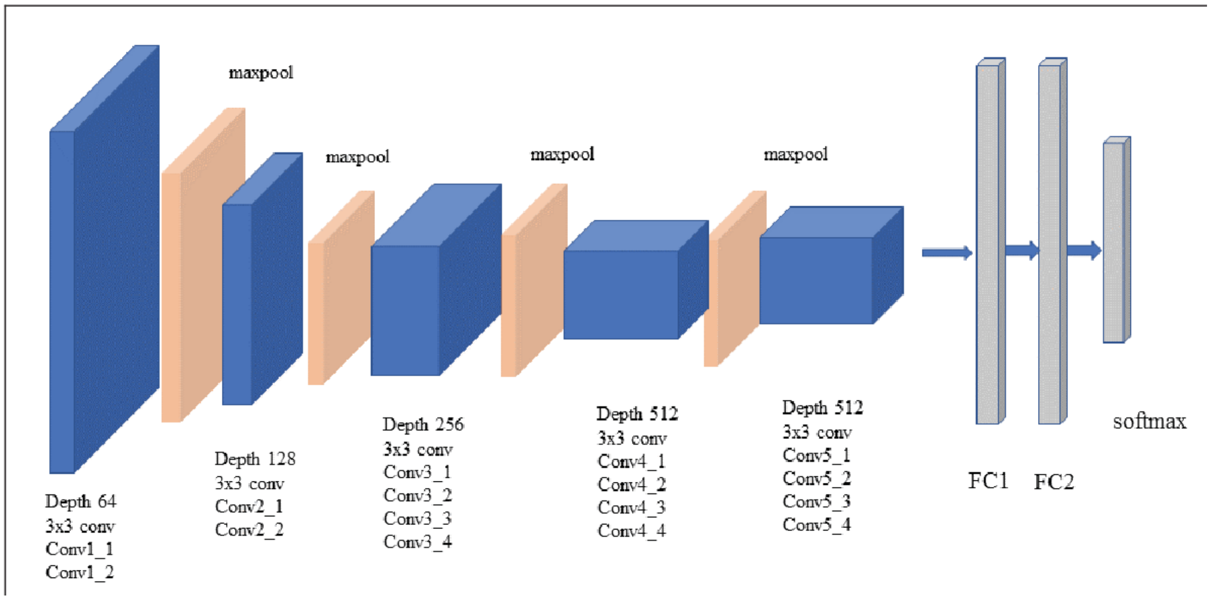


ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

The screenshot is from the original research paper. The different columns A, A-LRN to E shows the different architectures tried by the VGG team. The column E refers to VGG-19 architecture. The VGG-19 was trained on the ImageNet challenge (ILSVRC) **1000-class classification** task. The network takes a (224, 224, 3) RBG image as the input.

3.4.3. Architecture

Below diagram shows how layers are represented. conv<size of the filter>-<number of such filters>. Thus, conv3-64 means 64 (3, 3) square filters. Note that all the conv layers in VGG-19 use (3, 3) filters and that the number of filters increases in powers of two (64, 128, 256, 512). In all the Conv layers, stride length used in 1 (pixel) with a padding of 1 (pixel) on each side.



There are 5 sets of conv layers, 2 of them have 64 filters, next set has 2 conv layers with 128 filters, next set has 4 conv layers with 256 filters, and next 2 sets have 4 conv layers each, with 512 filters. There are max pooling layers in between each set of conv layers. max pooling layers have 2x2 filters with stride of 2 (pixels). The output of last pooling layer is flattened an fed to a fully connected layer with 4096 neurons. The output goes to another fully connected layer with 4096 neurons, whose output is fed into another fully connected layer with 1000 neurons. All these layers are ReLU activated. Finally, there is a softmax layer which uses cross entropy loss.

The layers with trainable weights are only the convolution layers and the Fully Connected layers. maxpool layer is used to reduce the size of the input image where softmax is used to make the final decision.

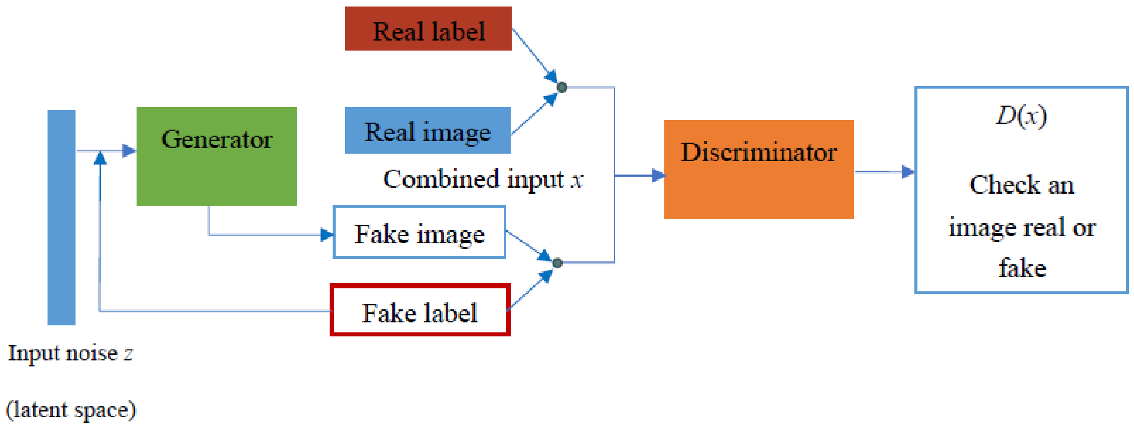
IV. Conditional GAN

4.1.1. Structure of CGAN

The cost function for CGAN is as shown below:

min$max'V(G,D) **= 𝔼0 log4*D*(*x*, *y*)5 + 𝔼7 log41 − *D*(*G*(*z*, *y*7), *y*7)5**

$D(x, y)$ and $G(z, y7)$ indicate that we are distinguishing the label y and generate a picture given by the label $y7$.



Conditional Generative Adversarial Networks (CGAN), an extension of the GAN, allows you to generate images with specific conditions or attributes. Same as GAN, CGAN also has a generator and a discriminator. However, the difference is that both the generator and the discriminator of CGAN receive some extra conditional information, such as the class of the image, a graph, some words, or a sentence. Because of that, CGAN can make generator to generate different types of images, which will prevent generator from generating similar images after multiple trainings. Also, we can control the generator to generate an image which will have some properties we want.

4.1.2. Algorithm

Initialize the Generator and Discriminator.

Single train iteration:

1. Fix the generator, then input the random vectors and random condition (category, image or description) into the generator to get the generated images. In order to train the generator to generate the specific images we want; we need to pull the sample images from the database. Then update the discriminator.

1. To update the discriminator, we have to adjust the parameter of it. For example, we can label the real images as 1, and generated images as 0. In addition, we also need to input the condition of the same attribute of images when we input the real images to discriminator. For fake images, we need to input the same condition used when generating them. After updating, the discriminator is supposed to classify the

real and fake images well. We can regard this problem as a classification problem and training the discriminator as training a classifier.

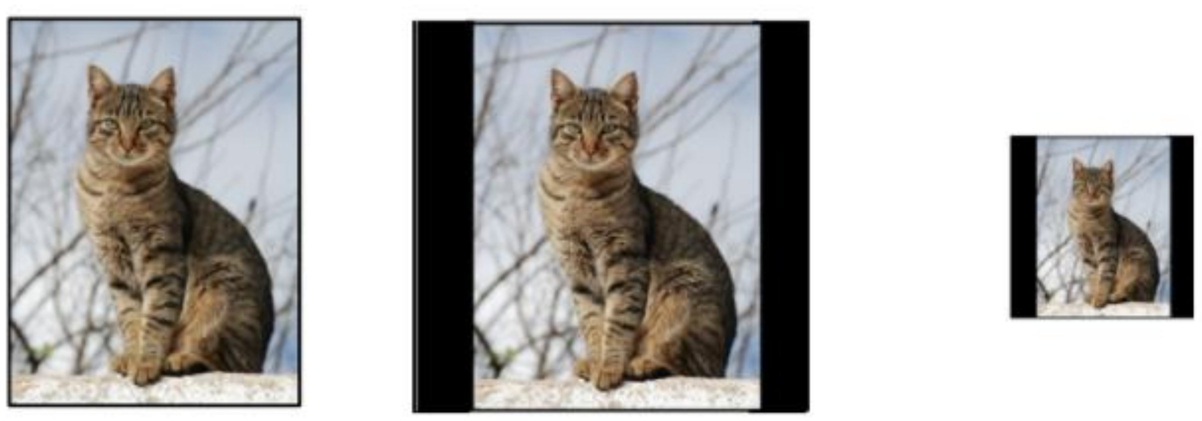
2. After training discriminator, we fix it, and adjust the parameters of generator. The goal of generator training is to generate the image that discriminator can grade it close to 1. This step is similar to the optimizer in the neural network that we learned about, but it's a gradient ascent process.

3. When we put these two processes together, it's a whole big network. As what we showed at the structure introduction. We put in the vector and the condition, then we end up with a number. But if we extract the output from the hidden layer in the middle, we get a complete image.

V. Experimental Setup

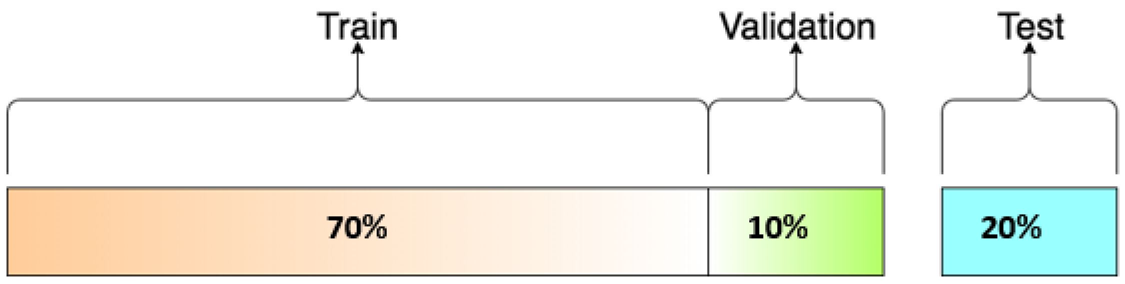
5.1 Pre-Processing

The Original 400x300 pixel raster as obtained from dataset is converted to 400x400 pixel raster upon resizing the image and then finally 64x64 pixel raster upon removing the padding from the resized image.



5.1.1 Training and Validation Sets

For training and testing purposes for our model, we have our data broken down into three distinct datasets. 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

The validation set 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 hyperparameters. 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

The test set 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.

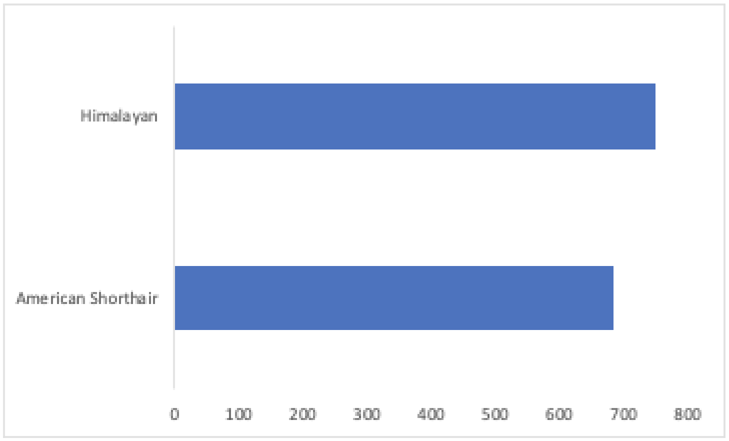
5.2. No Augmentation

After splitting the images were fed to pre trained models AlexNet, ResNext, DenseNet and Vgg 19

5.3. Conventional Data Augmentation

In order to make the most of our few training examples, we "augment" them via a number of random transformations, so that our model would never see twice the exact same picture. This helps prevent overfitting and helps the model generalize better. In this section, we used data augmentation techniques to increase the number of images in our dataset. We used classical methods of data augmentation for image data such as horizontally and Vertical flip images. Horizontally shifting and rotating them while keeping the object of interest in the image. In Keras this can be done via keras.preprocessing.image.ImageDataGenerator class.

Breed	Original Count	Flip	Shift	Rotate	**Final Count**
American Shorthair	685	0	0	0	**685**
Himalayan	125	2	2	1	**750**



5.4. Conditional Generative Adversarial Network (GAN)

5.4.1. Generator

Generator is more like the decoder part. As you feed the generator with a vector or matrix, the Generator will return a new image. From a vector to an image, up sampling methods are always applied into the Generator

5.4.2. Discriminator

Similar to the regular neuron networks (MLP, CNN), Discriminator works as a classifier which only needs to distinguish an image whether it is real or fake. Down sampling methods need to be applied into the Discriminator.

5.4.3. Training Discriminator

First, initializing all weights in this network. Second, using initialized Generator to get a group of fake images by some random noises. Third, the real images randomly chosen from the original dataset is labelled as class 1 and fake images created by Generator are labelled as class 0. Fourth, training Discriminator to classify these combined images and updating the weights of Discriminator.

5.4.4. Training Generator

Before training Generator, let the weights fixed in Discriminator. First, put a group of random noises into Generator to get fake images. Second, put these fake images into the trained Discriminator to see the probability they are real images. Third, to train the Generator, the target of input noise should be class 1. Thus, using the target 1 to calculate loss and gradient so that the weights of Generator will be trained.

VI. Results

6.1 Accuracy Report

Though accuracy is not a good performance metric and None of these findings were particularly useful for us but does imply that there are statistical differences which the GAN may take advantage of to increase its recall for each category of models.

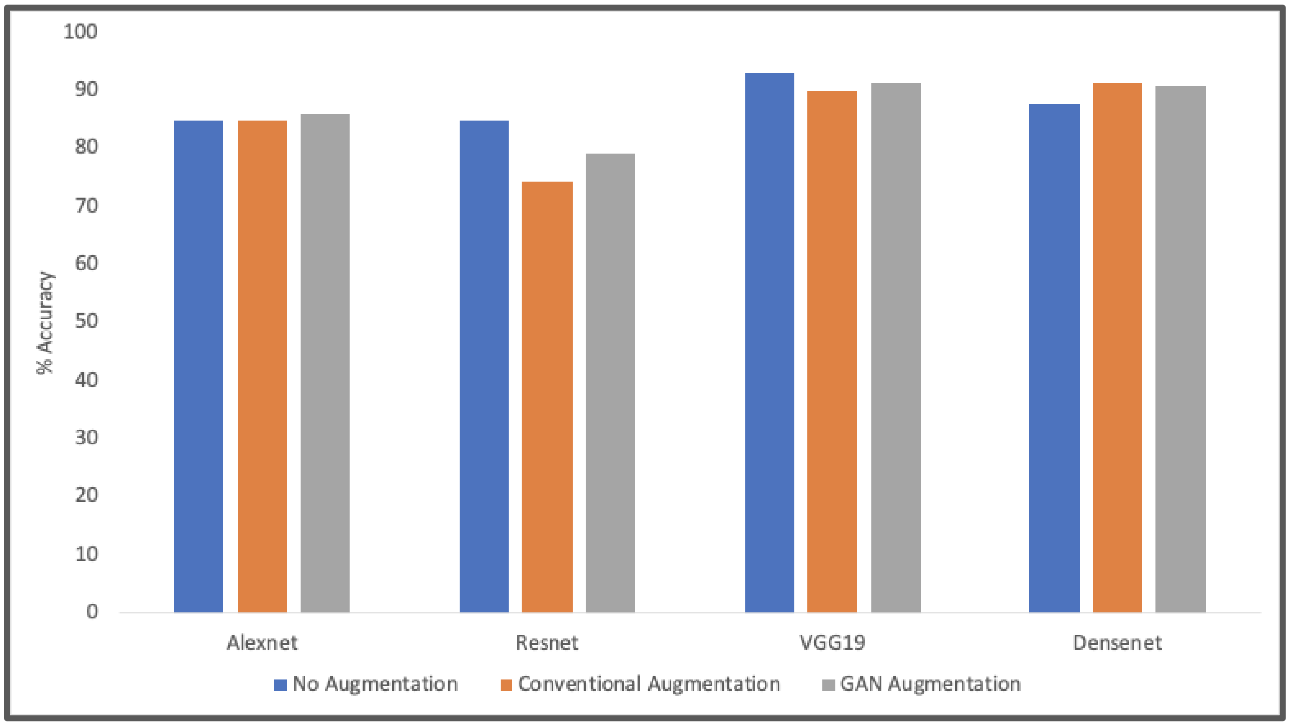
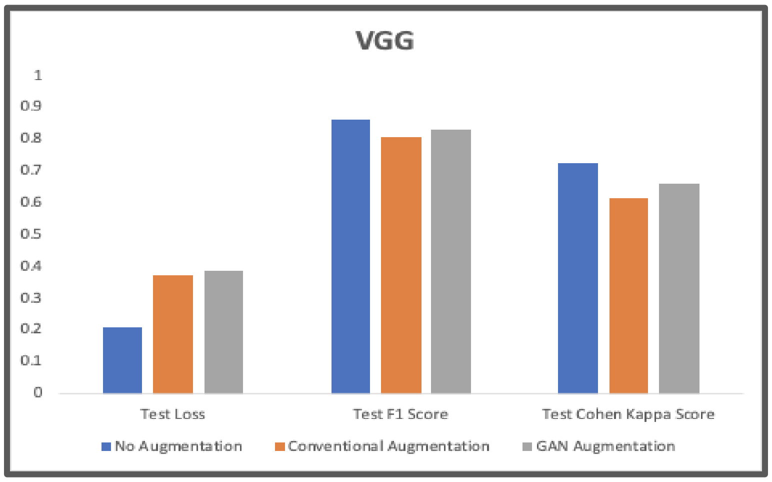


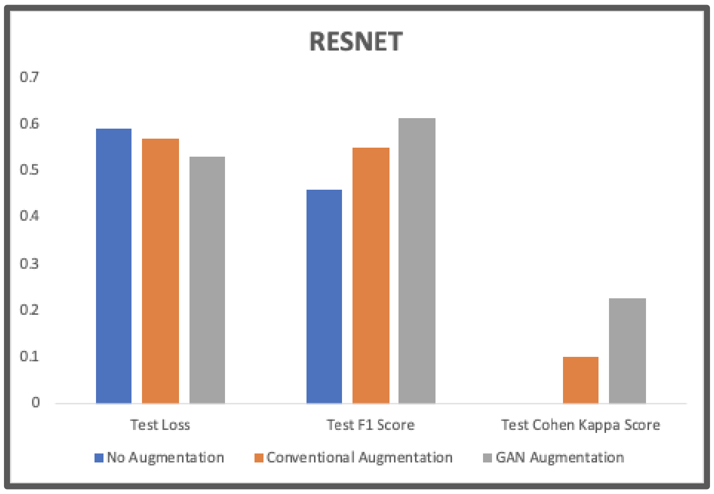
figure: Accuracy scores for pretrained models

6.2 Performance Metrics

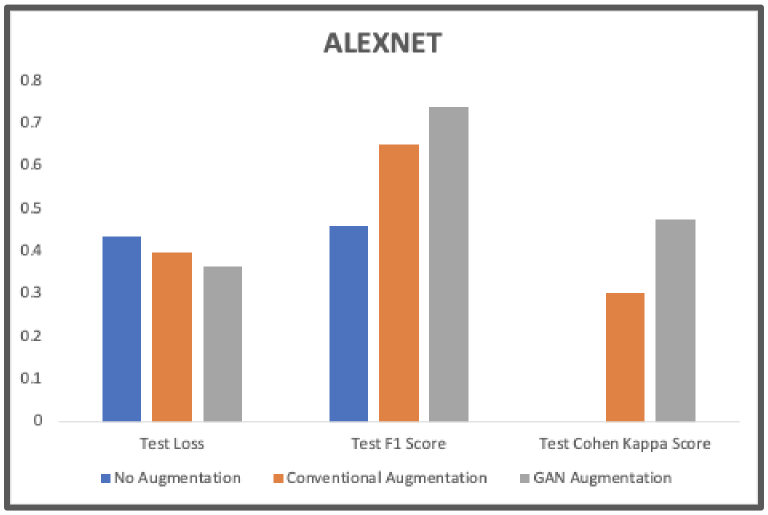
Performance metrics such as Test loss, Test F1 score and Test Cohen Kappa score for Vgg, ResNext, AlexNet and DenseNet is shown below. Besides vgg rest all models for performing fairly well and are improving with respect to versions such as No augmentation, Conventional augmentation and GAN augmentation respectively.



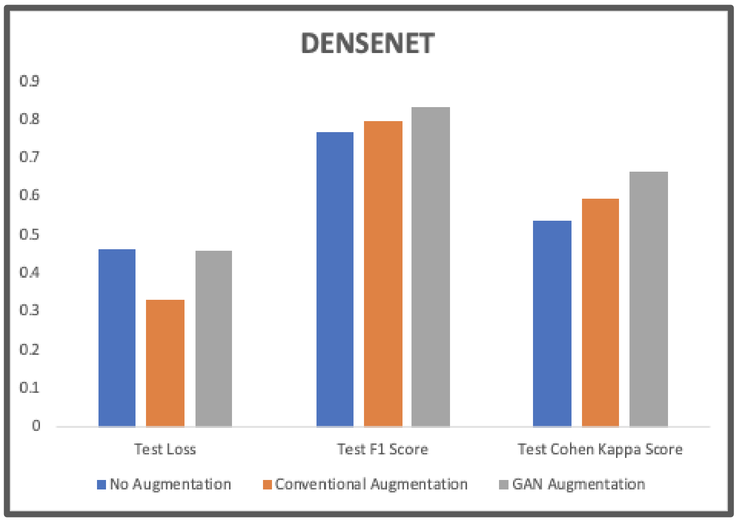
Performance Metrics for Vgg model



Performance Metrics for ResNext model



Performance Metrics for AlexNet model

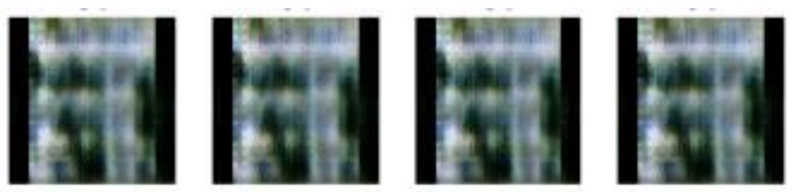


Performance Metrics for Vgg model

6.3 CGAN

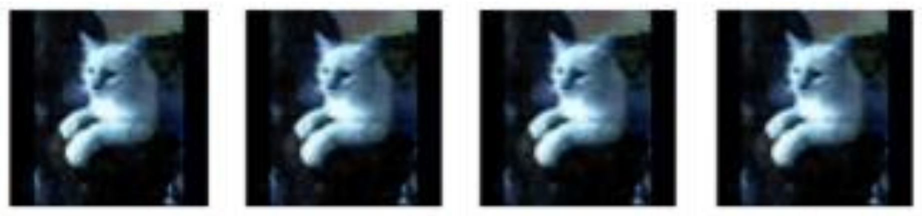
Model Performance

In the early training step, it cannot generate clear images.



Although still a bunch of things, some have the shape of a cat.

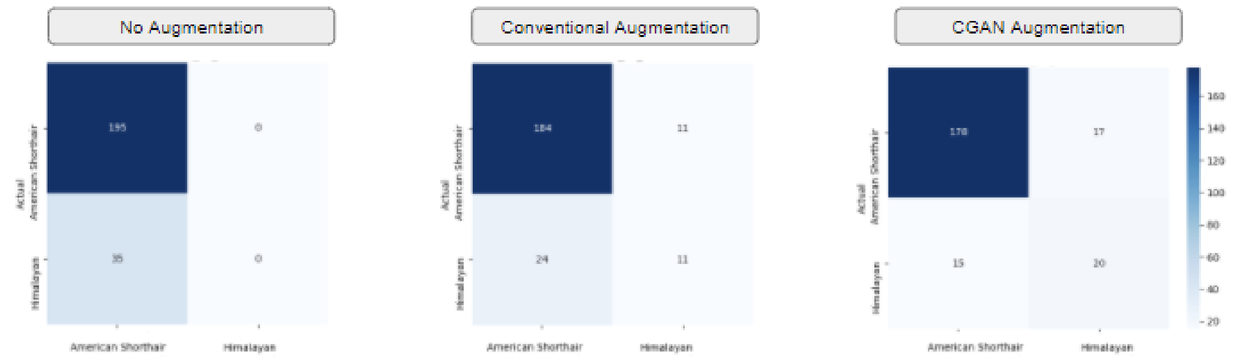
We find that the performance between 2000 and 3000 epochs are much better as the images look like a cat.



VII. Summary & Conclusions

7.1. Summary of Results

From below confusion matrix we can see results are better as we move from No Augmentation to conventional Augmentation and CGAN Augmentation.



7.1. Conclusion

In Conclusion, the purpose of all this is to try and add data to our training sets that result in better Deep Learning models. Amongst the breadth of deep learning research, there lie many creative ways of doing data augmentation. The study of data augmentation is very important for building better models and creating a system such that you do not have to gather an absurd amount of training data to have reliable statistical models.

7.2. Improvements

1. More aggressive data augmentation and aggressive dropout
2. Use of L1 and L2 regularization (also known as "weight decay")
3. Fine-tuning one more convolutional block (alongside greater regularization)

VIII. References

http://datahacker.rs/deep-learning-alexnet-architecture/

https://www.researchgate.net/figure/The-configurations-of-the-Alexnet-and-ZF-5net_tbl1_328948573

https://towardsdatascience.com/extract-features-visualize-filters-and-feature-maps-in-vgg16-and-vgg19-cnn-models-d2da6333edd0

https://towardsdatascience.com/understanding-and-visualizing-densenets-7f688092391a
https://pytorch.org/hub/pytorch_vision_densenet/

https://towardsdatascience.com/understanding-and-visualizing-resnets-442284831be8
https://towardsdatascience.com/generative-adversarial-networks-history-and-overview-7effbb713545

https://github.com/fhtanaka/directed_research_CS_2018

https://github.com/nicolas-gervais/data-augmentation-with-gan-and-vae

https://towardsdatascience.com/extract-features-visualize-filters-and-feature-maps-in-vgg16-and-vgg19-cnn-models-d2da6333edd0

https://machinelearningmastery.com/how-to-develop-a-conditional-generative-adversarial-network-from-scratch/

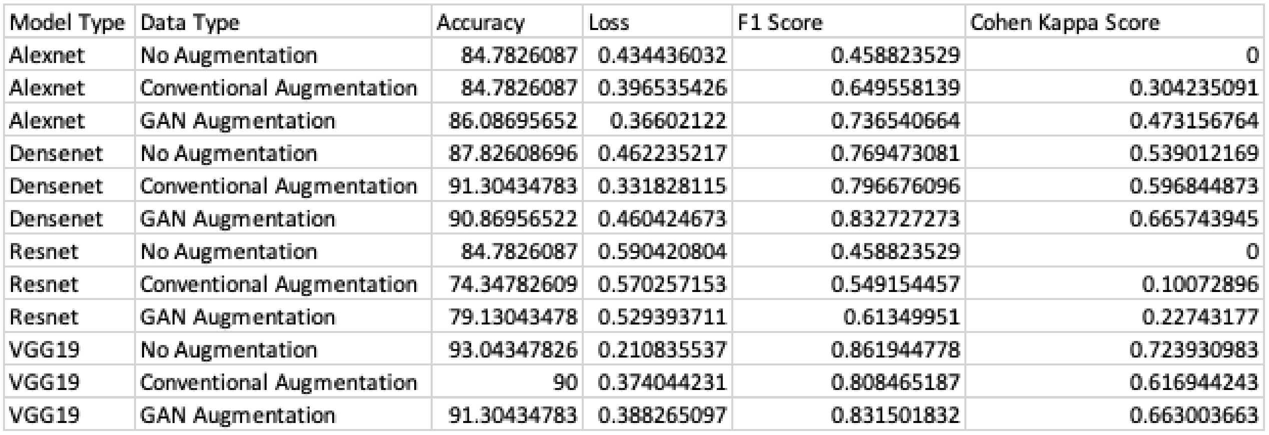
https://towardsdatascience.com/gan-ways-to-improve-gan-performance-acf37f9f59b

https://medium.com/@jonathan_hui/gan-cgan-infogan-using-labels-to-improve-gan-8ba4de5f9c3d

https://medium.com/@jonathan_hui/gan-why-it-is-so-hard-to-train-generative-advisory-networks-819a86b3750b

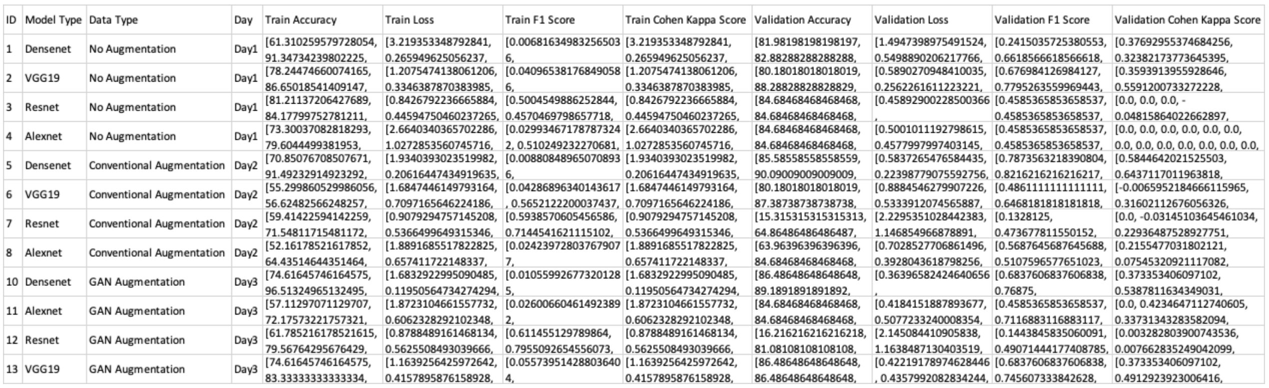
https://medium.com/@jonathan_hui/gan-dcgan-deep-convolutional-generative-adversarial-networks-df855c43
https://medium.com/@utk.is.here/keep-calm-and-train-a-gan-pitfalls-and-tips-on-training-generative-adversarial-networks-edd

Appendix



Model Type	Data Type	Accuracy	Loss	F1 Score	Cohen Kappa Score
Alexnet	No Augmentation	84.7826087	0.434436032	0.458823529	0
Alexnet	Conventional Augmentation	84.7826087	0.396535426	0.649558139	0.304235091
Alexnet	GAN Augmentation	86.08695652	0.36602122	0.736540664	0.473156764
Densenet	No Augmentation	87.82608696	0.462235217	0.769473081	0.539012169
Densenet	Conventional Augmentation	91.30434783	0.331828115	0.796676096	0.596844873
Densenet	GAN Augmentation	90.86956522	0.460424673	0.832727273	0.665743945
Resnet	No Augmentation	84.7826087	0.590420804	0.458823529	0
Resnet	Conventional Augmentation	74.34782609	0.570257153	0.549154457	0.10072896
Resnet	GAN Augmentation	79.13043478	0.529393711	0.61349951	0.22743177
VGG19	No Augmentation	93.04347826	0.210835537	0.861944778	0.723930983
VGG19	Conventional Augmentation	90	0.374044231	0.808465187	0.616944243
VGG19	GAN Augmentation	91.30434783	0.388265097	0.831501832	0.663003663

Summary of results



ID	Model Type	Data Type	Day	Train Accuracy	Train Loss	Train F1 Score	Train Cohen Kappa Score	Validation Accuracy	Validation Loss	Validation F1 Score	Validation Cohen Kappa Score
1	Densenet	No Augmentation	Day1	[61.310259579728054, 91.34734239802225,	[3.219353348792841, 0.265949625056237,	[0.00681634983256503 6,	[3.219353348792841, 0.265949625056237,	[81.98198198198197, 82.88288288288288,	[1.4947398975491524, 0.5498890206217766,	[0.2415035725380553, 0.6618566618566618,	[0.37692955374684256, 0.32382173773645395,
2	VGG19	No Augmentation	Day1	[78.24474660074165, 86.65018541409147,	[1.2075474138061206, 0.3346387870383985,	[0.04096538176849058 6,	[1.2075474138061206, 0.3346387870383985,	[80.18018018018019, 88.28828828828829,	[0.5890270948410035, 0.2562261611223221,	[0.676984126984127, 0.7795263559969443,	[0.3593913955928646, 0.5591200733272228,
3	Resnet	No Augmentation	Day1	[81.21137206427689, 84.17799752781211,	[0.8426792236665884, 0.44594750460237265,	[0.5004549886252844, 0.4570469798657718,	[0.8426792236665884, 0.44594750460237265,	[84.68468468468468, 84.68468468468468,	[0.45892900228500366 ,	[0.4585365853658537, 0.4585365853658537,	[0.0, 0.0, 0.0, - 0.04815864022662897,
4	Alexnet	No Augmentation	Day1	[73.30037082818293, 79.6044499381953,	[2.6640340365702286, 1.0272853560745716,	[0.02993467178787324 2, 0.510249232270681,	[2.6640340365702286, 1.0272853560745716,	[84.68468468468468, 84.68468468468468,	[0.5001011192798615, 0.4577997997403145,	[0.4585365853658537, 0.4585365853658537,	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
5	Densenet	Conventional Augmentation	Day2	[70.85076708507671, 91.49232914923292,	[1.9340393023519982, 0.20616447434919635,	[0.00880848965070893 6,	[1.9340393023519982, 0.20616447434919635,	[85.58558558558559, 90.09009009009009,	[0.5837265476584435, 0.22398779075592756,	[0.7873563218390804, 0.8216216216216217,	[0.584464202152550З, 0.6437117011963818,
6	VGG19	Conventional Augmentation	Day2	[55.299860529986056, 56.62482566248257,	[1.6847446149793164, 0.7097165646224186,	[0.04286896340143617 , 0.5652122200037437,	[1.6847446149793164, 0.7097165646224186,	[80.18018018018019, 87.38738738738738,	[0.8884546279907226, 0.5333912074565887,	[0.4861111111111111, 0.6468181818181818,	[-0.0065952184666115965, 0.31602112676056326,
7	Resnet	Conventional Augmentation	Day2	[59.41422594142259, 71.54811715481172,	[0.9079294757145208, 0.5366499649315346,	[0.5938570605456586, 0.7144541621115102,	[0.9079294757145208, 0.5366499649315346,	[15.315315315315313, 64.86486486486487,	[2.2295351028442383, 1.146854966878891,	[0.1328125, 0.473677811550152,	[0.0, -0.03145103645461034, 0.22936487528927751,
8	Alexnet	Conventional Augmentation	Day2	[52.16178521617852, 64.43514644351464,	[1.8891685517822825, 0.657411722148337,	[0.02423972803767907 7,	[1.8891685517822825, 0.657411722148337,	[63.96396396396396, 84.68468468468468,	[0.7028527706861496, 0.3928043618798256,	[0.5687645687645688, 0.5107596577651023,	[0.2155477031802121, 0.07545320921117082,
10	Densenet	GAN Augmentation	Day3	[74.61645746164575, 96.51324965132495,	[1.6832922995090485, 0.11950564734274294,	[0.01055992677320128 5,	[1.6832922995090485, 0.11950564734274294,	[86.48648648648648, 89.1891891891892,	[0.36396582424640656 ,	[0.6837606837606838, 0.76875,	[0.373353406097102, 0.5387811634349031,
11	Alexnet	GAN Augmentation	Day3	[57.11297071129707, 72.17573221757321,	[1.8723104661557732, 0.6062328292102348,	[0.02600660461492389 2,	[1.8723104661557732, 0.6062328292102348,	[84.68468468468468, 84.68468468468468,	[0.4184151887893677, 0.5077233240008354,	[0.4585365853658537, 0.7116883116883117,	[0.0, 0.4234647112740605, 0.33731343283582094,
12	Resnet	GAN Augmentation	Day3	[61.785216178521615, 79.56764295676429,	[0.8788489161468134, 0.5625508493039666,	[0.611455129789864, 0.7955092654556073,	[0.8788489161468134, 0.5625508493039666,	[16.216216216216218, 81.08108108108108,	[2.145084410905838, 1.1638487130403519,	[0.1443845835060091, 0.49071444177408785,	[0.003282803900743536, 0.007662835249042099,
13	VGG19	GAN Augmentation	Day3	[74.61645746164575, 83.33333333333334,	[1.1639256425972642, 0.4157895876158928,	[0.05573951428803640 4,	[1.1639256425972642, 0.4157895876158928,	[86.48648648648648, 86.48648648648648,	[0.42219178974628446 , 0.4357992082834244,	[0.6837606837606838, 0.745607333842628,	[0.373353406097102, 0.4912923923006416,

Summary of training