**Project Overview:**

Body condition scoring (BCS) is a useful management tool for distinguishing differences in nutritional needs of beef cows in the herd. This system uses a numeric score to estimate body energy reserves in the cow. Research indicates that there is a strong link between the body condition of a cow and her reproductive performance. The percentage of open cows, calving interval, and calf vigor at birth are all closely related to the body condition of cows both at calving and during the breeding season. All these factors play an important role in the economics of a beef cow-calf operation and help determine the percentage of viable calves each year. Monitoring body condition using the BCS system is an important managerial tool for assessing production efficiency.

**Problem Statement**



The BCS dataset was saved in local disk in respective folder labelled by identifying BCS condition in the image such as Class\_1, Class\_2, Class\_3, Class\_4.

The goal is to predict the likelihood that the BCS condition is from a certain class from the provided classes, thus making it a multi-class classification problem in machine learning terms.

Four target classes are provided in this dataset: Class\_1, Class\_2, Class\_3, Class\_4.

The goal is to train a CNN that would be able to classify the weather condition into these Four classes.

Deep-learning based techniques (CNNs)has been very popular in the last few years where they consistently outperformed traditional approaches for feature extraction to the point of winning ImageNet challenges. In this project, transfer learning along with data augmentation will be used to train a convolutional neural network to classify images of BCS to their respective classes.

Transfer learning is referred as a machine learning method where a model developed for a task is reused as the starting point for a model on a second task or in other words,Transfer learning refers to the process of using the weights from pre-trained networks on large dataset. As the pre-trained networks have already learnt how to identify lower level features such as edges, lines, curves etc with the convolutional layers which is often the most computationally time-consuming parts of the process, using those weights help the network to converge to a good score faster than training from scratch.

To train a CNN model from scratch successfully, the dataset needs to be huge (which is definitely not the case here, the available dataset is very small, only 132 images for training and validation) . networks such as ResNet, Inception, VGG pretrained on ImageNetchallenge is available for use publicly. I'll be using all of them and comparing them for best accuracy.

**Metrics**

Here, I’m using the metric for this project is multi-class logarithmic loss (also known as categorical cross entropy)

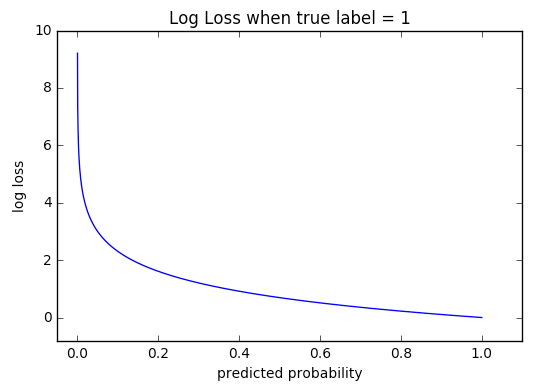
Here, each image has been labelled with one true class and for each image a set of predicted

probabilities should be submitted. ***N*** is the number of images in the test set, ***M*** is the number of image class labels, ***log*** is the natural logarithm, ***yij*** is 1 if observation ***i*** belongs to class ***j*** and 0 otherwise, and ***pij*** is the predicted probability that observation ***i*** belongs to class ***j***.

A perfect classifier will have the ***log-loss*** of *0*.

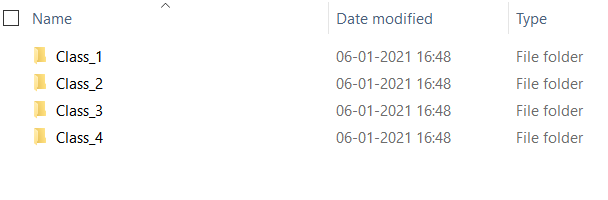
Multiclass *log-loss* punishes the classifiers which are confident about an incorrect prediction. In the above equation, if the class label is 1(the instance is from that class) and the predicted probability is near to 1(classifier predictions are correct), then the loss is really low as ***log(x) → 0*** as ***x → 1***, so this instance contributes a small amount of loss to the total loss and if this occurs for every single instance(the classifiers is accurate) then the total loss will also approach 0.

On the other hand, if the class label is 1(the instance is from that class) and the predicted probability is close to 0(the classifier is confident in its mistake), as ***log(0)*** is undefined it approaches →ꝏ so theoretically the loss can approach infinity. In order to avoid the extremes of the log function, predicted probabilities are replaced with *max(min(p, 1-1015),1015)* . Graphically[^1] , assuming the ***i th*** instance belongs to class ***j*** and ***yij*** = 1 , it's shown that when the predicted probability approaches 0, loss can be very large.

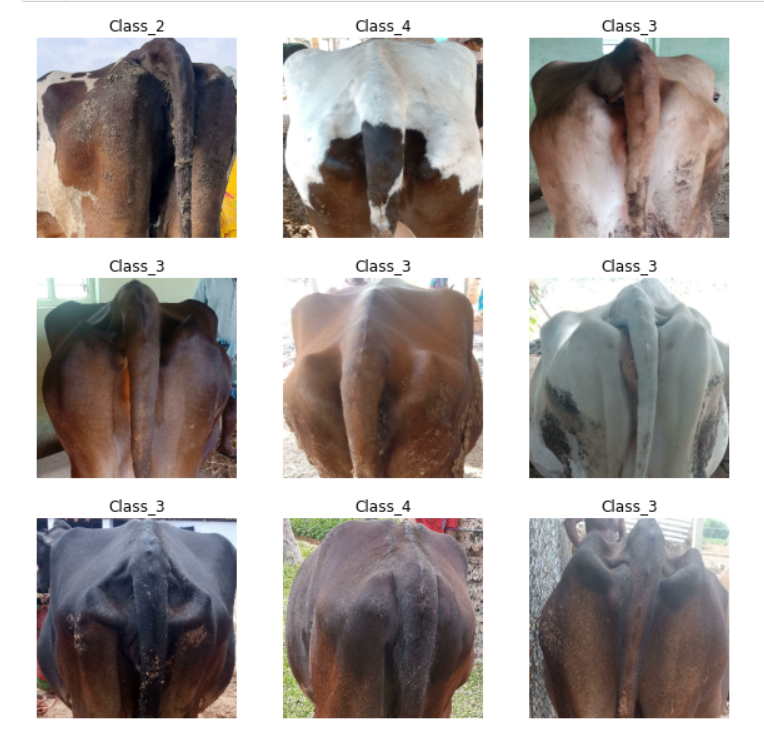


**Analysis**

Each image has only one BCS category and are saved in separate folder as of the labelled class. The images are saved in folders as shown in following figure:



Sample Different Class Images :

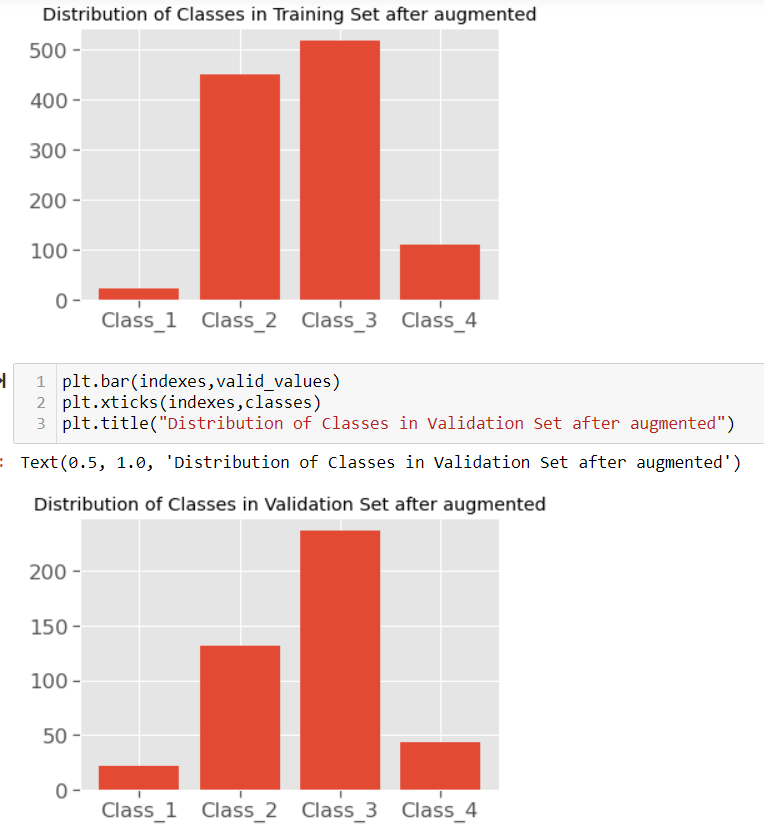


**Exploratory Visualization**

The number of images of each class in BCS dataset is imabalance. The distribution of these classes of dataset can be seen in following figure:

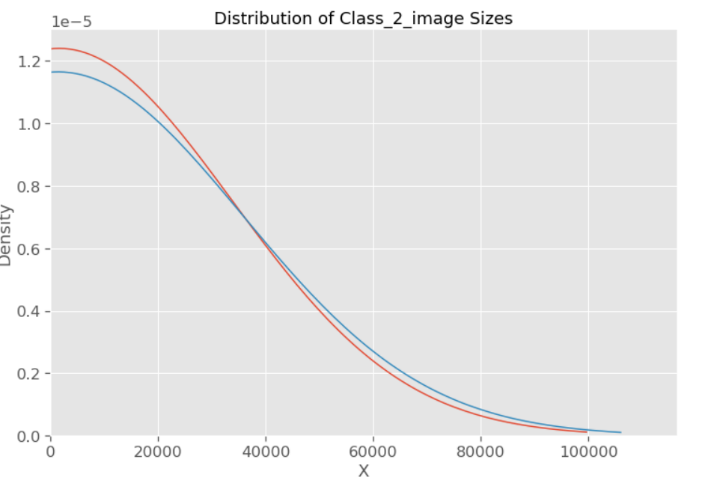
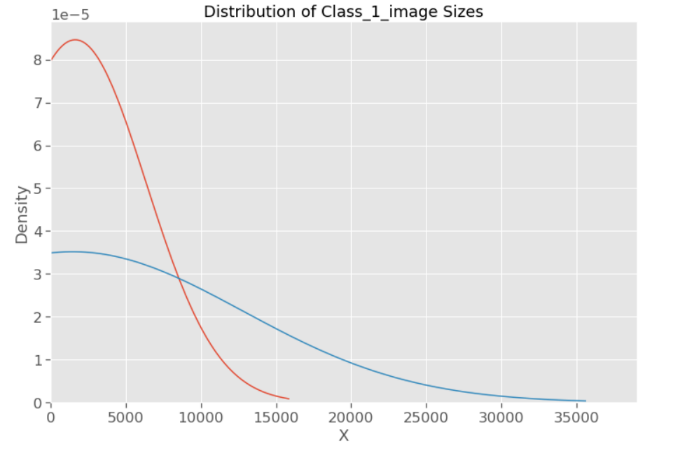
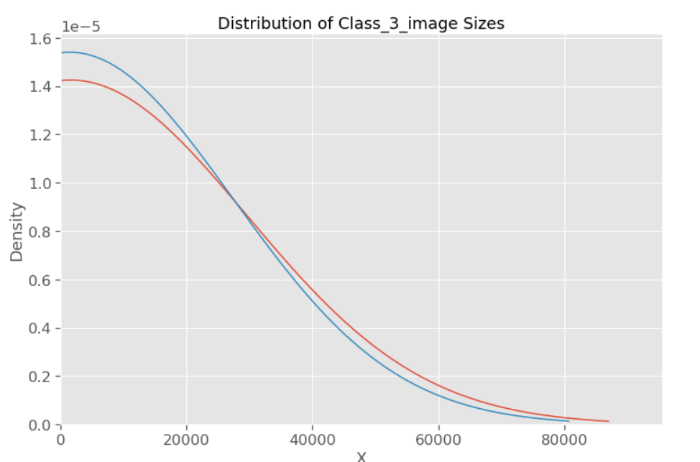
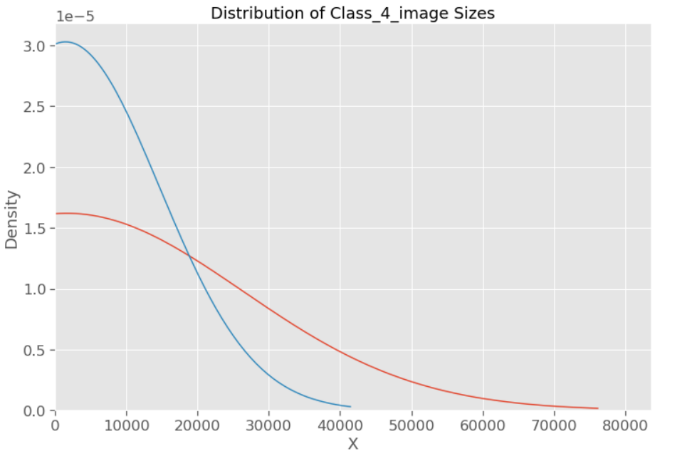


Because of less data we had applied image augmentation techniques on different classes so that we got some more images . the distribution of data is show below after augmented.



**Kernel-density plot of image sizes**

*As known as* Density Plots, Density Trace Graph*.* A Density Plot visualises the distribution of data over a continuous interval or time period. This chart is a variation of a Histogram that uses kernel smoothing to plot values, allowing for smoother distributions by smoothing out the noise. The peaks of a Density Plot help display where values are concentrated over the interval. An advantage Density Plots have over Histograms is that they're better at determining the distribution shape because they're not affected by the number of bins used (each bar used in a typical histogram). A Histogram comprising of only 4 bins wouldn't produce a distinguishable enough shape of distribution as a 20-bin Histogram would. However, with Density Plots, this isn't an issue.



**Algorithms and Techniques**

**Transfer Learning**:

Transfer learning refers to the process of using the weights of a pretrained network trained on a large dataset applied to a different dataset (either as a feature extractor or by finetuning the network). Finetuning refers to the process of training the last few or more layers of the pretrained network on the new dataset to adjust the weight. Transfer learning is very popular in practice as collecting data is often costly and training a large network is computationally expensive. Here, in this case the dataset is very small so weights from a convolutional neural network pretrained on ImageNet dataset is finetuned to classify weather condition.

**Benchmark method**:

**CNN:** A convolutional neural network (**CNN**, or **ConvNet**) is a class of deep neural networks, most commonly applied to analysing visual imagery. CNNs are regularized versions of multilayer perceptron (fully connected networks).

The name “convolutional neural network” indicates that the network employs a mathematical operation called convolution. Convolution is a specialized kind of linear operation. Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers.

A convolutional neural network consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of a series of convolutional layers that convolve with a multiplication or other dot product. The activation function is commonly a RELU layer, and is subsequently followed by additional convolutions such as pooling layers, fully connected layers and normalization layers, referred to as hidden layers because their inputs and outputs are masked by the activation function and final convolution. The final convolution, in turn, often involves backpropagation in order to more accurately weight the end product.

**Convolutional**: When programming a CNN, the input is a tensor with shape *(number of*

*images) x (image width) x (image height) x (image depth)*. Then after passing through a

convolutional layer, the image becomes abstracted to a feature map, with shape *(number of*

*images) x (feature map width) x (feature map height) x (feature map channels)*. A

convolutional layer within a neural network should have the following attributes:

• Convolutional kernels defined by a width and height (hyper-parameters).

• The number of input channels and output channels (hyper-parameter).

• The depth of the Convolution filter (the input channels) must be equal to the number

channels (depth) of the input feature map.

**Pooling:** Convolutional networks may include local or global pooling layers to streamline the

underlying computation. Pooling layers reduce the dimensions of the data by combining the

outputs of neuron clusters at one layer into a single neuron in the next layer. Local pooling

combines small clusters, typically 2 x 2. Global pooling acts on all the neurons of the

convolutional layer. In addition, pooling may compute a max or an average. Max pooling uses the maximum value from each of a cluster of neurons at the prior layer. Average pooling uses the average value from each of a cluster of neurons at the prior layer. **Fully Connected:** Fully connected layers connect every neuron in one layer to every neuron in another layer. It is in principle the same as the traditional multi-layer perceptron neural network (MLP). The flattened matrix goes through a fully connected layer to classify the images.

**Receptive Field:** In neural networks, each neuron receives input from some number of locations in the previous layer. In a fully connected layer, each neuron receives input from every element of the previous layer. In a convolutional layer, neurons receive input from only a restricted subarea of the previous layer. Typically, the subarea is of a square shape (e.g., size 5 by 5). The input area of a neuron is called its receptive field. So, in a fully connected layer, the receptive field is the entire previous layer. In a convolutional layer, the receptive area is smaller than the entire previous layer.

**Weights:** Each neuron in a neural network computes an output value by applying a specific function to the input values coming from the receptive field in the previous layer. The function that is applied to the input values is determined by a vector of weights and a bias (typically real numbers). Learning, in a neural network, progresses by making iterative adjustments to these biases and weights. The vector of weights and the bias are called *filters* and represent particular features of the input (e.g., a particular shape).

**Architecture of CNN:** A 13-layered network is used to predict the weather condition of each

class. The following layers are used in the Convent.

**Layers:**

• **Convolution:** Convolutional layers convolve around the image to detect edges, lines, blobs

of colours and other visual elements. Convolutional layers hyperparameters are the number

of filters, filter size, stride, padding and activation functions for introducing non-linearity.

• **MaxPooling:** Pooling layers reduces the dimensionality of the images by removing some of

the pixels from the image. MaxPooling replaces a n x n area of an image with the maximum

pixel value from that area to down sample the image.

• **Dropout:** Dropout is a simple and effective technique to prevent the neural network from

overfitting during the training. Dropout is implemented by only keeping a neuron active with

some probability p and setting it to 0 otherwise. This forces the network to not learn redundant information.

• **Flatten:** Flattens the output of the convolution layers to feed into the Dense layers.

• **Dense:** Dense layers are the traditional fully connected networks that maps the scores of the

convolutional

**Very Deep Convolutional networks (VGG):** Winner of the ImageNet ILSVRC-2014 competition, VGGNet was invented by Oxford's Visual Geometry Group, The VGG architecture is composed entirely of 3x3 Convolutional and MaxPooling layers, with a fully connected block at the end. The pertained model is available in Keras, TensorFlow, Caffe, Torch and many other popular DL libraries for public use.

In the original paper, they have shown the effect of the convolutional network depth on its accuracy in the large-scale image recognition setting. In the whole architecture they have used a very small (3x3) convolution filters, and shown a significant improvement on the prior-art configurations by pushing the depth to ***16-19*** weight layers. Their findings were the basis of ImageNet Challenge 2014 submission, where their team secured the first and the second places in the localisation and classification tracks respectively. They have made their two best-performing ConvNet models publicly available for further use.

**Architecture of VGG:**

**Layers:**

• **Convolution:** Convolutional layers convolve around the image to detect edges, lines, blobs of

colours and other visual elements. Convolutional layers hyper parameters are the number of

filters, filter size, stride, padding and activation functions for introducing non-linearity.

• **MaxPooling:** Pooling layers reduces the dimensionality of the images by removing some of the

pixels from the image. MaxPooling replaces a n x n area of an image with the maximum pixel

value from that area to down sample the image.

• **Dropout:** Dropout is a simple and effective technique to prevent the neural network from

overfitting during the training. Dropout is implemented by only keeping a neuron active with

some probability p and setting it to 0 otherwise. This forces the network to not learn redundant

information.

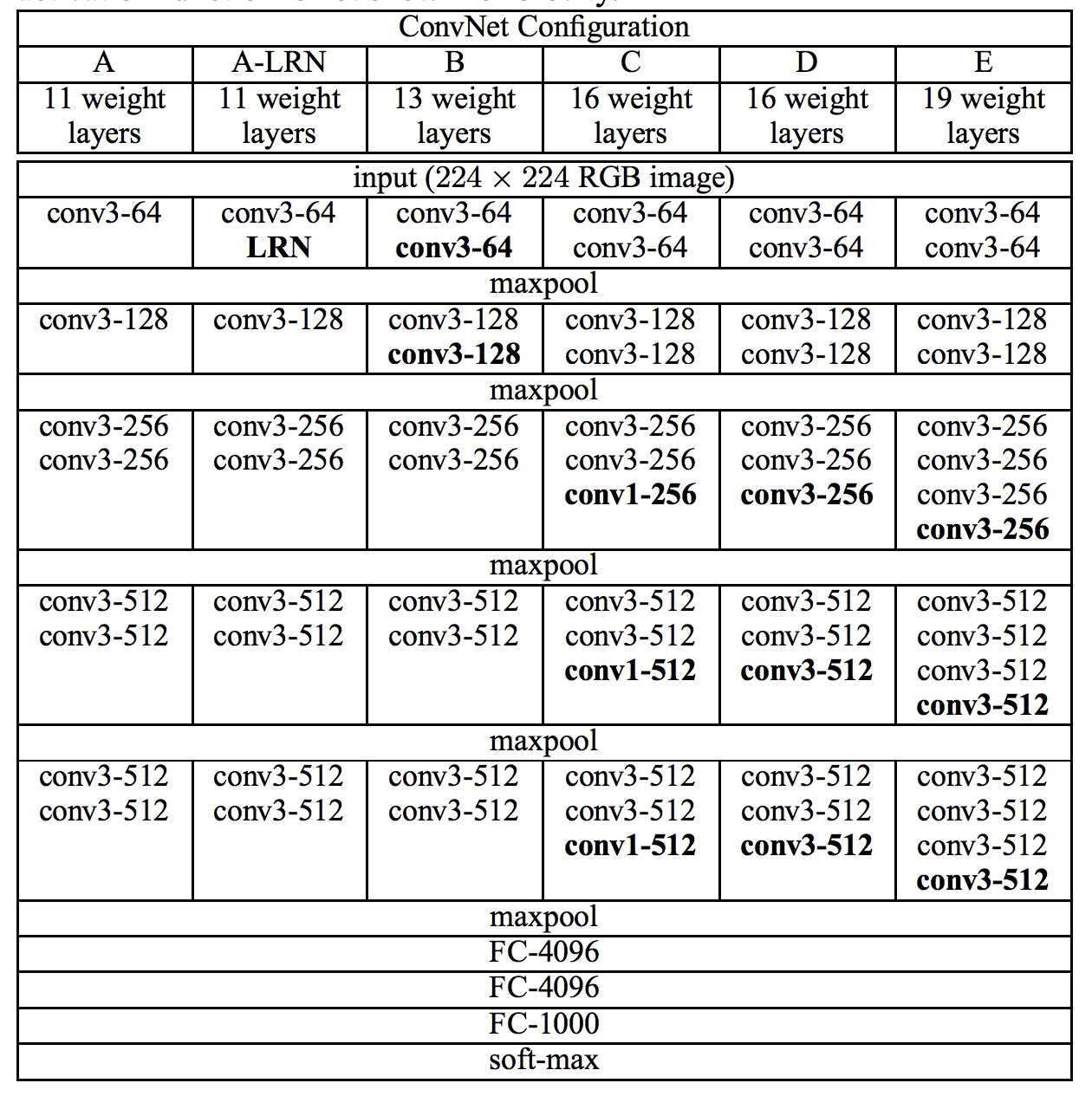
• **Flatten:** Flattens the output of the convolution layers to feed into the Dense layers.

• **Dense:** Dense layers are the traditional fully connected networks that maps the scores of the

convolutional layers into the correct labels with some activation function (SoftMax used here)

Below is a table taken from the paper; note the two far right columns indicating the configuration

(number of filters) used in the VGG-16 and VGG-19 versions of the architecture.



**Residual Network (ResNet):** Winner of the ImageNet ILSVRC & COCO-2015 competition, ResNet was invented by a group of researchers from Microsoft Research. The pretrained model is available in Keras, TensorFlow, Caffe, Torch and many other popular DL libraries for public use.

Their model had an impressive 152 layers. Key to the model design is the idea of residual blocks that make use of shortcut connections. These are simply connections in the network architecture where the input is kept as-is (not weighted) and passed on to a deeper layer, e.g. skipping the next layer.

A residual block is a pattern of two convolutional layers with ReLU activation where the output of the block is combined with the input to the block, e.g. the shortcut connection. A projected version of the input used via 1×1 if the shape of the input to the block is different to the output of the block, so-called 1×1 convolution. These are referred to as projected shortcut connections, compared to the unweighted or identity shortcut connections.

The authors start with what they call a plain network, which is a VGG-inspired deep convolutional neural network with small filters (3×3), grouped convolutional layers followed with no pooling in between, and an average pooling at the end of the feature detector part of the model prior to the fully connected output layer with a SoftMax activation function. The plain network is modified to become a residual network by adding shortcut connections in order to define residual blocks. Typically, the shape of the input for the shortcut connection is the same size as the output of the residual block.

**Architecture of ResNet:**

**Layers:**

• **Convolution:** Convolutional layers convolve around the image to detect edges, lines, blobs of

colours and other visual elements. Convolutional layers hyperparameters are the number of

filters, filter size, stride, padding and activation functions for introducing non-linearity.

• **MaxPooling:** Pooling layers reduces the dimensionality of the images by removing some of the

pixels from the image. MaxPooling replaces a n x n area of an image with the maximum pixel

value from that area to down sample the image.

• **Dropout:** Dropout is a simple and effective technique to prevent the neural network from

overfitting during the training. Dropout is implemented by only keeping a neuron active with

some probability p and setting it to 0 otherwise. This forces the network to not learn redundant

information.

• **Flatten:** Flattens the output of the convolution layers to feed into the Dense layers.

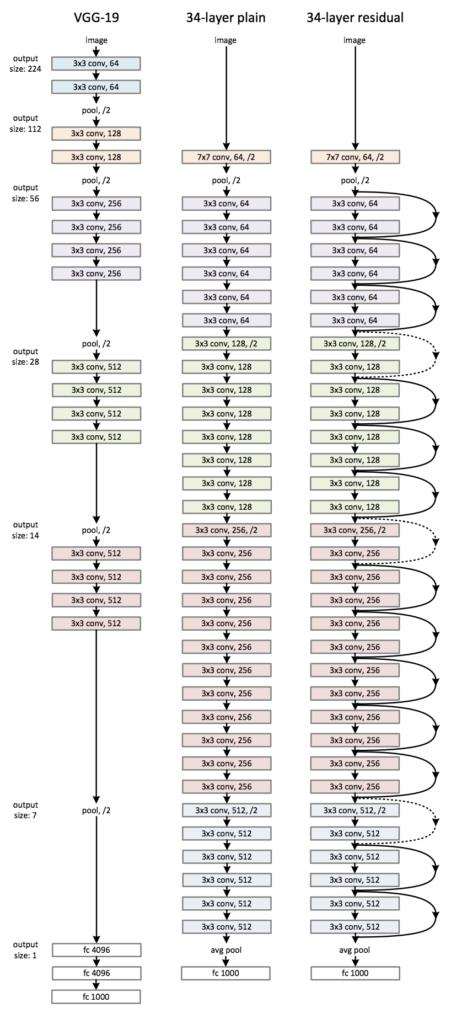
• **Dense:** Dense layers are the traditional fully connected networks that maps the scores of the

convolutional layers into the correct labels with some activation function (SoftMax used here)

The image below was taken from the paper and from left to right compares the architecture of a VGG

model, a plain convolutional model, and a version of the plain convolutional with residual modules,

called a residual network.



**Activation functions:** Activation layers apply a non-linear operation to the output of the other layers such as convolutional layers or dense layers.

• **ReLu Activation:** ReLu or Rectified Linear Unit computes the function $f(x)=max(0,x) to

threshold the activation at 0.

• **SoftMax Activation:** SoftMax function is applied to the output layer to convert the scores

into probabilities that sum to 1.

**Optimizers:**

• **Adam** (Adaptive moment estimation): is an update to RMSProp optimizer in which the

running average of both the gradients and their magnitude is used. In practice Adam is

currently recommended as the default algorithm to use, and often works slightly better than

RMSProp. In my experiments Adam also shows general high accuracy while Adadelta learns

too fast. I've used Adam in all the experiments because I felt having similar optimizer would be a better baseline for comparing the experiments.

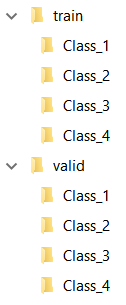
**Data Augmentation:** Data augmentation is a regularization technique where we produce more images from the training data provided with random jitter, crop, rotate, reflect, scaling etc to change the pixels while keeping the labels intact. CNNs generally perform better with more data as it prevents overfitting.

**Batch Normalization:** Batch Normalization is a recently developed technique by Ioffe and Szegedy which tries to properly initializing neural networks by explicitly forcing the activations throughout a network to take on a unit gaussian distribution at the beginning of the training. In practice, we put the BatchNormalization layers right after Dense or convolutional layers. Networks that use Batch Normalization are significantly more robust to bad initialization. Because normalization greatly reduces the ability of a small number of outlying inputs to over-influence the training, it also tends to reduce overfitting. Additionally, batch normalization can be interpreted as doing pre-processing at every layer of the network, but integrated into the network itself.

**Methodology:**

**Data Pre-processing**

As per using ResNet, VGG like architecture for transfer learning, images are pre-processed as performed in the original architecture mentioned in paper. Creators of these original Networks took the different inputs for their respective Nets like in VGG16 they had subtracted the mean of each channel (R, G, B) first so the data for each channel had a mean of 0. Furthermore, their processing software expected input in (B, G, R) order whereas python by default expects (R, G, B), so the images had to be converted from RGB -> BGR. For this purpose, a *preprocess\_input* function, imported from *keras application* module, is used for respective network. In this dataset input images also come in different sizes and resolutions, so they were resized to 256 x 256 x 3 to reduce size. This dataset does not have any validation set, so it was split into a training set and a validation set for evaluation. Out of 1500 images, 1274 images are in the training set and the remaining (0.15% of all classes) are in the validation set. Note that instead of using *train\_test\_split* methods in *scikit-learn*, I randomly took 0.15% of each class from the dataset to the validation set while preserving the directory structure.



**Implementation**

Initially the baselines with a deep ConvNet with and without augmentation were implemented for comparison. After that the models with transfer learning were used.

So, VGG and ResNet architecture without the last fully connected layers were used to extract the convolutional features from the pre-processed images. On the extracted features (CNN codes), a small fully connected model was applied first but unfortunately it didn't have a good result. After that I applied dropout and batch normalization to the fully connected layer which beat the CNN benchmark by 17.50.

**Results**

For each experiment only the best model was saved along with their weights (a model only gets saved per epoch if it shows higher validation accuracy than the previous epoch)

To recap, the best model so far uses transfer learning technique along with data augmentation and

batch normalization to prevent overfitting. To use transfer learning, I've collected pretrained weights

for the VGG and ResNet architecture from the keras official GitHub page and used the similar

architecture only with replacing the fully connected layers with different dropout and batch

normalization. I had used less aggressive dropout in my models.

I'he pre-processed all the images according to VGG and ResNet architecture directions (using the

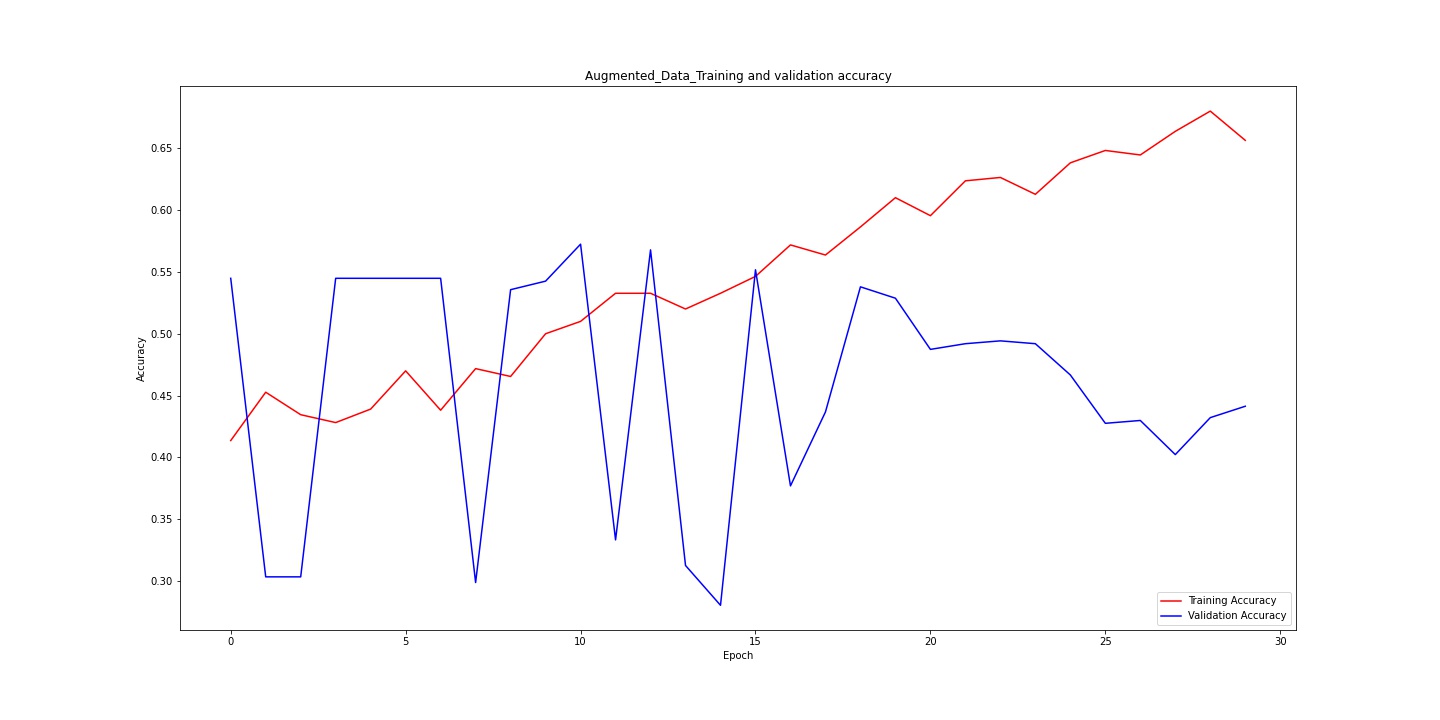
*preprocess\_input* function from *keras\_applications* module). For the final model I used the base

model of ***ResNet101*** excluding the fully connected layers along with the pretrained weights, added

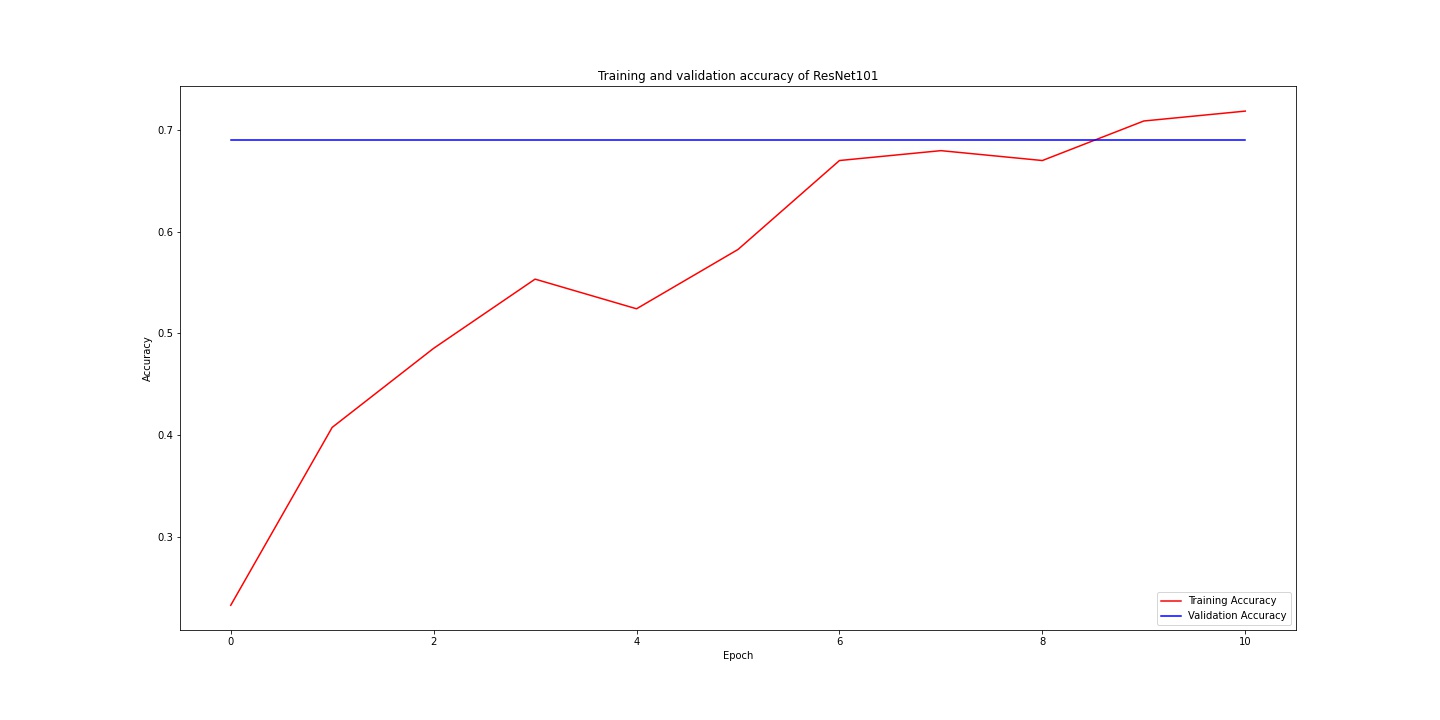
two new Dense layers with dropout and batch normalization on top of it and on top of these two, a

dense layer with *softmax* activation function to predict the final images.

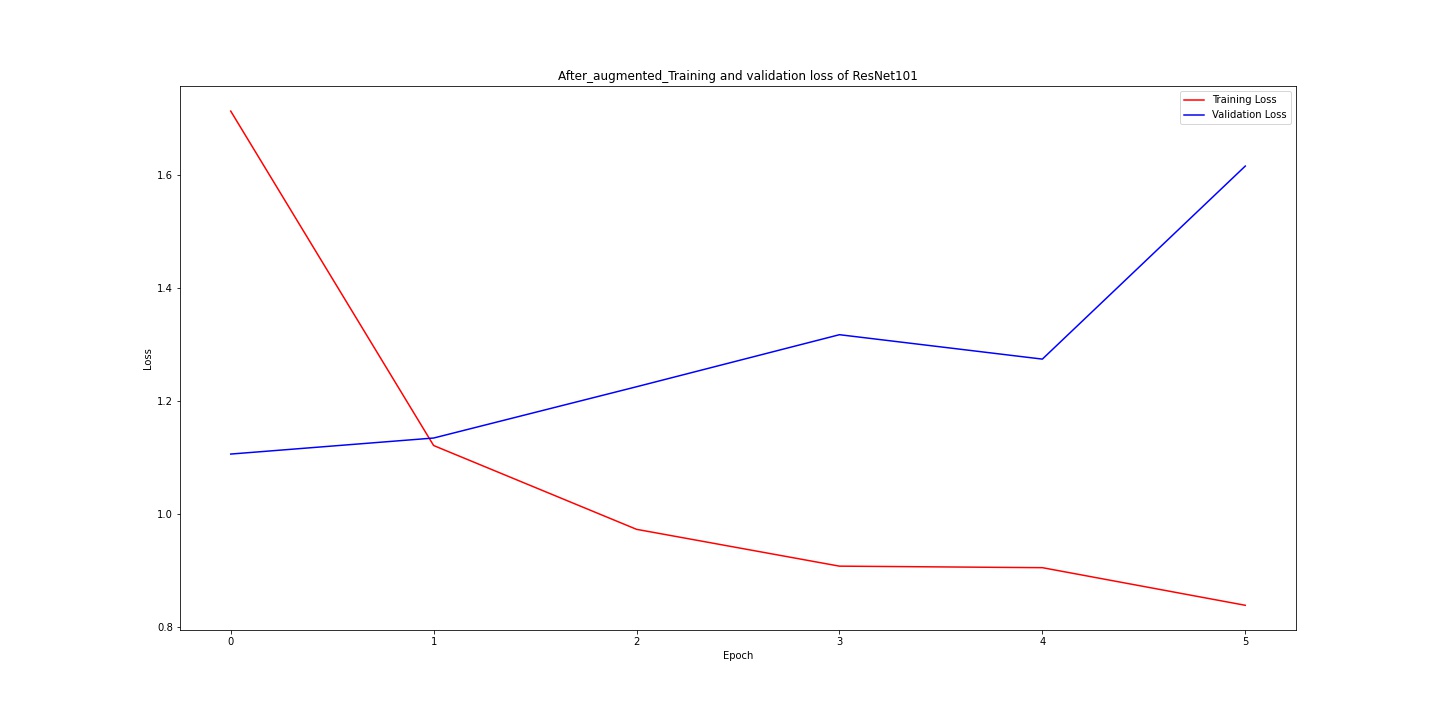
Augmented Data Training and Validation Accuracy



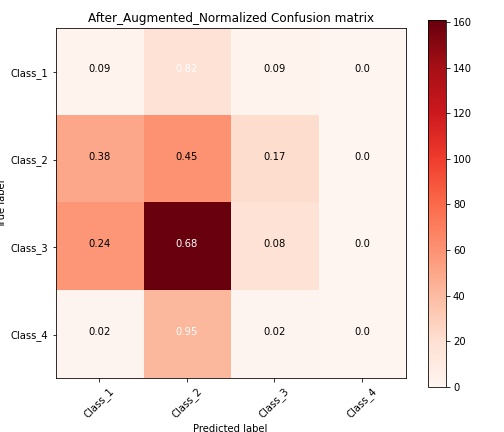
Training and Validation Accuracy of ResNet101



After Augmented Training and Validation loss of ResNet101



After Augmented Normalized Confusion Matrix



After Augmented Confusion Matrix Without Normalization

