ISSN: 0976-9102 (ONLINE)
DOI: 10.21917/ijivp.2022.0381

IMAGE CLASSIFICATION USING MODEL ENSEMBLING

Debabrata Datta [1], Anweshan Mukherjee[2], Soumen Mukherjee[3], Arup Kr. Bhattacharjee[4], Anal Acharya[5]
[1,2,5]Department of Computer Science, St. Xavier's College, India
[3]Department of Computer Application, RCC Institute of Information Technology, India
[4]Department of Computer Science and Engineering, RCC Institute of Information Technology, India

Abstract

Classifying images efficiently using various algorithms is very useful now-a-days given that the field of computer vision is growing rapidly. The research work highlighted in this paper focuses on the independent use of various models to classify images and then combining them together to form a better model in terms of performance than each of the individual models. The dataset used consists of 200 classes with 90,000 training images, 10,000 validation images and 10,000 test images. The data preparation step in this work involves resizing the images (data), shuffling them and transforming them into a data generator to provide input to the models. The images were also augmented using two different sets of image transformation effects to get more data for the models to train on. These data were then used to train five different models (one model trained from scratch and four other models using pre-trained weights and transfer learning) independently. The performance of each model was judged by checking two evaluation metrics – f1-score and categorical accuracy. The models were also tried to be fine-tuned to get a better performance, and finally the models were ensembled together to get a better categorical accuracy and f1-score on unseen (validation and test) data.

Keywords:

Image Classification, Convolutional Neural Networks, Image Augmentation, Model Ensembling, F1-Score

1. INTRODUCTION

Computer Vision is a field of study that requires collective knowledge from various scientific branches, focusing on training computers to learn from electronic videos or images. This field of artificial intelligence can be used to help humans by replicating the work done by the human visual system, or even obtain better results such as in low resolution images or videos [1].

Image Classification is one of the most widely used and studied sub-domains of computer vision. It is the task of predicting the class (label) an image belongs to by examining it using various algorithms [2]. Image Classification is difficult because what we see as an image, the computer sees it as an array of binary numbers, which are actually the pixel values (either RGB or Black and White) converted to binary. The pixel values of a colored image are represented as RGB values. So, a colored image is represented as a huge 3-dimensional array. And, this array changes with changing the size of the image or, changing the position of an object in the image or, even changing the brightness and contrast of the image. This means that we may have many different huge 3-dimensional arrays belonging to the same class. Thus, what seems very trivial to us, is actually not a very easy task to do while training the computer to do the same.

The method followed in this paper focuses on training multiple models (using different algorithms) independently and then joining them together to get a better result.

There were five models used here: a VGG16 inspired model trained from scratch; Xception, InceptionResNetV2, MobileNetV2 and DenseNet201 (all using ImageNet weights) for transfer learning. These models are then ensembled together to get the final model on which classification was done on unseen data.

The method described in this paper can be used to develop a web application or a mobile application and if developed further, can be applied in medical fields such as detection of pneumonia and cancer or in military uses such as classifying enemies in the battlefield.

2. RELATED WORK

The work of image classification generally starts off with getting a uniform size and shape of the input images (data). This is done for two reasons: firstly, the model to which the images are to be fed, needs a fixed size of the input; and secondly, the original data may be so small (size of the images are very small in dimensions), that it becomes difficult for the model to extract sufficient information from them.

Due to the large amount of data contained in each image, every method related to machine learning cannot be applied to the problem of image classification. Input images are made to go through various series of convolution layers so that the vastness of the data is reduced and a feasible amount of data can be obtained to perform the classification task.

The quest to classify images and proposing various algorithms to do that is not a very recent topic. It goes way back to the year 1998 when LeCun et al. [3] used convolutional neural networks to classify images. Their model named LeNet-5, contains 7 layers (other than the input layer), all of which contain trainable parameters (weights).

LeNet-5 is one of the oldest networks for image classification. In LeNet-5, deep neural networks were trained and optimized using gradient-descent algorithm and it showed how independent convolutional neural networks (CNNs) could be combined to get interdependent layers of the model to obtain outputs with a better accuracy [4].

Krizhevsky et al., in 2012, designed a network model called AlexNet [5], to classify ImageNet [6] [7] data [8]. The AlexNet architecture is famous because it focuses on the distribution of work between two different GPUs. This network is one of the pioneer networks able to achieve a significantly high classification accuracy on a standard image classification task. The fact that this network uses concepts like convolution layers, pooling and GPUs (for parallel computing) makes it a significant contribution to the image classification domain. It was the first CNN-based award winner at ILSVRC 2012 with a top-5 error rate of 16.4% [9].

Out of the five models trained and ensembled together in the research work mentioned in this paper, one of them was trained from scratch. It is based on the VGGNet architecture. VGG16 architecture was developed by Simonyan and Zisserman [10].

The VGG16 model uses many but small convolutional filters to extract information from the images [11]. This model was the winner in localization at ILSVRC 2014 with a top-5 error rate of 7.3% [12].

The problem with AlexNet and VGG16 was that they had too many parameters and VGG16 used too much of memory for training (almost 96 MB per image). Szegedy et al. [13] proposed the GoogLeNet architecture which solved these problems to a great extent. It uses an Inception module, which is not simply a convolutional layer but many different kinds of layers concatenated together and arranged in a way that reduces the number of computations.

The Inception module is based on the strategy to form a stack of well-designed local network topology modules (a network inside another network). The naïve version of the Inception module [13] has three convolution layers which intuitively allows one to see an image in different resolutions with respect to zooming in or out the image. The filters (three convolutions and one pooling operation) are applied parallelly and then the outputs of these filters are concatenated depth-wise. But, the number of operations in doing this is too high because along with the large number of computations in the convolution layers, the pooling layer also preserves feature depth. So, the images are never down sampled, which could result in an increase (but never decrease) in the total depth after filter concatenation is done at each layer.

To overcome this problem, an Inception module with dimension reduction strategy involved was introduced [13]. In that, there are several 1×1 convolution layers which serve as bottleneck layers. They reduce the feature depth of the output at that layer and as a result reduce the number of operations significantly.

In GoogLeNet, Inception modules are stacked one upon the other for dimension reduction. In such deep networks, one major problem is with the costly backpropagation. For that, GoogLeNet uses a trick to have auxiliary outputs in the middle of the network so that the backpropagation is made faster. The same output which is there at the final fully connected layer is also available as some intermediate Inception module’s auxiliary output. These auxiliary outputs serve as additional gradients at lower layers.

GoogLeNet has almost 12 times lesser number of parameters than AlexNet and it was classification winner at ILSVRC 2014 with a top-5 error of 6.7% [12].

He et al. [14] proposed another model, called ResNet, which uses very deep network with residual connections. Generally, in very deep networks, there is a disadvantage that the gradients are not backpropagated properly. ResNet kind of bypasses certain layers to solve this. There is a parallel path in the model which intuitively helps in bypassing the weight layers. This path helps in both forward and backward passes.

ResNet was the classification winner at ILSVRC 2015 with a top-5 error rate of 3.57% (better than human performance) [15]. As days go by, the fields of computer vision and image classification are developing more and more.

3. PROPOSED METHODOLOGY

The methodology proposed in this paper consists of training five different models (one from scratch and four others by transfer learning) and then ensembling them together for final classification.

3.1 DATA PREPARATION

The data (images) were initially kept at 64×64 dimensions but it was getting harder to extract sufficient information from them and the validation categorical accuracy was stuck at around 53%. For this reason, the input data dimension was changed to 128×128, which provided much better results. The pixel values of each image were also rescaled to 1/255 so as to make the computations easier. The images were then shuffled and transformed into a data generator for providing input.

3.2 TRAINING THE VGG16 INSPIRED MODEL FROM SCRATCH

The first model which was trained from scratch, was inspired by the VGG16 architecture [11]. The model takes as input RGB images of dimensions 128×128. The model has two convolution layers with 64 kernels, two with 128 kernels, three with 256 kernels, and six with 512 kernels. The kernel size in each convolution layer was 3×3. Each convolution layer has batch normalization integrated to let the network train faster [19]. ReLU activation has also been used in each convolution layer. Any two successive convolution blocks were separated from each other by a max-pooling layer of pool size 2×2 and a stride of 2 to reduce computational complexity.

The result of the final max-pooling layer was flattened to be passed through dense layers for classification. There were three dense layers with 4096 units, 512 units and 200 units respectively. The first dense layer (with 4096 units) had adropout of 50% following it to prevent overfitting [20]. The first two dense layers have ReLU activation while the final dense layer has softmax activation.

The final dense layer had given the output in the form of a vector which served as the probabilities of each of the 200 classes and thus enabled the model to perform soft classification.

3.2.1 Reasons for using Convolution Neural Networks:

There might be some images which were essentially the same but differed from each other drastically with respect to the pixel values. This happened because some images might be zoomed in or out; or the objects in the images were at different positions in different images. Due to this huge variability in data, it was not possible to use a dense neural network having so many parameters and still be able to classify the data efficiently [16].

To tackle this problem, convolutional neural networks were used which, intuitively, extracted some features or attributes from an image and made the work of classification much easier. For example, if a classifier has been given three images of a tiger as input, the attributes such as having a tail, black and yellow stripes on body and having two eyes will be the same for all images. So, if the classifier could extract these features, it would be able to easily classify that the images belong to the class ‘tiger’, though in one image the tiger could be running, while in the other images the tiger could be eating or sleeping.

For scanning the images for attributes or features, a kernel is used. A hop size (or, stride size) was selected which is the shift in kernel's position for scanning.

3.2.2 Reasons for using Max Pooling:

Pooling was done to down-sample the data (image) and thus reduce computational complexity of the succeeding operations [16]. For example, if an image of a tiger is being scanned with the kernel of its tail, the convoluted matrix would be almost 0's everywhere and 1 only at the specific pixel values where the tail has been found. These insignificant values make the data sparse and increase the computational complexity since the dimension of the data gradually increases.

To overcome this, max pooling was done, which is actually selecting a group of elements from the convoluted matrix and selecting the maximum of them to be considered as an element of max-pooled convoluted matrix.

3.2.3 Reasons for using Softmax and ReLU Non-Linearity:

The gradient descent algorithm [21] follows Eq.(1) for updating the weights.

$$w_{new} \leftarrow w_{old} - \eta \left.\frac{\partial L}{\partial w}\right|_{w=w_{old}} \tag{1}$$

where, w are the parameters; η is the learning rate and L is the loss function.

Thus, the non-linearity needs to be differentiable, which both Softmax and ReLU is. Using ReLU also tackled the problem of vanishing gradients [22]. Using Softmax, provided a vector asthe output, in which each element belonged to the interval [0, 1]; and, the sum of all these elements equal to 1. Thus, these values could be considered to be probability values for each class – allowing the model to perform soft classification.

The different hyperparameters used are:

- Optimizer: Stochastic Gradient Descent (SGD) [23] with a learning rate of 0.001 and momentum of 0.9
- Callback: ReduceLROnPlateau [24] with factor of 0.2, patience of 3 and minimum learning rate of 10^{-7} was made to monitor the validation loss. This was used to facilitate faster training when the model did not show any more improvement.
- **Loss function**: Categorical Cross-entropy [25]
- **Evaluation Metrics**: F1-Score and Categorical Accuracy

3.2.4 Stage 1 (Training on Original Data):

In this stage, the model was trained on original training data. Stochastic gradient descent with a learning rate of 0.001 and momentum of 0.9 was initially used as the optimizer. ReduceLROnPlateau callback was incorporated in the training process to prevent overfitting. This callback was made to monitor the validation loss and it reduced the learning rate by a factor of 0.2 whenever the model did not show any improvement for 3 consecutive epochs (patience = 3). However, a minimum learning rate was fixed at 10^{-7}, below which the callback could no longer reduce the learning rate. Categorical-cross entropy was used as the loss function. The evaluation metrics used were f1-score and categorical accuracy.

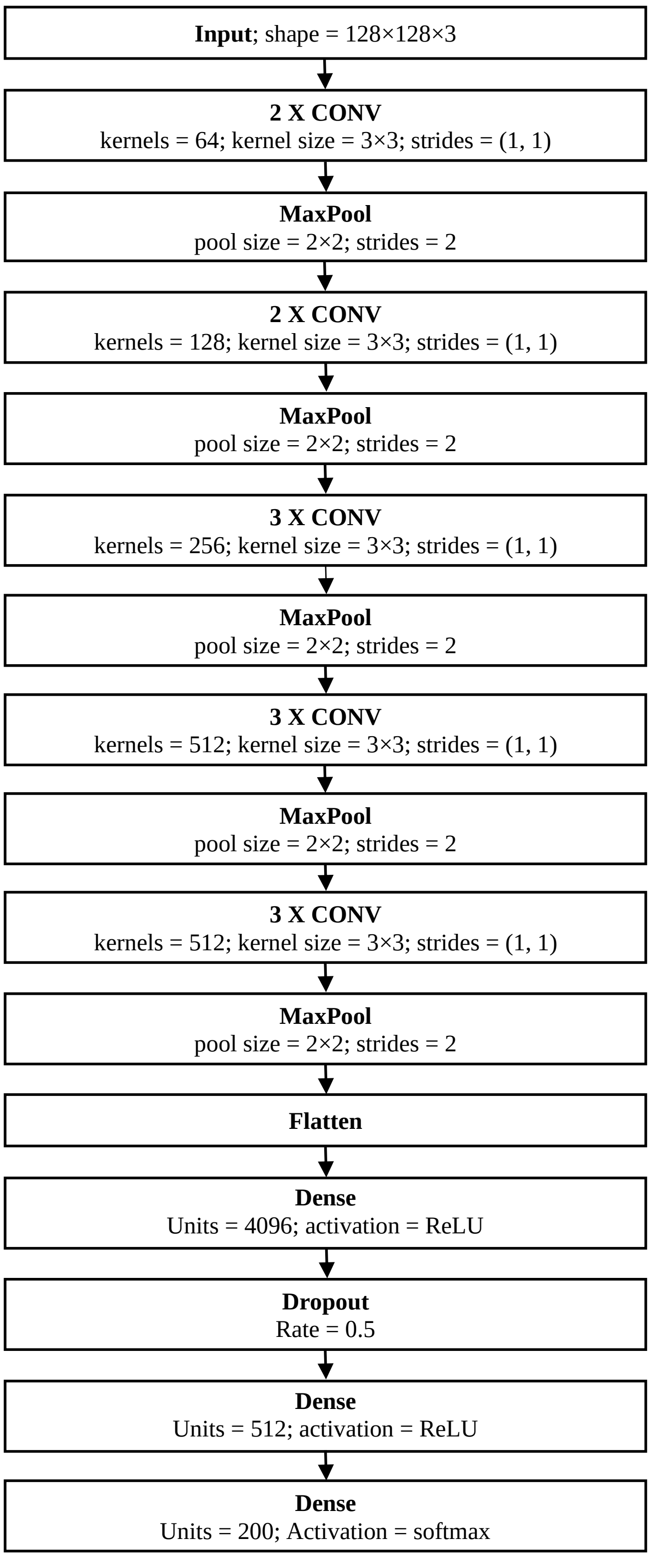

Input; shape = 128×128×3
2 X CONV
kernels = 64; kernel size = 3×3; strides = (1, 1)
MaxPool
pool size = 2×2; strides = 2
2 X CONV
kernels = 128; kernel size = 3×3; strides = (1, 1)
MaxPool
pool size = 2×2; strides = 2
3 X CONV
kernels = 256; kernel size = 3×3; strides = (1, 1)
MaxPool
pool size = 2×2; strides = 2
3 X CONV
kernels = 512; kernel size = 3×3; strides = (1, 1)
MaxPool
pool size = 2×2; strides = 2
3 X CONV
kernels = 512; kernel size = 3×3; strides = (1, 1)
MaxPool
pool size = 2×2; strides = 2
Flatten
Dense
Units = 4096; activation = ReLU
Dropout
Rate = 0.5
Dense
Units = 512; activation = ReLU
Dense
Units = 200; Activation = softmax

Fig.1. Model Architecture

The input images were shuffled to ensure that each image creates a unique and independent change in the model weights without being biased by the previous input data.

3.2.5 Stage 2 (Training on Augmented Data 1):

One of the most prioritized objectives in image classification is to get more and more data for training the model. Using image augmentation, more training data can be generated from the already existing data. It is very useful in applications where data is not so easily available [17].

For performing image augmentation, the already available data were made to undergo several transformations such as rotation, re-scaling of images and so on. These transformations were done keeping in mind the application that the model is being trained for and at the same time prevent overfitting of the model. This was because getting too much of data for one class might make the model biased towards that class and could provide not very satisfactory results for images belonging to other classes.

In this stage, the training data were augmented using a list of transformations as:

- Rescaling each pixel of the image by 1/255
- Rotating each image randomly within a range of 40°
- Shifting width of each image by a range of 0.2
- Shifting height of each image by a range of 0.2
- Shearing each image by a range of 0.2°
- Zooming each image by a range of 0.2
- Randomly flipping some images horizontally

After these transformations were done, the images were shuffled and fed to the model as input. All of these transformations were applied using ImageDataGenerator [26] class of keras.preprocessing.

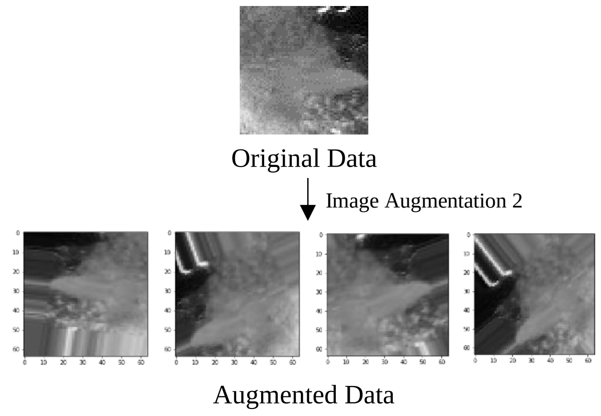


Fig.2. Images after applying Image Augmentation 1

The optimizer and the callback used in this stage was same as that in Stage 1 other than the initial learning rate which was reduced to 10^{-4} for stochastic gradient descent optimizer; and the patience and the minimum learning rate was changed to 4 and 10^{-14} respectively for the ReduceLROnPlateau callback.

3.2.6 *Stage 3 (Training on Augmented Data 2):*

In this stage, the input data were augmented using a different set of transformations from the ones in stage 1.

The transformations applied were:

- Horizontally flipping 50% of the images.
- Cropping images from each side by 0 to 4px (randomly chosen).
- Applying Gaussian blur with sigma between 0 and 2.5.
- Increasing or decreasing the contrast in each image by varying the linear contrast between 0.75 and 1.25.
- Applying additive Gaussian noise within a scale of 0 to 0.05×255.
- Scaling each image within the range of (0.8, 1.2) for both x- and y-axes.
- Rotating each image by some degree ranging from -20° to 20°.
- Dropping some pixels from each image, 2% to 5% of the original size, leading to large dropped rectangles.

All of these transformations were applied using imgaug [27] library. Unlike image augmentation 1, image augmentation 2 did not use all the transformations mentioned. It was made to randomly select an integer from 0 to 6, and that many transformations were applied on the images. This was done so that the images were not heavily augmented which could result in underfitting the model. The optimizer and the callback used in this stage was the same as that in stage 2.

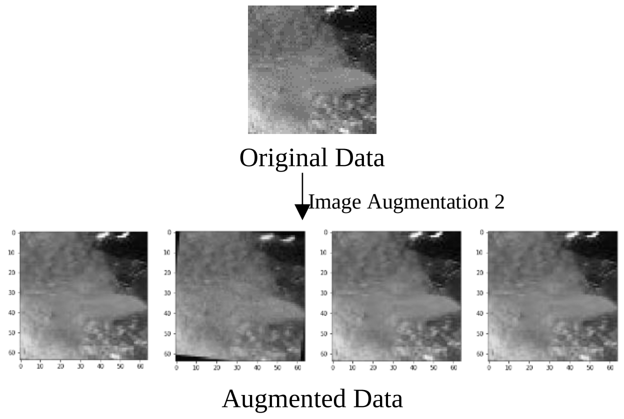


Fig.3. Images after applying Image Augmentation 2

3.3 TRANSFER LEARNING USING XCEPTION CONVOLUTION BASE AND IMAGENET WEIGHTS

Transfer learning is a technique in machine learning where an algorithm or a model which has been already used for a task, is used for another task. This facilitates for a better performance on the second task since the model being used is already pre-trained.

Xception [28] is a deep learning model for image classification that performs its task using point-wise convolution and depth-wise convolution. The convolution block for Xception is divided into 3 flows – entry flow, middle flow and exit flow.

In the entry flow, images of size 299 × 299 × 3 are taken as input (default input) and they are made to pass through two convolution layers – one having 32 kernels, kernel size = 3×3 and stride = 2×2, and the other having 64 kernels of size 3×3. Two separable convolution networks of 128 kernels of size 3×3, two of 256 kernels of size 3×3 and two of 728 kernels of size 3×3 are also present. All of these have ReLU non-linearity attached.

The entry flow also contains three max pooling layers of pool size 3×3 and stride 2×2. Three convolution layers with kernel size 1×1 and stride 2×2 are present as linear residual connections to various points in the entry flow. For a default input of 299×299×3 image, the entry flow outputs feature maps of dimension 19×19×728.

The middle flow takes as input the 19×19×728 feature maps from the entry flow and passes it through three separable convolution layers of 728 kernels of size 3×3 having ReLU activation. The middle flow has a linear residual connection and is repeated 8 times to produce 19×19×728 feature maps for the default input.

The exit flow takes as input 19×19×728 feature maps and is made to pass through four separable convolution layers of 728 kernels, 1024 kernels, 1536 kernels and 2048 kernels respectively each of size 3×3 and ReLU non-linearity. A convolution layer with kernel size 1×1 and stride 2×2 is attached to the linear residual connection. For pooling, a max pooling layer with pool size 3×3 and stride 2×2 and a global average pooling layer is present as the final layer for the convolution block. All the blocks in all the three flows have batch normalization integrated.

In the research work highlighted in this paper, the convolution block for Xception was imported with pre-trained weights obtained on feeding them the ImageNet [29] dataset. A dense layer of 200 units and softmax non-linearity was added on top for classification. The optimizer used was Adadelta [30] and the input images were rescaled to size 128×128 after undergoing image augmentation 2. ReduceLROnPlateau callback was not incorporated in the training process because the models were not trained for many epochs at a time due to their large size and usage limits on Google Colaboratory GPU. Reduction in learning rate, if required, was done manually.

3.4 TRANSFER LEARNING USING INCEPTIONRESNETV2 CONVOLUTION BASE AND IMAGENET WEIGHTS

InceptionResNetV2 [31] is a deep learning model which uses very deep convolution networks for image classification. It uses both residual connections and inception blocks for lesser computations and better training.

The schema for InceptionResNetV2 is divided into several blocks/stages, namely ‘Input’, ‘Stem’, ‘Inception-resnet-A’, ‘Reduction-A’, ‘Inception-resnet-B’, ‘Reduction-B’, ‘Inception-resnet-C’, ‘Average Pooling’, ‘Dropout’ and ‘Softmax’.

The default input in the Input layer is of size 229×229×3.The Stem layer takes this input and passes it through three convolution layers with 32, 32 and 64 kernels respectively, each of kernel size 3×3. The path from this output is split, made to pass through a max pooling layer of pool size 3×3 and strides 2, and a convolution layer with 96 kernels of size 3×3 and strides 2. The outputs from these are concatenated together using a ‘Filter concat’ layer.

The path from the Filter concat layer is again split into two paths: one with two convolution layers (one with 64 kernels of size 1×1 and another with 96 kernels of size 3×3), and another with four convolution layers (a 64 kernel of size 1×1, a 64 kernel of size 7×1, a 64 kernel of size 1×7, and a 96 kernel of size 3×3). The outputs from these paths are again concatenated and the paths are split: one to a convolution layer with 192 kernels of size 3×3 and the other with a max pooling layer with stride = 2.

The Stem layer is followed by five Inception-resnet-A layers. It is a 35×35 grid module consisting of three convolution layers of 32 kernels and size 1×1, three convolution layers with 32, 48 and 64 kernels respectively each of size 3×3 and a convolution layer with 384 kernels of size 1×1 to the linear residual connection. This layer has ReLU activation both at the starting and at the end.

The Reduction-A layer takes the filter concat version of the output from the Inception-resnet-A layer. The path is split into three ways: one with a max pooling layer of strides 2, one with a convolution layer with 284 kernels of kernel size 3×3, and the other with three convolution layers (one with 256 kernels of size 1×1, one with 256 kernels of size 3×3, and the last one with 384 kernels of size 3×3). The outputs from the three paths were again filter concatenated.

The Reduction-A layer is followed by ten Inception-resnet-B layers. It is a 17×17 grid module consisting of two convolution layers with 128 kernels and 192 kernels respectively, each of size 1×1, a convolution layer with 160 kernels of size 1×7, and a convolution layer with 192 kernels of size 7×1. A convolution layer with 1154 kernels of size 1×1 is also connected to the linear residual connection. Both the input to this layer and the output generated from this layer are ReLU activated.

The Reduction-B module reduces a 17×17 grid to 8×8 grid. From the Inception-resnet-B layers, it gets divided into four paths. One path contains a max pooling layer with pool size 3×3 and strides 2. Another path contains two convolution layers: one with 256 kernels of size 1×1 and another with 384 kernels of size 3×3. The third path also contains two convolution layers: one with 256 kernels of size 1×1 and another with 288 kernels of size 3×3. And, the final path contains three convolution layers: one with 256 kernels of size 1×1, one with 288 kernels of size 3×3, and one with 320 kernels of size 3×3. The outputs from all these paths were filter concatenated.

The output from the Reduction-B module is fed into five Inception-resnet-C modules. It is a 8×8 grid module consisting of two convolution layers with 192 kernels of size 1×1, one convolution layer with 224 kernels of size 1×3, one convolution layer with 256 kernels of size 3×1 and a convolution layer with 2048 kernels of size 1×1 connected to the linear residual connection. Both the input to this layer and the output generated from this layer are ReLU activated.

The output from Inception-resnet-C module is made to go through average pooling, dropout of 20% and finally a Softmax non-linearity. The 1×1 convolution layers are added to keep the dimension optimum since inception modules reduce dimensions.

The convolution block for InceptionResNetV2 was imported with pre-trained weights obtained on feeding them the ImageNet [29] dataset. A dense layer of 200 units and softmax non-linearity was added on top for classification. The optimizer used was Adadelta [30] and the input images were rescaled to size 128×128 after undergoing image augmentation 2.

3.5 TRANSFER LEARNING USING MOBILENETV2 CONVOLUTION BASE AND IMAGENET WEIGHTS

MobileNetV2 [32] is a deep learning model for image classification which is based on depth-wise separable convolutions and bottleneck layers. Depth-wise separable convolutions require much smaller number of computations than normal convolution layers. Generally, non-linearities tend to lose some information due to their behavior in certain intervals (for example, ReLU non-linearity loses negative data). This could deteriorate the performance of the model. Instead, if a linear bottleneck is used such that the last convolution block produces a linear output before it undergoes the initial non-linearities, it would be very useful.

In MobileNetV2 architecture, a default input of 224×224×3 is given to a convolution layer with 32 channels and strides 2. Seven bottleneck layers are used. The first bottleneck layer takes as input images of size 112×112×32, has 16 output channels, stride 1 and is not repeated. The second bottleneck layer takes as input images of size 112×112×16, has 24 output channels, strides 2 (for first layer only; 1 for the rest) and it repeated twice. The third bottleneck layer takes as input images of size 56×56×24, has 32 output channels, strides 2 (for first layer only; 1 for the rest) and is repeated thrice. The fourth bottleneck layer takes as input images of size 28×28×32, has 64 output channels, strides 2 (for first layer only; 1 for the rest) and is repeated four times. The fifth bottleneck layer takes as input images of size 14×14×64, has 96 output channels, strides 1 and is repeated thrice. The sixth bottleneck layer takes as input images of size 14×14×96, has 160 output channels, strides 2 (for first layer only; 1 for the rest) and is repeated thrice. The seventh bottleneck layer takes as input images of size 7×7×160, has 320 output channels, strides 1 and is not repeated. All the bottleneck layers except the first one has the input expanded by a factor of 6. Another convolution layer accepting inputs of size 7×7×320, having 1280 output channels and stride 1 is also present. An average pooling layer having input size 7×7×1280 is present. A convolution layer accepting input of size 1×1×1280 is present at the end.

In the research work mentioned in this paper, the convolution block for MobileNetV2 was imported with pre-trained weights obtained on feeding them the ImageNet [29] dataset. A dense layer of 200 units and softmax non-linearity was added on top for classification. The optimizer used was Adadelta [30] and the input images were rescaled to size 128×128 after undergoing image augmentation 2.

3.6 TRANSFER LEARNING USING DENSENET201 CONVOLUTION BASE AND IMAGENET WEIGHTS

DenseNet201 [33] is a deep learning model used for image classification. It is based on dense convolutional networks where the layers close to the input layer and the output layer have shorter connections between them. The feature-maps of the previous layers are used as inputs to the current layer and the feature-maps of the current layer are used as inputs to all the succeeding layers. Due to this, the flow of features from one layer to another is increased which reduces the overall number of parameters. The problem of vanishing gradients is also solved to a great extent.

The convolution block for DenseNet201 was imported with pre-trained weights obtained on feeding them the ImageNet [29] dataset. A dense layer of 200 units and softmax non-linearity was added on top for classification. The optimizer used was Adadelta [30] and the input images were rescaled to size 128×128 after undergoing image augmentation 2.

3.7 MODEL ENSEMBLING

Model ensembling is a type of regularization in machine learning where various models are trained independently to solve the same problem and then they are combined (taken average of the weights) to get better results [18]. The intuition behind this is that when several weak models are correctly combined, the result obtained is better (in terms of accuracy) than any of the combined models individually.

In the research work mentioned in this paper, after all the models were trained independently, they were ensembled together to get the final model for classification on unseen/test data.

These five models were ensembled together:

- VGG16 inspired model (trained from scratch)
- Xception (transfer learning)
- InceptionResNetV2 (transfer learning)
- MobileNetV2 (transfer learning)
- DenseNet201 (transfer learning)

3.7.1 Algorithm for Model Ensembling:

```
models = [VGG16_inspired_model, Xception_model, Inception ResNetV2_model, MobileNetV2_model, DenseNet201_model]
input = input layer of shape (128,128,3)
yhat = []
function ensemble_models (models, input)
{
    for each model in models
    yhat = yhat, [model(input)]
    yavg = average(elements of yhat)
    model_ens = new_model(inputs = input, outputs = yavg)
    return model_ens
}
```

4. RESULTS AND ANALYSIS

The various models were first trained individually and then they were ensembled together to get the final model.

The Dataset used [37] consists of 200 classes with 90,000 training images, 10,000 validation images and 10,000 test images. Size of each image is 64×64 pixels. The small dimension of each image made it harder to extract sufficient information from them.

∴ Training set : Validation set : Testing set = 9:1:1

4.1 TRAINING THE VGG16 INSPIRED MODEL FROM SCRATCH

4.1.1 Stage 1: Training on Original Data:

The model was trained for 24 epochs with a batch size of 64. After the 16th epoch, the learning rate was manually changed to 10^{-6}. This resulted in an improvement in both training and validation metrics. After stage 1, the model showed the results as given in Table.1.

4.1.2 Stage 2: Training on Augmented Data 1:

The model was trained for 24 epochs with a batch size of 64. Since the images used here were augmented, the overall performance of the model was poorer than that in stage 1. As the training went further, the performance of the model gradually got improved. After stage 2, the model showed the results as shown in Table.2.

4.1.3 Stage 3: Training on Augmented Data 2:

The model was trained for 24 epochs with a batch size of 64. After the 16th epoch, the learning rate was manually reduced to 10^{-5}, which resulted in better performance of the model. After stage 3, the model showed the results as shown in Table.3.

From Table.3, it can be seen that the model performed much better in stage 3 than in stage 2, both in terms of training and validation f1-score and categorical accuracy.

4.2 TRANSFER LEARNING USING XCEPTION CONVOLUTION BASE AND IMAGENET WEIGHTS

The model was trained for 12 epochs. Till epoch 8, the learning rate was fixed at 10^{-3}. After that, the learning rate was decreased to 10^{-4} for the model to be trained for 4 more epochs. Fine-tuning the last convolution block did not prove any improvement. Transfer learning the Xception model provided the results as shown in Table.4.

4.3 TRANSFER LEARNING USING INCEPTIONRESNETV2 CONVOLUTION BASE AND IMAGENET WEIGHTS

The model was trained for 16 epochs. Till epoch 8, the learning rate was fixed at 10^{-3}. After that, the learning rate was decreased to 10^{-4} for the model to be trained for 8 more epochs. Fine-tuning the last convolution block did not provide any improvement. Transfer learning the InceptionResNetV2 model provided the results as shown in Table.5.

4.4 TRANSFER LEARNING USING MOBILENETV2 CONVOLUTION BASE AND IMAGENET WEIGHTS

The model was initially trained for 24 epochs. Till epoch 16, the learning rate was fixed at 10^{-3}. After that, the learning rate was decreased to 10^{-4} for the model to be trained for 8 more epochs. Fine-tuning the last convolution block provided improvement in both training and validation loss and metrics. For fine-tuning the last convolution block, the learning rate was fixed at 10^{-4} and it was trained for 4 more epochs.

Transfer learning the MobileNetV2 model provided the results as shown in Table.6 and the results after fine-tuning the last convolution block are shown in Table.7.

4.5 TRANSFER LEARNING USING MOBILENETV2 CONVOLUTION BASE AND IMAGENET WEIGHTS

The model was trained for 26 epochs. Till epoch 8, the learning rate was fixed at 10^{-3}. After that, the learning rate was decreased to 10^{-4} for the model to be trained for 18 more epochs. Fine-tuning the last convolution block did not provide any improvement. Transfer learning the DenseNet201 model provided the results as shown in Table.8.

Table.1. Results after training model 1 on original data

Loss and Metrics vs Epochs	Training			Validation		
	Loss	F1-Score	Categorical Accuracy	Loss	F1-Score	Categorical Accuracy
Epoch 1	4.9837	8.2935×10^{-4}	0.0324	4.3599	0.0062	0.0782
Epoch 2	4.2236	0.0175	0.1057	3.8721	0.0251	0.1319
Epoch 3	3.7294	0.0576	0.1736	3.5868	0.0797	0.2021
Epoch 4	3.3929	0.1088	0.2287	3.1137	0.1123	0.2407
Epoch 5	3.1291	0.1606	0.2733	2.7731	0.1542	0.2661
Epoch 6	2.8988	0.2130	0.3151	3.5337	0.2150	0.2969
Epoch 7	2.6904	0.2665	0.3549	2.3900	0.2207	0.3027
Epoch 8	2.4951	0.3179	0.3919	2.8664	0.2796	0.3428
Epoch 9	2.3189	0.3631	0.4275	2.2265	0.2843	0.3384
Epoch 10	2.1391	0.4100	0.4637	3.8136	0.3221	0.3636
Epoch 11	1.9628	0.4574	0.5005	2.4065	0.3459	0.3776
Epoch 12	1.7969	0.5001	0.5358	3.0625	0.3530	0.3653
ReduceLROnPlateau reducing learning rate to 0.00020000000949949026						
Epoch 13	1.2356	0.6488	0.6728	3.4637	0.4438	0.4503
Epoch 14	1.0434	0.7020	0.7194	3.8950	0.4342	0.4271
Epoch 15	0.9236	0.7327	0.7485	1.8921	0.4457	0.4313
Epoch 16	0.8090	0.7631	0.7782	2.9619	0.4354	0.4234
Manually reducing learning rate to $1e^{-6}$						
Epoch 17	0.6851	0.7972	0.8158	2.6833	0.4550	0.4451
Epoch 18	0.6531	0.8074	0.8278	2.5127	0.4569	0.4508
Epoch 19	0.6388	0.8121	0.8312	1.7470	0.4591	0.4524
Epoch 20	0.6322	0.8154	0.8350	2.3139	0.4599	0.4541
Epoch 21	0.6292	0.8159	0.8366	1.8152	0.4620	0.4553
Epoch 22	0.6220	0.8175	0.8370	1.9120	0.4608	0.4542
ReduceLROnPlateau reducing learning rate to $1.9999999949504855e^{-07}$						
Epoch 23	0.6196	0.8176	0.8384	3.6468	0.4617	0.4544
Epoch 24	0.6174	0.8190	0.8376	1.8759	0.4593	0.4533

Table.2. Results after training model 1 on Image Augmentation 1 data

Loss and Metrics vs Epochs	Training			Validation		
	Loss	F1-Score	Categorical Accuracy	Loss	F1-Score	Categorical Accuracy
Epoch 1	2.9792	0.2545	0.3137	2.1032	0.3017	0.3182
Epoch 2	2.7904	0.2737	0.3412	2.9635	0.2854	0.2996
Epoch 3	2.7272	0.2838	0.3523	3.1365	0.3072	0.3238
Epoch 4	2.6693	0.2975	0.3642	3.0420	0.3159	0.3348
Epoch 5	2.6313	0.3052	0.3698	2.9908	0.3264	0.3371
ReduceLROnPlateau reducing learning rate to $1.9999999494757503e^{-05}$						
Epoch 6	2.5801	0.3156	0.3804	3.3823	0.3437	0.3592
Epoch 7	2.5648	0.3224	0.3839	3.0035	0.3507	0.3695
Epoch 8	2.5521	0.3245	0.3862	3.1798	0.3440	0.3580
Epoch 9	2.5427	0.3248	0.3851	3.1259	0.3483	0.3592
Epoch 10	2.5309	0.3300	0.3897	4.2754	0.3479	0.3604

Epoch 11	2.5224	0.3306	0.3905	2.5517	0.3441	0.3588
Epoch 12	2.5157	0.3327	0.3929	2.8440	0.3533	0.3673
Epoch 13	2.5014	0.3361	0.3952	2.8635	0.3596	0.3737
Epoch 14	2.4976	0.3362	0.3965	2.0916	0.3577	0.3716
Epoch 15	2.5006	0.3386	0.3961	2.3885	0.3695	0.3843
Epoch 16	2.4818	0.3433	0.3994	3.4623	0.3738	0.3894
Epoch 17	2.4853	0.3407	0.3973	3.5315	0.3814	0.3964
Epoch 18	2.4679	0.3454	0.4017	4.0982	0.3664	0.3777
Epoch 19	2.4720	0.3429	0.4020	3.0072	0.3534	0.3643
Epoch 20	2.4633	0.3467	0.4033	2.0370	0.3714	0.3831
Epoch 21	2.4565	0.3480	0.4049	3.0719	0.3743	0.3898
Epoch 22	2.4582	0.3457	0.4030	2.1521	0.3841	0.3994
Epoch 23	2.4468	0.3489	0.4059	2.1513	0.3773	0.3901
Epoch 24	2.4434	0.3497	0.4065	3.4570	0.3751	0.3847

Table.3. Results after training model 1 on Image Augmentation 2 data

Loss and Metrics vs Epochs	**Training**			**Validation**		
	Loss	**F1-Score**	**Categorical Accuracy**	**Loss**	**F1-Score**	**Categorical Accuracy**
Epoch 1	2.3762	0.3896	0.4380	3.0034	0.4498	0.4591
Epoch 2	2.1614	0.4321	0.4748	3.4550	0.4563	0.4594
Epoch 3	2.0806	0.4540	0.4928	2.0705	0.4654	0.4659
Epoch 4	2.0270	0.4677	0.5012	2.6434	0.4689	0.4686
Epoch 5	1.9907	0.4750	0.5078	3.4008	0.4713	0.4664
Epoch 6	1.9565	0.4836	0.5149	2.5456	0.4745	0.4727
Epoch 7	1.9340	0.4905	0.5200	2.6562	0.4770	0.4717
ReduceLROnPlateau reducing learning rate to $3.999999898951501e^{-06}$						
Epoch 8	1.9125	0.4962	0.5245	2.1752	0.4789	0.4749
Epoch 9	1.9480	0.4875	0.5180	2.2418	0.4633	0.4578
Epoch 10	1.8865	0.5057	0.5309	1.9330	0.4670	0.4626
Epoch 11	1.8444	0.5142	0.5399	2.3276	0.4699	0.4613
Epoch 12	1.8077	0.5230	0.5463	2.8729	0.4709	0.4622
Epoch 13	1.7801	0.5318	0.5540	1.8726	0.4744	0.4681
Epoch 14	1.7554	0.5364	0.5579	1.0774	0.4804	0.4703
Epoch 15	1.7379	0.5424	0.5640	2.4314	0.4719	0.4654
Epoch 16	1.7120	0.5492	0.5690	1.9042	0.4759	0.4726
Manually changing learning rate to $1e^{-5}$						
Epoch 17	1.6420	0.5642	0.5832	2.2347	0.4945	0.4893
Epoch 18	1.6195	0.5714	0.5914	1.7109	0.4987	0.4897
Epoch 19	1.6093	0.5749	0.5924	2.1635	0.4979	0.4897
Epoch 20	1.6080	0.5777	0.5934	2.6233	0.4975	0.4877
Epoch 21	1.6051	0.5782	0.5946	2.4903	0.4983	0.4915
Epoch 22	1.6010	0.5794	0.5966	1.9718	0.4978	0.4880
Epoch 23	1.6039	0.5779	0.5941	1.8970	0.5008	0.4892
Epoch 24	1.5897	0.5832	0.5976	2.1555	0.5017	0.4904

Table.4. Results after transfer learning Xception convolution base with a dense layer fitted on top on Image Augmentation 2 data

Loss and Metrics vs Epochs	**Training**			**Validation**		
	Loss	**F1-Score**	**Categorical Accuracy**	**Loss**	**F1-Score**	**Categorical Accuracy**
Epoch 1	2.3766	0.4194	0.4477	2.4278	0.5390	0.5319
Epoch 2	1.6484	0.5906	0.5883	1.6726	0.5895	0.5799
Epoch 3	1.3680	0.6580	0.6506	0.8879	0.6215	0.6070
Epoch 4	1.1751	0.7017	0.6925	1.8066	0.6457	0.6236
Epoch 5	1.0302	0.7352	0.7254	0.8405	0.6439	0.6225
Epoch 6	0.9003	0.7649	0.7565	1.9879	0.6424	0.6185
Epoch 7	0.7971	0.7887	0.7805	2.8998	0.6541	0.6314
Epoch 8	0.7067	0.8110	0.8031	1.1810	0.6572	0.6345
Manually changing learning rate to $1e^{-4}$						
Epoch 9	0.5428	0.8504	0.8480	1.2028	0.6906	0.6716
Epoch 10	0.5397	0.8511	0.8493	0.5792	0.6916	0.6719
Epoch 11	0.5404	0.8515	0.8492	1.8780	0.6915	0.6719
Epoch 12	0.5362	0.8516	0.8509	1.4777	0.6921	0.6727

Table.5. Results after transfer learning InceptionResNetV2 convolution base with a dense layer fitted on top on Image Augmentation 2 data

Loss and Metrics vs Epochs	**Training**			**Validation**		
	Loss	**F1-Score**	**Categorical Accuracy**	**Loss**	**F1-Score**	**Categorical Accuracy**
Epoch 1	2.6320	0.3468	0.3902	2.6481	0.4794	0.4929
Epoch 2	1.8931	0.5212	0.5320	2.6178	0.5086	0.5000
Epoch 3	1.5905	0.5971	0.5974	2.0026	0.5548	0.5438
Epoch 4	1.3528	0.6530	0.6496	0.8843	0.5875	0.5698
Epoch 5	1.1756	0.6960	0.6894	1.0983	0.6012	0.5813
Epoch 6	1.0143	0.7347	0.7267	2.8491	0.6053	0.5811
Epoch 7	0.8827	0.7669	0.7584	3.3372	0.6147	0.5897
Epoch 8	0.7671	0.7939	0.7870	1.8681	0.5997	0.5788
Manually changing learning rate to $1e^{-4}$						
Epoch 9	0.6697	0.8193	0.8117	1.7691	0.6222	0.5993
Epoch 10	0.5918	0.8398	0.8340	1.5458	0.6211	0.6001
Epoch 11	0.5165	0.8581	0.8520	2.1342	0.5733	0.5535
Epoch 12	0.4658	0.8718	0.8663	2.5184	0.6338	0.6148
Epoch 13	0.3700	0.8966	0.8933	2.9483	0.6485	0.6291
Epoch 14	0.3632	0.8983	0.8961	2.6975	0.6499	0.6299
Epoch 15	0.3639	0.8990	0.8963	1.8291	0.6495	0.6307
Epoch 16	0.3580	0.9001	0.8974	1.3072	0.6508	0.6319

Table.6. Results after transfer learning MobileNetV2 convolution base with a dense layer fitted on top on Image Augmentation 2 data

Loss and Metrics vs Epochs	**Training**			**Validation**		
	Loss	**F1-Score**	**Categorical Accuracy**	**Loss**	**F1-Score**	**Categorical Accuracy**
Epoch 1	3.0173	0.2406	0.3073	11.7397	0.0164	0.0169

Epoch 2	2.3245	0.3983	0.4359	9.1230	0.0536	0.0599
Epoch 3	2.0484	0.4710	0.4925	6.6012	0.1549	0.1512
Epoch 4	1.8744	0.5173	0.5293	5.1103	0.1634	0.1751
Epoch 5	1.7313	0.5553	0.5614	3.4618	0.3020	0.2963
Epoch 6	1.6110	0.5839	0.5863	5.6468	0.3184	0.3081
Epoch 7	1.5177	0.6068	0.6082	3.6592	0.3964	0.3816
Epoch 8	1.4328	0.6276	0.6258	2.4346	0.4355	0.4183
Epoch 9	1.3580	0.6483	0.6424	2.7962	0.4705	0.4542
Epoch 10	1.2763	0.6669	0.6623	2.6735	0.4589	0.4440
Epoch 11	1.2154	0.6789	0.6743	2.0015	0.4586	0.4425
Epoch 12	1.1602	0.6941	0.6868	2.4261	0.4804	0.4631
Epoch 13	1.1024	0.7059	0.6999	1.8106	0.4948	0.4777
Epoch 14	1.0518	0.7207	0.7145	1.8682	0.4655	0.4498
Epoch 15	1.0013	0.7314	0.7258	4.2774	0.5110	0.4861
Epoch 16	0.9638	0.7418	0.7344	1.5714	0.4687	0.4511
Manually changing learning rate to 1e^{-4}						
Epoch 17	0.8031	0.7788	0.7757	1.3163	0.6307	0.6117
Epoch 18	0.8118	0.7772	0.7741	1.6644	0.6390	0.6174
Epoch 19	0.8039	0.7786	0.7758	1.3992	0.6406	0.6195
Epoch 20	0.8018	0.7795	0.7777	2.0463	0.6394	0.6197
Epoch 21	0.8024	0.7788	0.7757	2.0415	0.6401	0.6206
Epoch 22	0.7992	0.7803	0.7777	1.3196	0.6409	0.6215
Epoch 23	0.7967	0.7804	0.7776	1.7722	0.6406	0.6218
Epoch 24	0.7938	0.7822	0.7786	2.5375	0.6407	0.6204

Table.7. Results after fine-tuning the last convolution block of MobileNetV2

Loss and Metrics vs Epochs	**Training**			**Validation**		
	Loss	**F1-Score**	**Categorical Accuracy**	**Loss**	**F1-Score**	**Categorical Accuracy**
Epoch 1	0.7048	0.8073	0.8015	1.3346	0.6553	0.6364
Epoch 2	0.7043	0.8096	0.8038	2.3650	0.6546	0.6354
Epoch 3	0.7106	0.8063	0.8002	1.9359	0.6551	0.6366
Epoch 4	0.7103	0.8061	0.7995	1.9308	0.6552	0.6356

Table.8. Results after transfer learning DenseNet201 convolution base with a dense layer fitted on top on Image Augmentation 2 data

Loss and Metrics vs Epochs	**Training**			**Validation**		
	Loss	**F1-Score**	**Categorical Accuracy**	**Loss**	**F1-Score**	**Categorical Accuracy**
Epoch 1	3.8564	0.1024	0.1747	3.2780	0.2292	0.2605
Epoch 2	2.8137	0.2742	0.3395	4.9444	0.2934	0.3098
Epoch 3	2.5553	0.3422	0.3912	2.9525	0.2958	0.3148
Epoch 4	2.2713	0.4119	0.4493	2.9322	0.3883	0.3864
Epoch 5	2.0819	0.4630	0.4874	1.9389	0.4762	0.4675
Epoch 6	2.0425	0.4791	0.4955	2.1287	0.4653	0.4626
Epoch 7	2.0010	0.4874	0.5031	2.7383	0.4826	0.4732
Epoch 8	1.8498	0.5293	0.5354	1.9524	0.5104	0.5111
Manually changing learning rate to 1e-4						
Epoch 9	1.6537	0.5712	0.5769	1.4003	0.5718	0.5675
Epoch 10	1.6433	0.5711	0.5807	1.3074	0.5745	0.5698
Epoch 11	1.6200	0.5768	0.5882	2.4057	0.5742	0.5693
Epoch 12	1.6488	0.5711	0.5834	1.7445	0.5731	0.5686
Epoch 13	1.6376	0.5725	0.5826	1.3678	0.5777	0.5714
Epoch 14	1.6261	0.5770	0.5845	1.5088	0.5794	0.5750
Epoch 15	1.5915	0.5851	0.5942	1.7061	0.5940	0.5879
Epoch 16	1.5390	0.6002	0.6080	1.6332	0.6043	0.5972
Epoch 17	1.5311	0.6045	0.6125	1.9447	0.6041	0.5980
Epoch 18	1.5191	0.6055	0.6137	1.7622	0.6056	0.5993
Epoch 19	1.4819	0.6135	0.6214	1.9073	0.6128	0.6049
Epoch 20	1.4336	0.6226	0.6352	1.7370	0.6188	0.6093
Epoch 21	1.4276	0.6279	0.6351	1.3144	0.6216	0.6154
Epoch 22	1.4094	0.6332	0.6376	2.0288	0.6268	0.6198
Epoch 23	1.4000	0.6353	0.6419	1.0301	0.6276	0.6204
Epoch 24	1.4001	0.6366	0.6418	1.3012	0.6290	0.6198
Epoch 25	1.3961	0.6367	0.6427	1.7881	0.6287	0.6212
Epoch 26	1.3959	0.6362	0.6428	1.6528	0.6292	0.6200

4.6 FINAL RESULTS OF ALL THE MODELS

The VGG16 inspired model was trained with 50.4 million parameters for 72 epochs (24 epochs for each of the three stages). After this, the model started overfitting.

The Xception model was trained with 21.2 million parameters (pre-trained ImageNet weights) for 12 epochs. The training had to be stopped after that since the validation metrics stopped improving further.

The InceptionResNetV2 model was trained with 54.6 million parameters (pre-trained ImageNet weights) for 16 epochs after which the model started overfitting.

The MobileNetV2 model was trained with 2.5 million parameters (pre-trained ImageNet weights) for 28 epochs (including 4 epochs for fine-tuning). It is the only model that showed improvement on fine-tuning.

The DenseNet201 model was trained with 18.7 million parameters (pre-trained ImageNet weights) for 26 epochs. The training was stopped after this since it started overfitting.

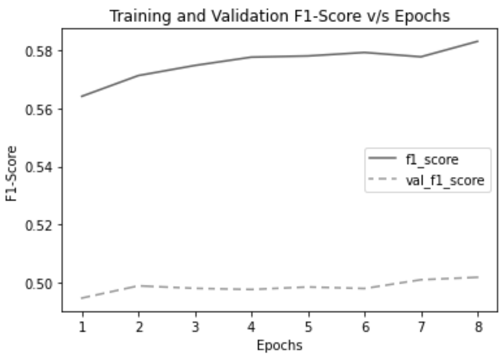
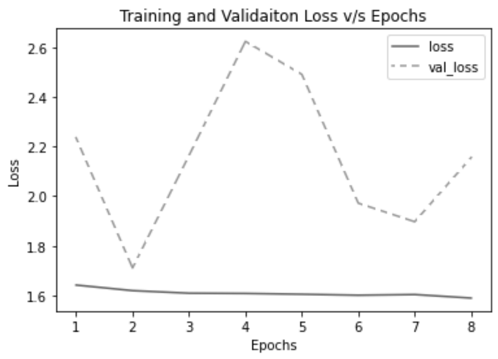
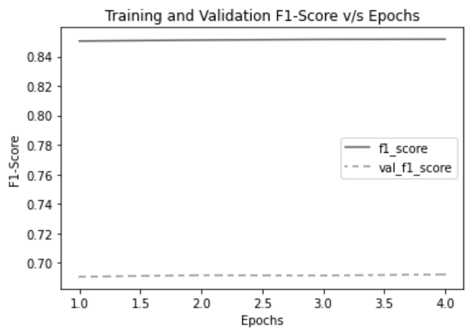
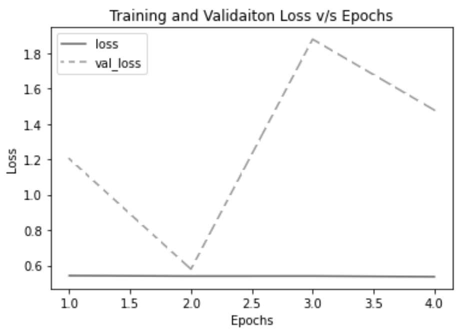
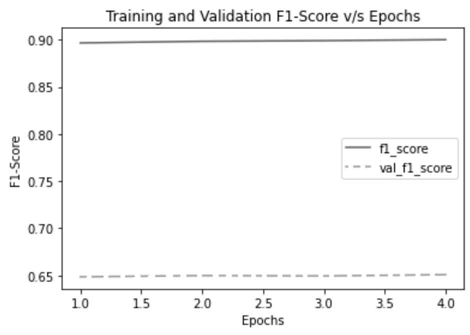
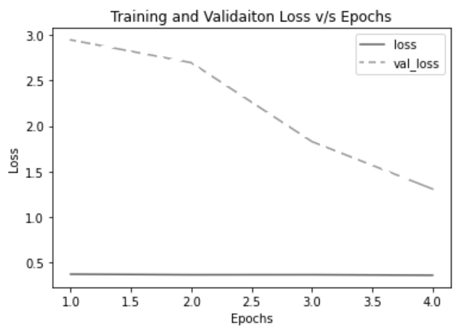
The ensembled model has 147.6 million parameters. It scored a categorization accuracy of 0.71300 on Kaggle [34], which means a categorical accuracy of 71.3%.

Thus, ensembling these models have provided better results than any of the above-mentioned models individually. The final results of the models have been compiled in Table.9 and the final plots of “f1-score v/s epochs” and “loss v/s epochs” for each model is given in Table.10.

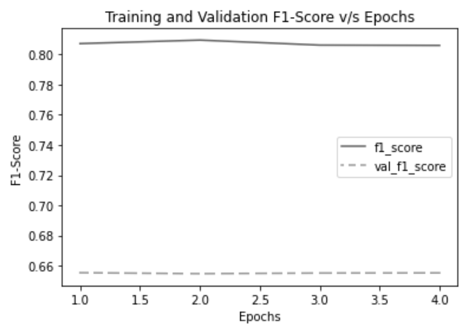
Table.9. Final results of all the models after training them on Image Augmentation 2 data

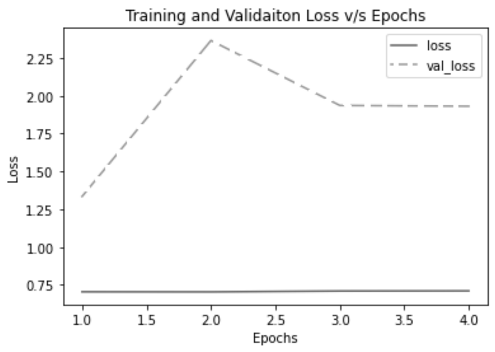
Loss and Metrics vs Models	**Training**			**Validation**		
	Loss	**F1-Score**	**Categorical Accuracy**	**Loss**	**F1-Score**	**Categorical Accuracy**
VGG16 inspired model	1.5897	0.5832	0.5976	2.1555	0.5017	0.4904
Xception	0.5362	0.8516	0.8509	1.4777	0.6921	0.6727
InceptionResNetV2	0.3580	0.9001	0.8974	1.3072	0.6508	0.6319
MobileNetV2	0.7103	0.8061	0.7995	1.9308	0.6552	0.6356
DenseNet201	1.3959	0.6362	0.6428	1.6528	0.6292	0.6200
Ensembled Model	Kaggle Best Categorization Accuracy Score = 0.71300					

Table.10. Final performance plots for each model

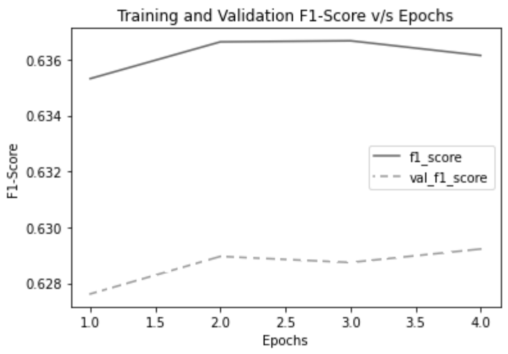
Performance plots vs. Models	**F1-Score v/s Epochs**	**Loss v/s Epochs**
VGG16 inspired model		
Xception		
InceptionResNetV2		

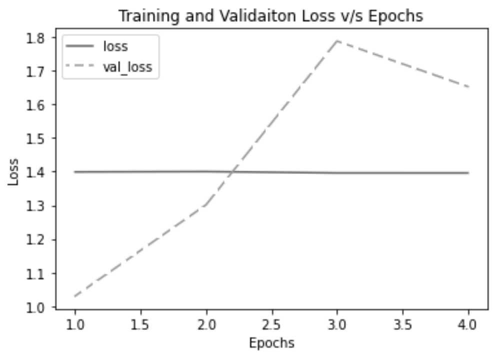
MobileNetV2





DenseNet201





4.7 FINAL RESULTS OF ALL THE MODELS

The ensembled model was tested on other data as well to check its performance, as shown in Table.11. During testing, images of different dimensions were given as input. The model was programmed to first convert each image to 128×128 size and then start the classification process.

Table.11. Testing the Ensembled Model on Various Data

Image	Predicted class	Result
	Grasshopper, Hopper	✓
	Syringe	×
	Parking Meter	✓
	Brown Bear	✓
	Wooden Spoon	✓ (Could be "ice-cream" as well)
	Goose	✓
	Golden Retriever	✓
	Grasshopper, Hopper	✓

4.8 COMPARISON OF RESULTS OBTAINED WITH RELATED WORK

The related works' models were trained on the same dataset to check where the final ensembled model's algorithm and training process stands with respect to other models. The related models which were trained were: LeNet-5 [3], AlexNet [5], VGG16 [10], GoogLeNet [13] and ResNet50 [14].

All these models were trained with the following properties:

- Every model was trained from scratch (no pre-trained weights were incorporated in the learning process)
- No image augmentation was done
- Each input image was rescaled to 1/255
- Each input image was resized to 128×128. Thus, input size of each image was 128×128×3 (due to RGB values)
- Batch size = 64
- An additional Dense (Fully Connected) Layer with 200 units and softmax non-linearity was attached at the top of each model for classification
- **Optimizer used**: Stochastic Gradient Descent (SGD) with learning rate = 0.001 and momentum = 0.9
- **Callback used**: ReduceLROnPlateau callback with factor of 0.2, patience of 3 and minimum learning rate of 10^{-7} was made to monitor the validation loss
- **Loss Function used**: Categorical Cross-entropy
- **Evaluation Metrics**: F1-Score, Categorical Accuracy

AlexNet model was trained on one GPU (Google Colaboratory GPU) only. The original model was trained on two GPUs simultaneously.

All of the training process was done till the respective models started overfitting. No image augmentation, manual change in learning rate or inspection on which optimizer works the best was done in training the related works' models.

This comparison was done only to check whether the procedure of the research work described in this paper provides more fruitful results than the results given by already established models when they are trained without any type of data augmentation. The results given by the above-mentioned models and their comparison are given in Table.12. Thus, the training process works quite well in comparison to the related works' models trained without data augmentation.

4.9 ANALYSIS OF OBTAINED RESULTS

The training process of each of the independent models was broken down into several steps of some epochs so that the loss and evaluation metrics of both training and validation data could be checked periodically and training of that model could be stopped when it started overfitting.

After each step of training, "f1-score v/s epochs" and "loss v/s epochs" graphs for both training and validation data were plotted to easily check whether the model needs to be trained further. Although in few cases, the training could not be stopped at the optimal epoch (since interrupting execution of the statement would result in a KeyboardInterrupt Error which could result in losing the data acquired from the previous epochs), the results obtained at the end was quite satisfactory in terms of validation f1-score and validation categorical accuracy.

The "f1-score v/s epochs" plot was mainly used to check rather than categorical accuracy because to get a higher f1-score, the model needs to perform well both in terms of precision and recall.

The ensembled model performs well on most of the images with some errors. Most of the errors by the ensembled model were mainly due to two reasons: low resolution of the images and, incapability to detect the primary object in an image where there are multiple entities present. For example, an image containing a cup of ice-cream and a wooden spoon, the model predicts it to belong to the class 'wooden spoon'.

The model works well otherwise. So, this model would work very well if it was deployed as an application which predicts the class an image belongs to by checking any content of that image.

Table.12. Comparison of performances of ensembled model with related works

Epochs and performance vs. Models	Number of epochs	Training categorical accuracy	Validation categorical accuracy
LeNet-5	40	0.4555	0.1200
AlexNet	80	0.8943	0.3616
VGG16	27	0.6647	0.2349
GoogLeNet	75	0.7826	0.3987
ResNet-50	12	0.7098	0.3041
Ensembled Model	Kaggle Best Categorization Accuracy Score = 0.71300		

5. CONCLUSION

Most of the research work mentioned in this paper was done using Keras [35] library. The primary goal was to achieve a top-1 test categorical accuracy of 60%. Submitting the results at Kaggle, the model earned a categorization accuracy score of 0.71300, that is 71.3%.

5.1 REASONS FOR USING THE PROPOSED MODELS

The models which were trained here are a VGG16 inspired model (trained from scratch), Xception, InceptionResNetV2, MobileNetV2 and DenseNet201. These models were selected specifically keeping in mind that the number of parameters to be trained does not increase too much and at the same time, we get a good performance from the ensembled model (both in terms of f1-score and categorical accuracy) on the test/unseen data. It is basically a trade-off between the number of parameters to be trained and the performance of the model (in terms of time required to train the model and produce the output).

5.2 ADVANTAGES OF USING THE ENSEMBLED MODEL FOR CLASSIFICATION

The final model here is the ensembled model of five models (weak learners), which are very efficient models themselves. That is why it has achieved an accuracy of 71.3% on unseen data. This model can be used as the backend for any mobile or web application to classify images among the 200 classes.

5.3 DISADVANTAGES OF USING THE ENSEMBLED MODEL FOR CLASSIFICATION

Although the model has achieved its primary goal of attaining at least 60% accuracy, it still has room for improvement. The final ensembled model has 147.6 million parameters, which is huge and takes some time to produce the output. Also, the model sometimes fails to detect the primary object in an image and classifies the image based on any object present in the image due to which this model sometimes fails to perform its work efficiently.

5.4 FURTHER IMPROVEMENTS

The main motive in the future would be to get a better f1-score and categorical accuracy, and at the same time make the model a light-weight one (with smaller number of parameters). Once this has been achieved, deploying this model into an application would be more fruitful since it would then take lesser time in providing the result to the end user. For more improvement, the model could be trained on other datasets containing more classes as well.

The models using which transfer learning was done here were attempted to be fine-tuned only on the last convolution blocks of the respective models. In the future, other combinations of convolution blocks could be tried to be fine-tuned hoping for a better result.

The models here were trained on scarce computation resources (due to usage limits on Google Colaboratory [36] GPU). In the future, it would be better if further training is done on a GPU as powerful as the ones Google Colaboratory offer but with more lenient usage limits.

REFERENCES

[1] D.H. Ballard and C.M. Brown, "*Computer Vision*", Prentice Hall, 1982.

[2] Github, "Convolution Neural Networks for Visual Recognition", Available at: https://cs231n.github.io/classification/, Accessed at 2020.

[3] Y. Le Cun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-Based Learning Applied to Document Recognition", *Proceedings of the IEEE*, Vol. 86, pp. 1-13, 1998.

[4] Y. Le Cun, Y. Bengio and G. Hinton, "Deep Learning", *Nature*, pp. 436-444, 2015.

[5] A. Krizhevsky, I. Sutskever and G. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks", *Neural Information Processing Systems*, Vol. 23, No. 1, pp. 1-14, 2000.

[6] J. Deng, W. Dong, R. Socher, L. Li, Kai Li and Li Fei Fei, "ImageNet: A Large-Scale Hierarchical Image Database", *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248-255, 2009.

[7] Image Net, "Image Net", Available at:http://www.image-net.org/. Accessed at 2020.

[8] F. Sultana, A. Sufian and P. Dutta, "Advancements in Image Classification using Convolutional Neural Network", *Proceedings of International Conference on Research in Computational Intelligence and Communication Networks*, pp. 122-129, 2018.

[9] Image Net, "ImageNet Large Scale Visual Recognition Challenge 2012". Available at: http://image-net.org/challenges/LSVRC/2012/, Accessed at 2020.

[10] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition", Proceedings of *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1-14, 2014.

[11] VGG16, Available at: https://neurohive.io/en/popular-networks/vgg16/.Accessed at 2020.

[12] ImageNet Large Scale Visual Recognition Challenge 2014, Available at: http://image-net.org/challenges/LSVRC/2014/.Accessed at 2020.

[13] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, "Going Deeper with Convolutions", *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1-9, 2015.

[14] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition", *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770-778, 2016.

[15] ImageNet Large Scale Visual Recognition Challenge 2015, Available at: http://image-net.org/challenges/LSVRC/2015/, Accessed at 2020.

[16] S. Albawi, T.A. Mohammed and S. Al Zawi, "Understanding of a Convolutional Neural Network", *Proceedings of International Conference on Engineering and Technology*, pp. 1-6, 2017.

[17] A. Mikołajczyk and M. Grochowski, "Data Augmentation for Improving Deep Learning in Image Classification Problem", *Proceedings of International Conference on Engineering and Technology*, pp. 117-122, 2018.

[18] W.H. Beluch, T. Genewein, A. Nurnberger and J.M. Kohler, "The Power of Ensembles for Active Learning in Image Classification", *Proceedings of International Conference on Computer Vision and Pattern Recognition*, pp. 9368-9377, 2018.

[19] V. Thakkar, S. Tewary and C. Chakraborty, "Batch Normalization in Convolutional Neural Networks - A Comparative Study with CIFAR-10 Data", *Proceedings of International Conference on Emerging Applications of Information Technology*, pp. 1-5, 2018.

[20] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", *Journal of Machine Learning Research*, Vol. 12, No. 1, pp. 1929-1958, 2014.

[21] E.M. Dogo, O.J. Afolabi, N.I. Nwulu, B. Twala and C.O. Aigbavboa, "A Comparative Analysis of Gradient Descent-Based Optimization Algorithms on Convolutional Neural Networks", *Proceedings of International Conference on Computational Techniques, Electronics and Mechanical Systems*, pp. 92-99, 2018.

[22] Data Science, "Vanishing Gradient Problem". Available at: https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484. Accessed at 2020.

[23] SGD Optimizer, Available at: https://keras.io/api/optimizers/sgd/. Accessed at 2020.

[24] Reduce LR on Plateau Callback, Available at: https://keras.io/api/callbacks/reduce_lr_on_plateau/, Accessed at 2020.
[25] Categorical Cross-Entropy Loss, Available at: https://keras.io/api/losses/probabilistic_losses/#categoricalcrossentropy-class, Accessed at 2020.
[26] Image Data Generator Class, Available at: https://keras.io/api/preprocessing/image/#imagedatagenerator-class, Accessed at 2020.
[27] Image Augmentation, Available at https://imgaug.readthedocs.io/en/latest/, Accessed at 2020.
[28] F. Chollet, "Xception: Deep Learning with Depthwise Separable Convolutions", *Proceedings of International Conference on Computer Vision and Pattern Recognition*, pp. 1800-1807, 2017.
[29] Image-Net Data, Available at: http://www.image-net.org/, Accessed at 2020.
[30] Ada Delta Optimizer, Available at https://keras.io/api/optimizers/adadelta/. Accessed at 2020.
[31] C. Szegedy,S. Ioffe,V. Vanhoucke andA. Alemi, "Inception-V4, Inception-ResNet and the Impact of Residual Connections on Learning", *Proceedings of International Conference on Artificial Intelligence*, pp. 1-7, 2016.
[32] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov and L. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks", *Proceedings of International Conference on Computer Vision and Pattern Recognition*, pp. 4510-4520, 2018.
[33] G. Huang, Z. Liu, L. Van Der Maaten and K.Q. Weinberger, "Densely Connected Convolutional Networks", *Proceedings of International Conference on Computer Vision and Pattern Recognition*, pp. 2261-2269, 2017.
[34] Kaggle Leaderboard, Available at: https://www.kaggle.com/c/image-detect/leaderboard, Accessed at 2020.
[35] Keras Library, Available at: https://keras.io/, Accessed at 2020.
[36] Google Colaboratory, Available at: https://colab.research.google.com/, Accessed at 2020.
[37] Image Dataset, Available at:https://www.kaggle.com/c/image-detect/data, Accessed at 2020.