INTRODUCTION

This report provides a detailed analysis on the work done in fulfilment of the image classification project. The objective of this project is to develop an image classifier to identify images of cherries, strawberries and tomatoes using keras with tensorflow backend. A total of 4500 images with 1500 in each class were provided to build the classifier. The goal here is to create a convolutional neural network using the given images which can then classify previously unseen images as well. A standard neural network will be used as a baseline to evaluate the performance of this CNN. Since training a high accuracy classifier from scratch usually requires a lot of time and computation transfer learning was used. Imagenet is a large visual database with over twenty thousand categories of images, each category having several hundred images. Since 2010, the ImageNet project runs an annual software contest, the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), where software programs compete to correctly classify and detect objects and scenes. The pre-trained weights for the top models are available as keras libraries. These weights were fine-tuned with the images from the three classes respectively. This approach was taken because the Imagenet database already had several thousand images of various fruits. Hence the pre-trained models will already be capable of identifying some patterns necessary for classifying images. All the coding work for this project was done in Google Colab since it provides a 12GB Tesla K80 GPU for free.

Problem Investigation

The dataset contains a total of 4500 images with 1500 images in each class; ‘cherry’, ‘strawberry’ and ‘tomato’. All the classes are mutually exclusive i.e each image belongs to one class only. All the images have been re-sized to the same dimensions 300X300 pixels. The images show different properties with respect to background, light condition, number of objects, resolution etc. The first step in building a classifier is pre-processing the data by exploratory data analysis.

The format of the images is a very important factor while building a classifier. Most image classification algorithms can only handle jpg, jpeg or png format images. A preliminary analysis of the dataset shows that all the images are in jpeg format. So no further conversion is necessary to pre-process the image format.

Data augmentation is extremely useful when the given dataset does not contain enough instances train a proper classifier. In this case the dataset only contains 1500 images per class which may not be sufficient for CNN to learn and identify different patterns. For images classification data augmentation may be defined as creating more images by making simple distortions to the given images like crops, scales and flips. These reflect the kind of variations we expect in the real world, and so can help train the model to cope with natural data more effectively.

In this project augmentation is performed by using the ImageDataGenerator() function in keras.preprocessing.image library. Various augmentation parameters can be set inside the

ImageDataGenerator function. It randomly selects one or more of the given augmentation operations and applies it onto the selected image in order to generate more images. The augmentation operations specified here are:

1. **rotation_range=10**

This parameter performs a random number of rotations on the given image. The value 10 specifies the degree of each rotation.

2. **width_shift_range=0.1**

This parameter is used for shifting the image horizontally. It can help the classifier identify off-center images properly.

3. **height_shift_range=0.1**

Height shifting moves the image vertically. Similar to width shifting it can help identify off-center images.

4. **shear_range=0.15**

This parameter randomly applies shearing transformations. The value 0.15 defines the shear angle in counter clockwise direction in degrees.

5. **zoom_range=0.1**

This parameter defines the range for a random zoom. A value of 0.1 indicates that the image can zoom anywhere between 0.9 and 1.1.

6. **channel_shift_range=10**

Channel shifting is the process of taking the red, green or blue values of pixels in an image and applying those values to pixels in different positions on the image. A value of 10 indicates very high channel shifting. This is done to ensure that the classifier can recognize the shape of fruit properly and thus be capable of working on noisy images.

7. **horizontal_flip=True**

Randomly flips input horizontally.

The code used for augmenting the dataset is given in ‘augment.py’. For each image one or more of the above operations were performed 5 times. Thus 5 additional augmented images were created for every image. Out of the 1500 original images in each of the classes 100 were separated and kept aside for validation. There were 1400 images left in each of the three classes; so a total of 4200 images. Since each image was augmented to create 5 additional images, the total number of images rose up to 25200.

The next pre-processing step is to convert the pixel ratio to the required target ratio of the pre-trained classifiers that are to be used. Here three different pre-trained models were tested to find out which performs best for the given classification problem. All of the three models required a 224X224 image ratio. Hence the given 300X300 ratio needs to be converted to the target ratio. This was done using target_size parameter in the flow_from_directory() function within ImageDataGenerator(). It was done inside the main program file 'train.py'.

Methodology

In this section the various aspects taken into consideration for training the convolutional neural network shall be explained. The various specifications used for building the classifier are given below.

1. Using the given images

The given dataset contained a total of 4500 images with 1500 images belonging to each class. As mentioned before from the 1500 images of each class 100 were taken to form the validation set and the remaining 1400 formed the training set. So there were a total of 4200 images in the training set and 300 in the validation set. In the previous it was explained how the 4200 images were augmented to form 25200 images. During training the 25200 images were used to train the classifier and the remaining 300 to validate it during each epoch. The batch training approach was used here to build the network. In batch training the adjustment delta values are accumulated over all training items, to give an aggregate set of deltas, and then the aggregated deltas are applied to each weight and bias. This is contrary to online training where weights and bias values are updated for every training item. The batch size was set to 300 for the training set and 50 for the validation set.

2. Loss function

The loss function used for this network is categorical crossentropy. In information theory, the cross entropy between two probability distributions p and q over the same underlying set of events measures the average number of bits needed to identify an event drawn from the set, if a coding scheme is used that is optimized for an "artificial" probability distribution q, rather than the "true" distribution p. If only two classes need to be classified binary crossentropy can be used, but since there are three classes here categorical crossentropy is used. When dealing with a classification problem it is generally better to use a crossentropy error function rather than classification error or mean squared error function. Both CNN and traditional neural networks output their predictions in terms of probabilities i.e they calculate the probability of an instance belonging to a particular class. Since it uses a natural logarithm function, cross-entropy takes into account the closeness of a prediction and is a more granular way to compute error. When compared to cross-entrophy mean squared error gives too much emphasis for incorrect inputs. It should be noted that this is only useful during training. Afterwards during validation a regular classification error works better.

3. Optimization method

An optimization method is the technique used by machine learning algorithms to reduce the error rate or loss of the classifier. The optimizer used here is Adam which stands for Adaptive Moment Estimation. Adaptive Moment Estimation (Adam) is a method that computes adaptive learning rates for each parameter. A learning rate is maintained for each network weight (parameter) and separately adapted as learning unfolds. This is shown to have a significant advantage when classifying images.

4. Activation Function

An activation function is a node that is connected to the output of a network. It maps the resulting output values between 0 and 1 or -1 and 1 etc depending upon the function. Sigmoid function works by converging the output to a value between 0 and 1 making an S-shaped curve. The softmax function is a more generalized sigmoid activation function which is used for multiclass classification. It was used here because it ensured that the probabilities of all the three classes added up to 1. This is unlike binary classification where probabilities can sum up to more or less than 1. Using the softmax activation function followed by log loss function categorical crossentropy leads to a model that predicts values in the range of true probabilities, thus making them interpretable. Generating a more interpretable model helps identify how well the classifier is able to label each image.

5. Hyper-parameter settings

When building any machine learning classifier one of the most important aspects is to tune the hyper-parameters properly. These determine how the system evolves over time. In this particular classification problem the learning rate and the performance metric for testing the classifier were set specifically.

Learning rate is a hyper-parameter that controls how much we are adjusting the weights of our network with respect the loss gradient. The lower the value, the slower the system travels along the downward slope. As mentioned before for this problem, transfer learning is used to fine tune a pre-trained model. So the model should have some idea on how to identify the curves and shapes necessary for classifying an image. Hence the weights and biases should already be within reasonable margin of the optimal value. Considering this the learning rate was set to really low value of 0.0001. This should fine tune the model and improve its accuracy. Since the values are already within reasonable margin it probably won't add much overhead to the computation time.

The performance metric for evaluating the classifier is another hyper-parameter to be considered. It was set to 'accuracy'. In an unbalanced dataset accuracy would not serve well as the ideal performance metric. But for this dataset all the classes have exactly the same number of images. Hence it is perfectly balanced. So accuracy was chosen as the suitable performance metric here.

6. Transfer Learning

Transfer learning describes relatively broad speaking, the process of using knowledge gained by solving one set of problems, to solve your current problem. In this case we will use a model that is already familiar with a vast database of images. As mentioned before the models pre-trained using the Imagenet database satisfy this criterion. In any CNN different convolutional layers extract different features from the images. The lower layers of the model would extract very basic features from the image like borders or curves. These lower layers can be kept intact while fine-tuning the higher layers to identify the complex shapes unique to this task.

For this project the primary factors that have been considered in selecting a pre-trained model was depth and size of the model. The depth of the model is the number of layers of nodes that are in it and the size denotes the size of the model file. If the depth of the model is too large it would take a long time to identify how many layers to fine tune to gain optimal performance. This is because the number of layers at the end to be trained can only be identified by trial and error since it varies depending on the task. And obviously the lower the size of the model the easier it would be to save, load and use.

At first the VGG16 and VGG19 models were considered. They only had 23 and 26 layers respectively which are the lowest among all the models. However they had a lot of tuneable parameters, 138 million and 143 million respectively which meant that it would have large file sizes. Both of them had file sizes over 500MB. Since the model already had so many tuned parameter values available, it would be reasonable to assume that it would perform better than smaller models. Hence only the final layer was fine tuned. The model was first converted to a sequential model using the keras sequential API. The final layer was changed to work with three classes instead of the 1000 that comes with the Imagenet weights. This layer was then fine-tuned with all the specifications that were mentioned in this section. Additionally the steps per epoch during training were set to 84 and the number of epochs at 30. After training the model managed to achieve a validation accuracy of 86% during the final epoch.

But since these models had large file sizes the possibility of a smaller model had to be explored. The mobilenet model which was taken had a depth of 88 and file size slightly over 20MB. However this is not a sequential model. Hence the sequential API in keras could not be used. So keras functional API was used. The output node was connected to the sixth to last layer of the network and the final 5 layers were discarded. From the remaining layers the optimal number of layers to be re-trained had to be found. The initial assumption while starting with this was that the model would already have some familiarity with the classes present since the Imagenet database contained some images of fruits. So the final five layers were fine-tuned and a validation accuracy of 86% was obtained. In an attempt to improve accuracy the number of trainable layers was changed a few times. After a lot of trial and error it was found that training the last 23 layers yielded a validation accuracy of 89% in the final epoch.

Since a higher accuracy and much smaller size were achieved with mobilenet model it was selected as the final model. The training algorithm for VGG16 and VGG19 can found in the file 'VGG16&VGG19_train.py'. The training algorithm for the final mobilenet model is given in 'train.py'. The final fine-tuned model which is based on mobilenet can be found inside 'model' folder as 'model.h5'.

RESULTS

In order to obtain a baseline performance for the CNN a regular MLP classifier was built and trained. This model needed to be trained from scratch. It would take a lot of time and computation to train the model if the input images need to be 300X300 pixels since that would amount to 90000 input nodes. So the image was resized to 100X100 pixels by using the ImageDataGenerator() function in keras. This would amount to a total of 10000 input nodes. Two hidden layers with 5120 nodes each were added. The final output layer had three nodes since there are only three classes. The learning rate was set to 0.01 and the optimization method was 'adam'. The activation function was softmax.

The regular MLP classifier built here achieved an accuracy of 58% after 30 epochs as opposed to the 89% accuracy obtained by the final fine-tuned mobilenet model. The regular MLP classifier took over 10 hours to train whereas the final mobilenet model only took around 3 hours for fine-tuning. The lower accuracy of the regular classifier can be explained by the absence of convolutional layers. Convolution operations are highly adept at finding borders, shapes and edges in images. This is why CNNs work exceptionally well in classifying images. In addition to this the CNN had already been pre-trained with thousands images from the Imagenet database. This obviously gave it an advantage over simple MLP classifier which only received the images available here.

CONCLUSION AND FUTURE SCOPE

In conclusion it should be noted that the mobilenet model developed here has some advantages and disadvantages. The primary advantage of this model is its small size and high accuracy. It shows that fine-tuning is extremely effective way to develop a good model when the available dataset is small. A limitation that was encountered while building this model was overfitting. The training accuracy had reached 100% after 10 epochs whereas the validation accuracy remained at 89% even at 30 epochs.

A possible improvement that can be made to this model is to use mobilenet v2 instead of mobilenet. This upgraded version was released a year after the original came out. An attempt was made here to use the v2 model but keras has some problems downloading the model files. Another option is to try the NASNetMobile or any of the DenseNET models. Two or more of these models can also be ensemble to get a better classification performance.