Project Report

Object Classification/Recognition using Using CNN Networks and Transfer-learning with EfficientNet-B0

Team Name: DL-Drivers

Team Members

Akanksha Rawat Karishma Kuria Nisha Mohan Devadiga

CMPE 297 - Fall, 2022
11/10/2022

1 Abstract

One of the sub-type of deep neural networks is Convolutional Neural Network (CNN) which has its main applications in the field of image recognition and object detection. CNNs are especially appropriate for these applications because the inbuilt layers in CNN reduces the high dimensionality of any image without losing the important features of the image. This project uses CIFAR-100 which has images from 100 classes.
The project's main objective is to create a convolutional neural network that can accurately identify and classify these items in the photos into 100 classes. To accomplish picture recognition, we employed the cutting-edge EfficientNet-B0 model trained on the ImageNet dataset. For image classification, transfer learning is applied, and the network has been scaled for improved performance. The model only needed 15 training epochs to get 82% accuracy.

2 Introduction

Object detection and image recognition is a very straightforward task for human eyes, but when it comes to machines it's a complicated task. There are several factors that affect the accuracy of the machines for image recognition such as brightness, angle and position. CNN has shown remarkable results in this field. CNN has a great ability of hierarchical feature learning, with all its layers extracting the important features from the image and training the model to learn those features. Studies have shown that the features extracted have good discrimination abilities than the hand-crafted features.
The goal of this project is to develop a convolutional neural network model using transfer learning that can accurately identify and categorize colored photographs of objects into one of the 100 available classes. For humans, picture identification is a pretty straightforward process, but for machines, it can be difficult because of the nearly limitless number of possible permutations for an object depending on its position and the effect of lighting.

The 80 million tiny images dataset CIFAR-100 is a labeled subset, and CIFAR stands for Canadian Institute For Advance Research. Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton gathered the photographs. The dataset comprises 60000 colored images of 32 by 32 pixels in 100 classes, divided into 20 super classes (50000 training and 10000 test). There are fine labels (classes) and coarse labels for each image (super class). This dataset's Python version was taken from the University of Toronto Computer Science website and utilized for the research.

3 Related Work

With the development of deep learning, methods for improving the accuracy of object detection for small objects are now available. The following three areas are the primary focus of deep learning method research.

3.1 Data Augmentation

Because of the uneven distribution of objects of different sizes in preparing tests, data augmentation is the most direct technique to work on the improvement of object recognition. Yu and others propose a scale matching technique that minimizes the loss of information about small objects by cropping objects of varying sizes and minimizing the difference between objects of varying sizes.By replicating small objects in the image, Kisantal et al. propose a method of replication enhancement that increases the number of training samples for small objects. As a result, its ability to detect small objects increases.Chen and others improve the data augmentation result by proposing an adaptive resampling strategy that takes into account the object context information. By increasing the number of small object data samples, data augmentation improves the performance of object detection. On the other hand, this approach necessitates more calculation. However, by employing inappropriate data augmentation strategies, it introduces additional noise.

3.2 Transfer learning

Transfer learning is a well known deep learning technique that follows the methodology of utilizing the information that was learned in an errand and applying it to take care of the issue of the related target task. Therefore, we "transfer" the learned features, which are essentially the network's "weights," rather than building a neural network from scratch. We use "pre-trained models" to put transfer learning into practice.

3.3 EfficientNet

Mingxing Tan and Quoc V. Le of Google Research, Brain team in their research paper ‘EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks’,proposed the ‘EfficientNet’ model that was presented in the International Conference on Machine Learning, 2019. They came up with a novel approach to scaling that scales the network uniformly across its depth, width, and resolution. They developed a brand-new baseline network by employing the neural architecture search, then scaled it up to produce a family of deep learning models known as EfficientNets, which outperform previous Convolutional Neural Networks in terms of accuracy and efficiency. In this project, we have considered EfficientNetB0 as the base model.

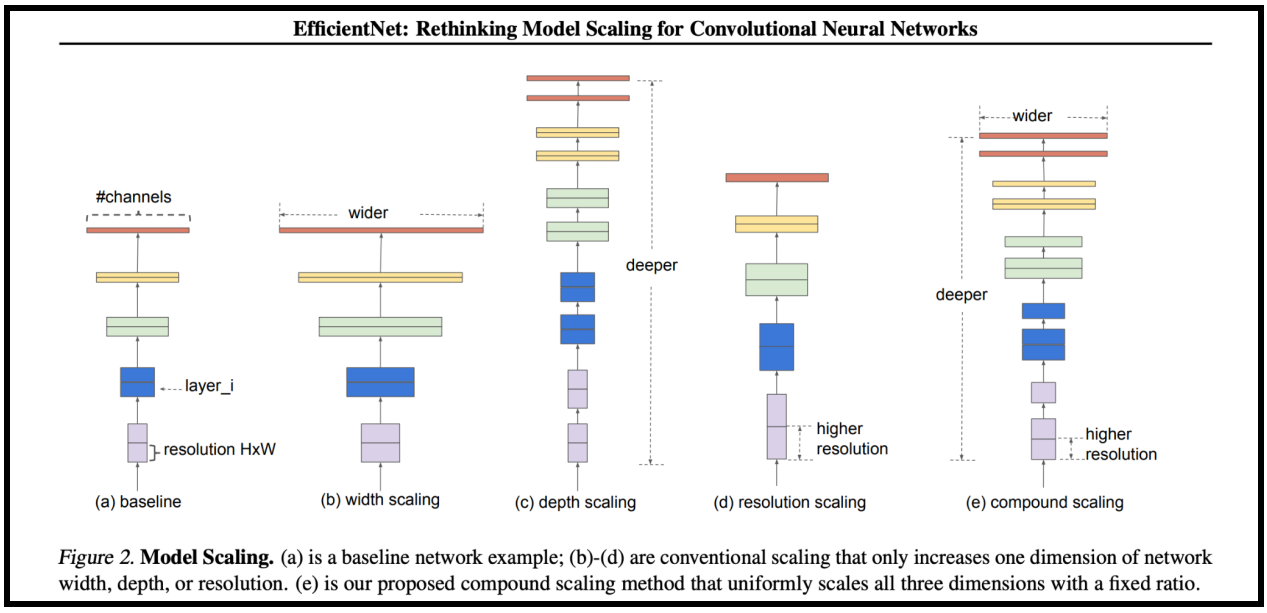


Figure 2. **Model Scaling.** (a) is a baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is our proposed compound scaling method that uniformly scales all three dimensions with a fixed ratio.

Image Source - TowardsDataScience link

Table 1. **EfficientNet-B0 baseline network** – Each row describes a stage i with $\hat{L}_i$ layers, with input resolution $\langle \hat{H}_i, \hat{W}_i \rangle$ and output channels $\hat{C}_i$. Notations are adopted from equation 2.

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224×224	32	1
2	MBConv1, k3x3	112×112	16	1
3	MBConv6, k3x3	112×112	24	2
4	MBConv6, k5x5	56×56	40	2
5	MBConv6, k3x3	28×28	80	3
6	MBConv6, k5x5	14×14	112	3
7	MBConv6, k5x5	14×14	192	4
8	MBConv6, k3x3	7×7	320	1
9	Conv1x1 & Pooling & FC	7×7	1280	1

Image Source - Kernel Size, resolution, channels, and no. of layers information.

4 Methodology

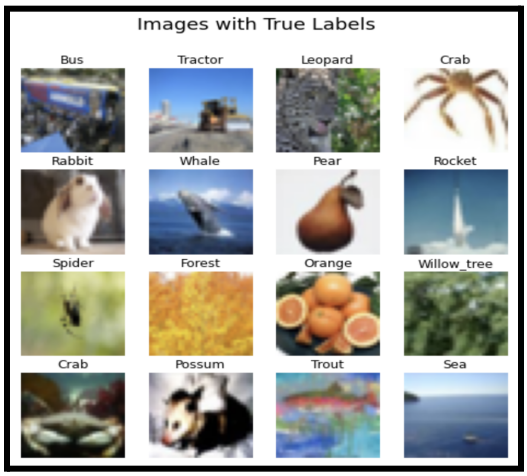
4.1 Business understanding

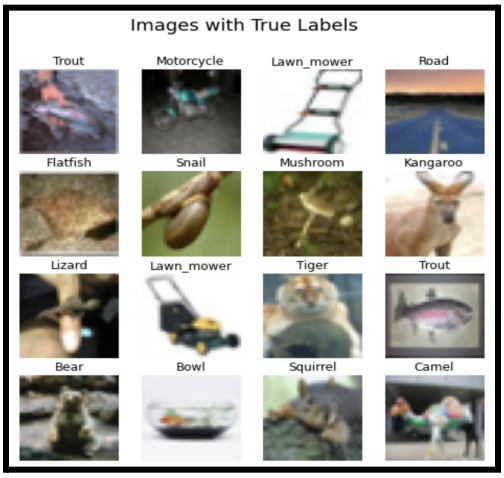
Interactive prototype to take input image frames and provide object classification for the input. Potential application in the future could be to use this model in object analysis tools to recognize and classify colored images.

The difficulty of obtaining a high accuracy score (greater than 59% as accomplished using a 9-layer convolutional neural network constructed earlier) is the driving force behind working on this dataset utilizing transfer learning. The dataset comprises 100 classes, yet there are only 600 photos in each class (500 for training and 100 for testing). The visual quality of this dataset is what we found most intriguing. Each image in the dataset is 32*32 pixels, which makes machine recognition difficult. Transfer learning has therefore been utilized to educate the computer to correctly identify and classify the photos more accurately than it did previously. However, memory is the biggest obstacle to creating a deep neural network for CIFAR-100 with millions of parameters. However, we thought that facing all of these difficulties would be a great learning experience, so we chose to move forward with this dataset.

4.2 Data understanding

The CIFAR-100 dataset, which contains numerous images, has been imported. There are only 600 images in each of the 100 classes in the dataset—500 for training and 100 for testing. The image quality of this dataset is the most intriguing aspect. Since each of the images in the dataset has 32 x 32 pixels, it is hard for computers to recognize them. Therefore, a transfer learning approach has been utilized to train the machine to correctly recognize and classify the images more effectively than in the past. Memory, on the other hand, is the primary drawback when developing a deep neural network for CIFAR-100 with millions of parameters.





We have used a Balanced dataset. In this project, we have used the aligned and cropped object images available to train the models. We have also used image augmentation to expand the training dataset using techniques like Image Shifting, Image Flipping, Image Zooming and Image Tilting /Perspective.Therefore, this dataset is intended to represent a more challenging classification task for our Transfer learning.

4.3 Data preparation

The University of Toronto Computer Science website has been used to download the Python version of the dataset used in this project. The accessible files were Python pickled objects created utilizing Pickle. These Python objects have been serialized or deserialized with the help of the Pickle module. These files were read and their structure and images were analyzed using Pickle's load() method.

After loading the dataset we have added a function to extract data from the dataset which results in creation of a dataframe with features such as fine label (class) and a coarse label (superclass).
After doing this preprocessing we have done further cleaning of data and performed EDA.

Before providing our data to standard CNN model and EfficientNetB0, we have used a helper object which worked as a data generator for our CIFAR-100 dataset. This will generate batches of data which will be fed to the multiple object classification model with both the images and their labels. We have used various supervised classifiers and divided the data into test and train. To make data fit for the model we did data normalization by dividing the data by 255.

4.4 Modeling

One of the best ways to use image classifiers in Deep learning is using Convolutional

Neural Networks (CNN). CNN is a class of Artificial Neural Network used in image analysis processes. It can also be referred to as a regularized version of multilayer perceptrons. The Keras library in Python helps to build the model.

We built the model using EfficientNet-B0 with ImageNet weights and our own pooling and dense layers. In order to control the overfitting issue, a global average pooling layer has been introduced to the model to lower the spatial size of the representation (downsampling). This has led to a decrease in the number of parameters and computation costs.
To ensure that the final activations add up to 1 and so satisfy the probability density restrictions, the Softmax activation function has been utilized in the output layer.
Later, when analyzing the model prediction for a few fresh random photos, these probabilities proved helpful.

4.4.1 Model construction

A. Full Connected CNN Model

In the first part, we have built a Fully Connected Convolutional neural networks architecture and Keras-high level API of tensorflow framework in multiple layers.

- The output contains 100 values which shows 20 categories.
- Number of filters tends to increase with depth of the model when more representational capacity is required in the model.
- Size of filters is mostly evenly distributed.

Full Connected CNN Model Summary:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 128)       3584

conv2d_1 (Conv2D)            (None, 32, 32, 128)       147584

max_pooling2d (MaxPooling2D  (None, 16, 16, 128)      0
       )

dropout (Dropout)            (None, 16, 16, 128)       0

conv2d_2 (Conv2D)            (None, 16, 16, 256)       295168
```

```
conv2d_3 (Conv2D)            (None, 16, 16, 256)       590080

max_pooling2d_1 (MaxPooling  (None, 8, 8, 256)         0
       2D)

dropout_1 (Dropout)          (None, 8, 8, 256)         0

conv2d_4 (Conv2D)            (None, 8, 8, 512)         1180160

conv2d_5 (Conv2D)            (None, 8, 8, 512)         2359808

max_pooling2d_2 (MaxPooling  (None, 4, 4, 512)         0
       2D)

dropout_2 (Dropout)          (None, 4, 4, 512)         0

flatten (Flatten)            (None, 8192)              0

dense (Dense)                (None, 1000)              8193000

dropout_3 (Dropout)          (None, 1000)              0

dense_1 (Dense)              (None, 1000)              1001000

dropout_4 (Dropout)          (None, 1000)              0

dense_2 (Dense)              (None, 100)               100100

=================================================================
Total params: 13,870,484
Trainable params: 13,870,484
Non-trainable params: 0
_________________________________________________________________
```

- Here , we are using 3 stacks of layer combinations - Conv2D , Conv2D, MaxPool2D and Dropout layers.

- Each stack has two Conv2D layers with the same padding , ReLu activation, followed by MaxPool2D with pool size of 2, strides of 2. Dropout layer is the layer of the stack having dropout value of 0.2 which gradually increases to 0.5 in next 2 stacks. Dropout layer prevents the network from overfitting and helps the network generalize better.

- In conjunction with training using model.fit() to save a model or weights (in a checkpoint file) at some interval ModelCheckpoint callback is used, this way the model or weights can be loaded later and continue the training from the state saved.

- The output layer is the ‘Dense’ layer. Before connecting to the Dense layer, we have a ‘Flatten’ layer, that serves as a connection between the convolution and dense layers.

- The last layer is the softmax activation.So that the output can be interpreted as probabilities, Softmax makes the output sum equal to 1. After that, the model will base its prediction on the option with the highest probability.

B. EfficientNet Model Summary

In the second part, we are using EfficientNet models designed to perform image classification. To improve the model performance, the standard CNN model is systematically studied for scaling and identifies that carefully balancing network depth, width, and resolution.

Here before providing the train dataset to the model, we use stratified shuffle split to preserve the percentage of samples in each of the 100 classes.

The model has also been kept from overfitting using the dropout technique. By randomly sampling the outputs of the layers involved in dropout, overfitting was avoided, which may have happened as a result of using millions of parameters to train a deep neural network. With over 4.2 million parameters, the model created for this project had a high likelihood of overfitting, which was prevented using dropout.

Since it aided in a faster convergence of the model and was therefore more computationally efficient than other optimization methods, the Adam optimization algorithm has been utilized in the model. It also needed less RAM, ran well with my model, and produced good results with very little hyperparameter modification. Categorical cross entropy loss was utilized because the dataset required multiple class classification. The classification model's accuracy score was calculated using accuracy as a performance parameter.

The model has employed early stopping and reduced learning rate on plateau strategies to track validation loss. Early stopping allows the training to be halted as soon as the model on the validation dataset stops improving. This has made it easier to avoid choosing either too few or too many epochs, which would result in underfitting. Reducing learning rate when training reaches a plateau aids in modifying learning rate as soon as the plateau is noticed. An argument for patience has been added to add a pause in monitoring to look for any signs of additional

improvement because the initial indication of no further progress may not be the ideal time to stop.

The model was trained using an 8-batch size across 15 epochs. Batch size, a hyperparameter, has been carefully adjusted so that the number of samples processed before updating the model parameters is not a significant quantity. It assisted in giving the model a regularization effect and even reduced the generalization error. In comparison to training the network using the whole training set or a smaller batch size, utilizing this batch size utilized less RAM overall.
Using the lowest validation loss as the preservation criterion, the best model has been preserved. To visualize the performance, the training and validation loss versus the number of epochs and the training and validation accuracy versus the number of epochs have been shown.

EfficientNet Model Summary

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 efficientnet-b0 (Functional  (None, 7, 7, 1280)       4049564
 )

 global_average_pooling2d (G  (None, 1280)             0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 1280)              0

 dense (Dense)               (None, 100)               128100

=================================================================
Total params: 4,177,664
Trainable params: 4,135,648
Non-trainable params: 42,016
_________________________________________________________________
```

4.4.2 Compiling the model

Once the model is well constructed we compiled our model with parameters - Loss and Accuracy.

```
 optimizer = Adam(lr=0.0001)
 #early stopping to monitor the validation loss and avoid overfitting
 early_stop = EarlyStopping(monitor='val_loss', mode='min', verbose=1,
patience=10, restore_best_weights=True)

 #reducing learning rate on plateau
 rlrop = ReduceLROnPlateau(monitor='val_loss', mode='min', patience= 5,
factor= 0.5, min_lr= 1e-6, verbose=1)

 model.compile(optimizer=optimizer, loss='categorical_crossentropy',
metrics=['accuracy'])
```

The model is then compiled as and when a sufficient number of layers are added.
Keras communicates with TensorFlow for the model's construction following compilation.

4.4.3 Training the model

To overcome the problem of overfitting, we use Early stopping strategy. This is an effective technique to regularize the neural network. On the other hand, we also use the ReduceLROnPlateau technique for reducing the learning rate when a metric stops getting better. Once learning stagnates, models frequently benefit from reducing the learning rate by 2-10. This callback keeps an eye on a number, and the learning rate slows down if there is no change after a certain number of "patience" epochs.

To train our model, we have used the 'fit()' method on our model with the following parameters: training data , steps_per_epoch, callbacks, validation data, validation steps, and the number of epochs.

The number of times the model will cycle through the data is the number of epochs. In this case, we have used epochs=15 here.

```
model_history = model.fit_generator(train_data_generator,
                                    validation_data=valid_data_generator,
                                    callbacks=[early_stop, rlrop],
                                    verbose=1,epochs=epochs)
```

4.5 Deployment

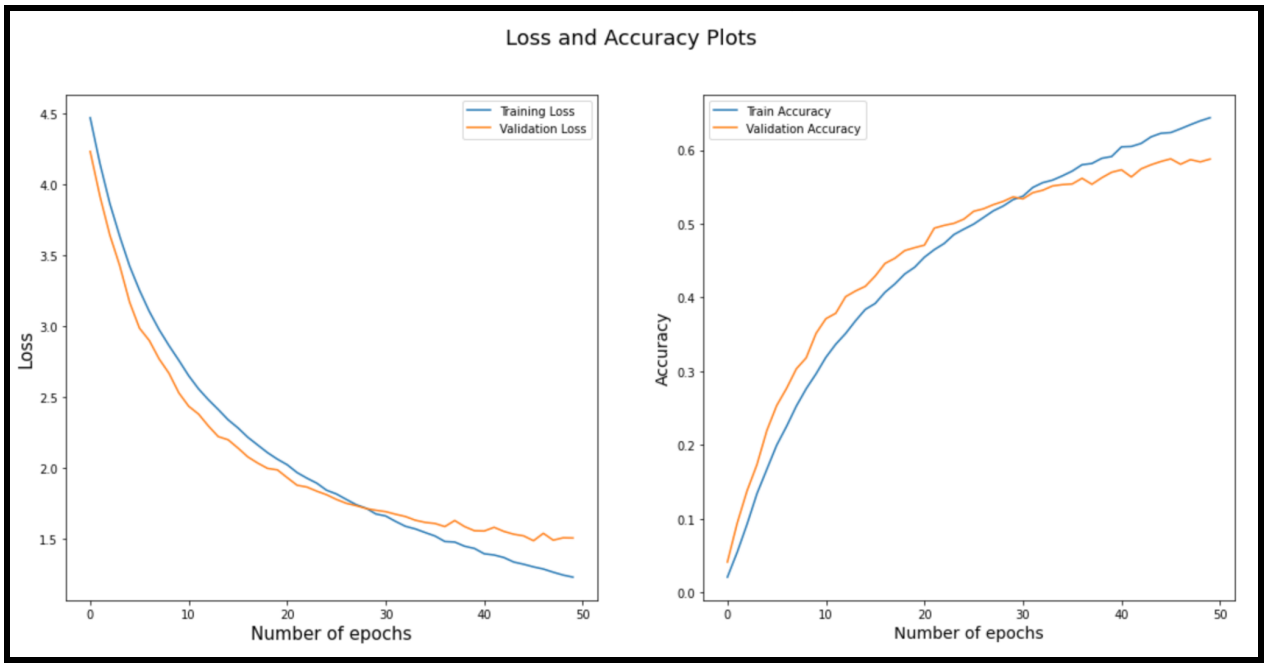
The first step of the deployment process was to create a basic user interface that would help the users to interact with our application and upload the input image frames using the upload button and then receive a classification of the object from our trained model. We have used Streamlit to build this UI for our model.

Next step was to deploy this model as a working web application. We have used PaaS Heroku for deployment of models. The reason we chose Heroku is because it integrates well with Github where we have our code and Streamlit where we have user interface.

The below screenshot shows the build logs for the created app in Heroku.

5 Experiments and Results

For experimentation, the basic CNN model is compiled and run first. The results are captured are shown below:-

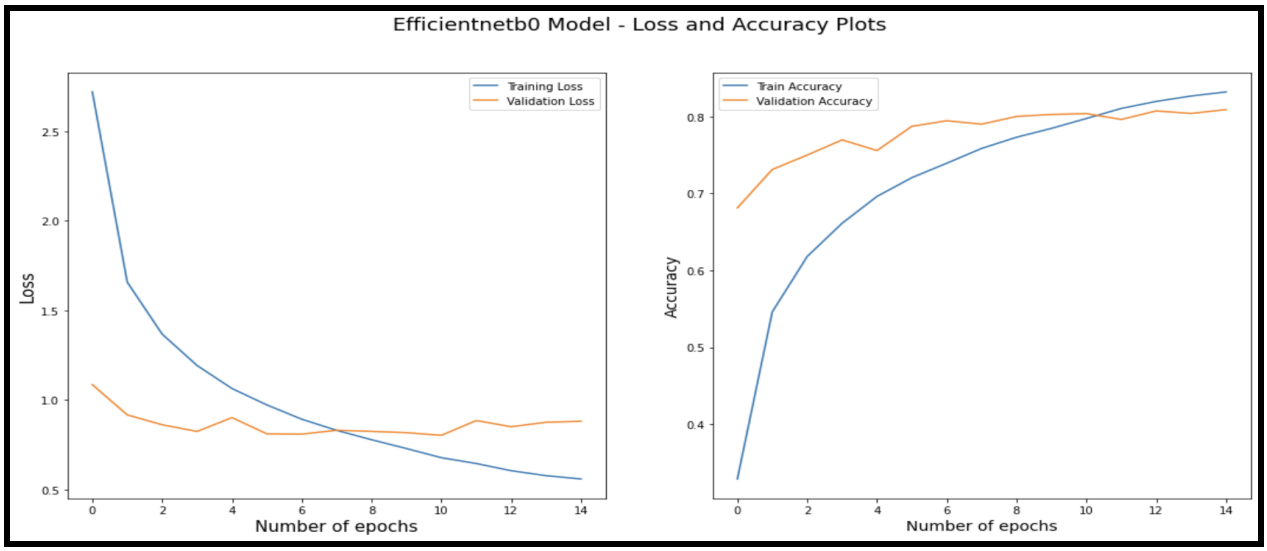


As the number of epochs increases, both training and validation loss gradually decreases. It can be noted that after epoch = 28, validation loss is higher than the training loss. It means that the model is overfitting to the train dataset and failing to generalize to the validation dataset. Similarly until epoch = 28, Validation accuracy is higher than training accuracy which indicates

that the model has generalized fine. The model completes with the validation accuracy of 59.96 % and test accuracy of 59.82 % and notes the validation loss of 1.44 and test loss of 1.43.

The result for true and predicted image after model validation can be seen below:-

For next experimentation, using transfer learning techniques on the EfficientNet model, the results are recorded and captured as shown below.

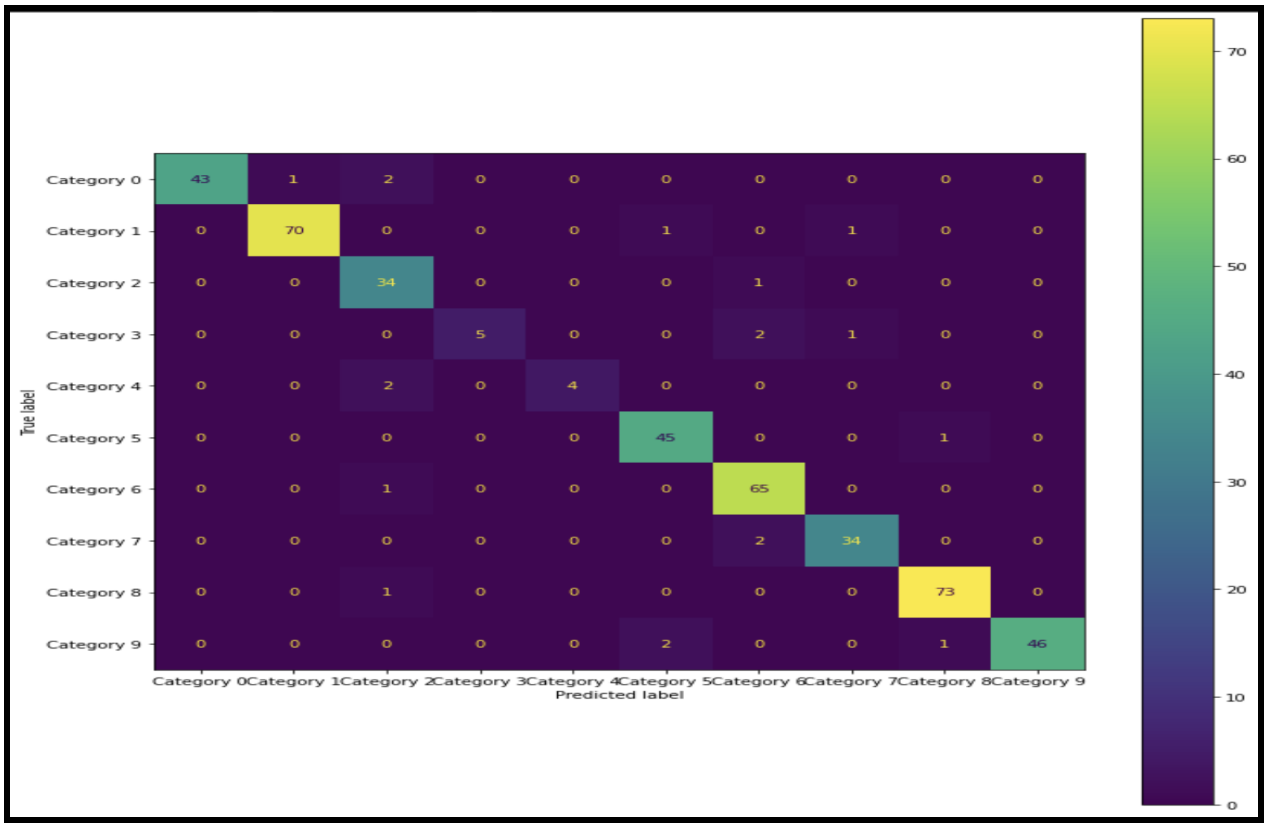


As the number of epochs increases, both training loss gradually decreases. However validation loss oscillates around 0.9 to 1. It can be noted that after epoch = 7, validation loss becomes higher than the training loss. It means that the model is overfitting to the train dataset and failing to generalize to the validation dataset. The model completes with the validation accuracy of

80.89 % and test accuracy of 80.55 % which is better as compared to the basic CNN model built in prior step.

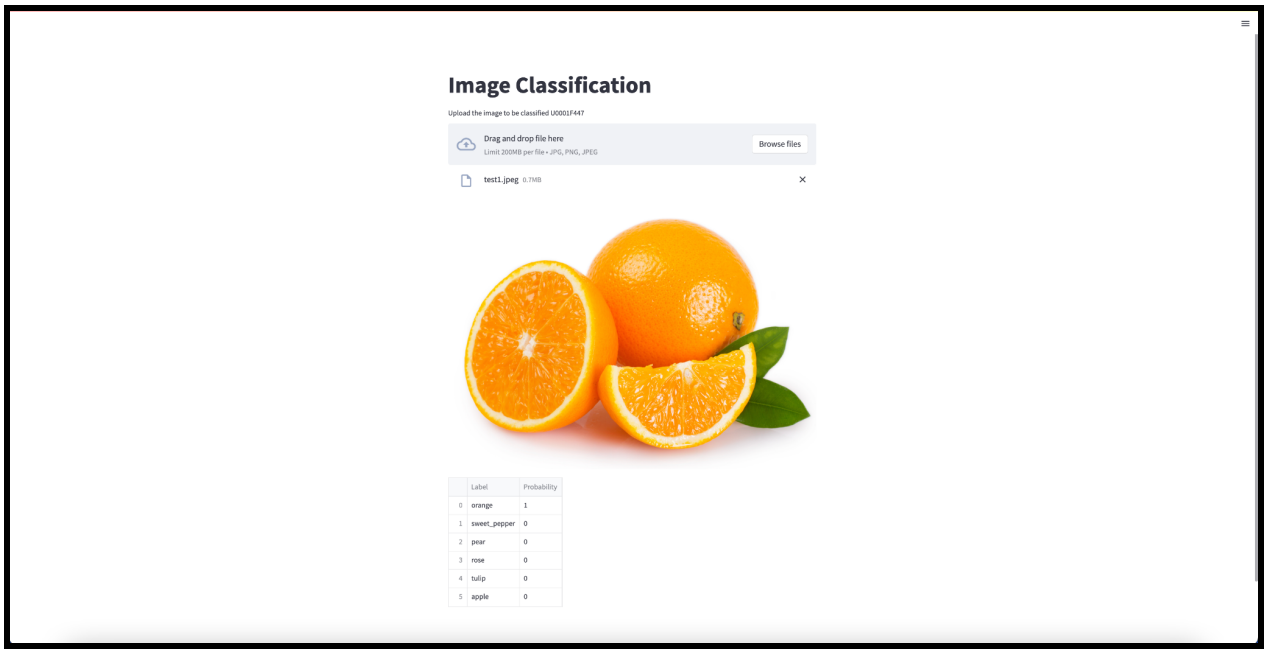
The result for true and predicted image after model validation can be seen below:-

The confusion matrix can be used for evaluating the performance of the new model where it compares the actual target values with those predicted by the machine learning model. The below figure shows the confusion matrix *for first 10 categories* -



Application / Model prediction using Streamlit:

Here we have used an open-source library called Streamlit to enable to build a unser interface for our model using Python.



6 Conclusion

Recognition of different images is a simple task for we humans as it is easy for us to distinguish between different features.We have been able to distinguish between features (images) without putting in a lot of effort because our brains have been subconsciously trained with images of the same kind, which has helped us do so. For instance, after seeing a few cats, we are able to

identify nearly every variety of cat we come across. However, feature extraction requires extensive machine training due to the high cost of computation, memory requirement, and processing power. With an accuracy of 59%, the nine-layer deep neural network model developed for the CIFAR-100 dataset can identify and classify colored images of objects into one of the 100 categories that are available. The model's ConvNet architecture consists of three CONV-RELU stacks, a POOL stack, two fully connected (FC) RELU stacks, and a fully connected output stack. The model has been trained for an hour and a half on an 8vCPU GPU using 13,870,484 trainable parameters. The training process, which involved 100 epochs and 64 batch sizes, was supported by the Adam optimizer with a learning rate of 0.0001 and categorical cross entropy loss. 1.47 is the reported loss. To avoid overfitting, the model used dropout and early stopping. For a few images, the class was incorrectly predicted even after the model was trained with millions of parameters. It's possible that the mediocre accuracy was caused by the small size of the dataset for each class, given that it is thought that a deep learning model performs better when more data is used for training. By working on various aspects of model building and hyper parameter tuning, it is anticipated that the accuracy of this dataset can be further enhanced.

During the design of the model, we learned how important it is to use the right learning rate because if it is too low, training will be slower and may remain stuck with a high training error forever. We also learned how to pre-process the data so that the model can be trained with a variety of images and perform well on test data or any user-supplied input image frame.

One of the potential applications of our project could be in object detection applications like Inventory Management , Bird Classification, Foot Traffic Analysis and many more.

With 82% accuracy, the transfer learning model created for the CIFAR-100 dataset in this project recognizes and classifies colored photos of objects into one of the 100 categories that are accessible. 0.19 is the reported loss. To prevent overfitting, the model employed strategies such as early stopping, reducing learning rate at a plateau, and dropping out.

Even after the model had been trained with millions of parameters, it still incorrectly predicted the class for some photos. A deep learning model's performance is thought to improve with the volume of data utilized in its training. The use of the other EfficientNet versions is thought to considerably increase the accuracy of this dataset.

7 Future Work

There are numerous categories in the CIFAR-100 dataset, which is a sizable dataset used to classify the photos. The model's performance has been boosted using transfer learning. However, the performance can be further enhanced by utilizing the other cutting-edge EfficientNet versions. According to the original paper, employing the EfficientNet-B0 model, the maximum accuracy on the CIFAR-100 was 92%.

The performance of the model can be monitored using optimizers like SGD and Nesterov momentum, which are supposed to shorten training and considerably enhance convergence. We tried utilizing RMSProp in the model, but there was not much of a performance change. However, we believe that less hyperparameter tuning for this optimizer may be to blame. All the hyperparameters that were used to generate this model can be fine-tuned further using other values and combinations. By further adjustment, there is a strong probability that the model performance will increase. The dataset has relatively few photographs in each class, therefore adding more images to it could potentially improve the model's performance

To come up with the optimal augmentation approaches for a dataset, thorough consideration of the dataset context is needed when selecting the data augmentation methodology. This is yet another area where performance can be enhanced.

8 Supplementary Material:

- **References**
  - https://towardsdatascience.com/using-convolutional-neural-network-for-image-classification-5997bfd0ede4
  - https://www.irjet.net/archives/V7/i11/IRJET-V7I11204.pdf
  - https://journalofbigdata.springeropen.com/articles/10.1186/s40537-021-00444-8
  - https://www.analyticsvidhya.com/blog/2020/02/learn-image-classification-cnn-convolutional-neural-networks-3-datasets/
  - https://keras.io/examples/vision/image_classification_efficientnet_fine_tuning/

- Deployment Reference
  - Building a Playground with Streamlit

    https://hackernoon.com/how-to-use-streamlit-and-python-to-build-a-data-science-app
  - Heroku deployment without the app being at the repo root (in a subfolder)

    https://coderwall.com/p/ssxp5q/heroku-deployment-without-the-app-being-at-the-repo-root-in-a-subfolder
  - https://github.com/timanovsky/subdir-heroku-buildpack
  - Heroku how to switch deployment from github to heroku-git with app changes in github

    https://help.heroku.com/CKVOUPSY/how-to-switch-deployment-method-from-github-to-heroku-git-with-all-the-changes-app-code-available-in-a-github-repo