Indian Institute of Technology, Palakkad

Course Code : CS5620

Computer Vision

Submitted to : Dr. Satyajit Das

Multi-label Image Classification

Project Report

IIT PALAKKAD

Team :

Himani Jain [111801016]

Ashish Kumar Gupta [111801052]

Branch : Computer Science and Engineering

B.Tech, Final Year

Table of Contents

Abstract

Multi-Label image classification is a well-known recent and hot problem and is a vast area for research. The project tries to implement various imagenet pretrained models on a dataset, compare and analyse the performance results. The report includes the details of the dataset used as a part of this project, the methodology followed including the details of the models implemented. Towards the end, the results obtained from these models are compared with logical inference.

Introduction

Multi-label image classification is an important task in computer vision with various applications, such as scene recognition, multi-object recognition, human attribute recognition, etc. Compared to single-label image classification, which has been extensively studied, the multi-label problem is more practical and challenging, as real-world images are usually associated with multiple labels, such as objects or attributes. [2]

We aimed to classify the image into multiple labels. The labels will represent the objects in the image. The primary goal of this project was to try out various pre-trained models and compare the results on the chosen dataset.

We trained different models including VGG16, InceptionV3, Resnet152 and NASNetLarge. We did transfer learning for all the models, in which we first used pre-trained layers of the different models mentioned and used their output to train a Feed-Forward Neural Network which does the need. We then did comparison of the results and analysis on the test images.

Dataset

The dataset used for this project is freely available on Kaggle [3]. It is derived from the famous dataset COCO, which is used for multi-object detection. The labels in the dataset are modified to

support multi-label image classification instead of object detection.

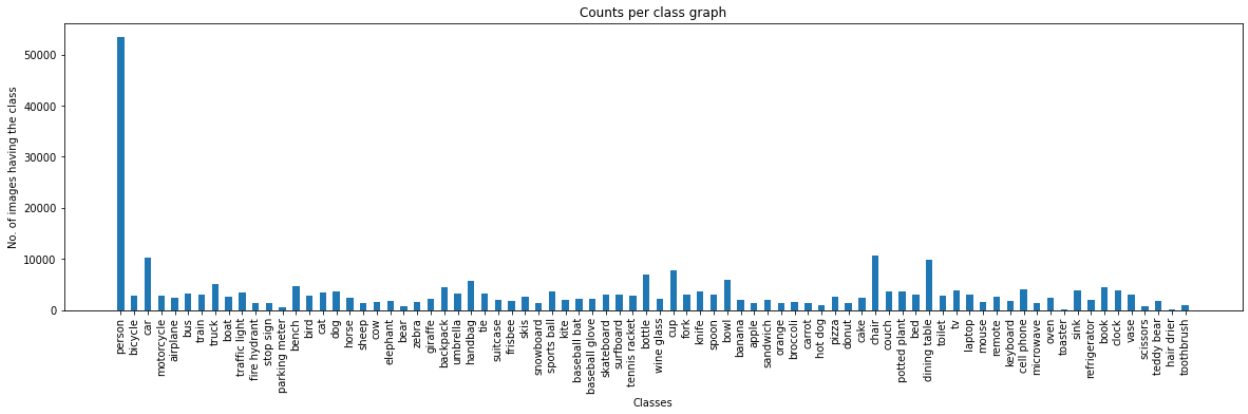


Figure 1. Distribution in original dataset

The original dataset available from Kaggle contains around 90K images with 80 different categories. Due to GPU limitations, we had to curb down the dataset to 5K images. Even in those 5K images, some of the classes were highly skewed, having only a frequency of 50-60 in a set of 5K images. So all the classes which had frequency less than 100, were dropped from the reduced dataset we formed, which removed 29 classes, leaving us with 51 classes to classify in.

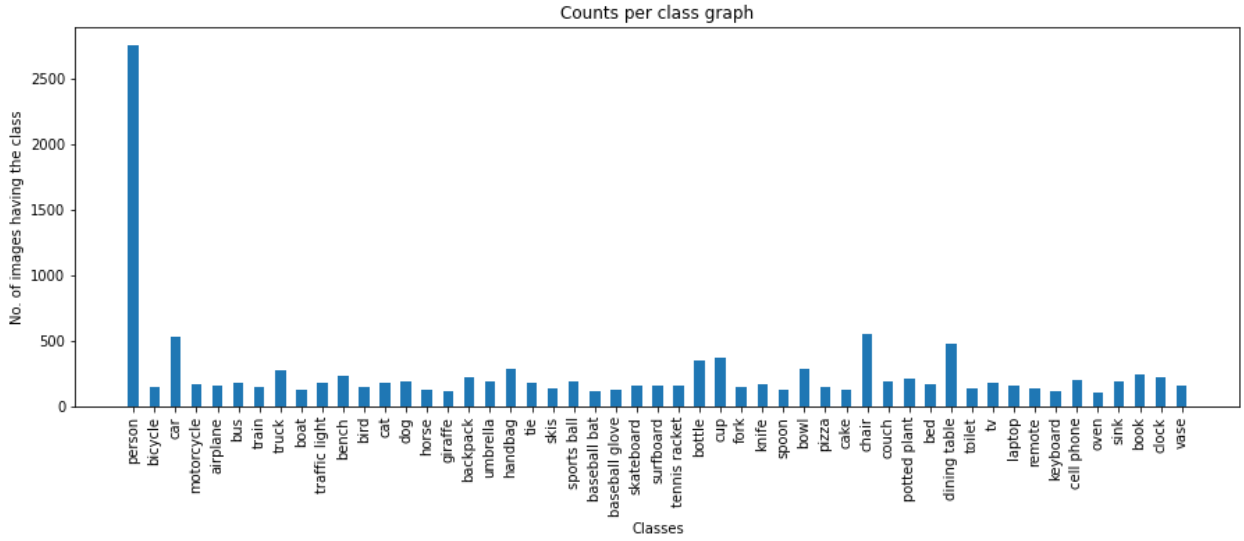


Figure 2. Distribution in the final dataset used

We saved this reduced dataset to train and fine tune different models. The training and test dataset were formed using stratified splitting so that the ratio of classes in test and train are the same. Moreover, before training, the dataset was processed using the *preprocess_input* function available for each of the models available.

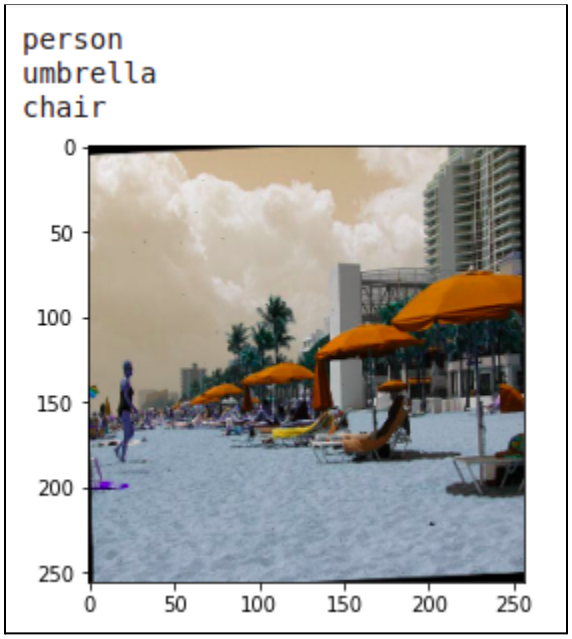


Figure 3. Sample from the training dataset

Methodology

For all the models trained, we followed a similar approach with small tweaks in each training. The approach includes, initializing models with pretrained weights and marking layers trainable and non-trainable, then training the models with specific hyperparameters and comparing the results based on some chosen metrics. This section includes the details about how we handled the skewness in the data while training so that our model doesn't output only one class after training. It includes the reason for the choice of metrics we used to test and check the performance of the

model. Finally, it includes the details of hyper-parameters used for training each of the models and the results obtained.

Handling Skewed data

As can be seen from Figure 2 [Fig 2], the dataset is highly skewed. The class 'person' has a very high frequency while some other classes have very low. So, to not make the model predict 'person' for every image we assigned weights to each of the class. The weighted loss while training is calculated using these weights. The weights are chosen such that the class which has a high frequency will have lower weight assigned to it, so that the accumulation of loss for that class is still comparable to other classes which have lower frequencies. The weights decided are **1/sqrt(frequency of the class)**. However, while training it was still noticed that the interference of 'person' class was still considerable due to very high disproportion of data. As a result we further reduced the weight of the 'person' class by half. After doing so the results looked nice and the models learnt better.

Choice of Metrics

Evaluation metrics play an important role in determining the performance of our model. However, we simply cannot use accuracy as our evaluation metric since in case of multi-label classification, there may be predictions which are neither completely wrong nor right. A prediction containing a subset of the actual classes should be considered better than a prediction that contains none of them, i.e., predicting two of the three labels correctly is better than predicting no labels at all.

Thus, for this reason we choose to use **Area under ROC curve**, **Recall** and **Precision**. If T

denotes the true set of labels for a given sample, and P the predicted set of labels, then the following metrics can be defined on that sample as shown below:

a. Precision = |TΛP| / |P|

b. Recall = |TΛP| / |T|

ROC is a probability curve and AUC represents the degree or measure of separability Figure 4 [Fig 4]. It tells how much the model is capable of distinguishing between classes. However in the case of multilabel classification, in order to measure a multi-class classifier we have to average out the classes somehow. There are two different methods of doing this called micro-averaging and macro-averaging. In micro-averaging all TPs, TNs, FPs and FNs for each class are summed up and then the average is taken. In macro-averaging we just take the average of the precision and recall of the system on different sets. Figure 5 [Fig 5] depicts the difference between micro-averaging and macro-averaging.

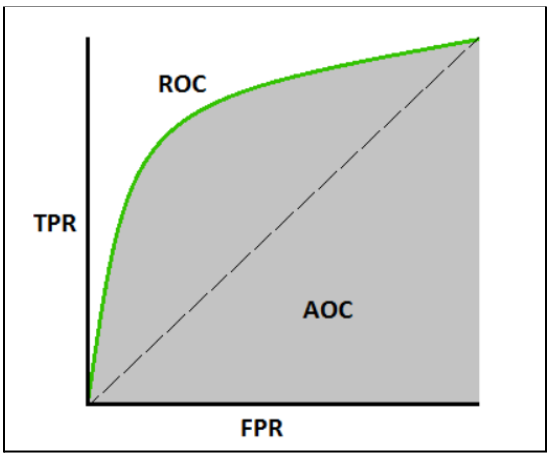


Figure 4. Area under curve of Recall v/s Precision

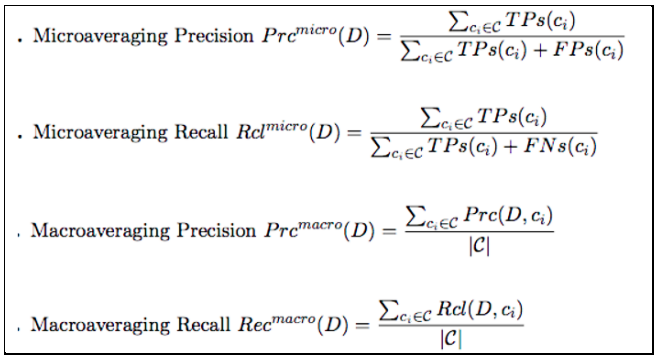


Figure 5. Micro Averaging v/s Macro Averaging

Transfer Learning

Now since we have multiple classes and multiple objects need to be detected in the same image, creating a CNN and training it from scratch is an overwhelming and time taking process. Instead, we will follow the general approach which researchers follow, we used the concept of Transfer Learning. The basic premise of transfer learning is simple: take a model trained on a large dataset and transfer its knowledge to a smaller dataset. For object recognition with a CNN, we freeze the early convolutional layers of the network and only train the last few layers which make a prediction. The idea is the convolutional layers extract general, low-level features that are applicable across images — such as edges, patterns, gradients — and the later layers identify specific features within an image such as eyes or wheels. Thus, we can use a network trained on unrelated categories in a massive dataset (usually Imagenet) and apply it to our own problem because there are universal, low-level features shared between images. [1]

The models which we used in our case are VGG16, Google Net(InceptionV3), ResNet152 and NAS Net Large. The following subsection contains the details of each of the models and the results obtained after training them.

1. VGG 16

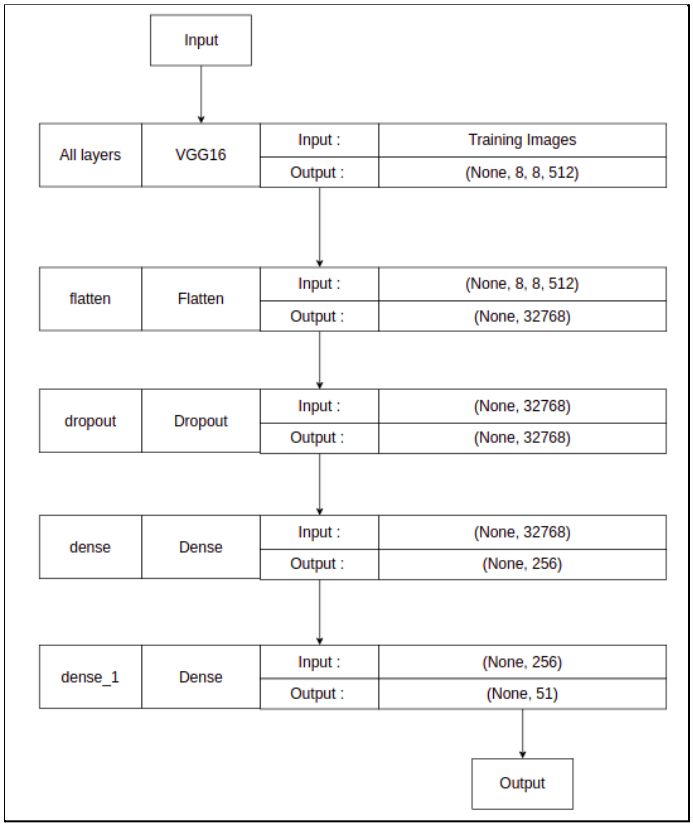


Figure 6. Transfer learning for VGG16 model

The figure [fig 6] shows how transfer learning using VGG16 has been done. We added a flatten layer in front of the head of the VGG16 model. The flatten layer flattens out the output of the last layer of VGG16. That is then passed to a dropout layer. The dropout chosen had a value of 0.6, this value was chosen by doing hyper-parameter tuning by using a validation set. After the dropout layer, a dense layer of 256 units was added and after that the output layer with 51 units (since we have 51 labels to train on). The dense layer 1 has relu as the activation function while the output layer uses sigmoid. We are not using softmax because we are now interested in the probability of surety that this

particular label exists in the image. So, sigmoid provides that flexibility as it will tell the probability of occurrence for each label contrary to softmax which provides overall probability.

The optimizer used for training is Adam, with initial learning rate as 0.001. The loss used to train is 'binary-cross entropy' which is widely used for training multi-label image classification models. The batch size was chosen as 128 and the model was trained for 40 epochs without early stopping callback. The class-weights and the metrics described earlier were used to train and monitor the performance of the model.

At 40th epoch, the verbose looks like this:

```
Epoch 40/40
36/36 [==============================] - 52s 1s/step - loss: 0.0011 - auc: 0.9668 - precision: 0.8977 - recall: 0.8965
```

Figure 7. Training result of last epoch for VGG16

The plots of metrics for the model is:

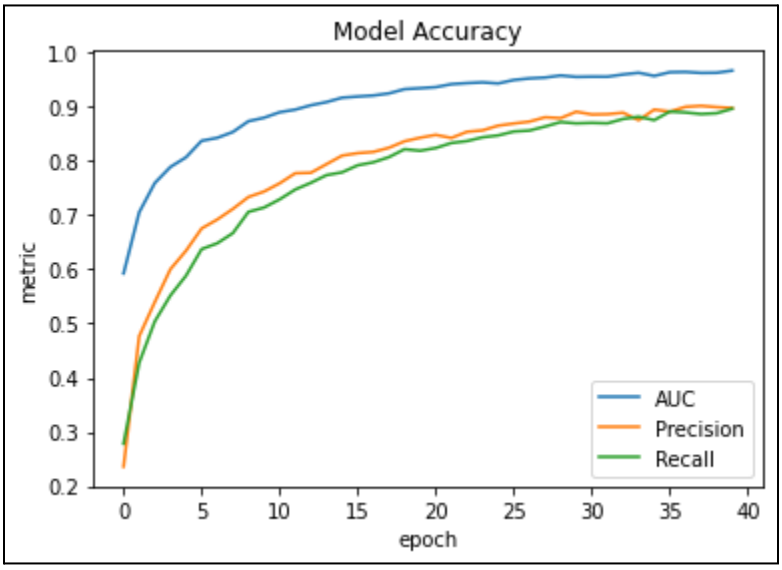


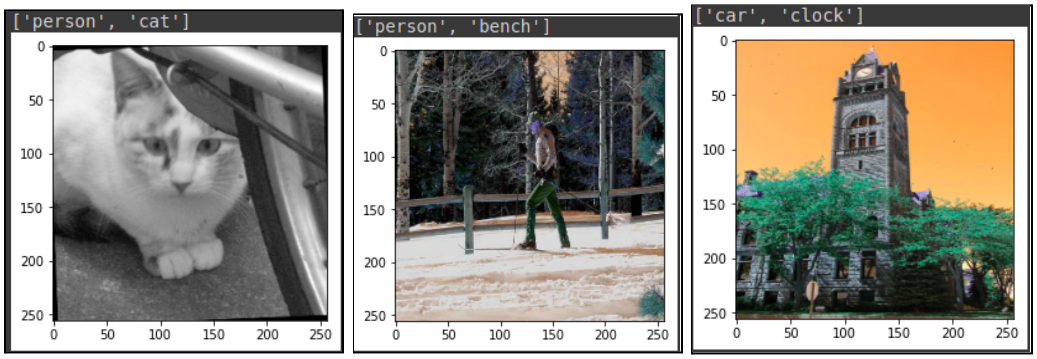
Figure 8. Training metrics for Inception VGG16

The metrics value on the test data obtained was:

- AUC - 0.6636
- Precision - 0.6416
- Recall - 0.4068

Good results on the test data:

Some bad results on test data:



In the first image, the cat is correctly identified, but there is no person in the picture. In the second picture, a person is there, but the bench is not there. In the third picture, the car is not there, the clock is there.

2. *Inception V3*

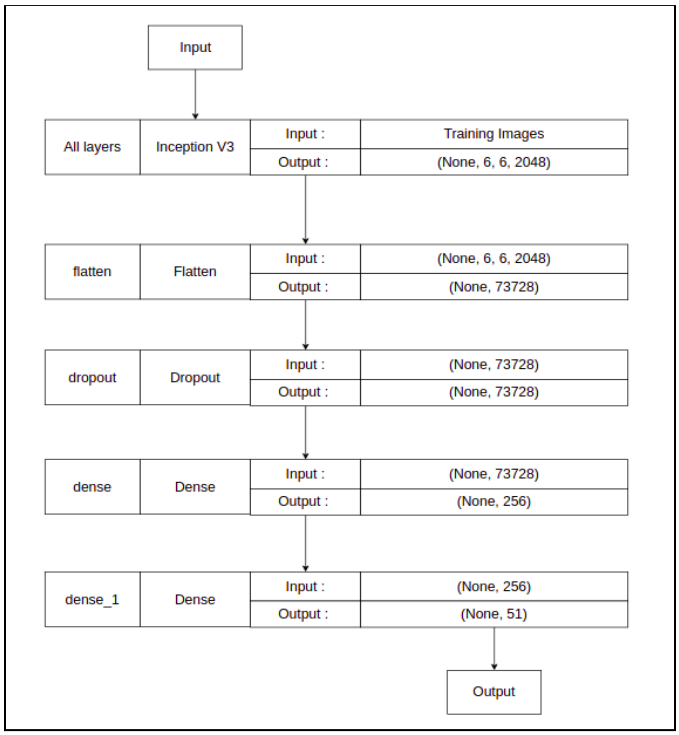


Figure 9. Transfer learning for Inception V3 model

The figure [fig 9] shows how transfer learning using InceptionV3 has been done. We added a flatten layer in front of the head of the InceptionV3 model. The flatten layer flattens out the output of the last layer of InceptionV3. That is then passed to a dropout layer. The dropout chosen had a value of 0.7, this value was chosen by doing hyper-parameter tuning by using a validation set. After the dropout layer, a dense layer of 256 units was added and after that the output layer with 51 units (since we have 51 labels

to train on). The dense layer 1 has relu as the activation function while the output layer uses sigmoid. We are not using softmax because we are now interested in the probability of surety that this particular label exists in the image. So, sigmoid provides that flexibility as it will tell the probability of occurrence for each label contrary to softmax which provides overall probability.

The optimizer used for training is Adam, with initial learning rate as 0.001. The loss used to train is 'binary-cross entropy' which is widely used for training multi-label image classification models. The batch size was chosen as 64 and the model was trained for 40 epochs without early stopping callback. The class-weights and the metrics described earlier were used to train and monitor the performance of the model.

At 40th epoch, the verbose looks like this:

```
Epoch 40/40
71/71 [==============================] - 29s 408ms/step - loss: 4.6143e-04 - auc: 0.9890 - precision: 0.9185 - recall: 0.8939
```

Figure 10. Training result of last epoch for Inception V3

The plots of metrics for the model is:

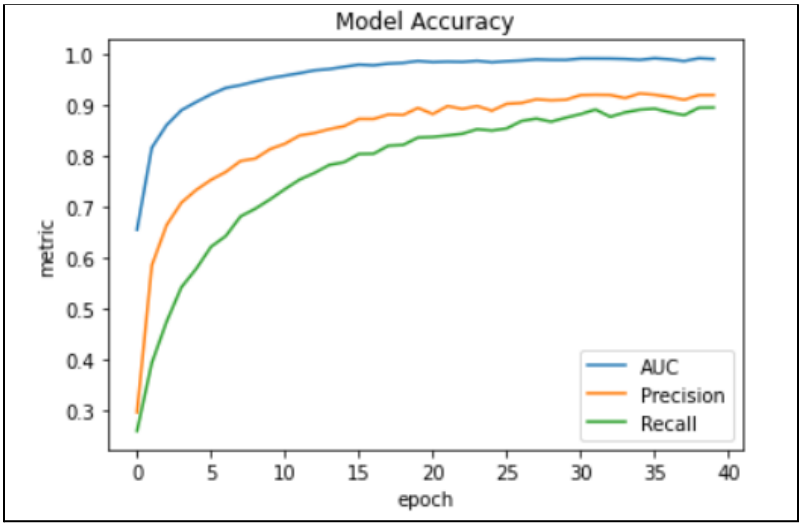
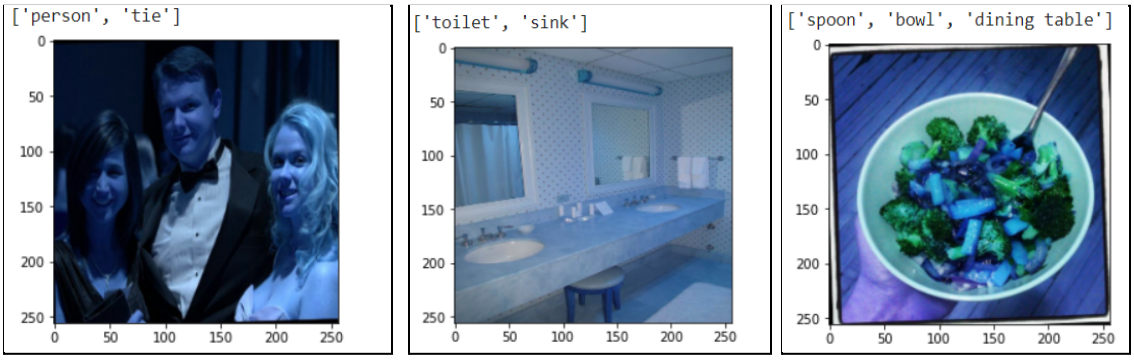


Figure 11. Training metrics for Inception V3 model

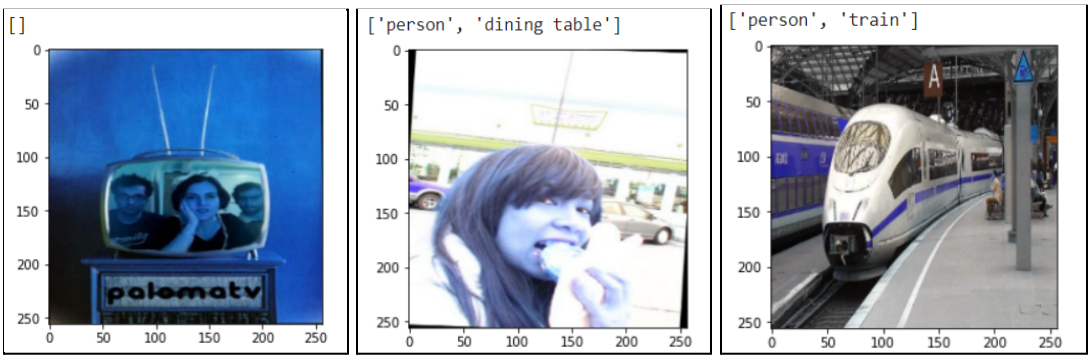
The metrics value on the test data obtained was:

- AUC - 0.7713
- Precision - 0.6457
- Recall - 0.4797

Good results on the test data:



Some bad results on test data:



As can be seen from the results, in the first image the model is not able to detect the person and the television. We have kept the confidence to be at least greater than 0.5, to select the label. However in this case the model is not able to identify it with high confidence because such kinds of images where a person is on the television must have been less in the training data. Similarly, in the second image the model is correctly

identifying the person and it is correlating the food in the person's hand and outputting the dining table as the label.

3. *ResNet (152 layers)*

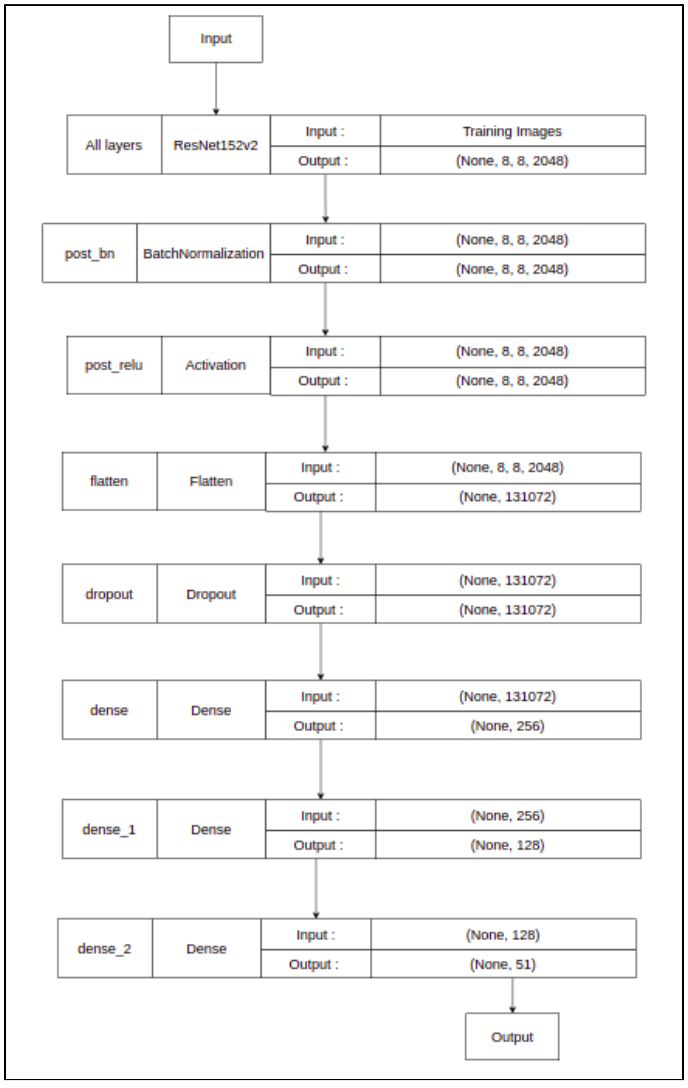


Figure 12. Transfer learning for ResNetV2 152 model

The figure [fig 12] shows how transfer learning using ResNetV2 152 has been done. We

added a flatten layer in front of the head of the ResNet152V2 model. The flatten layer flattens out the output of the last layer of ResNet. That is then passed to a dropout layer. The dropout chosen had a value of 0.6, this value was chosen by doing hyper-parameter tuning by using a validation set. After the dropout layer, two dense layers of 256 and 128 units were added and after that the output layer with 51 units (since we have 51 labels to train on). The dense layers had relu as the activation function while the output layer uses sigmoid. Similar to above mentioned models here too, we are not using softmax because we are now interested in the probability of surety that this particular label exists in the image. So, sigmoid provides that flexibility as it will tell the probability of occurrence for each label contrary to softmax which provides overall probability.

The optimizer used for training is Adam, with initial learning rate as 0.001, same as previous models. The loss used to train is 'binary-cross entropy' which is widely used for training multi-label image classification models. The batch size was chosen as 32 and the model was trained for 40 epochs without early stopping callback. The class-weights and the metrics described earlier were used to train and monitor the performance of the model.

At 40th epoch, the verbose looks like this:

```
Epoch 40/40
141/141 [==============================] - 87s 620ms/step - loss: 5.1813e-04 - auc: 0.9879 - precision: 0.9288 - recall: 0.9126
```

Figure 13. Training result of last epoch for ResNetV2 152

The plot of metric for this model is:

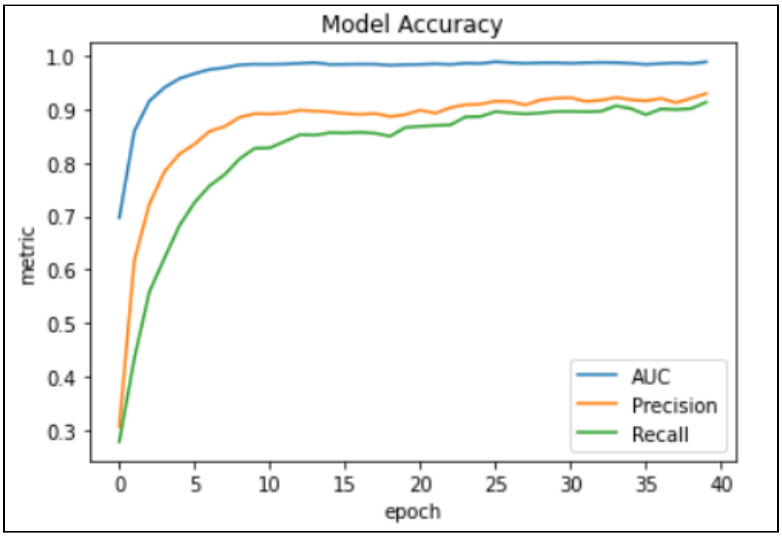
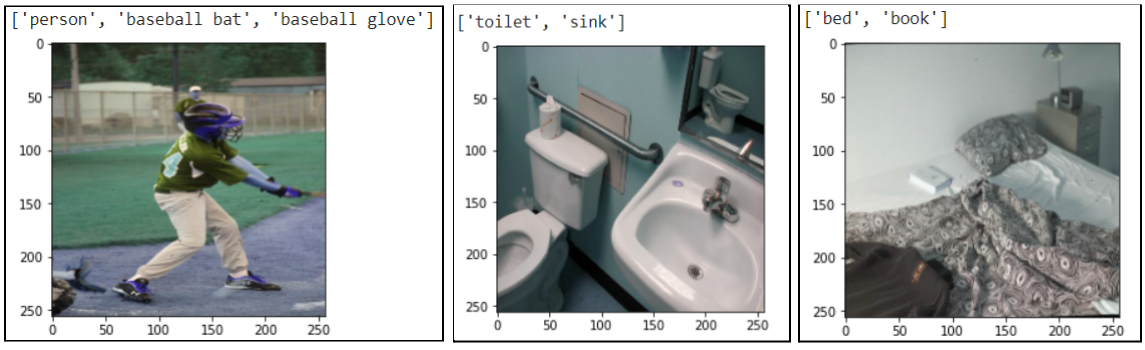


Figure 14. Training metrics for ResNetV2 152 model

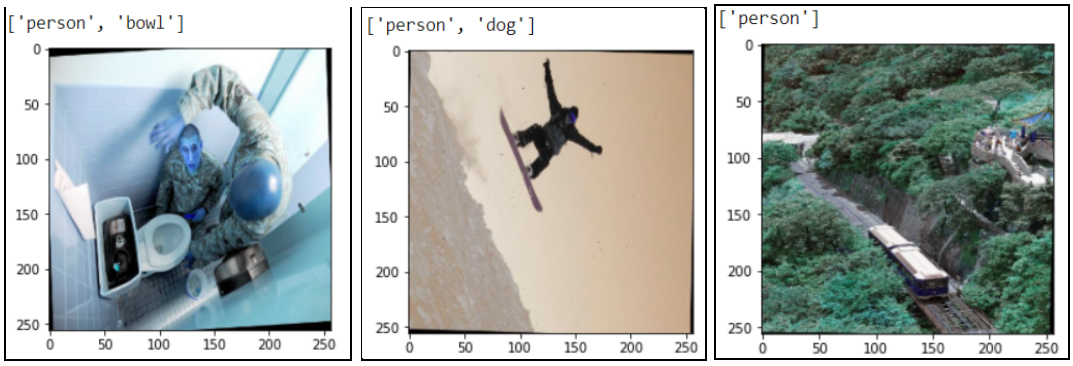
The metrics value on the test data obtained was:

- AUC - 0.7630
- Precision - 0.6292
- Recall - 0.4988

Good results on the test data:



Some bad results on test data:



In the first image, the toilet is identified as a bowl which is wrong. Similarly in the second image there is no dog but still its identifying a dog. In the third image, there is no person in the picture but a train, however the model is predicting a person. A reason can be since there was a lot of data with the label 'person' in the training set, class-weights were not enough to reduce the effect.

4. *NASNetLarge*

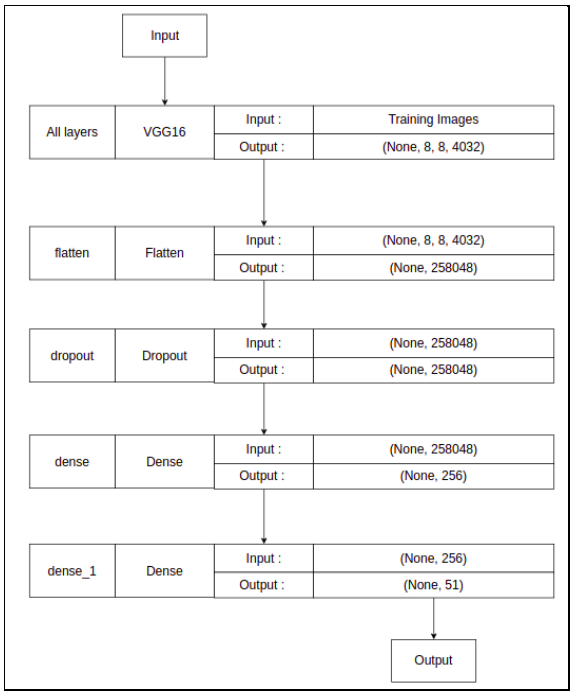


Figure 15. Transfer learning for NASNet model

The figure [fig 15] shows how transfer learning using NASNet has been implemented. We added a flatten layer in front of the head of the NASNet model. The flatten layer flattens out the output of the last layer of the NASNet model. That is then passed to a dropout layer. The dropout chosen had a value of 0.7, this value was chosen by doing hyper-parameter tuning by using a validation set. After the dropout layer, a dense layer of 256 units was added and after that the output layer with 51 units (since we have 51 labels to train on). The dense layer 1 has relu as the activation function while the output layer uses sigmoid. We are not using softmax because we are now interested in the probability of surety that this particular label exists in the image. So, sigmoid provides that flexibility as it will tell the probability of occurrence for each label contrary to softmax which provides overall probability.

The optimizer used for training is Adam, with initial learning rate as 0.001. The loss used to train is 'binary-cross entropy' which is widely used for training multi-label image classification models. The batch size was chosen as 32 and the model was trained for 40 epochs without early stopping callback. The class-weights and the metrics described earlier were used to train and monitor the performance of the model.

At 40th epoch, the verbose looks like this:

Figure 16. Training result of last epoch for NASNet model

The plots of metrics for the model is:

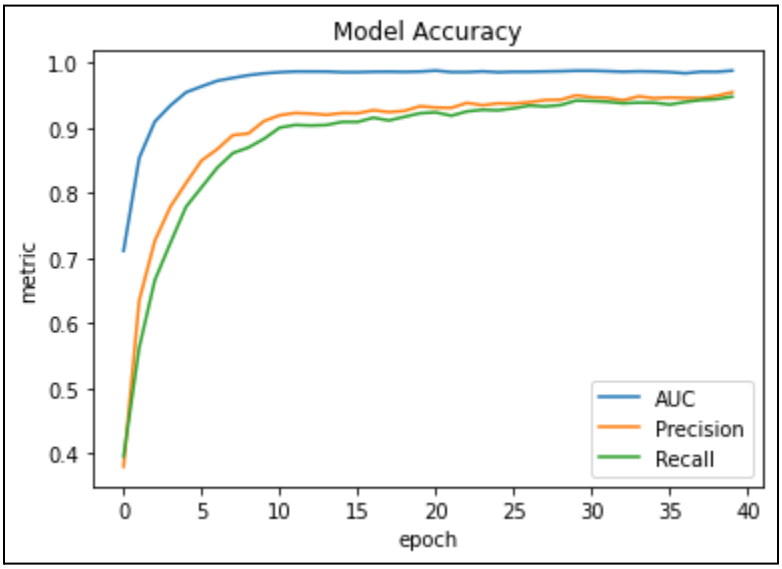
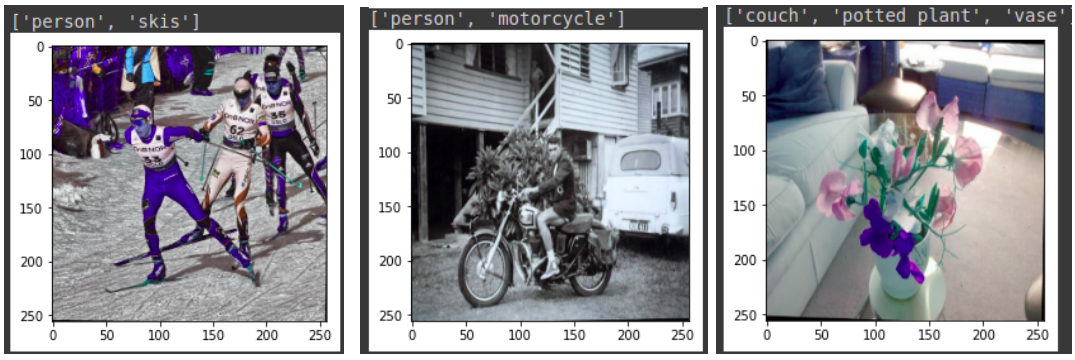


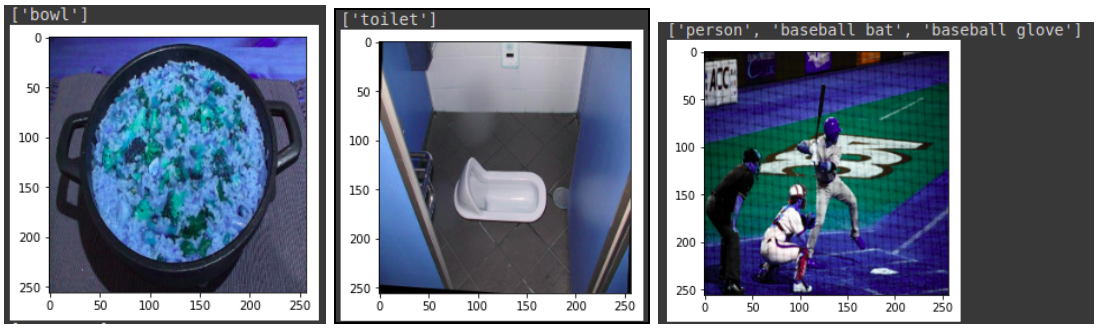
Figure 17. Training metrics for ResNetV2 152 model

The metrics value on the test data obtained was:

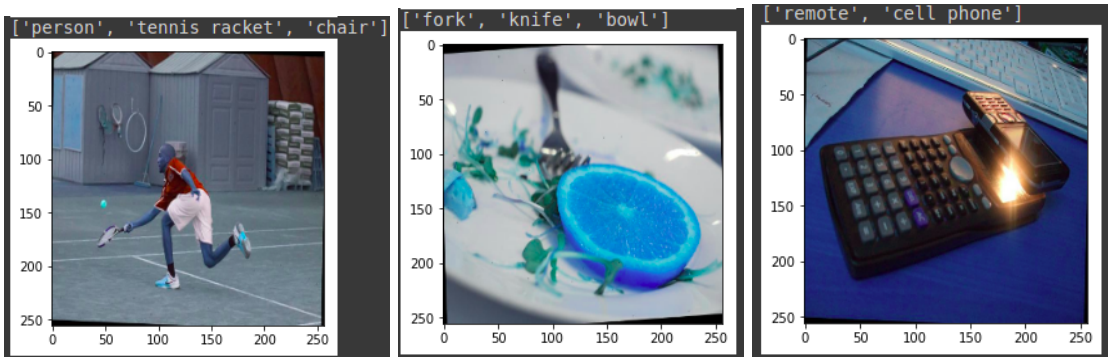
- AUC - 0.7506
- Precision - 0.6631
- Recall - 0.5196

Good results on the test data:





Some bad results on test data:



In the first image, the chair is not there, but a person and tennis racket is there. In the second image, the fork and bowl kinda structure is there, but the knife is not there. In the third image, the cell phone is there, but the remote is not there. Although it looks like a remote.

Comparison of Models

The models were trained with almost the same hyperparameters for 40 epochs with early stopping disabled. The table [table1] shows the difference in training parameters for each of the models.

MODEL	DROPOUT	DENSE LAYERS (units)	BATCH SIZE
VGG16	0.6	256	128
Inception V3	0.7	256	64
ResNet152 V2	0.6	256, 128	32
NAS Net Large	0.7	256	32

Table 1. Training Parameters used for each model

The time taken to train each model grew with the no. of layers and parameters which are there. NASNet Large took the most time followed by ResNet and VGG16. Since the parameters to train were less in InceptionV3, it was the fastest in terms of time to train.

Comparison based on training metrics:

MODEL	AUC	PRECISION	RECALL
VGG16	0.9668	0.8977	0.8965
Inception V3	0.9890	0.9185	0.8939
ResNet152 V2	0.9879	0.9288	0.9126
NAS Net Large	0.9883	0.9552	0.9484

Table 2. Training Results for each model

The training AUCs for InceptionV3, ResNet 152 and NAS Net were almost equal and larger than VGG16's AUC. However the precision and recall of training NAS Net were comparatively larger than others. The precision and recall for ResNet is slightly higher than Inception's while VGG16 has the lowest. It can be inferred that NAS Net is extracting much more features than others and it is true too, since NASNet has the lowest loss among the models when trained on

ImageNet dataset. So, relevant features are best identified using NASNet. The results of Inception and ResNet are almost similar. The basic reason for ResNet being better can be because of much deeper architecture with residual layers attached which were the reason for ResNet's better performance on ImageNet dataset as well. VGG16 as expected is less performant of them all.

Note that the ImageNet dataset was for image classification into single class, here we have multiple classes (multi-label). However, since we have only taken up the head part of those models, we are only concerned with CNN part of them and how well they are able to extract the features from it. The remaining dense layers addition is almost similar in all the models with little tweaks that were done based on hyperparameter tuning when the models were trained with a validation set.

Comparison based on test metrics:

MODEL	AUC	PRECISION	RECALL
VGG16	0.6636	0.6416	0.4068
Inception V3	0.7713	0.6457	0.4797
ResNet152 V2	0.7630	0.6292	0.4988
NAS Net Large	0.7506	0.6631	0.5196

Table 3. Test Results for each model

Here too, in terms of Precision and Recall, NAS Net outperforms all other models. The basic reason for NASNet being better than others is the much better CNN head of the model which can extract the detailed features from the image.

However, we can see that test output is considerably less as compared to training. This is because we trained the model on the subset of the actual dataset and not whole. Also, the dataset we chose to train is highly skewed. Also, all of the models we chose were trained on ImageNet dataset not on COCO. So, detecting multiple features becomes inherently tough for these models.

Challenges Faced

1. Dataset : Original dataset included 90K images. 5K images were taken from that dataset, but that was highly skewed. It was a challenge to deal with them. The details of the solution are included in the section Handling Skewed Data.
2. Resources : We used Google Colab for the project. There is a limit on using the GPU of google colab. Without GPU, the training takes a lot of time.
3. Hyperparameter Tuning : Because of the less availability of resources, to tune the hyperparameters in order to get the best results was a challenge.

Conclusion

The discrepancy in training and test results can be seen. The test results are quite bad when compared to training results. Still the results obtained were quite good based on the less amount of training. These results will improve if we are able to train the CNN head part as well as use the complete dataset for training. However because of the limited amount of RAM and GPU, we

were forced to curb our testing limits. In spite of all this, we were able to draw out a comparison and distinction between the models and at the same time develop multi-label image classification models.

From the above observations, it can be inferred that we can use models trained on ImageNet to do multi-label image classification with little tweaks on how we deal with output of the head of those models. Some other deep learning issues which can be seen are - skewness of data and how it can be reduced to a little extent. Also, from the comparison it is clear that NASNet has a much better CNN part than the others and it is able to extract out more features when compared to other models.

References

1. https://towardsdatascience.com/transfer-learning-with-convolutional-neural-networks-in-pytorch-dd09190245ce
2. https://openaccess.thecvf.com/content_cvpr_2017/papers/Zhu_Learning_Spatial_Regularization_CVPR_2017_paper.pdf
3. https://www.kaggle.com/c/hbku2019/data