MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

DEPARTMENT OF MECHATRONICS

Comparison of Indoor Scene Classification Models

Mini Project Report

Date: 01/11/2021 to 28/11/2021

Submitted by

NAME: ARYAN JAMES PHILIP

REG NO: 180929144

SECTION B

Under the guidance of

Dr. ASHA C.S

DEPARTMENT OF MECHATRONICS

November 2021

Contents

1) Introduction

Humans for many years have yearned to understand the intricacies which evade them, and continuous learning has helped them achieve many problems which we thought were unsolvable. However, humans fall behind in a vital part of learning i.e., repetition. The reasons for this have long been studied and have always concluded that we are slow in repetitive tasks, but this is not the case for computers. Computers excel in repetitive tasks which have helped humans achieve the solution to many problems. This comes in full light in the field of Machine Learning. Machine Learning is the ability of computers to learn and repeat without being explicitly programmed. It is a novel field that has applications in the field of medicine, physics, chemistry, mathematics, and computer science. This project mainly targets the comparison of different supervised learning classification models on image data, and which one performs the best. This project is based on the paper titled **'Indoor Scene Recognition' CVPR '09** which tries to give us an understanding of how indoor scene classification works but has not gone into testing the many models which improve its accuracy.

Machine learning can be broadly classified into 3 categories:

1.1) Unsupervised learning:

This is a type of machine learning in which our data does not have *labels* and the algorithm would try to find meaning or patterns in the data and try clustering them.

1.2) Supervised learning:

This is a type of machine learning in which our data is *labeled*, for example, an email could be labeled as spam or non-spam. The algorithm tries to make a model which would correctly predict whether or not a new email, in this case, is spam or non-spam with the help of the data of previous emails.

1.3) Reinforcement learning:

Reinforcement learning is a type of machine learning in which the algorithm tries to maximize its reward function in a particular situation to find the best possible, say, a path it should take while traveling on a road or any environment.

There are many efficient algorithms today used for different applications and research is going on to find better ones.

I am focusing on the Supervised Classification models of machine learning which helps in matching input data to their respective labels using CNN and finding out which model has the best accuracy and why.

2) Objectives and Weekly Breakup

2.1) Objectives:

This training helped me achieve the following objectives:

a) Understanding different concepts and fundamentals of machine learning.
b) Getting acquainted with the different terminologies of statistics.
c) Gaining and applying mathematical knowledge into a field that has vast applications of it and working on computer vision tools.
d) Understanding the working and use of different machine learning and deep learning algorithms.
e) Implementing various algorithms on a particular dataset and measuring its performance.
f) Learning how to represent data by plotting graphs, histograms, etc. using the matplotlib library in python.
g) Using NumPy arrays in python to make algorithms work fast and efficiently.
h) To get a better understanding of the uses of Ensemble learning techniques on image datasets in Machine Learning.
i) Perform a comparative study on different types of models used.

2.2) Weekly breakup:

My weekly breakup of activities for this mini-project can be summarized in the following manner:

Week 1

a) Learning about CNN and VGG
b) Learning to use OpenCV, matplotlib, seaborn
c) Implementing simple algorithms on the CIFAR-10 dataset
d) Reading papers on VGG as a feature extractor

Week 2

a) Building a CNN model on MNIST dataset
b) Implementing hyperparameter tuning
c) Applying data augmentation techniques and optimizers
d) Paper Reading

Week 3

a) Collecting and cleaning the dataset
b) Learning to implement Principal Component Analysis
c) Implementing Xgboost and Random Forest

Week 4

a) Paper Reading and Implementation
b) Project Report

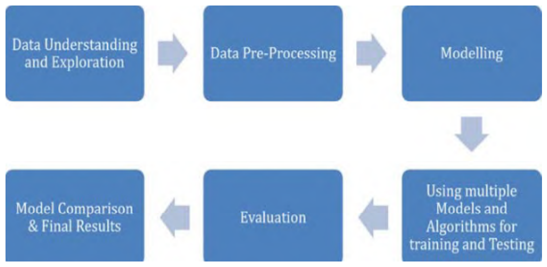

Data Understanding and Exploration
Data Pre-Processing
Modelling
Using multiple Models and Algorithms for training and Testing
Evaluation
Model Comparison & Final Results

Fig: This Figure indicates the steps taken to train an ML mode

3) Dataset

To train the models for image classification we need to collect image data. Any Image classification task requires many corrupt-free images belonging to a particular class. I have taken images from across the web using selenium as well as using some images from the original paper. The dataset originally consisted of 67 classes, with each class having an unbalanced number of images, to prevent issues with overfitting and imbalanced datasets, I ensured that only 11 classes were selected out of which 100 images were used for training and 20 images for testing. Due to hardware constraints, the entire dataset couldn't be trained however they need to be explored further with modern techniques.
Some of the more difficult classes were chosen and tested upon.
The problem with image classification tasks lies in the selection of good training data, for example, two images belonging to 2 different classes in our eyes could look very similar to the model.

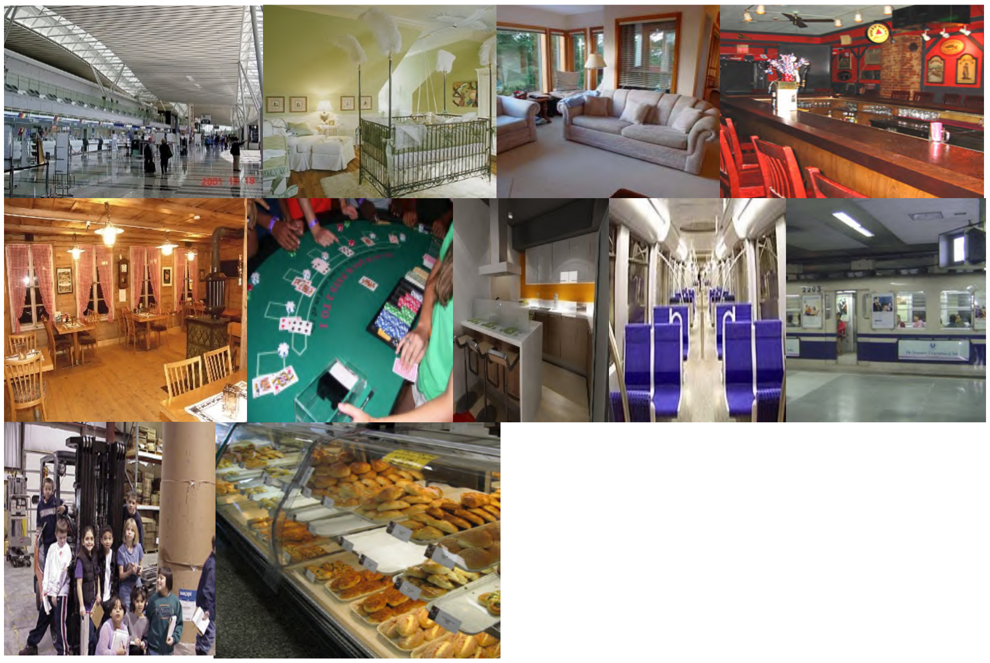
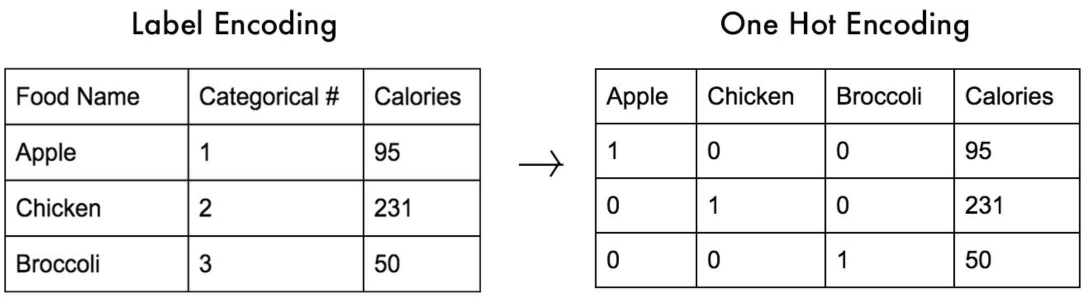


Fig: The above images show the 11 classes chosen for classification. class0-airport class1-bakery, class2-bar, class3-bedroom, class4-casino, class5-inside_subway, class6-kitchen, class7-livingroom, class8-restaurant, class9-subway, class10-warehouse.

The images were separated into different classes using one-hot encoding to label them as integers from their respective categorical integer data types. It is represented by ‘1’ and ‘0’’s indicating true and false. One-hot encoding is a process by which categorical variables are converted into a form that could be provided to ML algorithms to do a better prediction.
The main difference between Label Encoder and One-hot is that one-hot generates Boolean values for each category and only one of these categories can take the value 1 for each sample
Hence, the term one-hot encoding.


Label Encoding

Food Name	Categorical #	Calories
Apple	1	95
Chicken	2	231
Broccoli	3	50

$\rightarrow$

One Hot Encoding

Apple	Chicken	Broccoli	Calories
1	0	0	95
0	1	0	231
0	0	1	50

Fig: This depicts the difference between label encoding and one-hot encoding

The dataset also went through preprocessing techniques to ensure a quality dataset that may not result in a high number of true negatives or false positives. All the images were resized to 128x128 and converted to grayscale. Finally, the images were split into their respective training and testing groups, they were also normalized to ensure that the values remained between 0 and 1.

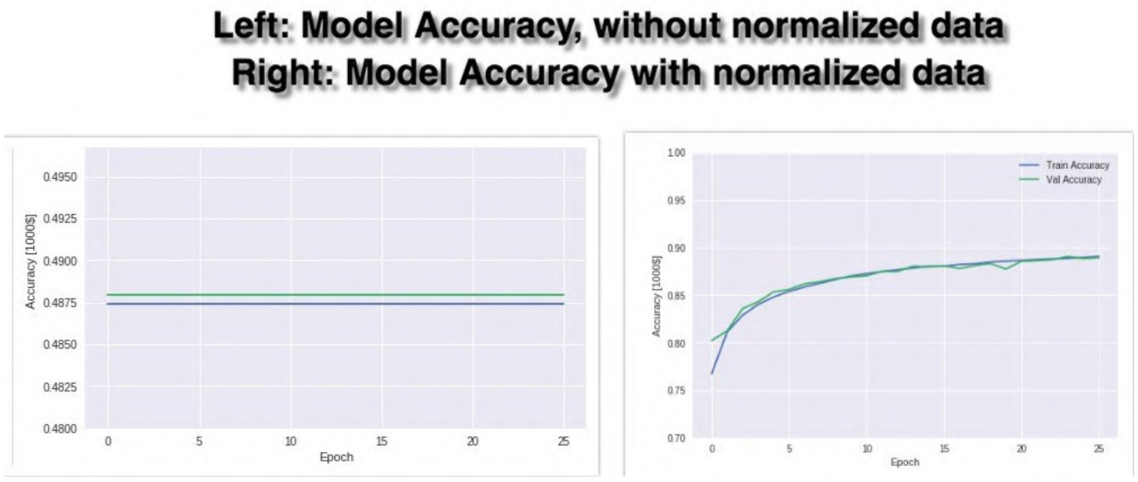


Fig: This figure depicts the difference between normalized data and un-normalized data. A clear indication that the model accuracy is not varying with changes in epochs.

4) Exploring Transfer Learning

Transfer Learning is a novel method of training models using pre-trained weights. Transfer learning generally refers to a process where a model trained on one problem is used in some way on a second related problem.

In deep learning, transfer learning is a technique whereby a neural network model is first trained on a problem like a problem that is being solved. One or more layers from the trained model are then used in a new model trained on the problem of interest.

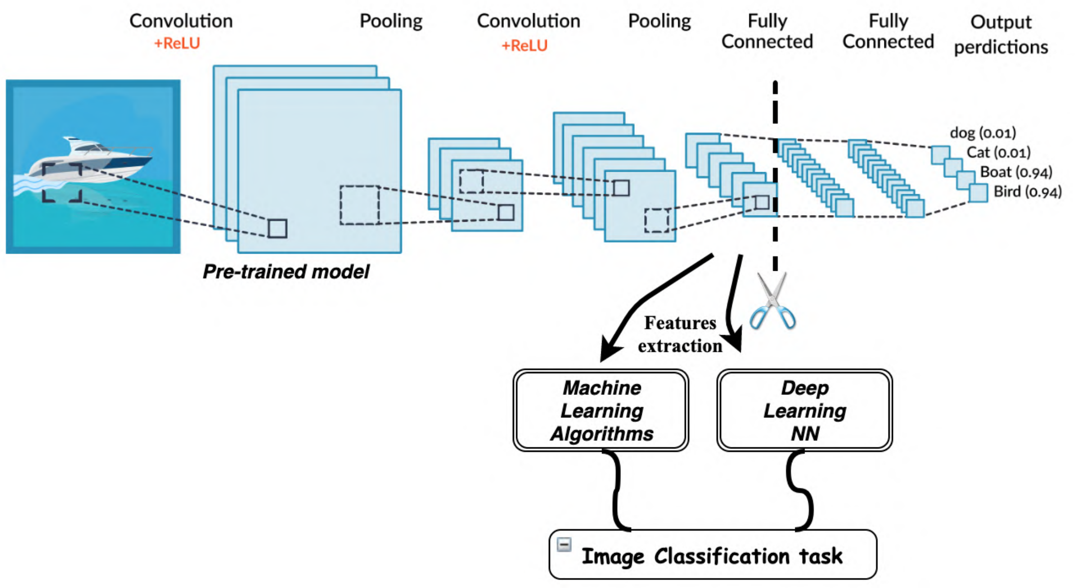


Fig: Image depicts transfer learning for feature extraction.

The question we all want to ask is why transfer learning models such as VGG16 or Resnet work on new input data. To realize this, we must understand the architecture as well as the concept of pre-trained weights.

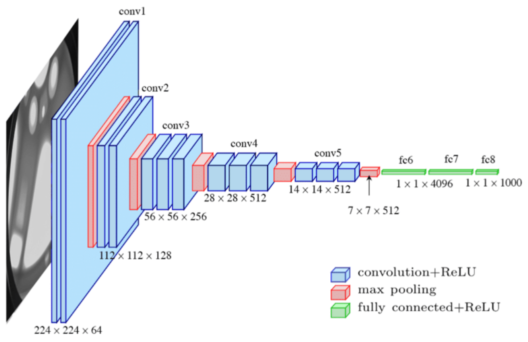


Fig: This image depicts the architecture of VGG-16

Transfer learning often involves taking the pre-trained weights in the first layers which are often general to many datasets and initializing the last layers randomly with and training them for classification purposes. Thus, in the transfer learning approach, learning or backpropagation occurs only at the last layers initialized with random weights. Meanwhile, there are several approaches to transfer learning and the approach we use is dependent on the nature of our new dataset we want to classify with respect to the dataset of the pre-trained models.

I have used a dataset that is smaller compared to the original ImageNet dataset and is completely different from the original, to handle this, I followed the following steps:

- I removed the end of the fully connected neural network and some CNN layers at the end of the network.
- I added a new fully-connected layer that has an output dimension equal to the number of classes (11) in the new data set.
- I randomized the weights of the new fully connected layer.
- I freeze all the weights from the remaining pre-trained CNN network
- I trained the network to update the weights of the new fully connected layer

This may lead to issues of overfitting; however, this can be evaded using novel techniques such as decision trees and gradient boosting.

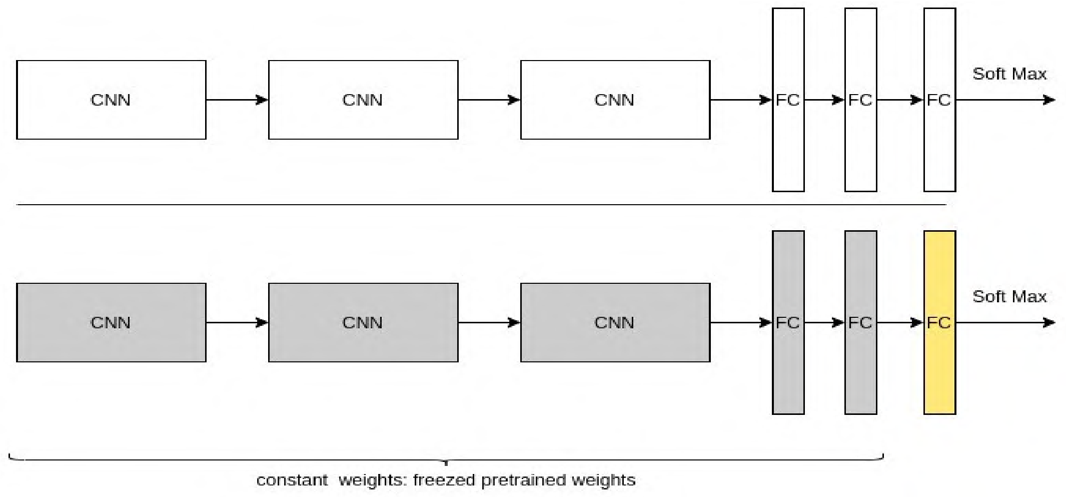
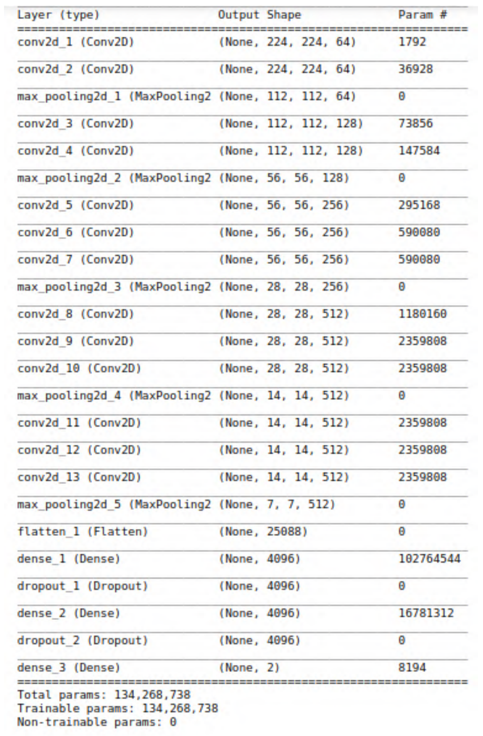


Fig: This is a visualization of how transfer learning works compared to CNN.

In the following sections, we will look into why a pretrained network works better on smaller datasets compared to a custom-built CNN.

5) CNN vs VGG? Which is better?

Now before we decipher why one is better over another, we need to understand that there is no actual difference between both. While we build a CNN using our choice of 2D convolution layers max-pooling layers and batch normalization layers, VGG is a pre-built model that is trained on the Image-net dataset in the ILSVR(ImageNet) competition in 2014.

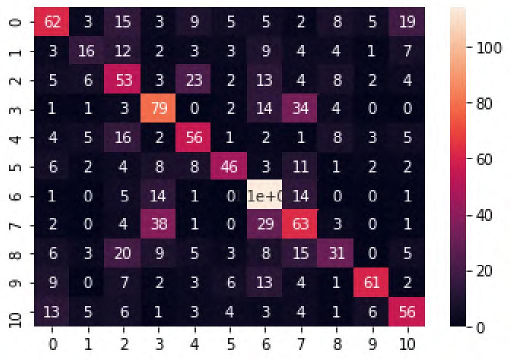


```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 224, 224, 64)      1792
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 224, 224, 64)      36928
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 112, 112, 64)      0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 112, 112, 128)     73856
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 112, 112, 128)     147584
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 56, 56, 128)       0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 56, 56, 256)       295168
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 56, 56, 256)       590080
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 56, 56, 256)       590080
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 28, 28, 256)       0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 28, 28, 512)       1180160
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 28, 28, 512)       2359808
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 28, 28, 512)       2359808
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 14, 14, 512)       0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 14, 14, 512)       2359808
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 14, 14, 512)       2359808
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 14, 14, 512)       2359808
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 7, 7, 512)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 25088)             0
_________________________________________________________________
dense_1 (Dense)              (None, 4096)              102764544
_________________________________________________________________
dropout_1 (Dropout)          (None, 4096)              0
_________________________________________________________________
dense_2 (Dense)              (None, 4096)              16781312
_________________________________________________________________
dropout_2 (Dropout)          (None, 4096)              0
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 8194
=================================================================
Total params: 134,268,738
Trainable params: 134,268,738
Non-trainable params: 0
```

Fig: This depicts the summary of a VGG model

Now in my model I have only used VGG as a feature extractor without the fully connected layers, this was done to apply bagging and boosting techniques as a classifier. To get a sense of how the classification has worked on my dataset we require different performance metrics. The extracted features were fed to an RF classifier or Xgboost/AdaBoost and then classified.

The accuracy without using Fully Connected layers to classify the model was 50%, however, the reason for this could be attributed to the fact that the training dataset was very small, and no image preprocessing was done. The heatmap also shows us some interesting features that some classes were classified much better than the others. The reason for this could be attributed to the images themselves, E.g.: A living room can look very much comparable to a bedroom and vice-versa.



	0	1	2	3	4	5	6	7	8	9	10
0	62	3	15	3	9	5	5	2	8	5	19
1	3	16	12	2	3	3	9	4	4	1	7
2	5	6	53	3	23	2	13	4	8	2	4
3	1	1	3	79	0	2	14	34	4	0	0
4	4	5	16	2	56	1	2	1	8	3	5
5	6	2	4	8	8	46	3	11	1	2	2
6	1	0	5	14	1	0	1e+0	14	0	0	1
7	2	0	4	38	1	0	29	63	3	0	1
8	6	3	20	9	5	3	8	15	31	0	5
9	9	0	7	2	3	6	13	4	1	61	2
10	13	5	6	1	3	4	3	4	1	6	56

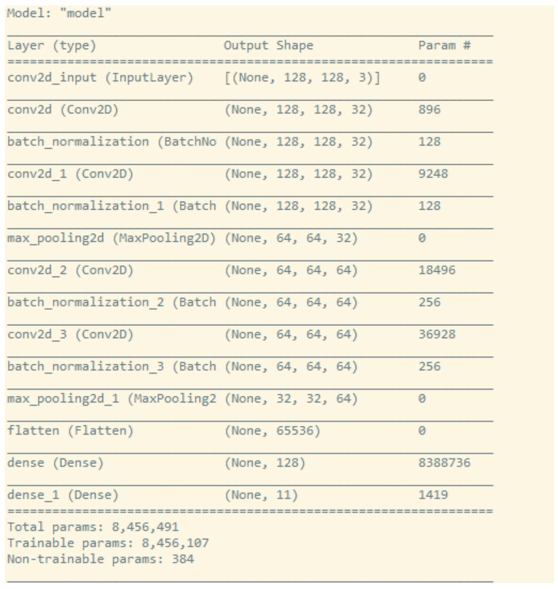
Fig: The above image shows the confusion matrix of the VGG model. class0-airport class1-bakery, class2-bar, class3-bedroom, class4-casino, class5-inside_subway, class6-kitchen, class7-livingroom, class8-restaurant, class9-subway, class10-warehouse.

Interestingly, we can observe that there are major misclassifications between class7 and 6, i.e. kitchen and living room as well as class 8 and 2, class 7 and 3.

To improve the classification accuracy, we use bagging and boosting techniques.

Before we jump into Ensemble techniques, let us look if our custom-built CNN model fared any better.

My **CNN** performed much worse as compared to the pre-trained model VGG 16. This can be attributed to the fact there are several non-trainable parameters in my CNN model along with the fact the VGG model has been trained on large datasets with a lot of hyperparameter tuning and conditioning resulting in faster training times and better accuracy.



```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_input (InputLayer)    [(None, 128, 128, 3)]     0
_________________________________________________________________
conv2d (Conv2D)              (None, 128, 128, 32)      896
_________________________________________________________________
batch_normalization (BatchNo (None, 128, 128, 32)      128
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 128, 128, 32)      9248
_________________________________________________________________
batch_normalization_1 (Batch (None, 128, 128, 32)      128
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 64, 64, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 64, 64, 64)        18496
_________________________________________________________________
batch_normalization_2 (Batch (None, 64, 64, 64)        256
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 64, 64, 64)        36928
_________________________________________________________________
batch_normalization_3 (Batch (None, 64, 64, 64)        256
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 32, 32, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 65536)             0
_________________________________________________________________
dense (Dense)                (None, 128)               8388736
_________________________________________________________________
dense_1 (Dense)              (None, 11)                1419
=================================================================
Total params: 8,456,491
Trainable params: 8,456,107
Non-trainable params: 384
_________________________________________________________________
```

Fig: The image shows my CNN model summary.

The dataset was subjected to one-hot encoding and label encoding before training it for 50 epochs. The validation accuracy came out to be **40%.**

Looking further into the confusion matrices of both models, we will see some slightly better predictions and fewer misclassifications in the VGG model; however, both models do not do any justice to certain complimentary classes which have similar features.

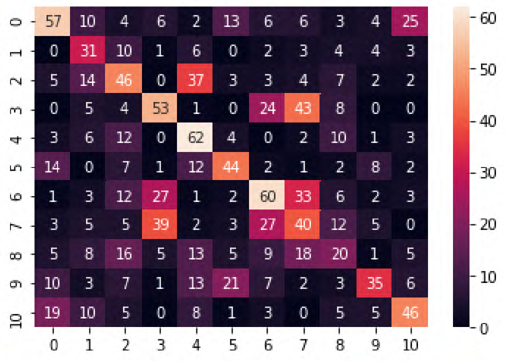


Fig: This shows the confusion matrix of the CNN model without any ensembling. class0-airport class1-bakery, class2-bar, class3-bedroom, class4-casino, class5-inside_subway, class6-kitchen, class7-livingroom, class8-restaurant, class9-subway, class10-warehouse.

My observations with recall, precision scores, and other performance metrics showed a major downgrade and showed much lower values as compared to the VGG model. Also, one important parameter to be remembered is that we are training our model on very small datasets which often in many cases lead to overfitting and performs poorly on the testing set.

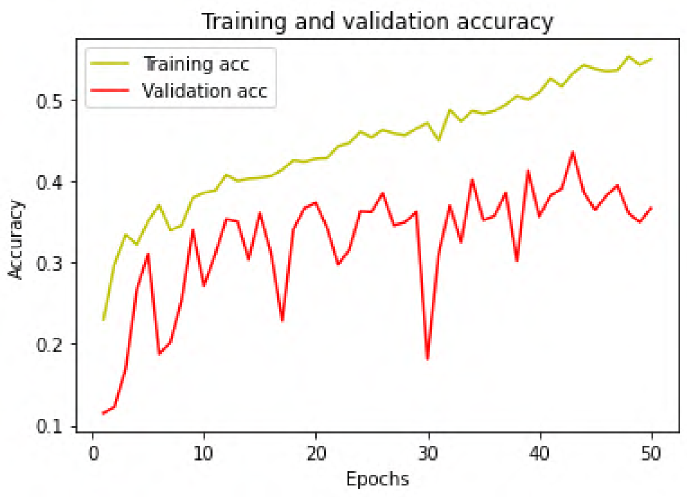


Fig: Image depicts the training and validation accuracy of our model.

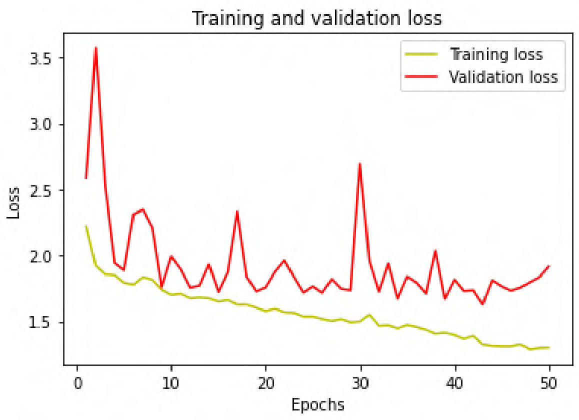
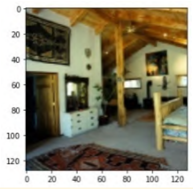
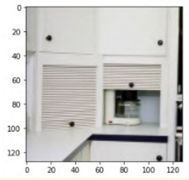


Fig: This image depicts the training and validation loss.

Examples:

```
The prediction for this image is:  ['inside_subway']
The actual label for this image is:  inside_subway
```
Fig: This shows a correct prediction made by the CNN model.


```
The prediction for this image is:  ['livingroom']
The actual label for this image is:  bedroom
```
Fig: This shows an incorrect prediction made by the CNN model


```
The prediction for this image is:  ['kitchen']
The actual label for this image is:  kitchen
```
Fig: This shows a correct prediction for the VGG model

From these examples we can extract some interesting observations:

- The classes such as living room and bedroom are often difficult to classify due to lots of overlapping features.
- The feature extraction process may sometimes lead to unwanted edges/corners leading to misclassification.
- Another important observation is that we realize our model is not overfitting the training data and we are moving in the right direction.

6) Random Forest Classifier. Is it good?

Before we answer the question of whether the RF classifier is good or not? We must understand what Random Forest is.

To break it down further we must understand the concept of Decision Trees.

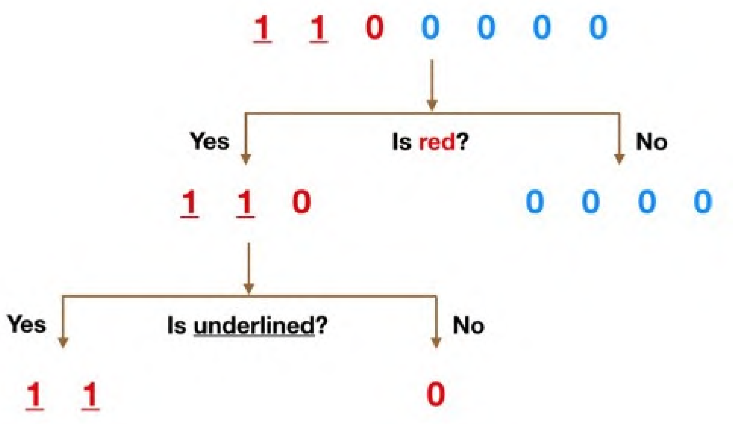


Fig: This shows how a decision tree works on a simple problem.

Decision trees are a foray into the well-known data structure known as trees which has a parent node (root) and many children nodes. The decision trees make use of 'yes' or 'no' to give an output based on very specific classifying parameters. The main challenge comes in the form of what is our classifying parameter. It uses Gini (change) or Entropy to give us an appropriate decision with maximum information gain.

Random forest consists of many individual **decision trees** that operate as an ensemble. Each tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction.

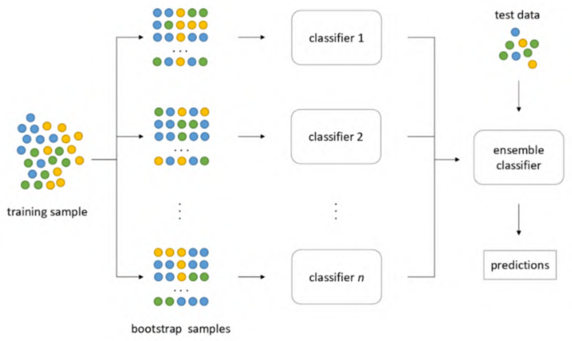
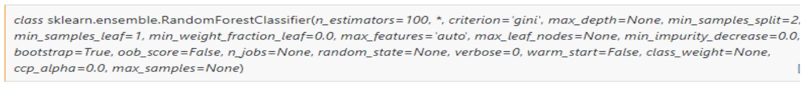


Fig: This image depicts the process of bootstrapping and classification.

Random forest is a combination of different decision trees all making their class predictions and the majority vote is often selected as the correct class.



```
class sklearn.ensemble.RandomForestClassifier(n_estimators=100, *, criterion='gini', max_depth=None, min_samples_split=2,
min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, class_weight=None,
ccp_alpha=0.0, max_samples=None)
```

Fig: This depicts the RF classifier package.

In my model, RF was used as a classifier on the extracted features from the convolutional filters. I had chosen 500 Decision Trees/estimators and a random state of 42 which gives us a highly randomized bootstrapping split.

Usually, the error reduces with the number of estimators; however, at a certain value the error rate saturates and if the number of estimators is increased further the error rate can increase due to smaller sample size for each classifier leading to poor classifications. To tune, we need to use different hyperparameter tunings like GridSearchCV, K-fold Cross-validation, etc.

On my model, I realized that RF does not perform much better than the CNN model the misclassifications are worse. The validation accuracy was only **42%.**

The reasons my model does not perform very well and why RF does not work well with image data can be attributed to the following reasons:

- **Sparsity** - When the data are very sparse, it's very plausible that for some nodes, the bootstrapped sample and the random subset of features will collaborate to produce an invariant feature space. There's no productive split to be had, so it's unlikely that the children of this node will be at all helpful. **XGBoost** can do better in this context.
- **Random forests basically only work on tabular data**, i.e. there is not a strong, qualitatively important relationship among the features in the sense of the data being an image, or the observations being networked together on a graph. These structures are typically not well-approximated by many rectangular partitions. If your data are a series of images the random forest will have a very hard time recognizing that.
- **Random Forest performs poorly on diagonal decision boundaries or even decision boundaries of higher dimensions** because even for simple boundaries e.g. just to distinguish between 2 features requires many iterations to get a perfect fit whereas an SVM model performs much better.

We can observe from the heatmap that our model does not perform very well for some classes as compared to the CNN model with fully connected layers.

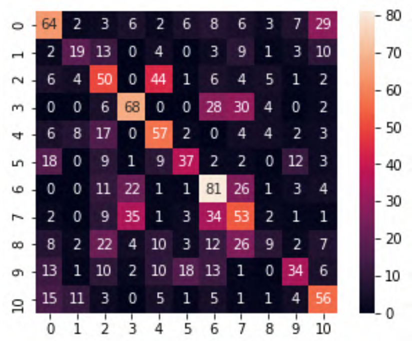


Fig: This depicts the heatmap for our CNN_RF model. class0-airport class1-bakery, class2-bar, class3-bedroom, class4-casino, class5-inside_subway, class6-kitchen, class7-livingroom, class8-restaurant, class9-subway, class10-warehouse.

We can observe that especially class 8 is performing very poorly, and the RF classifier is unable to distinguish between class 8 and other classes very well.

Now to further investigate methods to improve our model we should look at using Principal Component Analysis (PCA) along with any of the transfer learning techniques.

PCA is a method for simplifying a multidimensional dataset to lower dimensions for analysis, visualization, or data compression.

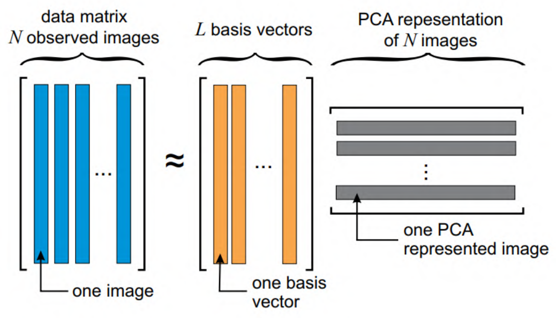


Fig: This image depicts how PCA works on image data by separating it into a basis vector and a principal component.

PCA helps determine which features are important by using Singular Value Decomposition (SVD) and projects the features to lower feature space such that the majority of the information (variance between features) is retained. Then the features are mapped (transform) to both the training and testing set.

The VGG filters are used as feature extractors and are then fed into the PCA model to extract a few features, these features are then sent into the fully connected layers and dense layers to give an output.

The PCA model gave the best validation accuracy after just 20 epochs of **68%** and there were fewer misclassifications were very few. Also, the other performance metrics performed quite admirably with all the scores being an average of **75%**.

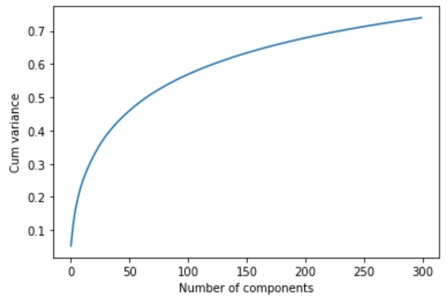


Fig: The image indicates the cumulative variance (information gain) as the number of components increases.

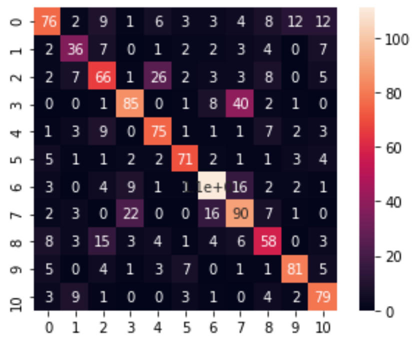


Fig: This image shows the confusion matrix of the VGG PCA model. class0-airport class1-bakery, class2-bar, class3-bedroom, class4-casino, class5-inside_subway, class6-kitchen, class7-livingroom, class8-restaurant, class9-subway, class10-warehouse.

The confusion matrix shows that most of the misclassifications are less than 10 with the only significantly misclassified class being the bedroom and the living room.

However, when the same model was classified using RF (bagging) techniques the model was much more poorly with a validation accuracy of only **49%**, due to the above-explained reasons.

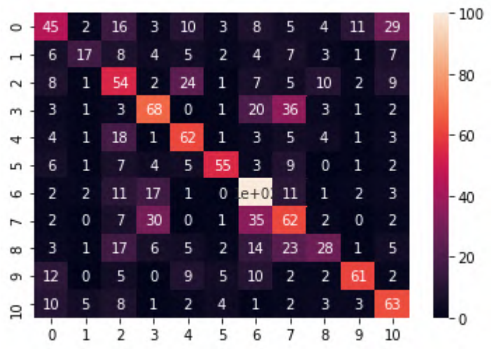


Fig: This image shows the confusion matrix of the VGG PCA RF model. class0-airport class1-bakery, class2-bar, class3-bedroom, class4-casino, class5-inside_subway, class6-kitchen, class7-livingroom, class8-restaurant, class9-subway, class10-warehouse.

So, our question is why PCA models perform much better with fully connected layers rather than ensemble techniques. To understand this we need to look into the concepts of SVD and understand how relevant features are extracted and described.

7) Gradient Boosting. The ultimate weapon

Gradient Boosting is one of the foremost ensemble boosting techniques. Gradient Boosting involves building an ensemble of weak learners.

It follows 2 key insights:

- If we can account for our model's errors, we will be able to improve our model's performance.
  E.g.: Assume we have a regressive model that predicts 3 for a test case with a result of 1. If we know the error (2 in this case), we may fine-tune the forecast by subtracting the error (2 in this case) from the initial prediction (3 in this case) to get a more accurate prediction of 1. This leads to our second insight, "How can we know the error made by our model for each given input?"
- We can train a new predictor to predict the errors made by the original model. We can now increase the accuracy of any predictive model by first training a new predictor to forecast its present errors. Then, a new improved model is created, with the result being a fine-tuned version of the initial prediction. The improved model is now called an ensemble of the two predictors because it requires the outputs of both the original predictor and the error predictor. This is repeated an arbitrary number of times in gradient boosting to continuously enhance the model's accuracy. Gradient boosting is based on this repetitive process.

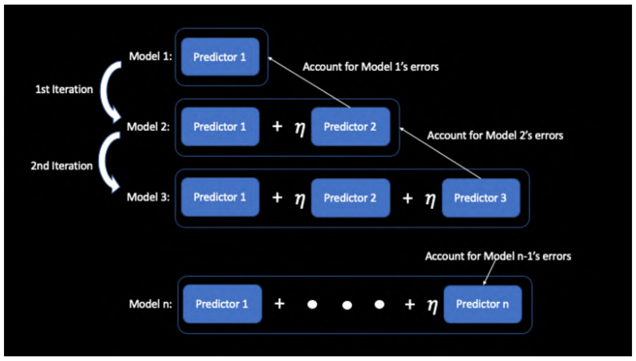

Model 1:
Predictor 1
Account for Model 1's errors
1st Iteration
Model 2:
Predictor 1 + η Predictor 2
Account for Model 2's errors
2nd Iteration
Model 3:
Predictor 1 + η Predictor 2 + η Predictor 3
Account for Model n-1's errors
Model n:
Predictor 1 + • • • + η Predictor n

Fig: This shows how an ensemble of weak learners improves using previous errors (an iterative process).

When training a new error-predicting model to predict a model's current errors, we regularize its complexity to prevent overfitting. This regularized model will have errors when predicting the original model's errors. Because the prediction of the new improved model is dependent on the forecast of the new error-predicting model, it will also include mistakes, though they will be smaller than previously.

To counteract this, we take two steps. First, we assign a tiny weight to the output of each single error predictor to limit our dependence or faith in it, η (usually between 0 and 0.1). Then, rather than terminating after one iteration of improvement, we repeat the process several times, learning new error predictors for newly built improved models, until the accuracy or error is acceptable. The equations below summarize this, where x is an input.

```
improved_model(x) = current_model(x) + η × error_prediction_model(x)

current_model(x) = improved_model(x)

Repeat above 2 steps till satisfactory
```

Fig: This depicts the basic mathematical pseudocode for boosting.

Let us look at one of the most popular boosting algorithms which were used in my model XGBoost. To understand XGBoost further we need to understand the mathematical intricacies.

XGBoost is a flavor of gradient boosting machines that use Gradient Boosting Trees (gbtree) as the error predictor. It starts with a simple predictor which predicts an arbitrary number (usually 0.5) regardless of the input. That predictor has a very high error rate. Then, the above idea is applied until the error is brought to a minimum. In XGBoost, training of the error prediction model is not done by trivially optimizing the predictor on (feature, error) pairs.

In essence, the gbtree's goal is to minimize the loss as well prevent the model to not overfit. Let us only consider one iteration in a gbtree for simplicity. To understand it better, let's start with the simplest possible tree that doesn't split and predicts the same value no matter what the input. This tree is extremely simple, independent of the input, and certainly under-fitted. However, it can still help reduce losses.

$$Loss(o) = \min_{o} \sum_{i=1}^{N} loss(y_i, f(x_i) + o) + \frac{1}{2}\lambda o^2$$

where N is the number of samples, f is the original model,
λ is the L2 regularization parameter and o is the value which we want to find

Fig: A simple MSE loss with L2 regularization to penalize.

L2 regularization is applied to penalize the model to prevent it from overfitting. To get a simplified expression to quantify the loss we can use a Taylor series approximation.

$$o = \frac{\sum_{i=1}^{N} g_i}{\sum_{i=1}^{N} h_i + \lambda}$$

Fig: A simplified loss function after Taylor series approximation.

My XGboost model uses the same features extracted from the VGG Convolutional filters and are fit into the XGB classifier. The model performed much more admirably giving an accuracy of **62%**. The confusion matrix also gives us a better picture of how much better gradient boosting performs as compared to the RF classifier.

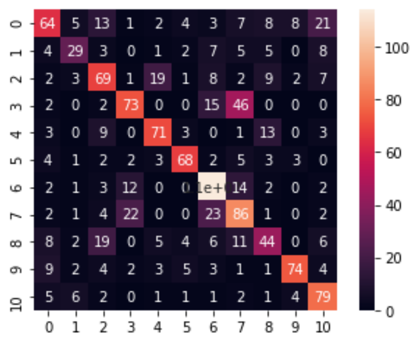


Fig: This shows the confusion matrix of the VGG XG classifier model.

As we can see, the XGboost performs much better than random forest, this is can be attributed to the following reasons:

- Xgboost performs pruning on the tree with a parameter called ‘Similarity Score’, this prevents the model from overfitting.
- Xgboost works well with unbalanced datasets in the sense that when the model fails to predict something on the first iteration it gives it more weightage in the following iterations
- Xgboost reliance on hyperparameters is much less as compared to random forest; It gives more importance to functional space.

8) Conclusion

While many models work well with different datasets, we can learn the concept of ‘No free lunch theorem’, which states that no model is perfect for all types of data. It is the duty of the engineer to understand, mix and match different models to find which works best on that dataset.

The objective of this project was to understand how different combinations of models work on small datasets. While I learned that most deep learning models do not work very well on small data, machine learning classifiers perform quite admirably. The main difficulty lies in feature engineering and selection, with many complimentary features across different classes, certain classifiers perform quite poorly.

However, I have managed to achieve quite admirable results, but of course, further investigation is needed in the form of larger sample sizes and better image preprocessing.

9)References and Codes

[1] http://people.csail.mit.edu/torralba/publications/indoor.pdf
[2] https://www.sciencedirect.com/science/article/pii/S2214317316300099
[3] https://arxiv.org/abs/2101.02767
[4] https://towardsdatascience.com/extract-features-visualize-filters-and-feature-maps-in-vgg16-and-vgg19-cnn-models-d2da6333edd0
[5] https://medium.com/geekculture/xgboost-versus-random-forest-898e42870f30
[6] https://github.com/bentrevett/pytorch-image-classification
[7] Code: https://github.com/disturbed-mystic1/IndoorSceneRecognition_ITR

9) Annexures

8.1) Annexure 1:

PO	• Tick	Page. No	Section No	Guides Observation
PO1	✓	11	8	
PO2	✓	9	4	
PO3	✓	17	6	
PO4	✓	19	6	
PO5	✓	19	6	
PO6	✓	6	3	
PO7				
PO8				
PO9				
PO10	✓	9	3	
PO11				
PO12				

Table 8.1: PO Mapping

PSO	• Tick	Pg. No		Section No	Guides Observation
PSO1					
PSO2	✓	11		5	

Table 8.2: PSO Mapping

8.2) Annexure 2:

Sl	PLO	· Tick	Pg. No	Section No	Guides Observation
1	C1.	✓	11	5	
2	C2.	✓	14	6	
3	C3.	✓	17	7	
4	C4.	✓	17	7	
5	C5.				
6	C6.	✓	9	4	
7	C7.				
8	C8.				
9	C9.				
10	C10.				
11	C11.				
12	C12.				
13	C13.	✓	14	6	
14	C14.				
15	C15.				
16	C16.				
17	C17.				
18	C18.				