Machine Learning Engineer Nanodegree

Capstone Project Report

Yiu Shun Wilson, LAM

6th December, 2020

Indoor Scene Recognition

Contents

1. Definition

1.1 Project Overview

Indoor scene recognition has a wide variety of applications: in robotics, for example, it plays a major role in environment cognition and navigation[1]; in assistive technology recognizing indoor scene and environment helps develop applications and solutions for individuals with visual impairment to safely navigate indoor[2], and potentially, from the perspective of a speech-language pathologist, forms the ground of developing applications that are able to automate the recommendation / selection of dynamic, environment- and context-sensitive communication content for augmentative and alternative communication (AAC) (e.g. on smartphones or tablets) for end-users accordingly such that they can save time flipping / navigating over categories or pages on their AAC device, while the buttons / cards on the screen are semantically relevant to their physical and/or social context, and hence communication can become more efficient and effective.

However, indoor scene recognition remains a challenging open problem in the field of computer vision and artificial intelligence. Compared to open scene recognition, indoor scenes present vast variations, rich contents in the input and overlapping features among output classes; some indoor scenes are well defined by global spatial properties while some by local objects they contain. Optimizing indoor scene recognition models therefore yields importance to both machine learning/computer vision and real-world applications.

1.2 Problem Statement

Indoor scene recognition is a multinomial classification problem. Given an image as input and N categories of scene, the goal of an indoor scene classifier is to predict which category the image belongs to as output.

Unlike other classification tasks such as fraud detection and text classification, image classification requires us to represent (color) images as 3-dimensional matrices with color RGB (Red, Green, Blue) channels of range 0 – 255 (Fig. 1).

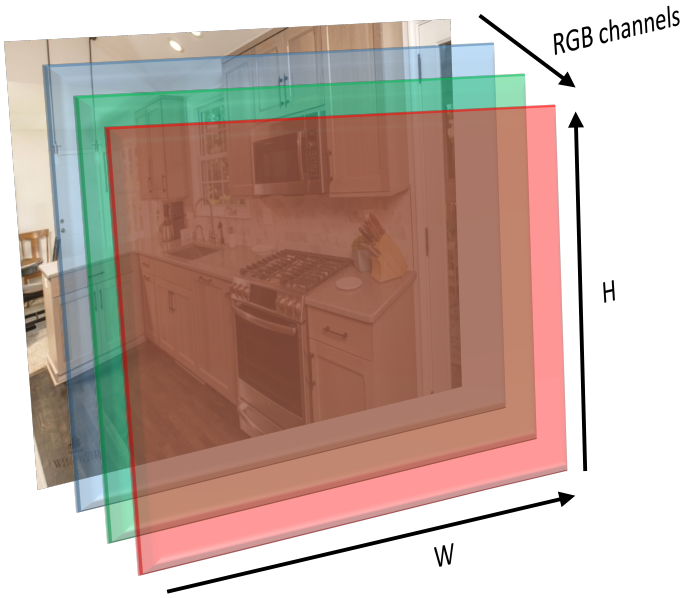

Fig. 1 Representation of Images as 3-dimensional RGB matrices (3 x W x H)

To capture and therefore classify the delicacy of such large number of data points, i.e. number of images x (3 x W x H) per image, models such as deep neural network (DNN), or particularly Convolutional Neural Network (CNN), would be the viable option.

1.3 Evaluation Metrics

1.3.1 Top-1 accuracy

Multiple class image classification is usually evaluated mainly on the Top-1 accuracy on a pre-defined test set. In simple terms, the number of matches between the most probable predicted class from a softmax layer to the ground truth label will be computed and divided the by the number of test set samples. More technically, the accuracy of each class (Eq.1) could be computed by the sum of *true positives(TP)* and *true negatives (TN)* divided by the sum of TP, TN, and *false positives(FP)* and *false negatives(FN)*:

$$Accuracy_k = \frac{TP + TN}{(TP + TN + FP + FN)}$$

Eq. 1 Accuracy of class k

We could also use a confusion matrix (Fig. 2), where the actual number of accurate and inaccurate predictions are represented in the diagonals and the rows and columns except the diagonals, to visualize the overview of results and compute the metrics.

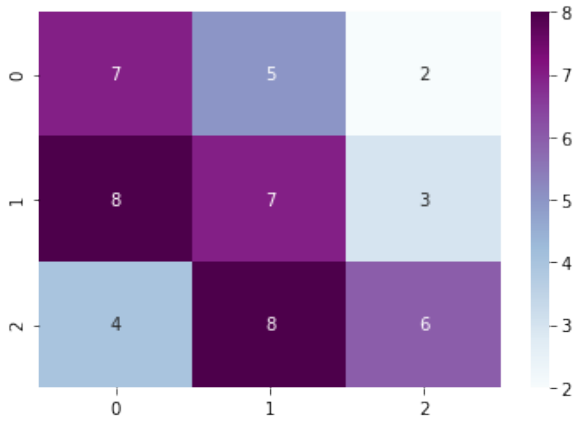

Fig 2. Sample 3 x 3 Confusion Matrix (3-classes)

Formally, given a $n \; x \; n$ confusion matrix M, the overall Top-1 accuracy of multinomial classification model is defined as the trace of the matrix, i.e. the sum of the diagonals, divided by the sum over the whole matrix (Eq. 2), where such element represents the top-1 predictions from the model:

$$tr(M) = \sum_{i=1}^{n} M_{ii}$$

$$Accuracy = \frac{tr(M)}{\sum_{i=1}^{n} \sum_{j=1}^{n} M_{ij}}$$

Eq. 2 Overall Top-1 Accuracy

1.3.2 Precision and recall

Precision and recall are more delicate measures of the performance of a model. Specifically, *precision*, also known as positive predictive value (PPV), measures the ratio of correct positive predictions to the total positive predictions, and *recall*, a.k.a. true positive rate or sensitivity, measures the ratio of correct positive predictions to the total positive ground truth labels (Eqs. 3). Simply put, in the context of indoor scene classification, precision tells, for example: among the group of image the model predicts as *kitchen*, how many of them are

actually kitchen, and recall tells, for example: among the group actual image of *kitchen*, how many the model has identified as *kitchen*.

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

Eqs. 3 Precision and recall

1.3.3 f1 score

In the application of image classification, especially in rehabilitation and assistive technology, we might want both precision and recall to be as high as possible, and as we shall see, our dataset might also be unevenly distributed or imbalanced; hence, we would also evaluate a model on the weighted average of precision and recall, a.k.a *f1-score* (Eq. 4).

$$f1\ score = 2 \times \frac{(Precision \times Recall)}{(Precision + Recall)}$$

Eq. 4 f1 score

The above only shows the calculation of each class, and while we are working with multi-class classification, we would also look at the macro (global) average and weighted average (for imbalanced classes) of precision, recall, and f1.

2. Analysis

2.1 Dataset and Inputs

This project used the MIT Indoor 67 dataset, which was developed in 2009 in MIT [3] and subsequently used by studies on indoor scene recognition [e.g. 1,2,4] and QSTP in-class competition on Kaggle. It contains 67 label categories and 15620 images, with each category containing around 100 images; the original, standard train-test split was 80:20 for each category.

2.1.1 Size variations

As shown in the Fig.3 below, our dataset has images of various size. This was expected since raw images are usually captured or taken with various devices or aspect, with or without manual cropping or resizing.

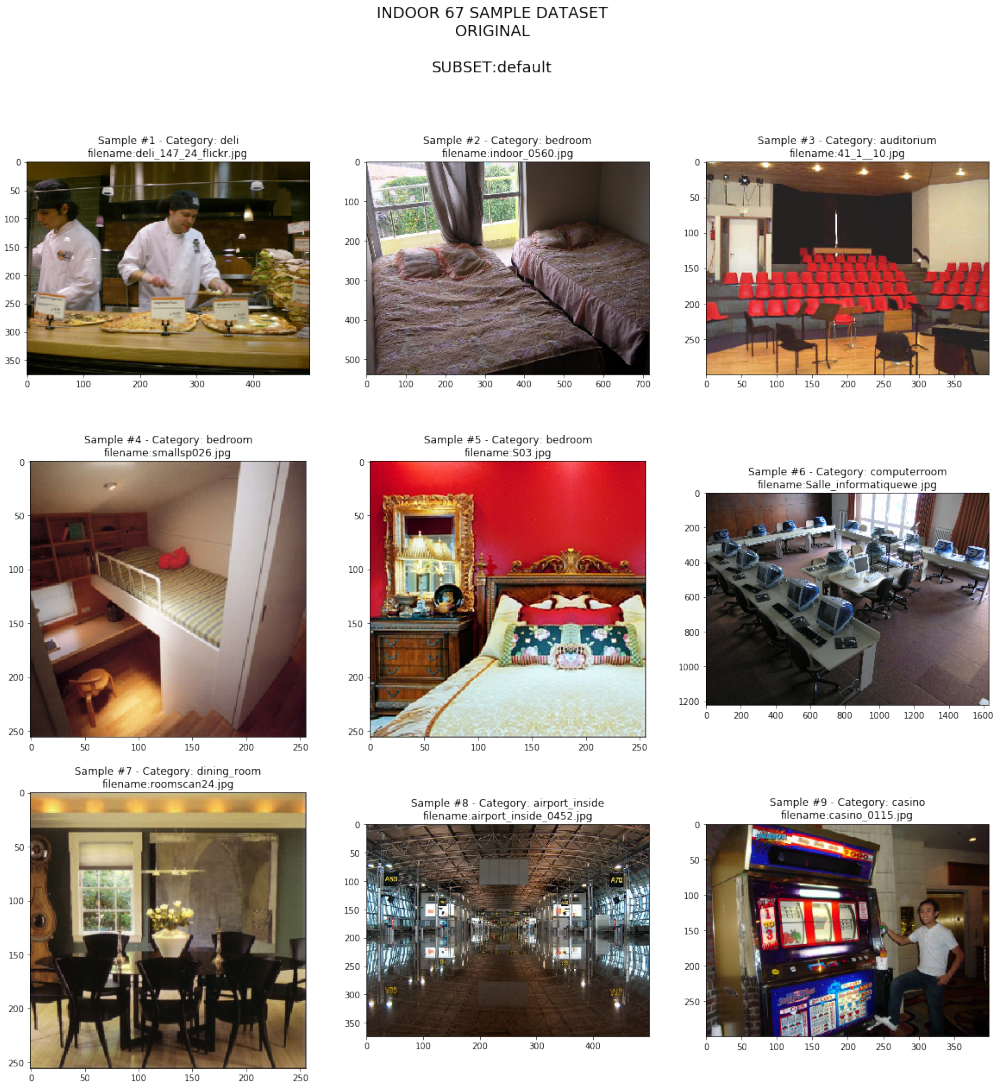

Fig. 3 Visualization of 9 random images from MIT Indoor 67

2.1.2 Class distributions

Our dataset seems to be unevenly distributed, i.e. imbalanced. This was also expected and makes sense since there are indoor space people may visit more frequently or staying longer, such as kitchen, living room, and bedroom (the top 3 elements in the dataset), while greenhouse, elevator, lobby (the bottom 3 elements) are less likely to be visited or stayed. However, this would be an issue that we shall address later on.

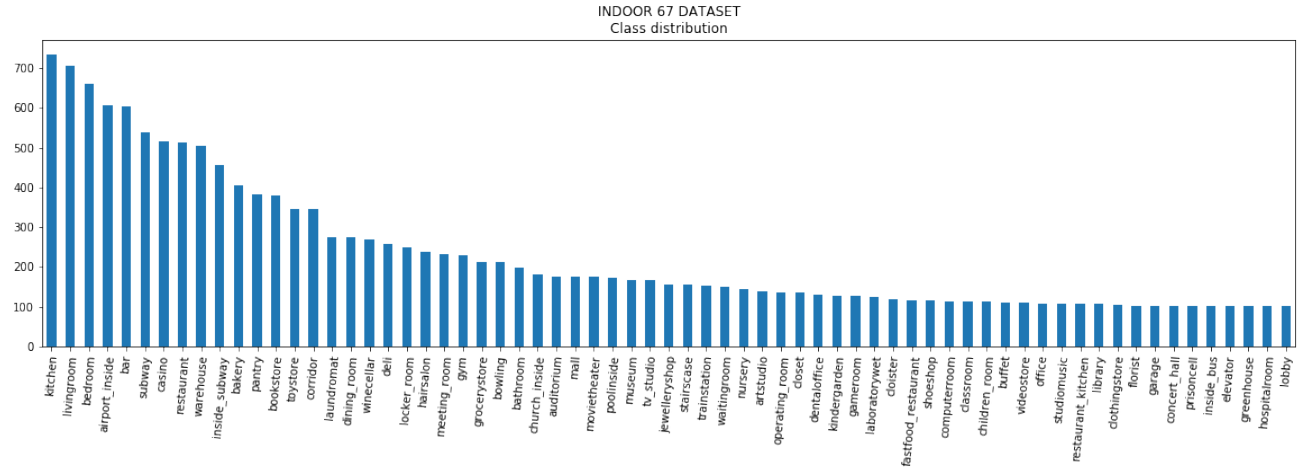

Fig. 4 Class distribution of MIT Indoor 67

2.2 Algorithms and Techniques

2.2.1 Convolutional Neural Networks

One challenge in image classification is the massive number of parameters fed into a model Consider a small 32 x 32 RGB image. It contains 3,072 parameters and while image classification models hunger for large dataset, the size of parameters would increase massively, overloading the model and therefore becoming computationally expensive. In the context of deploying models on edge devices later on, efficiency is certainly a primary concern.

Fig. 5 Sample CNN architecture (VGG-16) for a 224 x224 RGB image.
Source: Shi, et al. (2018). Learning better deep features for the prediction of occult invasive disease in ductal carcinoma in situ through transfer learning, *Proc. SPIE 10575, Medical Imaging 2018: Computer-Aided Diagnosis*, 105752R (27 February 2018)

We would address this issue by using Convolutional Neural Networks (CNNs) and the optimization techniques that come with (Fig. 5). Essentially, CNNs are deep neural networks and could be implemented recursively with:

(1) *Convolution*: Reduce the dimension of data image by walking through the image by small steps (strides), selecting small portion of the image, applying a filter matrix (e.g. a 3x3 matrix) and then multiplying the filter with original small matrix followed by summing up the matrix plus a biased constant to produce stacks of convolutions. Intuitively, the network is extracting and learning *features* of images from a human's perspective.

(2) *Batch Normalization*: Select a mini-batch of the convolutions and normalize the mini-batch to avoid the range of values changing as we move along the network (covariance shift) such that the network runs faster in training and whenever new data comes in, we do not need to retrain the network; it also offers slight regularization effect to prevent overfitting.

(3) *Activation*: Apply a non-linear function such as a sigmoid function (Eq. 5) to project probability or a range of 0 to 1, or -1 to 1, such that we could map outputs of hidden layers to predict certain features or classes in the next layer or the final softmax layer.

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Eq.5 Sigmoid function, where z is a linear function

(4) *Pooling*: Reduce the size of feature maps by apply another window matrix (e.g. a 3x3 matrix) and selecting either the max value of the matrix (max pooling).

2.2.2 EfficientNet – A Rethinked Approach of Network Scaling

Scaling up a CNN could help improve accuracy of models; this could be done by making the network deeper (depth scaling), wider (width scaling), using higher resolution, or a mix (compound scaling) (Fig. 6), but as Tan and Le [5] pointed out model scaling has been rather arbitrarily without careful handling.

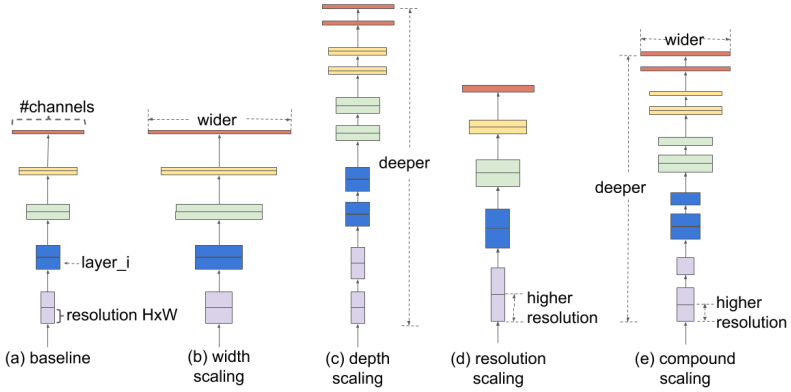

Fig. 6 Model Scaling – (a) baseline, (b) width scaling, (c) depth scaling, (d) resolution scaling, (e) compound scaling
Source: [5]

EfficientNet was hence proposed as a family of CNN models with compound scaling implemented in a principled way of uniform scaling by a compound coefficient ϕ (Eqs. 6).

$$depth: d = \alpha^{\phi}$$
$$width: w = \beta^{\phi}$$
$$resolution\ r = \gamma^{\phi}$$
$$s.t.\ \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$$
$$\alpha \geq 1, \beta \geq 1, \gamma \geq 1$$

Eq. 6 Compound scaling method of *EfficientNet*
Source: [5]

Their proposed method has been empirically shown to produce high accuracy on ImageNet with fewer Parameters and Floating Point Operations per Second (FLOPS) [5]. Subsequent studies (e.g. [2]) has also reported state-of-the-art accuracy.

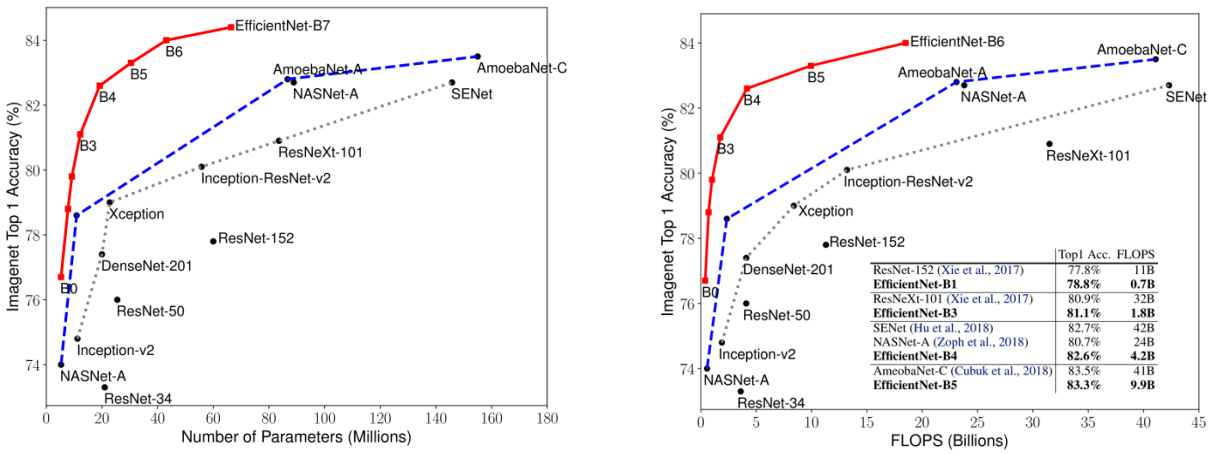

Fig. 7 Accuracy & Number of Parameters (Left) and Accuracy & FLOPS (Right)
Source: [5]

2.2.3 Data augmentation

To help CNN models to generalize to unseen data and avoid overfitting, i.e. learning or evening memorizing the train set while failing to predict accurately on unseen data, we can augment images by random rotation, cropping, flipping, adding noise, blurring. In this way, we also engineer more training data for CNN models to learn from.

2.2.4 Transfer learning and fine-tuning

Training neural network from scratch not only requires more time for model to learn but, as in the case of MIT indoor 67, such small size of data would also make the model vulnerable to overfitting. Transfer learning from models that have been trained on larger dataset, in the case of computer vision on ImageNet, would be viable option. One form of such approach is *fine tuning*, where we add a final fully connected layer on top of the pretrained model, with or without unfreezing layers of the original model, depending on the similarity of the current dataset and the pretrained dataset. In the case of *EfficientNet*, it has been suggested to unfreeze an entire block together, with batchnorm2d layers remained frozen since they were highly optimized by large dataset [10].

2.2.5 Adaptive Moment Estimation with Weight Decay (AdamW)

Intuitively, minimizing a loss function is like walking down a hill and finding the deepest valley (global minimum). We might also want to take larger step (larger learning rate) when we reach a flat land and smaller steps (smaller learning rate) when we walk down a steep slope such that with fewer steps we could reach the valley. This could be achieved by adaptive moment estimation (Adam): keeping track of past moving average and the current gradient to adapt step size for each individual weights [6]. To help with generalization and avoid overfitting as well as fixing the 'missing effect' of L2 regularization in the original Adam[8], weight decay would also be applied after controlling the step size of each element [6, 7].

2.2.6 Check-pointing and early stopping

To make sure we obtain the optimal model, whenever the model improves on lowering the loss or raising the accuracy, we would save that model and if it is observed that the validation loss starts increasing over a certain amount of epochs (patience), where we suspect the model starts overfitting, we would stop the training.

2.3 Benchmark Model

While it is known that many published studies use a lot more complex CNN architecture on the MIT 67 Indoor dataset (e.g. [4]), our benchmark model was a relatively simple, fine-tuned ResNeXt-101_32x16d by Ashrut Kumr in 2019 QSTP in-class Kaggle competition (source code). ResNeXt-101 is 101-layer CNN network that falls into the family of ResNet, which is trained by residual learning, i.e. network with 'skipped connection' to avoid vanishing gradient in deep networks, plus additional cardinality layers. Since the reported accuracy (84.4%) of this model was calculated on the entire dataset, the benchmark model was replicated from its source code, trained with our train set with data augmentation and the original hyperparameters (batch size: 50, stochastic gradient descent with momentum of 0.9, learning rate: 0.001) and obtained an overall top-1 accuracy of 81.56% against our test set.

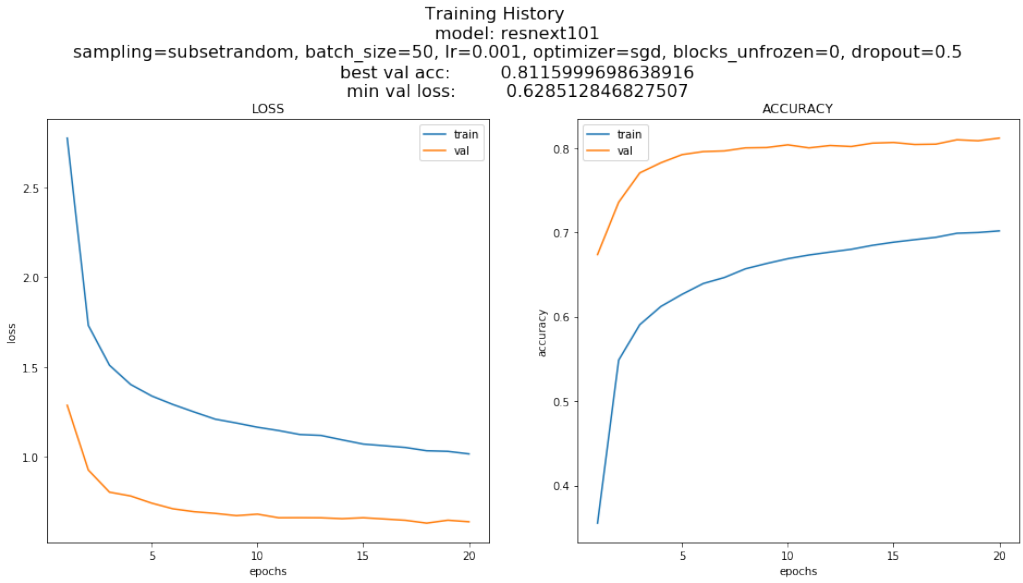

Fig. 8 Benchmark model training

As shown in Fig. 8, the benchmark model was still learning (under-fitting) after 20 epochs but started converging and showed pretty well generalization in the validation set. But as a point of comparison, training was only conducted for 20 epochs.

3. Methodology

All preprocessing, training, and testing were implemented on Amazon Web Service (AWS) Sagemaker with Python3 and PyTorch framework (version: 1.6.0) on ml.t2.medium, ml.p3.2xlarge, and ml.t2.medium instances respectively.

3.1 Data Preparation: Train-Validation-Test Split

Following the Indoor 67 standard protocol, 80:20 train-test split, and a further 80:20 train-validation split were performed.

3.1.1 Imbalanced class

As we noticed, classes are unevenly distributed in this dataset and hence when a naïve splitting was performed with sklearn.model_selection.trian_test_split(), the class distributions in different subsets (train, validation, and test) were not proportional to one another (Fig. 9). We could have used weighted (or under- and over-) sampling later on but sampling would be biased towards the train set.

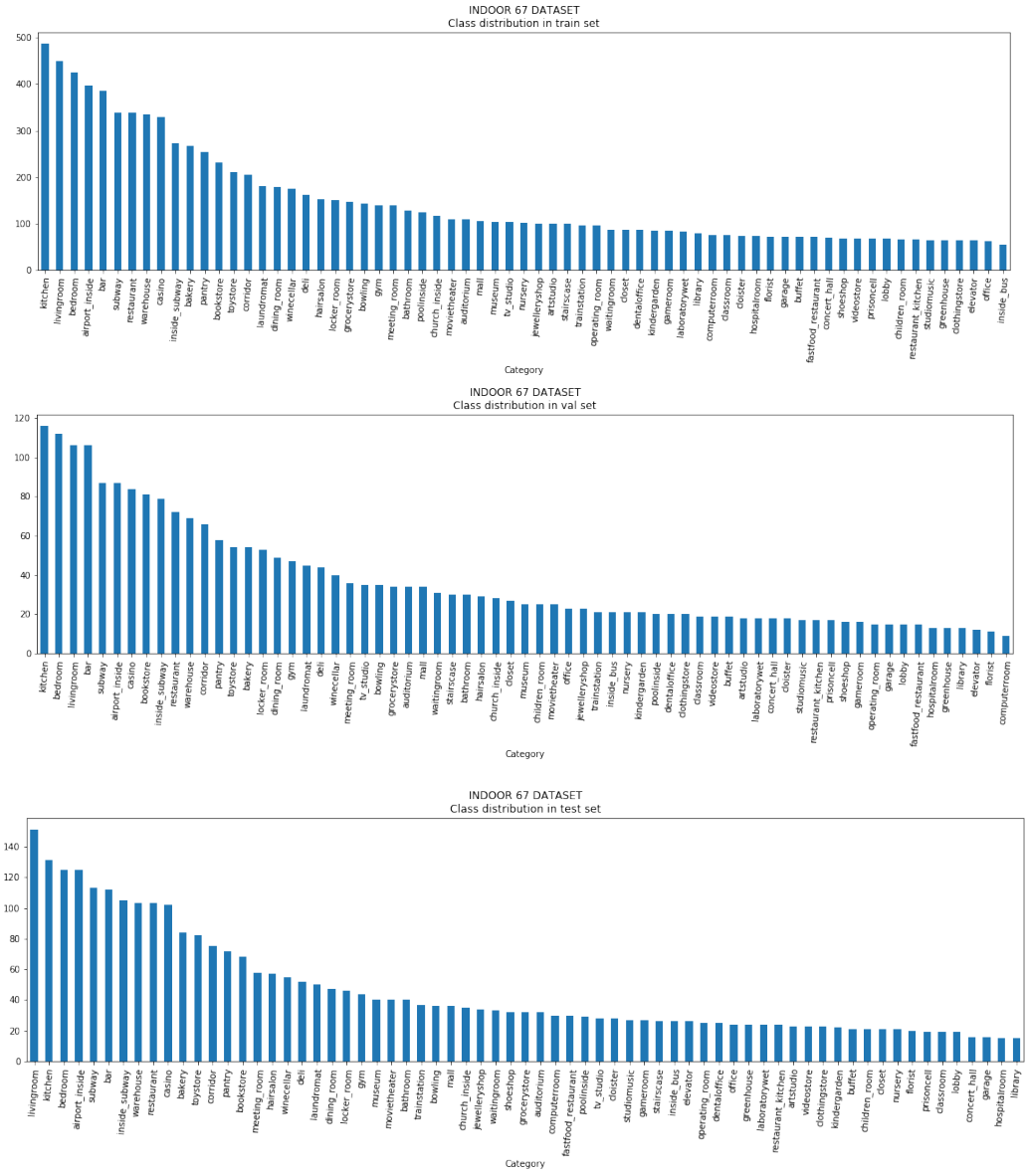

Fig. 9 Class distributions without stratified random splitting

Hence, stratified splitting was performed and hence we obtained subsets with similar disitrbutions (Fig. 10).

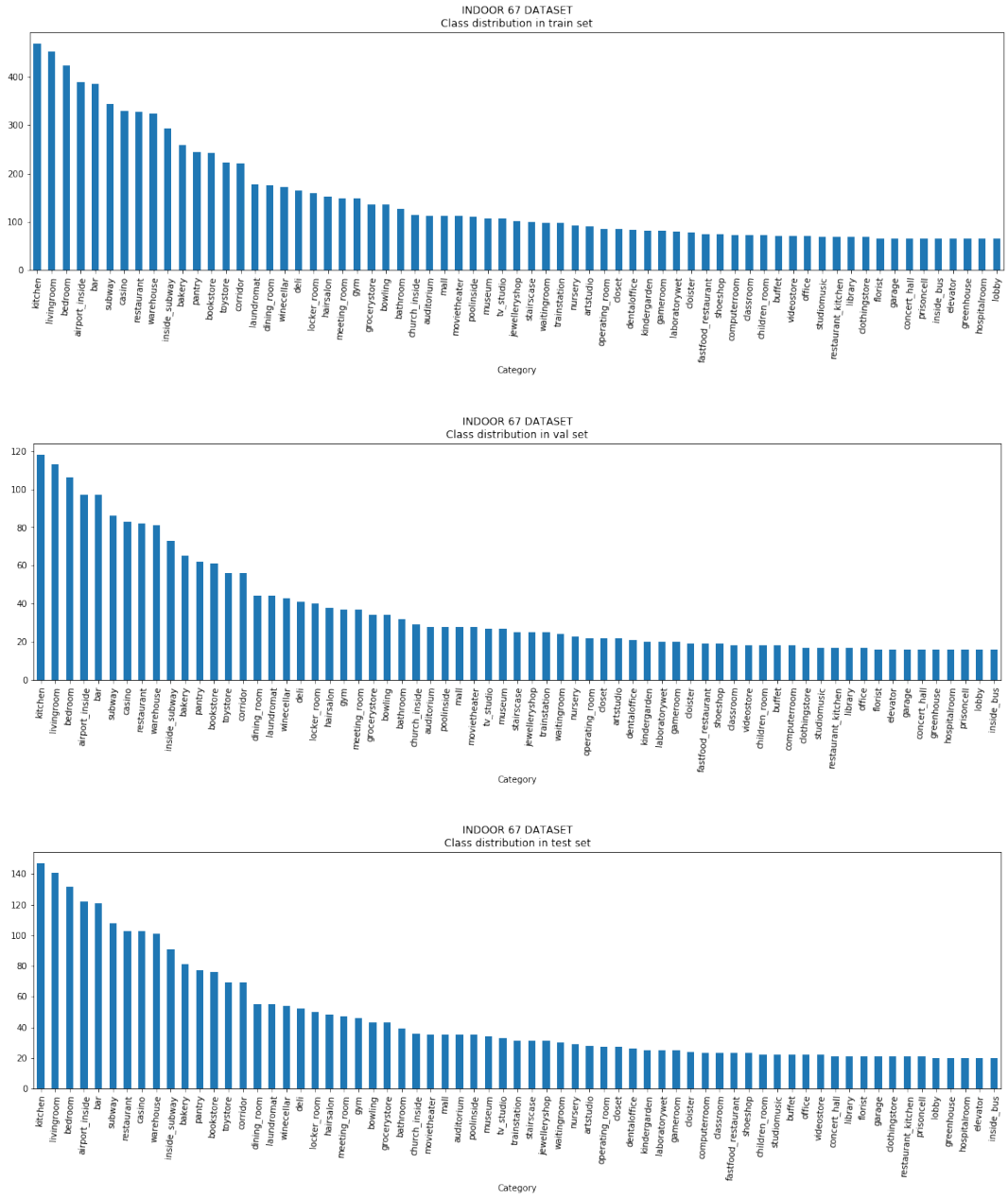

Fig. 10 Class distributions with stratified random splitting

Upon splitting, we obtained a train set, validation set, and test set of 9,996 images, 2500 images, and 3124 images respectively.

3.2 Data Preprocessing

Images were then preprocessed in advanced to speed up training on AWS [8]:

(1) Data augmentation: a mix of random cropping, flipping, rotation, brightness adjustments were applied to the original image; this process was repeatedly implemented 2 times and the augmented data were concatenated with the original image, such that the final train set size was 29, 988. Fig. 11 shows the samples of original and augmented images.

Fig. 11 EfficientNet-b3 original and augmented images: random categories (Left) and *kitchen* (Right)

(2) Resizing: As we are performing compound scaling in EfficientNets, the specific resizing image resolution was implemented with reference to each version [9].

(3) Conversion to Tensor and Normalization: the images were converted into Tensor objects using the PyTorch library and normalized with the tensor array [[], []]. This step was done right before the data were fed into the network.

3.3 Implementation

3.3.1 Initial Solution: Base Model

A base model was first trained by (1) loading weights of EfficientNet-b3, pre-trained with ImageNet, in an open source library for PyTorch, (2) adding a top classification layer of 67 classes, and (3) using the default setting of AdamW optimizer, (4) with a batch size of 32, dropout probability of 0.5, and 20 epochs. EfficientNet-b3 was chosen since this version was compared with ResNeXt-101 in original EfficientNet paper[5].

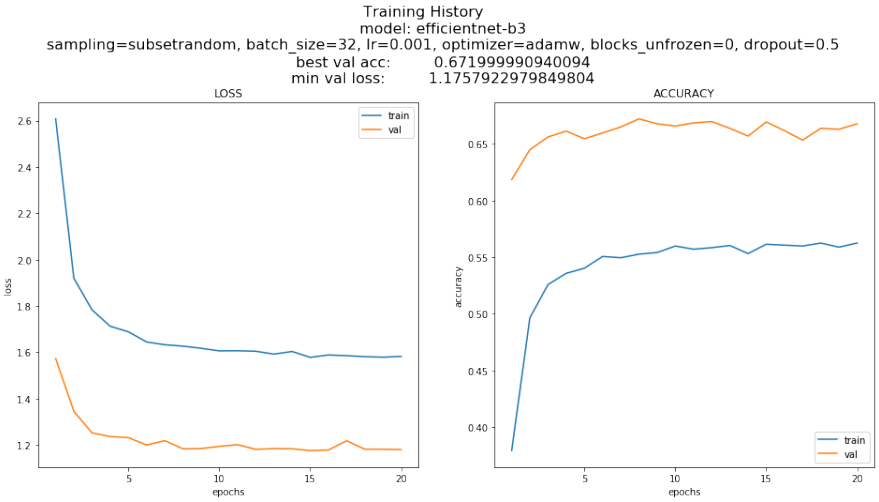

Fig.12 *EfficientNet-b3* base model training

The base model, with the best validation accuracy 67.19%, seemed to under-fit and struggled to further learn from the train set after 20 epochs. This could probably be due to the dissimilarity between ImageNet (the pre-trained set) and MIT Indoor 67, where the form mainly contains images of objects while the later of scenes which is more complex by nature. Fine-tuning with unfreezing layers and/or more complex fully connected layer is/are certainly warranted here.

3.3.2 Refinement: Model Architecture and Hyperparameter Tuning

Instead of using a discrete trial and error approach, Bayesian search with respect to minimum validation loss (objective metric) was adopted to tune hyperparameters and architectures using AWS Hyperparameter Tuner; the base model was loaded for training and patience was set to 3 epochs before early stopping. Table 1 below shows the ranges searched:

Hyperparameter / Architecture	**Search range**
Learning rate & weight decay	0.0001 – 0.1
Dropout probability	0.1 – 0.9
Unfrozen blocks	0 – 26 (maximum block for EfficientNet-b3)
Depth of fully connected layer	1 (classification only) - 3

Table 1. Search range in Hyperparameter and Architecture Tuning

3.3.3 Fine-tuned EfficientNet-b3

A model was found to perform particularly well on train accuracy and loss, and converge as fast as in 3 epochs (Fig. 13), with learning rate of 0.0001, weight decay of around 0.0072, dropout around 0.7005, unfrozen blocks of 23, after a few rounds of hyperparameter tuning. As compared to the base model, we can see a significant improvement: minimum validation loss of 0.653 and best validation accuracy of 82.76%.

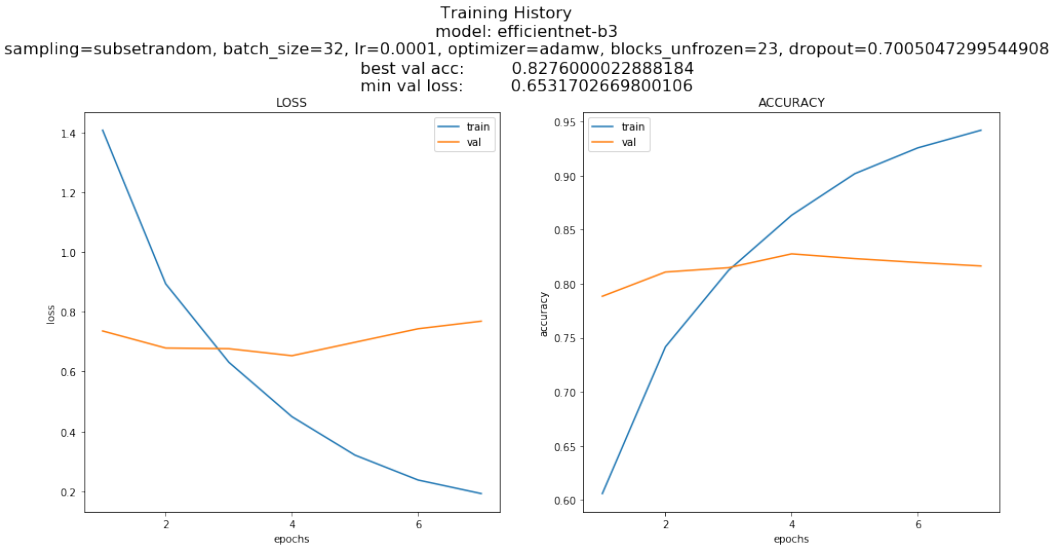

Fig. 13 Best fine-tuned EfficientNet-b3 model

The model learned pretty quickly but meanwhile also overfit the train set. While hyperparameters such as learning rate, dropout, and weight decay, and the unfrozen blocks were optimized by Bayesian search, it could have been trained with further regularization techniques such as more augmented data or smaller batch-size to observe improvement in overtfitting. Other comparable optimization algorithms such as Stochastic Gradient Descent and Adabound[13] were also attempted preliminarily but they required more rounds of trials with Bayesian search to optimize their hyperparameters. These were not pursued due to limited computational resources.

4. Results

The benchmark model and the final model were evaluated on the test set with respect to overall Top-1 accuracy, macro and weighted average precision, recall, and f1-score. These are summarized in Table 2 below.

Metrics	ResNeXt-101_32x16d_WSL (Benchmark)	Fine-tuned EfficientNet-b3
Top-1 accuracy	81.56%	81.95%
Macro average precision	80.23%	81.56%
Weighted average precision	82%	82.24%
Macro average recall	79.46%	79.83%
Weighted average recall	81.56%	81.95%
Macro average f1-score	79.34%	79.83%
Weighted average f1-score	81.37%	81.81%

Table 2. Comparison of evaluation metrics

The results were encouraging: the fine-tuned EfficientNet-b3 model slightly outperformed the benchmark model on every evaluation metric by less than 1%, except macro average precision. This could be visualized through their confusion matrices (Fig. 14), where we could clearly see that the models got similar 'noises' in the misclassified regions (cells except the diagonal), while there are a few cells with low correct classification and a few with high wrong classification in the benchmark model.

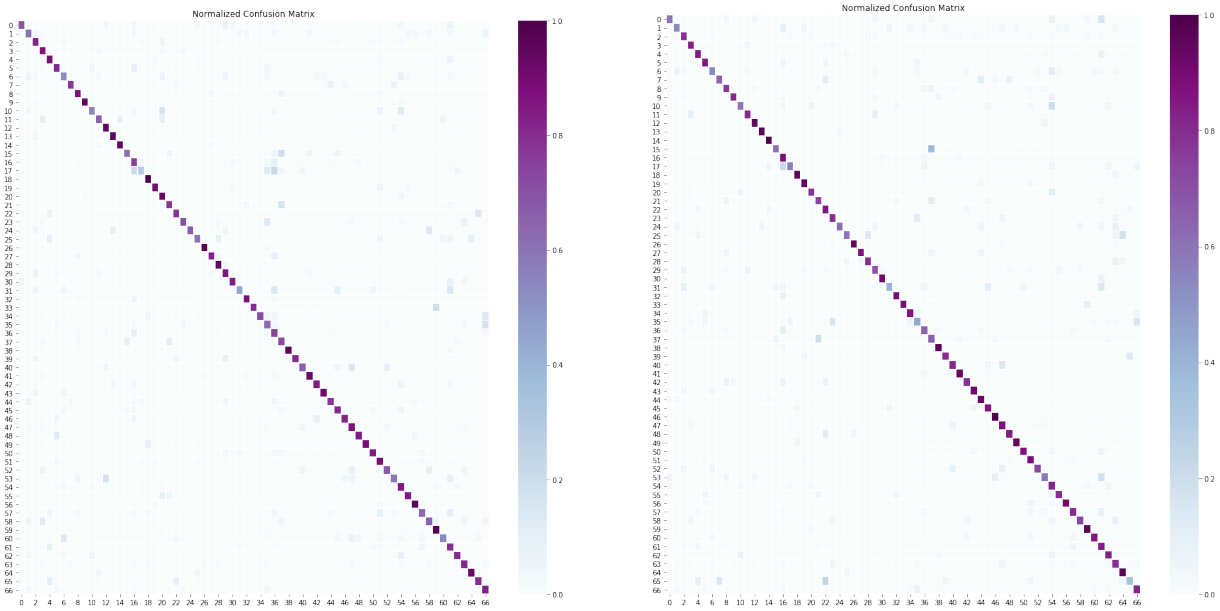

Fig. 14 Normalized Confusion matrices of fine-tuned EfficientNet-b3 (Left) and ResNeXt-101_32x16d_WSL (Right)

However, both models suffered from either under-fitting (Benchmark) or over-fitting (current model), meaning that further optimizations were warranted. But as discussed in Section 5, this could also be due to the challenge of distinctiveness of images.

5. Conclusions

5.1 Feature Distinctiveness

Interestingly, both models seemed to struggle with differentiating *bar* (class 16) from *fastfood restaurant* (class 17) when they were given images of the latter. This makes some sense in that these places are *visually similar* (Fig. 15): they both get a bar table / counter, chairs by the bar table, tables and chairs set up in lines, while *bar* has got extra features like specific decors and pool tables, making them *visually* distinguishable from *fastfood restaurant* but not the other way round.

This also applied to cases like *lobby* (class 31) vs. *living room* (class 61) – both models, *operating room* (class 22) vs. *hospitalroom* (class 65), and *deli* (class 35) vs. *bakery* (class 66) – ResNeXt-101, where one scene is more distinguishable from another while the other is less differentiable. ResNeXt-101 seemed to make more 'confusion' in this regard and hence underperformed – probably because it was not optimally fine-tuned; recall that the optimal hyperparameters were not reported by the original author.

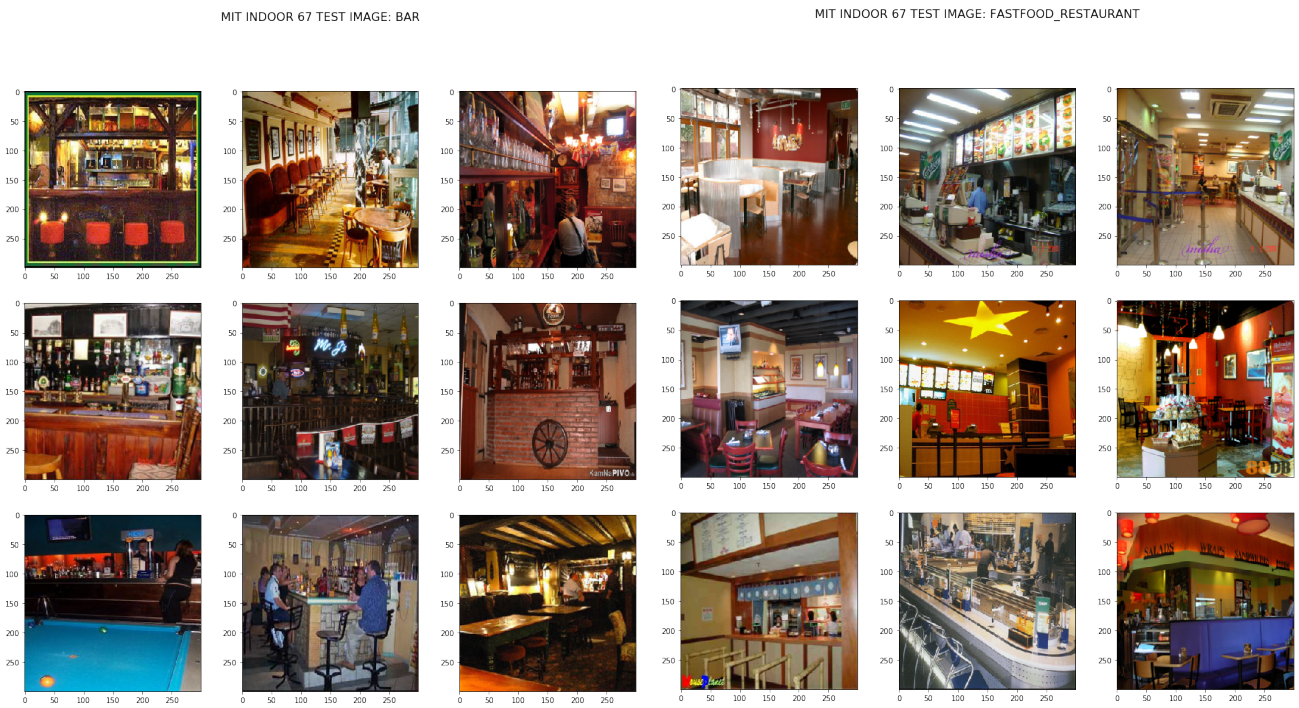

Fig. 15 Images of *bar* (class 16) and *fastfood restaurant* (class 17)

5.2 Training Efficiency and Optimization

In addition to the performance metrics, training of EfficientNets, given fewer FLOPS, was fast, making optimization and tuning more efficient, using fewer computational resources (especially when using Cloud computing services like AWS), more trials of hyperparameter architectural tuning therefore could be administered, making them 'easier' to tune, and hence more likely to identify an optimal model. From Figure 16, we could see that given early stopping an EfficientNet, with unfrozen blocks, could be trained in around 20 minutes while a freezed ResNeXt-101 network took 2 hours, not to mention if a fraction of the network is unfrozen – for such time period, there would have been 6 trials of hyperparameter tuning for EfficientNets.

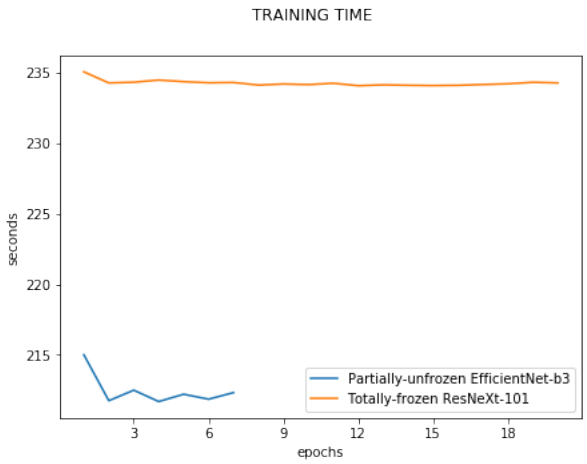

Fig. 16 Training time of totally-frozen ResNeXt-101 vs. Partially-unfrozen EfficientNet-b3

5.3 Transfer learning in scene classification

Unfreezing 23 out of 26 blocks of layers in fine-tuning, while models with fewer unfrozen blocks seemed to generalize poorer, re-indicates that scene images are pretty different from object images from ImageNet: ImageNet and MIT indoor 67 probably only share very low-level features such as edges. A better approach could have been pre-training EfficientNet with larger scene dataset such as Places365[11].

5.4 Future Direction

The current approach and model could be seen as a preliminary step towards training a CNN scene classifier, given its complexity: feature distinctiveness/ambiguity and challenge of transfer learning from available scene dataset. Given more computational resources, optimization of various hyperparameters, higher versions of EfficienNets, and larger augmented datasets could be attempted to see if improvements could be made.

5.5 Application

Given the motivation of the current project being the development of scene classification for assistive technology, a well-optimized version of EfficientNet could be deployed at Cloud endpoints or training a *EfficientNet-lite*[12] with the optimal hyperparameters and optimizations identified in EfficientNets and deploying the lite-model at edge devices such as smart phones or a raspberry-pi.

References

1. Liu, S. & Tian, G. (2019). An Indoor Scene Classification Method for Service Robot Based on CNN Feature. *Journal of Robotics, Vol. 2019*.

2. Afif, M., Ayachi, R., Said, Y., & Atri, M. (2020). Deep Learning Based Application for Indoor Scene Recognition. *Neural Processing Letters, 51*.

3. Quattoni, A. & Torralba, A. (2009). Recognizing Indoor Scenes. *2009 IEEE Conference on Computer Vision and Pattern Recognition*, Miami, FL, 2009, pp. 413-420

4. Seong, H., Hyun, J., & Kim, E. (2019). FOSNet: an end-to-end trainable deep neural network for scene recognition. *IEEE Access*, 8, pp. 82066-82077.

5. Tan, M., & Le, Q. V. (2019). EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. *Proceedings of the 36 th International Conference on Machine Learning*, Long Beach, California, PMLR 97.

6. Graetz, F.M. (3rd June, 2018). *Why AdamW matters* (blogpost), retrievable at https://towardsdatascience.com/why-adamw-matters-736223f31b5d.

7. Gugger, S., & Howard, J. (2nd July, 2018). *AdamW and Super-convergence is now the fastest way to train neural nets* (blogpost), retrievable at https://www.fast.ai/2018/07/02/adam-weight-decay/

8. Loushchilov, I., & Hutter, F. (2019). Decoupled Weight Decay Regularization, *ICLR' 2019*, retrievable at https://arxiv.org/pdf/1711.05101.pdf

9. Keras. (n.d.). *EfficientNet B0 to* B7, Keras API Documentation. https://keras.io/api/applications/efficientnet/.

10. Chengwei (Jun, 2019). *How to do Transfer learning with Efficientnet* (blogpost), retrievable at https://www.dlology.com/blog/transfer-learning-with-efficientnet/.

11. Zhou, B., Lapedriza, A., Khosla, A., Oliva, A., & Torralba, A. (2017). Places: A 10 million image database for scene recognition, *IEEE TPAMI*, *40(6)*.

12. Tensorflow (n.d.). *EfficientNet-lite* (Github repository). Retrievable at https://github.com/tensorflow/tpu/tree/master/models/official/efficientnet/lite.

13. Luo, L., Xiong, Y., Liu, Y., & Sun, X. (2019). Adaptive Gradient Methods with Dynamic Bound of Learning Rate, *ICLR' 2019*, retrievable at https://arxiv.org/pdf/1902.09843.pdf.