Classification of domestic environments

A project to embed in a real world scenario.

Emanuele Rucci

Abstract

The task chosen for this homework is the classification of domestic enviroments. The main goal behind this choice is to include the project in a real world website. The purpose of this website is to give people the opportunity to find new apartments in new buildings that meets specified requirements. So, when the rendering of an apartment are uploaded on the website, the inference process will attach the label of classification to the image; this information will be useful to then create automatic new filters based on new contents periodically added.

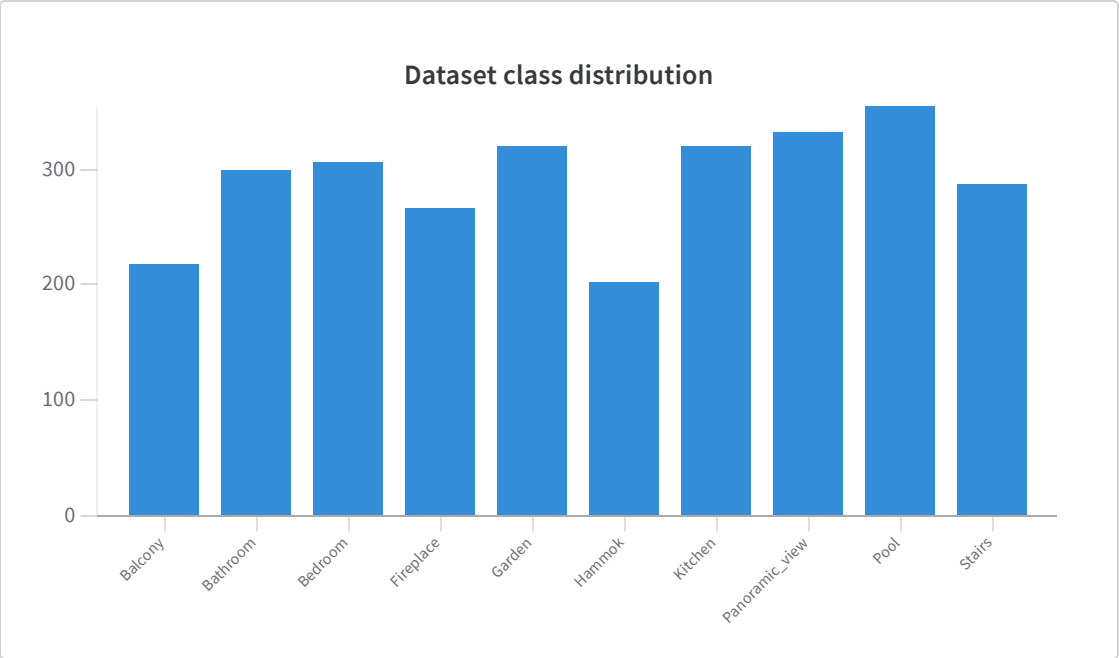
For this project the library chosen is pytorch, weight and biases has been use to generate this report and most of the data visualization graphs and images in this document. Furthermore have been choosen two differents type of model, one Deit, a VisionTransformer model based on the transformers architecture, and the other one, EfficientNet based on the classic convolutional neural networks architecture.

Data distribution

The dataset is self made and is almost a balanced dataset; to create the dataset has been used a google chrome extension that allows to

download all the images given from a google image search.

The dataset is composed of 2911 images divided in 10 classes; In the interactive graph below is it possibile to see the exact number of images for each class: Balcony, Bathroom, Bedroom, Fireplace, Garden, Hammok, Kitchen, Panoramic view, Pool, Stairs.



Have been used dataloaders to manage the training validation and testing division of the entire dataset.

The ratio for the division is as follow: training 80%, validation ad test 10%. The batch size for training is 128 images and the batch size for validation and test is 32. Given this, there are 19 batches (2324 images) for training, 9 batches for validation and 10 batches for test.

All the test have been mede with the same random seed for splitting to ensure that the comparison is correct.

Data preprocessing

To perform data augmentation and to preprocess the dataset have been used 2 pipelines using the utilities functions of *torchvision.transform.*

Pipeline for training:

- *RandomHorizontalFlip*: randomly flips the input image horizontally with a probability of 0.5, or 50%. This means that the image will be flipped horizontally with a probability of 1/2, or approximately half of the time.
- *RandomVerticalFlip*: randomly flips the input image vertically with a probability of 0.5, or 50%. This means that the image will be flipped vertically with a probability of 1/2, or approximately half of the time.
- *RandomApply ColorJittel*: the ColorJitter transformation is a specific type of transformation that randomly adjusts the brightness, contrast, saturation, and hue of an image. The RandomApply transformation will apply the ColorJitter transformation to the input with a probability of 0.25, or 25%. This means that the ColorJitter transformation will be applied to approximately 1/4 of the images in the dataset.
- *Resize*: transformation that resizes the input image to a size of 256x256 pixels. If the aspect ratio of the original image is not the same as the target size, the image will be resized such that the shorter dimension is 256 pixels, while the longer dimension is adjusted accordingly to preserve the aspect ratio.
- *CenterCrop*: transformation that crops the input image to a size of 224x224 pixels around the center of the image. If the aspect ratio of the original image is not the same as the target size, the image will be padded with zeros to match the target aspect ratio before cropping. This transformation is needed to apply the model that have been choose, infact they both accept input images of 224x224.
- *ToTensor*: transformation converts the input image to a tensor with shape (C, H, W), where C is the number of channels, H is the height, and W is the width.
- *Normalize:* transformation that scales and shifts the input image such that its mean and standard deviation match the given mean and standard deviation values. This is often used to normalize the input images to the same distribution as the training data, which can improve the performance of the model. The mean and standard deviation values are typically calculated from the training data and are used to normalize the input images to a zero mean and unit

variance. In this normalization has been used the mean and the standard deviation of the ImageNet dataset.
- *RandomErasing*: transformation that erases a random region of the input image with a probability of 20% by replacing the pixel values with a random value between the image range.

Pipeline for validation and testing:

- *Resize* to 256x256px
- *CenterCrop* 224x224px
- *ToTensor* transformation
- *Normalization* to ImageNet mean and standard deviation

The purpose of the transformations in training is to augment the training data by changing randomly elements. This can help the model generalize better and reduce overfitting. Data augmentation is a common technique used to improve the performance of machine learning models, especially in tasks such as image classification.

While in training are performed the discussed transformation, in validation and testing the inference is performed on real images to evaluate the performance.

Models & Architectures

In this work have been used 2 different models: VisionTransformer and EfficientNet.

VisionTransformer Deit

A Vision Transformer (ViT) is a type of neural network architecture that was introduced in a paper by Google Research in 2020. It is designed for image classification tasks and is based on the transformer architecture, which was originally developed for natural language processing (NLP).

The key idea behind the transformer architecture is to use self-attention mechanisms to process input sequences in a parallel and efficient manner, without the need for recurrent connections or convolutions. In the context of image classification, a ViT takes an image and represents it as a sequence of patches (or "tokens"), which are fed into the transformer architecture to compute a prediction.

One of the main advantages of the ViT architecture is that it can handle very large images, such as high-resolution photographs, without the need for down-sampling or pooling operations. This is because the self-attention mechanisms in the transformer architecture can attend to any position in the input sequence, allowing the model to learn long-range dependencies and global context directly from the input.

In addition to the self-attention mechanisms, a ViT also has a few other components:

- A patch embedding layer: This layer maps each image patch to a high-dimensional embedding space, which captures the visual features of the patch.
- A multi-headed attention layer: This layer applies self-attention mechanisms to the patch embeddings to compute a weighted sum of the patch embeddings, which represents the context of the image.
- A feed-forward network (FFN): This is a simple neural network that processes the patch embeddings and produces a prediction for the image.

DEiT (Distillation-aware Embedding and Transformation) is a family of efficient image classification models developed by Facebook AI. DEiT models are trained to perform image classification tasks on the ImageNet dataset.

DEiT models are based on a transformer-based architecture and are available in several sizes, ranging from small and efficient models such as DEiT-Tiny to larger and more powerful models such as DEiT-Base and DEiT-Large; For this project has been used the **DEiT-Tiny** version.

Efficient Net

EfficientNet is a family of image classification models introduced by Google AI in 2019. The main idea behind EfficientNet is to use a scaling method to scale up CNN models (EfficientNets are CNNs) in a more structured and principled way, so that the model can achieve better accuracy with fewer parameters and FLOPS (floating point operations per second).

To scale up a CNN model, one can simply increase the number of filters in each layer and the resolution of the input image. However, this simple approach leads to a significant increase in the number of parameters and FLOPS, which can be computationally expensive and may not necessarily lead to better accuracy. EfficientNet addresses this problem by using a compound scaling method that scales up the network in a more balanced way, taking into account the dimensions of the layers and their importance in the network.

The *compound scaling* method consists of three factors:

- Width: The number of filters in each layer is multiplied by a width factor (w). This controls the number of parameters in the model.
- Depth: The number of layers in the network is increased by a depth factor (d). This controls the capacity of the model.
- Resolution: The input image resolution is increased by a resolution factor (r). This controls the amount of computation required by the model.

By carefully choosing the values of these scaling factors, the authors of EfficientNet were able to achieve state-of-the-art accuracy on several image classification benchmarks while using fewer parameters and FLOPS than previous models.

In addition to the compound scaling method, EfficientNet also uses other techniques to improve efficiency, such as using a mobile-inspired network architecture and utilizing weight-sharing and depthwise convolutions.

For this task have been used EfficientNet-B0; it is the baseline model of the EfficientNet family, it has the smallest size and the lowest accuracy among all the models in the family. This model has been trained trained on the ImageNet dataset using the compound scaling method.

EfficientNet-B0 was obtained by applying the compound scaling method with scaling factors of 1.0 for all three dimensions: width, depth, and resolution. This resulted in a model that is relatively small and fast, while still achieving good accuracy on the ImageNet dataset.

EfficientNet-B0 has around 5.3 million parameters and can process images up to a resolution of 224x224 pixels. It has achieved an accuracy of 77.1% on the ImageNet dataset, which is competitive with other state-of-the-art models at the time of its release.

Runs

Both the model's architecture have been slightly adapted to the specific problem requirements such us output number of classes.

VisionTransformer Deit

Supposing that the baseline model, trained on ImageNet, has learnt differents features from various types of objects and subjects, here the only part trained is an head added to the model.

The head is composed as follow:

- *Fully connected:* from the 1000 classes that are the classes of ImageNet dataset as well the output of the baseline EfficientNet-B0, to 512 output values.
- *ReLu:* the Rectified Linear Unit activation function to be sure to capture non linear pattern between data in the last layers.
- *Dropout:* with a value of 30%; Dropout is a regularization technique for reducing overfitting in neural networks. It works by randomly

setting a fraction of the input units to zero during training. This has the effect of "dropping out" the corresponding hidden units in the forward pass, and forcing the network to learn a more robust model that can handle missing input units.
- *Fully connected:* from the 512 classes to the specific number of classes for this task (10).

The high parameters and their values for this configuration as follow:

- *learning rate:* initialized to 0.001
- *number of epochs: 300*
- *training batch size: 128 images*
- *validation and test batch size: 32 images*
- *loss function*: the LabelSmoothingCrossEntropy from the timm package; Is the same as Negative Log Likelihood loss with label smoothing. Label smoothing increases loss when the model is correct x and decreases loss when model is incorrect x_i. This has been used with the default value of smoothing = 0.1
- *learning rate scheduler:* an utility for adjusting the learning rate of a neural network during training. This specific learning rate scheduler in this case is a step scheduler that decreases the learning rate by a factor of gamma every step_size epochs. Here step_size is 3 and gamma is 0.97: the learning rate will be multiplied by 0.97 every 3 epochs.

Efficient Net

The **first test** that has been done with Efficient Net model the model was to infer on the test set with the baseline model, to be sure that could be possibile to obtain significant performance increase after finetuning.

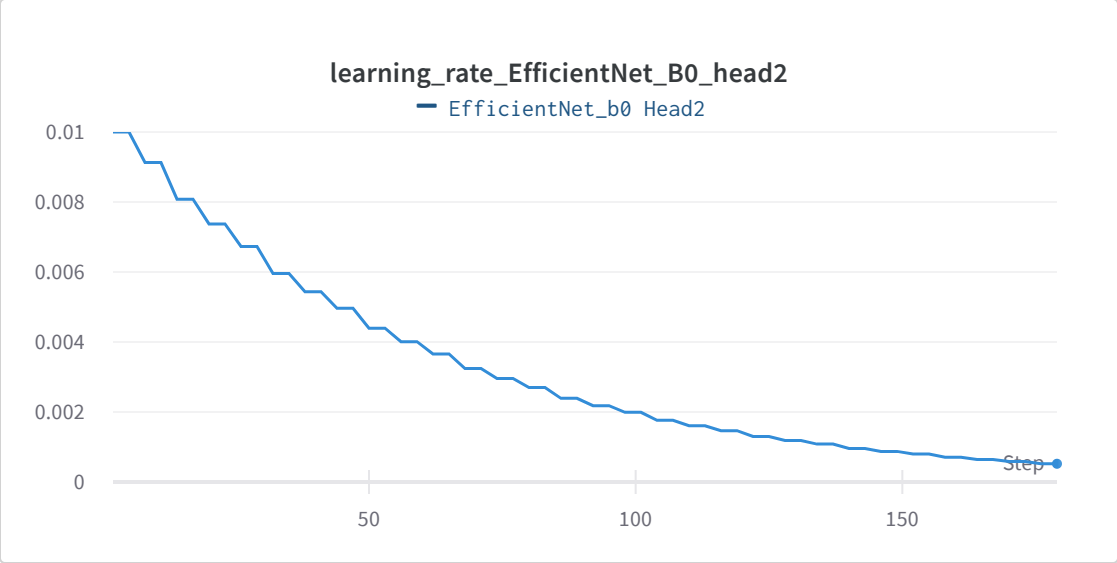
The **second test** consist of an adding of an head (type 1) exactly like the head added to the VisionTransformer model and the same high parameters.

The **third test** head (type 2) has the following layers:

- *Batch normalization:* help to stabilize the training process and improve the model's generalization ability.
- *Fully connected* from 1000 to 1024
- *ReLu*
- *Dropout* 50%
- *Fully connected* from 1024 to 512
- *ReLu*
- *Fully connected* from 512 to 256
- *Relu*
- *Dropout* 30%
- *Fully connected* from 256 to 10

The high parameters and their values for this configuration remain the same as before excluding the *learning rate* that has been initialized to value 0.01.

The following graph shows the value of the learning rate that has been decreased by the scheduler; Note: has been reported just the one for the learning rate starting from 0.1 (head type 2).



Results

Note that to calculate the metrics has been used the weighted-averaging: a method of averaging the scores for each class in a multi-class classification problem. It is similar to macro-averaging, which involves computing the scores for each class separately and then averaging them. However, in weighted averaging, different weights are assigned to each class based on the number of samples in that class. This has been done using sklearn.metrics package.

The metrics calculated are precision, recall and f1 measure; weighted-averaging has been chosen with respect to macro averaging to take into account the small changes of classes distribution in the dataset.

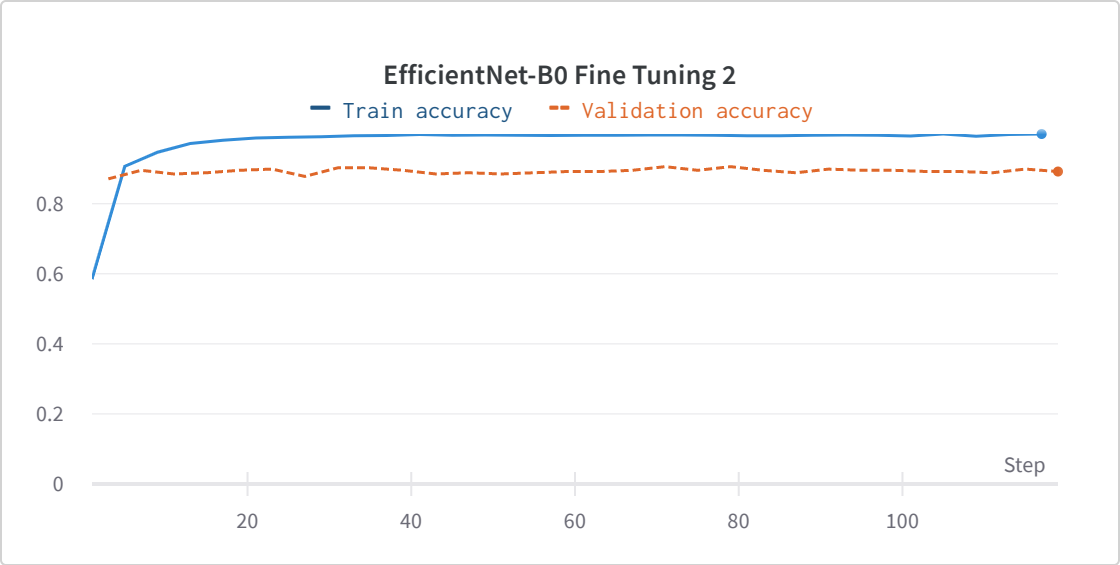
Vision Transformer Deit

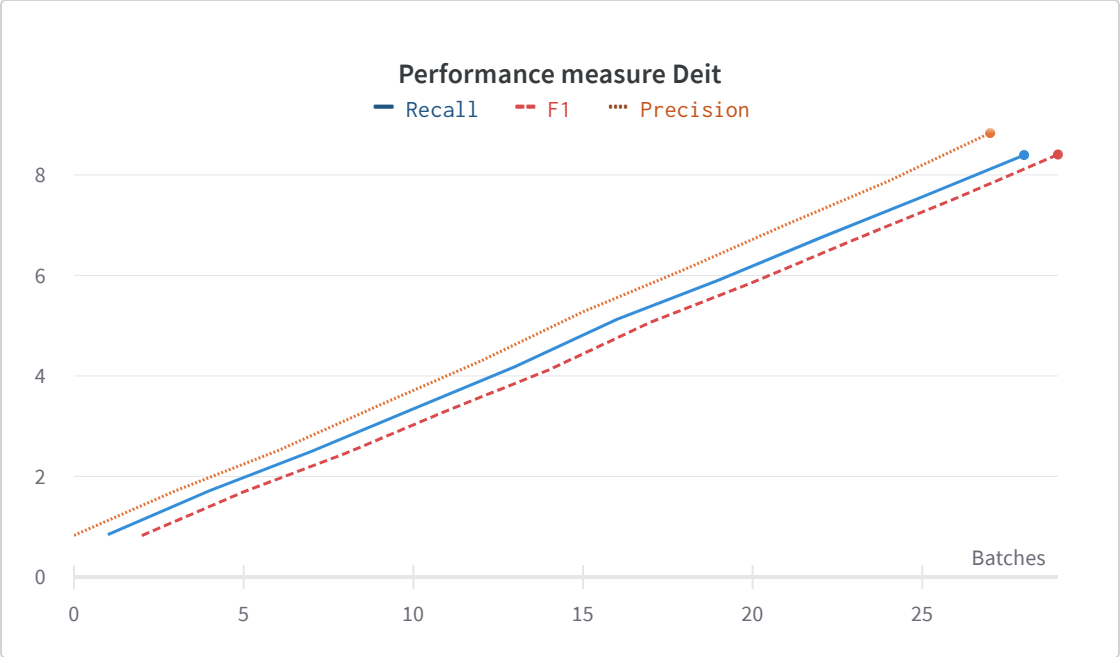
The **Deit** got the following measures of performance on the test set:

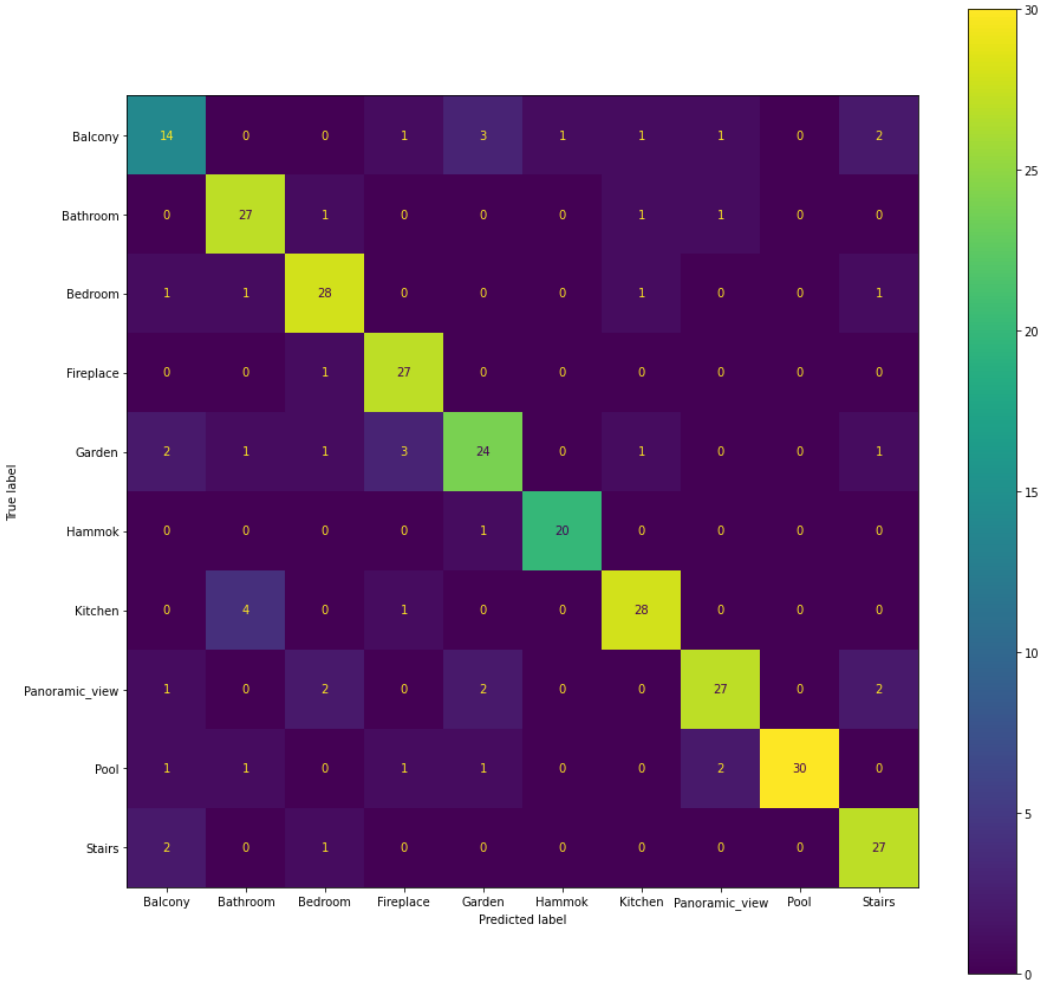
- *Test Loss:* 0.0316
- *Train Accuracy: 84*% (256 correct)

In details this are the accuracy performance for each class:

Classes	Accuracy
Balcony	61%
Bathroom	89%
Bedroom	86%
Fireplace	96%
Garden	72%
Hammok	95%
Kitchen	84%
Panoramic_view	79%
Pool	84%
Stairs	90%


EfficientNet-B0 Fine Tuning 2
Train accuracy
Validation accuracy
0.8
0.6
0.4
0.2
0
Step
20
40
60
80
100


Performance measure Deit
Recall
F1
Precision
8
6
4
2
0
Batches
0
5
10
15
20
25



Confusion Matrix Deit, a Vision Transformer family model

Efficient Net

Head type 1

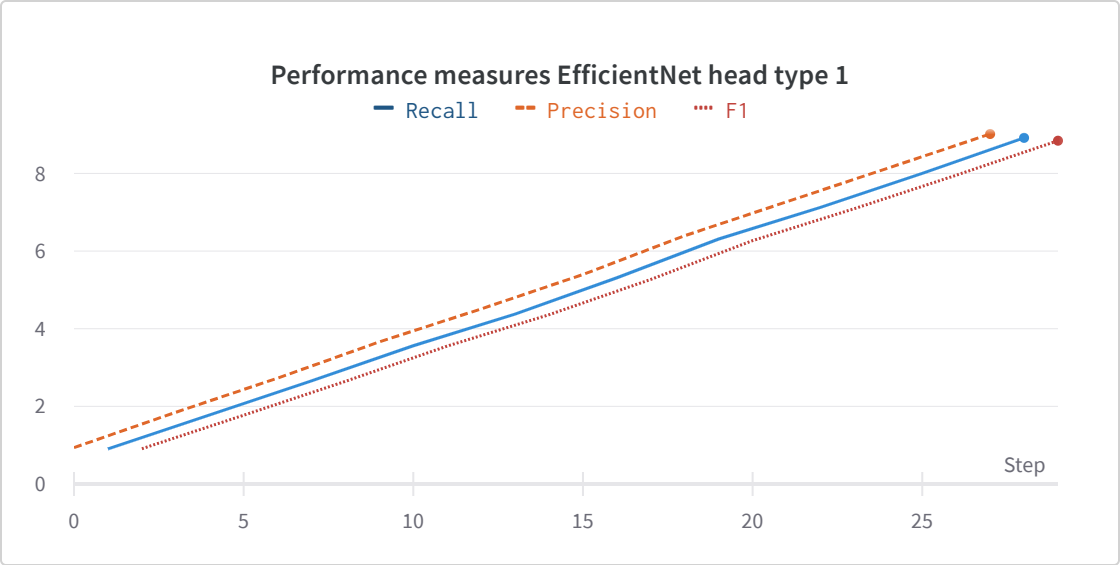
The **type 1 head** got the following measures of performance on the test set:

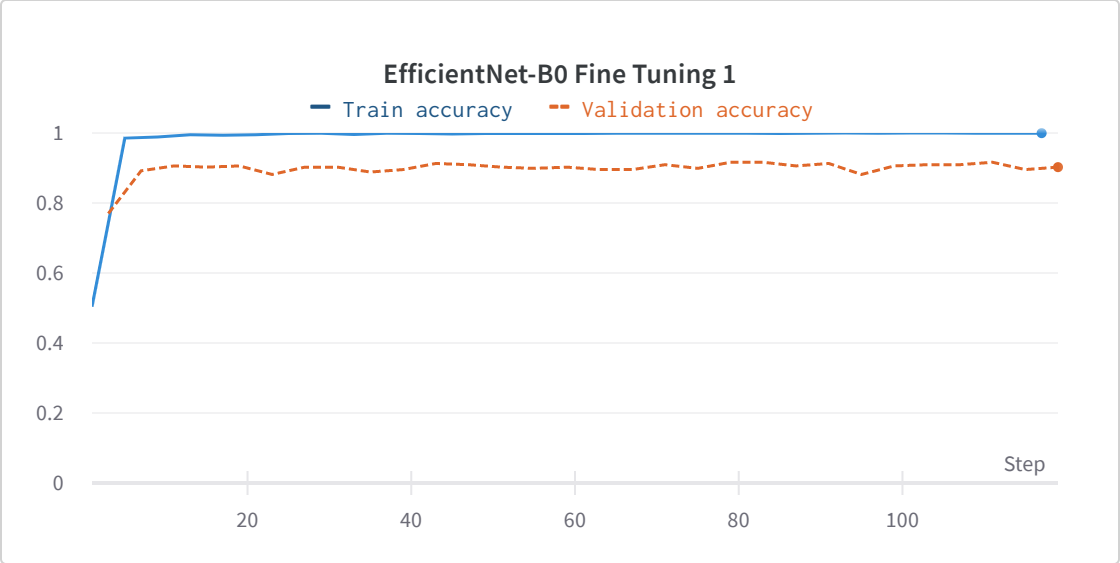
- *Test Loss:* 0.0536
- *Train Accuracy: 88*% (256 correct)

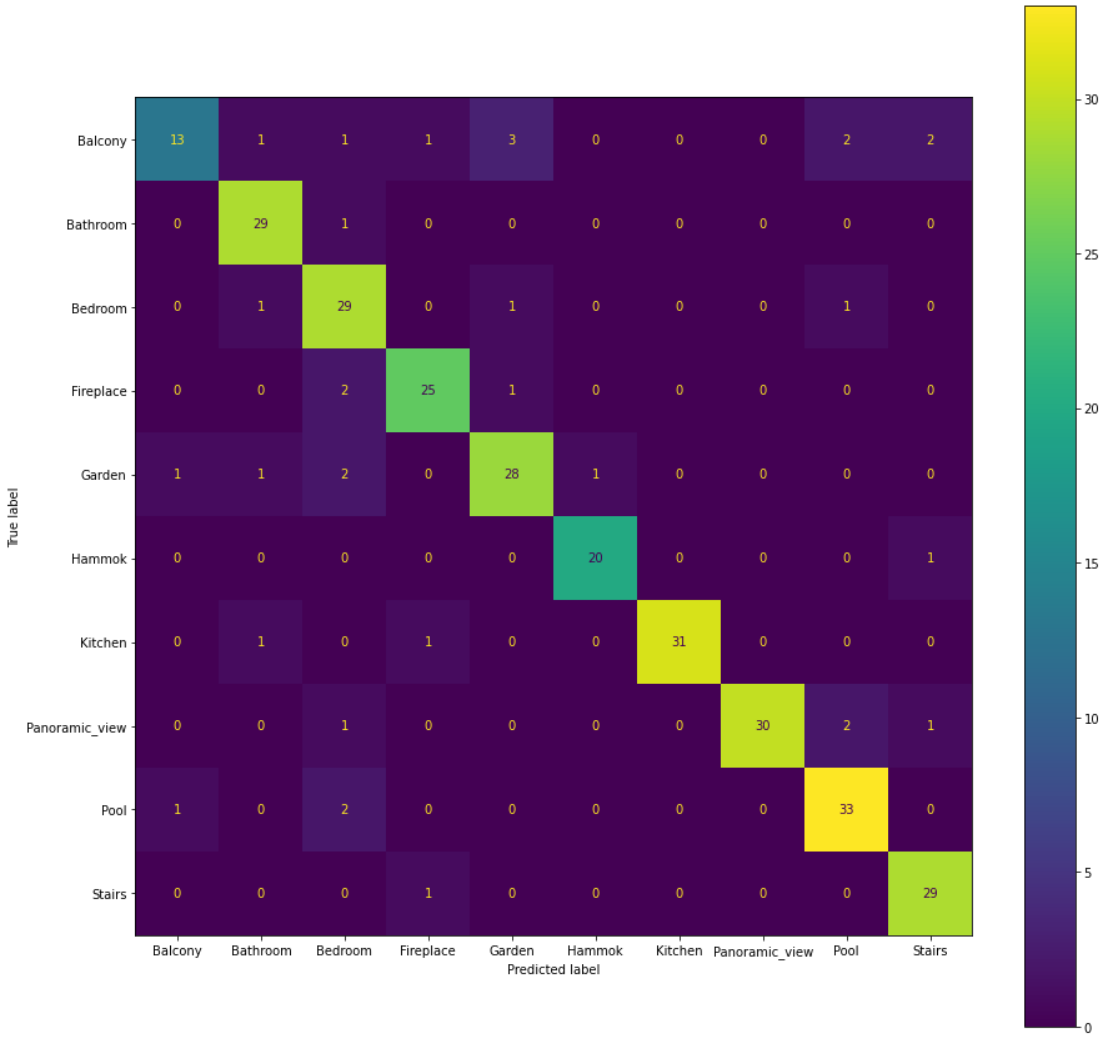
In details this are the accuracy performance for each class:

Classes	Accuracy
Balcony	55%
Bathroom	96%
Bedroom	90%

Classes	Accuracy
Fireplace	88%
Garden	83%
Hammok	95%
Kitchen	93%
Panoramic_view	88%







Confusion Matrix EfficientNet-B0 with head type 1

Head type 2

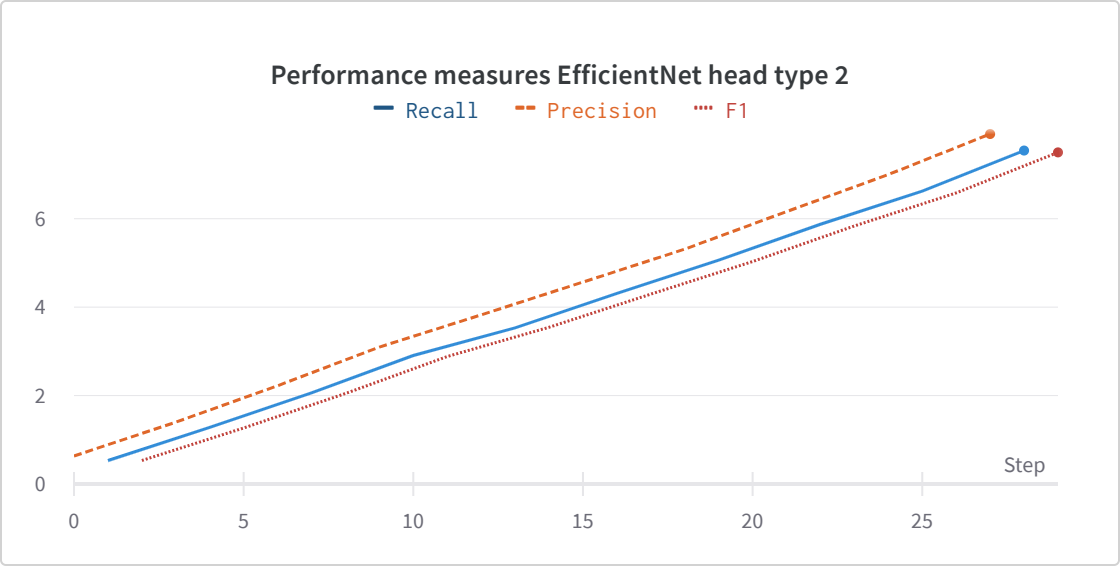
The **type 2 head test** got the following measures of performance

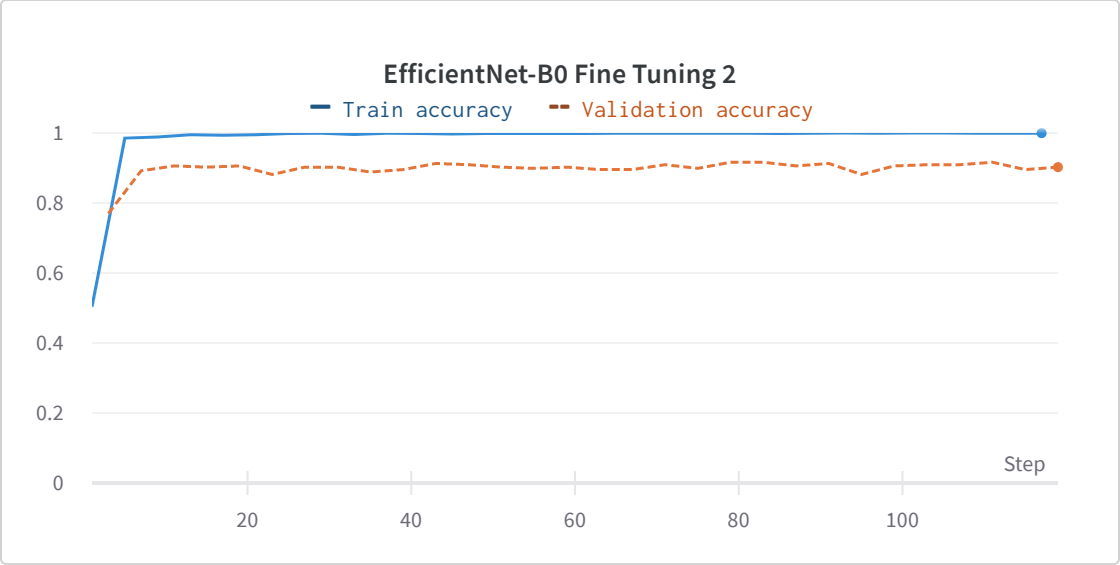
- *Test Loss:* 0.0511
- *Train Accuracy:* 73% (212 correct)

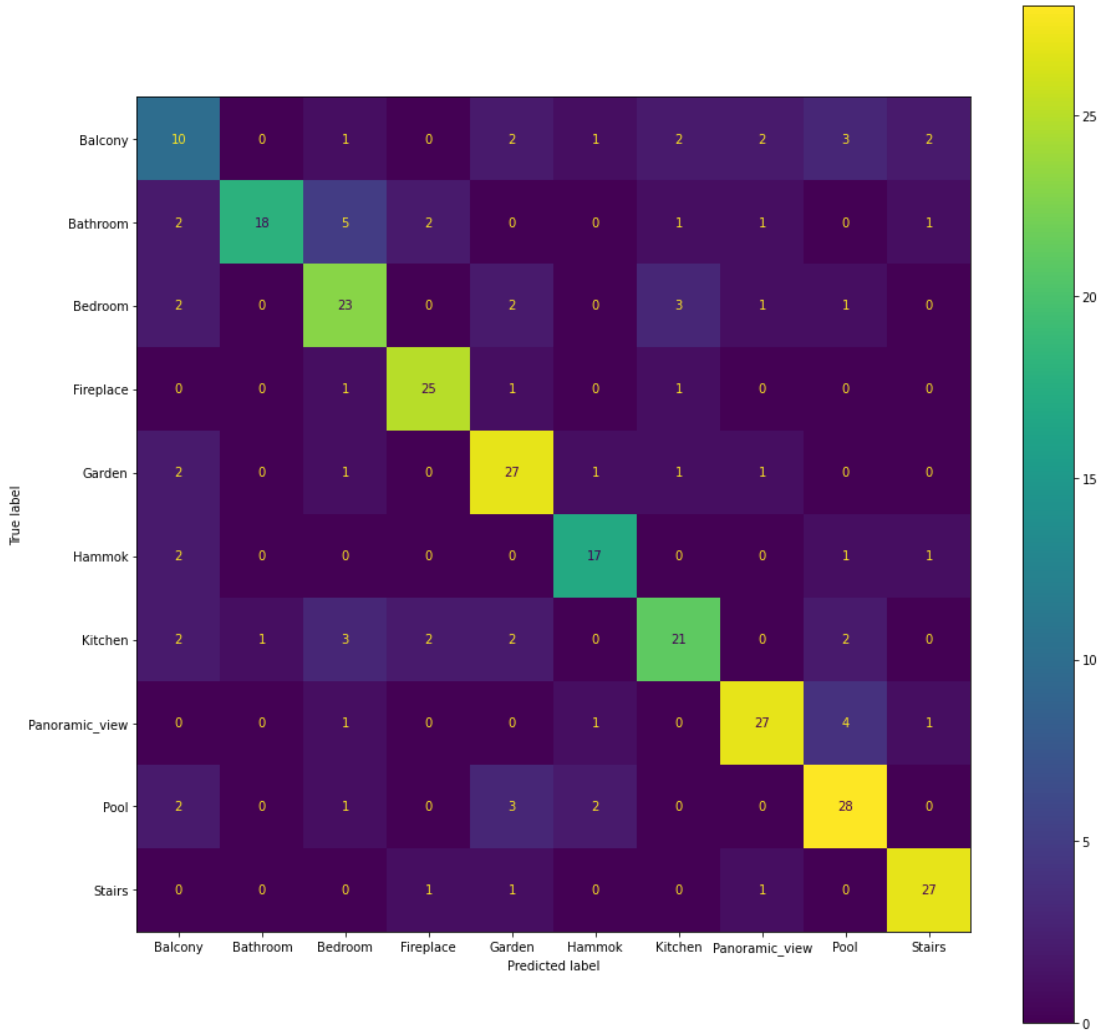
In details this are the accuracy performance for each class

Classes	Accuracy
Balcony	42%
Bathroom	57%
Bedroom	70%
Fireplace	88%

Classes	Accuracy
Garden	81%
Hammok	80%
Kitchen	63%
Panoramic_view	78%
[illegible]	[illegible]







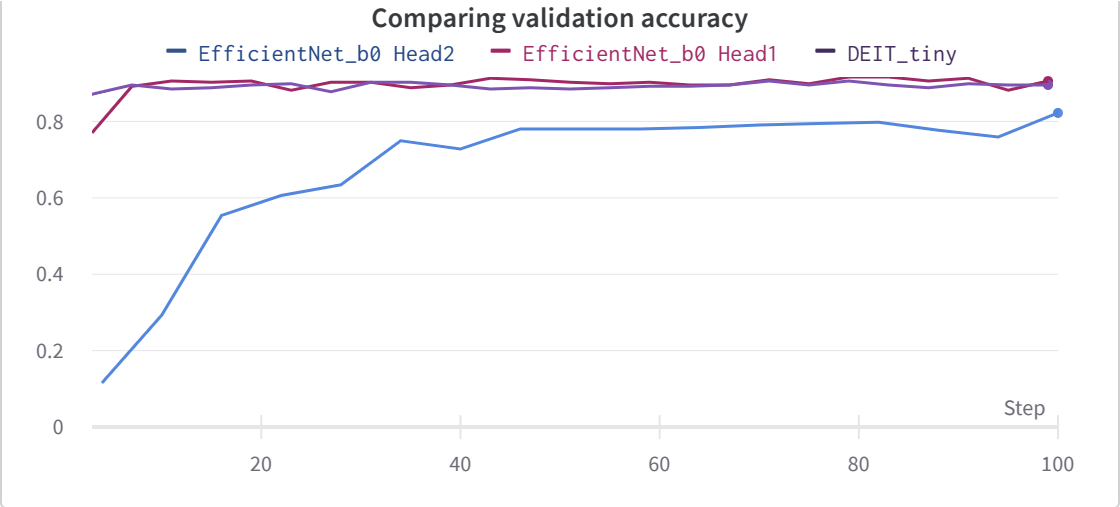
Confusion Matrix EfficientNet-B0 with head type 2

Conclusion

Taking a look to the test set accuracy reached from the three tests, the best model is **EfficientNet with head type 1** with 88%, then the VisionTransform model (Deit) got 84% and the last, EfficientNet with head type 2 got 73%.

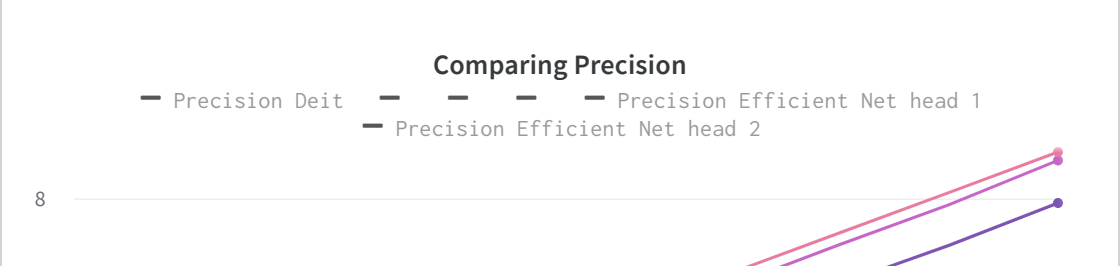
An observation on the accuracy values of the single classes in test set: it looks like the most difficult class to correct classify is the 'Balcony' class; Something that would be useful to increase performace is to give the model more sample in that class (there are slightly less samples than the other classes) or to perform other augmentation technique.

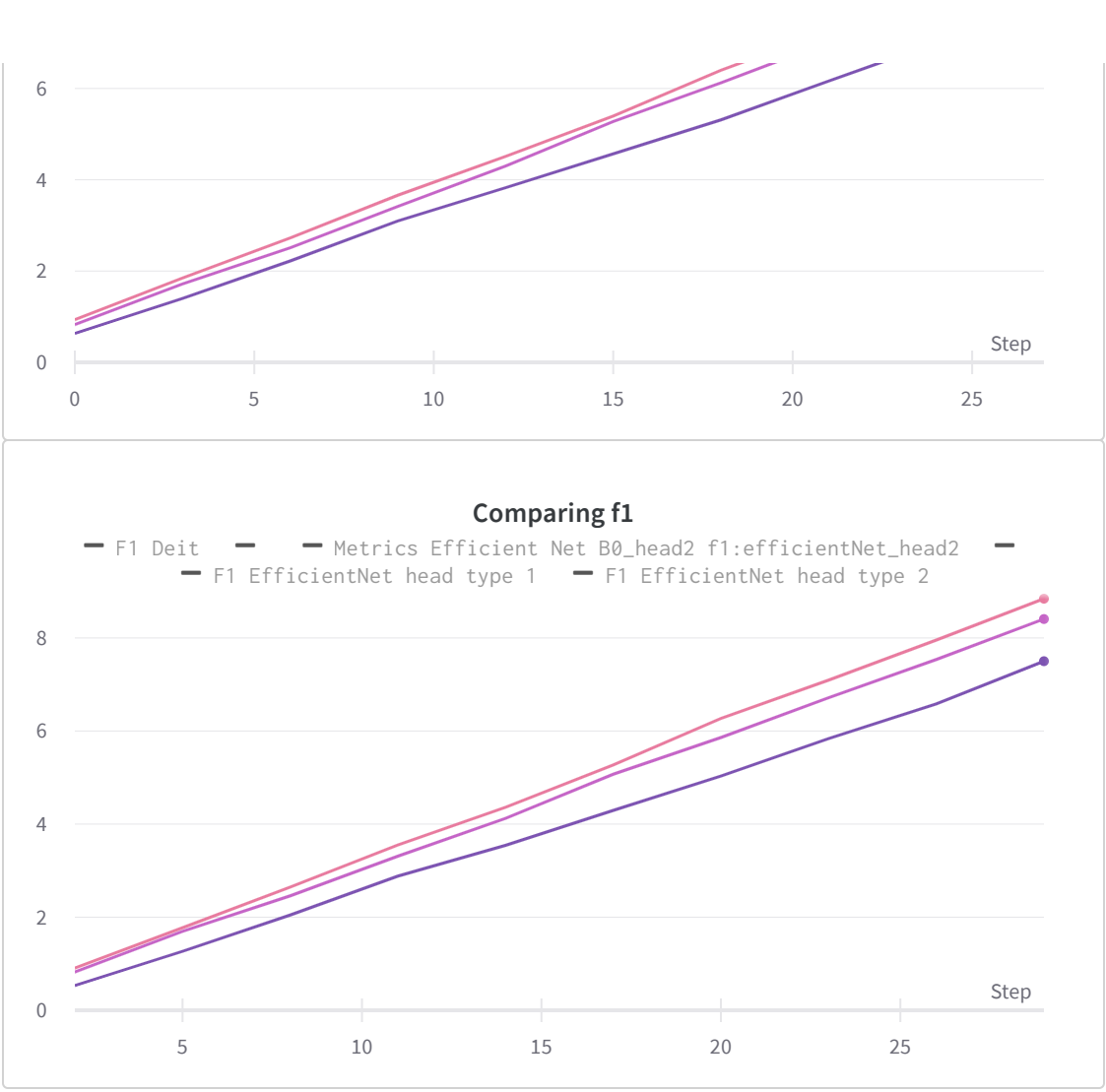
Looking at the validation set accuracy over the training phase:



From this plot is possible to see that, even if there is a small advance in test set accuracy for the **EfficientNet with head type 1** respect to the Deit model, the validation accuracy doesn't change too much and they looks very similar;

Other observations that can be made concern the other performance measures (precision,recall and f1) obtained from the models.





It is easy to observe that in every batch of the test set, in every metric, the Efficient Net Head type 1 reaches the highest values while the Efficient Net Head type 2 is down below the other two models.

In conclusion even if the head 2 for efficient net was supposed to create a better bottleneck to have the last 10 classes in the last layer, it has been the worst case in terms of performances. While, looking at the other two tests, it's worth noting that the Deit model has been freezed in the orginal architecture, so it didn't recieve an update of the weights with a consequent difficulty to learn and recognize the features; this could be a reason for which it performed less then the EfficientNet head type 1.

Created with ❤️ on Weights & Biases.

https://wandb.ai/sapienza_ml_2022_23/homework2_ML/reports/Classification-of-domestic-environments--VmlldzozMjM0NTkx