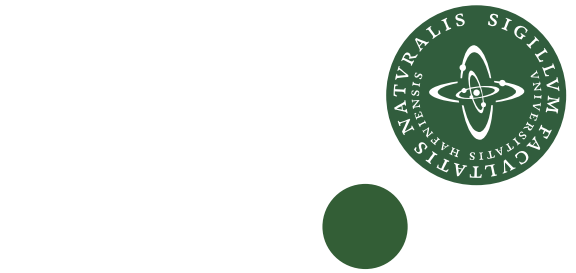


Transfer-learned Deep Autoencoders for Visual Similarity

Image Similarity in the Danish Online Housing Market

Author:
Adam Frederik Ingwersen Linnemann
gqr701
Supervisor:
Desmond Elliott

Datalogisk Institut
Københavns Universitet

January 16, 2020

Contents

Abstract

Abstract

Here's an abstract

1 Introduction

The aim of this report is to investigate the abilities of autoencoder networks to reduce dimensionality and obtain meaningful representations from pretrained neural networks using images from the danish housing market. This constitutes an unsupervised learning task, that is evaluated based on a models ability to identify similar images to a given input image. Throughout this report, key theoretical concepts relating to autoencoders and transfer learning are outlined and an extensive analysis of the appropriateness of autoencoders for this setting is conducted. The focus of this report to conduct experiments in order to ascertain whether lower-cost methodologies can be leveraged to provide solutions in predicting image similarity on images of danish homes.

1.1 The Domain Problem

Individuals looking to find a house or an apartment of any sort in Denmark will inevitably rely on primitive aggregation sites, which allow them to filter based on some particular features for homes. According to **Danmarks Statistik**[1], between 500-1000 new homes entered the market every month in the Copenhagen City Area and Frederiksberg alone in 2019 - that number is increasing.

This trend indicates that homeseekers will have to devote more time and effort in searching for a new home as they have no sophisticated tools at their disposal to do so. Homeseekers may also only periodically have the time to be alert on homes moving on and off the market and as such, they may not be aware that the perfect home has entered the market.
It may also be the case, that homeseekers are actively looking regularly, using an aggregation service but are missing out on discovering homes that do not match their filters. An example of this could be; a homeseeker has a strong preference for 1920's style apartments, enjoys large windows and open kitchens - also the homeseeker believes that the ideal apartment should be located in Frederiksberg and should be larger than 80sqm. This homeseeker may set up filters according to preferences wrt. size and location, but will have to sieve through many uninteresting prospects. Furthermore, the homeseeker may be willing to consider a 79sqm apartment in an adjacent zipcode - given that the aesthetic requirements are fulfilled.

1.2 A Possible Solution

Whilst the sale of fashion and lifestyle products has seen significant innovation with services like **Instagram** and **Pinterest**, many other industries are severely lacking behind. The technologies within image similarity put forward by e.g. **Pinterest[Jing2015]** suggest that consumers intuitively understand visual similarity for search purposes.

[1] Table: UDB010, https://rkr.statistikbank.dk/204

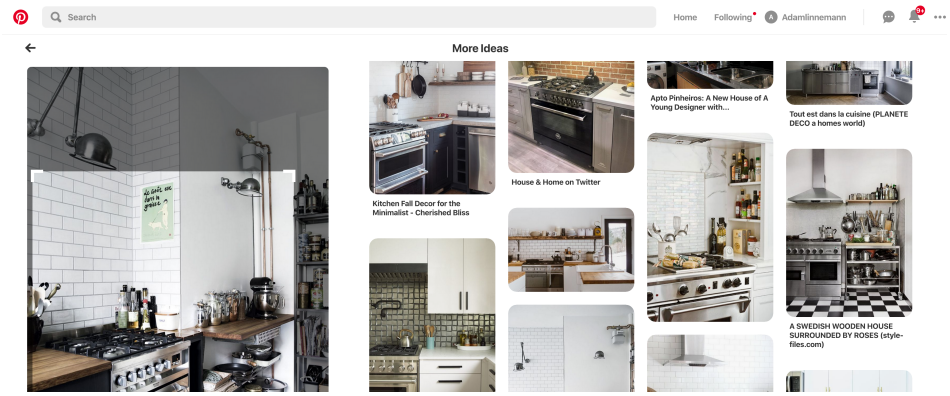


Figure 1: Pinterest Visual Search

Tools similar to those of **Pinterest** may aid homeseekers in:

1. Filter away homes that do not meet their aesthetic preferences
2. Identify homes that match their aesthetic preferences that they might not otherwise have found

An improvement to the current services surrounding the danish real estate market may be to allow homeseekers to query for similar homes to ones they like based on visual similarity. Making it easier for homeseekers to identify e.g. 5 homes that suit their preferences wrt. aesthetics could be an aid to homeseekers and allow them to focus only on candidate homes that acutally suit them. The report will address the feasibility of such visual search based on images from the danish real estate market.

2 Theory

2.1 Perceptual Hashing

A Hashing algorithm is intended to produce a *fingerprint* of a file, e.g. an image. Cryptographic hashing algorithms such as `bcrypt` and `MD5`, produces this fingerprint in a way such that the same image, with only a single pixel altered, would result in a significantly different hash - this is called the *The Avalanche Effect*. Perceptual hashing algorithms on the other hand, would in the same scenario produce two very similar hashes. Perceptual hashing algorithms are therefore quite useful in detecting e.g. copyright infringements on photographs, logos, etc. There are many different perceptual hashing algorithms available - their fingerprints can typically be compared using the *Hamming Distance*.

In addressing similarity of images in terms of, say, aesthetic, layout, etc. perceptual hashing methods are probably not very effective. This is due to the fact, that perceptual hashing methods rely on a per-pixel comparison, which will not hold for the proposed problem, as a photograph of the same kitchen may be taken from several different angles - and thus yield very dissimilar

hashes. There are more sophisticated methodologies to address the problem of finding semantic similarities in images - some of which will be discussed further below.

2.2 Neural Networks

In understanding the subsequent sections, it will prove useful to have a grasp on the fundamental ideas behind neural networks - and in particular, deep neural networks. Neural networks were first introduced as a way of describing the complexities of the human brain in 1943 by McCulloch and Pitts [**pitts**]. Neural networks today, though, are significantly more complicated than the initial proposition by McCulloch and Pitts; they have become deeper and have adopted new techniques such as backpropagation (commonly attributed to Paul John Werbos) allowing for neural networks to learn good internal representations. Furthermore, the advance in computing power, accessibility of large computing resource via cloud computing and the open-sourcing of many machine learning libraries such as Tensorflow, Keras, Caffe and Torch have allowed for rapid advancement within the field.

These factors, and many others, have contributed to neural networks succeeding in tasks such as image classification, object detection, natural machine translation etc.

2.3 Autoencoders

Over the last decade, autoencoders have been thought of as potentially useful for compression and decompression. In practice, though, autoencoders are not ideal for compressing e.g. images or sound - and have failed to replace compression algorithms such as `JPEG` and `MP3`. Autoencoding procedures are best suited for tasks that require data-specificity - i.e. an autoencoder trained to compress images of cats, typically will not do well in compressing images of cars.
Currently, autoencoders are commonly used for data-denoising, e.g. removing static noise from images. Autoencoders are also utilized for their ability to compress information into a smaller space - this can be useful in generating visualizations and assessing similarity of data.
Autoencoders come in many different varieties; denoising, deep, convolutional, etc. Each of these have their own drawbacks and merits and will be discussed further.

Within the field of machine learning, an autoencoder refers to a neural network architecture which is trained to replicate its input to its output[**Bengio2009**].

An autoencoder utilizes an encoder, that compresses its input to a lower dimension vector - and a decoder that seeks to replicate the original input from this lower dimension representation. Autoencoders are symmetrical in that the encoding layer is mimicked in the decoding layer as an inverted version of the encoding layer.

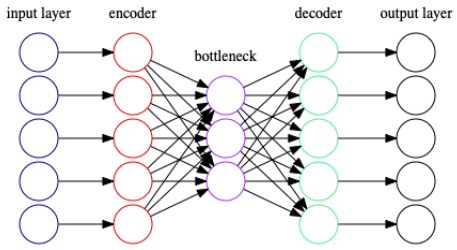


Figure 2: Quintessential autoencoder architecture

Formally, this can represented as follows: The autoencoder takes an input and maps that into a hidden representation (bottleneck) using an encoder. This happens via a deterministic mapping:

$$x \in [0,1]^d \xrightarrow{f(x)} y \in [0,1]^{d'}, \qquad f(x) = s(Wx + b)$$

where s represents a non-linear activation function; e.g. `sigmoid` or `ReLU`. From this, the decoder maps y into z, where z is an attempt at reconstructing x from y; x and z are of same shape. This occurs through a mapping that is inverted from the mapping stated above:

$$y \in [0,1]^{d'} \xrightarrow{f'(y)} z \in [0,1]^{d}, \qquad f'(x) = s(W'y + b')$$

The weights, W, W' are tuned to minimize the error rate of the reconstruction z compared to the original input, x. Two commonly used reconstruction errors are `MSE` and `Binary Cross-Entropy`. We denote the Mean Squared Error (MSE) as:

$$L(xz) = ||x - z||^2$$

Penalizing the autoencoder towards minimizing the reconstruction error is an effort towards achieving a hidden representation, y, that captures the most important features of the data. As such, the autoencoder is trained to distill the input data in a way, that it can still be reconstructed with the least amount of data fidelity loss in z.

The autoencoders hidden representation does not retain perfect information - and is as such a lossy compression of x. The encoder is optimized towards compressing the data on which it is trained, and as such, autoencoders do not generalize well.

2.4 Denoising Autoencoders

A denoising autoencoder is similar to any autoencder in form, with the distinction, that the input to a denoising autoencoder is corrupted prior to training. Also, the reconstruction error is computed by comparing the reconstruction, $\widetilde{z}$ to the non-corrupted input, x. Thus, the autoencoder is trained to denoise the corrupted input, $\widetilde{x}$. In line with notation from **??**, we can describe the denoising autoencoder as:

$$\widetilde{x} \sim q_D(\widetilde{x}|x)$$

$$\widetilde{x} \in [0,1]^d \xrightarrow{f(\widetilde{x})} \widetilde{y} \in [0,1]^{d'}, \qquad f(\widetilde{x}) = s(W\widetilde{x} + b)$$

$$\widetilde{y} \in [0,1]^{d'} \xrightarrow{f'(\widetilde{y})} \widetilde{z} \in [0,1]^{d}, \qquad f'(\widetilde{y}) = s(W'\widetilde{y} + b')$$

The `MSE` can then be computed as:

$$L(x\widetilde{z}) = ||x - \widetilde{z}||^2$$

One way of defining the corruption, q_D, is zeroing out random values[**Vincent2008**] (in black-white images this is referred to as salt-noise)

A denoising autoencoder is due to its ability to reconstruct tainted input a more robust version of the autoencoder, and is thought to be slightly more generalizable than the vanilla autoencoder.

2.5 Deeper Autoencoders

An autoencoder model is said to be deep (also referred to as stacked), when the encoder and decoder parts, respectively are constructed of multiple layers - often example fully-connected layers. Deep autoencoder architectures have shown to be proficient at mapping images to meaningful hidden representations[**Krizhevsky2010**].

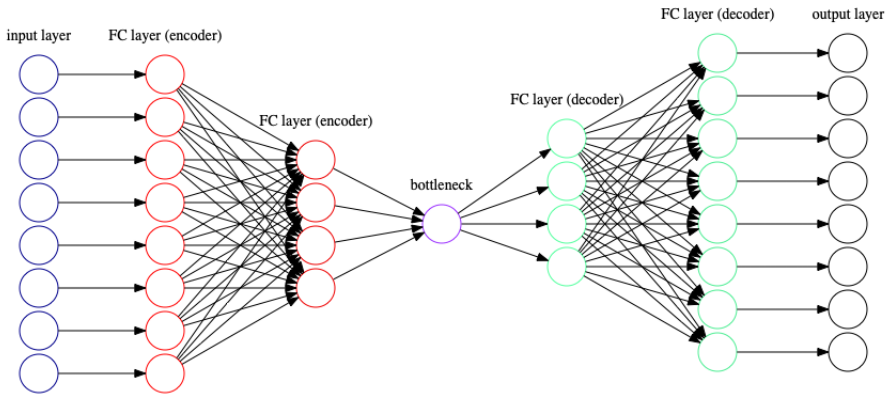


Figure 3: Example of Deep Autoencoder architechture

Deeper autoencoders may have the capacity to capture more intricate mechanisms, allowing for better performance in some tasks at the cost of higher training cost and risk of overfitting. In older litterature, it is suggested to do greedy layer-wise training of deeper autoencoder arhitechtures - as we will see in **??** better approaches have since been established.

2.6 Transfer Learning

The motivation behind transfer learning is to transfer generalizable features from one network to another - leveraging learning from different settings than the downstream task. A useful notation for describing transfer learning was provided in (Pan and Yang, 2010)[**PanYang2010**].

Consider a data domain, $\mathcal{D}$, there exists a complete feature space $\mathcal{X}$. When training models, limited data is available and we rely on a sample of that feauture space; $X = x_1, ..., x_n \in \mathcal{X}$. The marginal distribution of $\mathcal{D}$ is $P(x)$. We describe $\mathcal{D} = \{\mathcal{X}, P(X)\}$. A task, $\mathcal{T}$ has label space $\mathcal{Y}$ and a conditional probabiltiy distribution, $P(Y|X)$ learned from the training sample.
Transfer learning is concerned with mapping learned features from one domain and target to another such that information learned in the source setting $\mathcal{D}_S, \mathcal{T}_S$ can aid in identifying the conditional probability distribution $P(Y_T|X_T)$ of a different target domain, $\mathcal{D}_T$.
This notation implies that four transfer learning scenarios exist. The scenario of interest for this report is the case where $P(X_S) \neq P(X_T)$, such that the marginal distributions of the source and target domains are dissimilar, this is known as **domain adaptation**.

2.7 Similarity

In measuring similarity between images many different proposals for candidate methods surface. However, in the context of this project it is the aim to determine similarity between feature vectors obtained from e.g. autoencoders bottleneck layers. This significantly simplifies the task as these feature vectors are 1-d arrays.
A well-established and conceptually simple metric for this usecase is the cosine similarity. It is commonly used in determining word-to-word similarity and naïve recommender systems. The cosine similarity is defined as:

$$similarity(A, B) = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

Trivially, the distance between two vectors can be obtained by:

$$cdist(A, B) = 1 - similarity(A, B)$$

3 Related Works

The problem of determining similarity between data objects is a well-known and exhaustively researched area within computer vision and machine learning. A significant amount of the work for these tasks revolve around deep neural network architectures and large amounts of data - and are as such computationally expensive, and not well suited for experimentation with limited resources. This project set out to examine low-cost methodologies for obtaining

useful results. This section proposes a combination of techniques (as suggested by the literature) that may allow for fast, inexpensive training of models intended to perform reasonably well on image similarity tasks - not only for this particular domain, but perhaps many others. Literature of particular interest for this project are; autoencoders and transfer learning and will be discussed below.

3.1 Autoencoders as Feature Extractors

Several works have addressed the capabilities of autoencoders as feature extraction models. Using autoencoders for feature extraction relies on the autoencoders ability to produce a low-dimensional representation of high-dimensional data within its bottleneck layer.
In [**HintonScience2006**], **Hinton and Salakhutdinov** show that autoencoders can be trained to learn the most representative features in the bottleneck layer, allowing a decoder layer to reconstruct images from the Olivetti Face Dataset. In the article, they achieve good reconstruction and feature extraction results employing a 30-dimensional autoencoder architecture. They show that this methodology generalizes well to also other domains. They find that a 2000-500-250-125-2 autoencoder learns to distinguish between several categories of documents.
Autoencoders often learn data-specific features, and may in some scenarios not generalize well to new input - even in the same domain. **Vincent et al.**[**Vincent2008**] outline an approach for increasing robustness of autoencoder architectures by introducing the concept of Denoising Autoencoders. This technique relies on corrupting input data prior to training such that the autoencoder learns useful internal representation by effectively reconstructing the corrupted input to its original state. They conclude by experimentation that corrupting the input to autoencoders is advantageous across several benchmarks.
Supporting the idea that autoencoders are well suited as feature extractors, **Krizhevsky and Hinton**[**Krizhevsky2010**] propose the use of deep autoencoders for image retrieval. They exploit the autoencoders bottleneck layer to map imags to much more compact representations to aid in searching for images in databases. Their work is based on a dataset consisting of 80 million tiny images and propose that deep autoencoders are capable of extracting more semantic information than other techniques. Furthermore, they claim that this methodology allows for construction of better image hashes allowing for faster, more scalable than other methods.
Going further, **Pulgar et al.**[**Pulgar2018**] propose a classifier based on autoencoders and `kNN` clustering. Here, they employ an autoencoder to reduce data dimensionality allowing the `kNN`-classifier to achieve significantly better classification results. This is possible due to the distance vectors calculated by the `kNN` are more significant due to the informational, lower-dimensional feature vectors produced by the autoencoder.

3.2 Training Strategies for Autoencoders

Until quite recently, the training strategy of choice for autoencoders was greedy layer-wise pre-training. **Zhou et al.[Zhou2014]** argue that this approach has severe drawbacks in the form of suboptimal performance due to a discrepancy in learning of higher and lower layers of deeper models. In this article, they show that joint training provides better learning of the data distribution model in autoencoders. It is suggested that this may hold for both supervised and unsupervised training scenarios with autoencoders. Furthermore, they find that employing regularization in the joint training scheme is crucial to model performance. Zhou et al. concluded that joint learning methods "...learn better data models, but also more representative features for classification as compared to the layerwise method".

3.3 Transfer Learning Strategies

Many machine learning methods have shown to work well under the assumption that training and test data are drawn from the same distribution. In cases where that distribution is subject to change - e.g. new data that is dissimilar to the original data on which the model was trained, machine learning models are not capable of adapting. Thus, it is often the case, that models need to be retrained and/or reformulated to fit the new data - which is often impractical, expensive or even impossible. Furthermore, training a new model requires that new data is plentiful enough for the model to learn features related to this new underlying distribution. The idea behind transfer learning is to transfer features learned by one model into another.
Pan and Yang[PanYang2010] describe different forms of transfer learning in their survey article. They introduce the notion of inductive and transductive transfer learning settings - and elaborate on how to categorize transfer learning strategies for varying settings. In particular, `Domain Adaptation` is relevant to the work conducted in this project. In the article, Pan and Yang compares the performance of transfer-learned models - their results suggest, that it is often the case, that transfer learning can improve performance compared to other training strategies. Furthermore, their work suggest that transfer learning allow models to be trained in settings that would otherwise have been infeasible. The transferability of features is determined by what a model learns. Work by **Yosinski et al.[Yosinski2014]** and **Razavian et al.[Razavian2014]** suggest that deep neural networks learn generalizable features in the lower layers. Generic descriptors obtained from deep neural networks' lower layers appear to learn features that are akin to Sobel, Gabor and color blob filters - which are applicable across a multitude of computer vision related settings. This suggests, that the computational expenditures related to training a model for a new domain can be somewhat alleviated by exploiting features from other pretrained networks.

3.4 A modelling strategy

Given the theory established so far, and the insight from previous scholars, an unsupervised domain adaptation feature transfer using autoencoders could yield useful results. Below an outline of the modelling strategy is established. Pass image data through a pretrained network in to an output layer before the final classification layers, feature layer. Train an autoencoder to replicate the feature layer. Extract features at the bottleneck layer of the autoencoder and obtain the cosine distance matrix and construct a prediction from the cosine distance matrix by finding the index with the largest similarity coefficient to the source data.

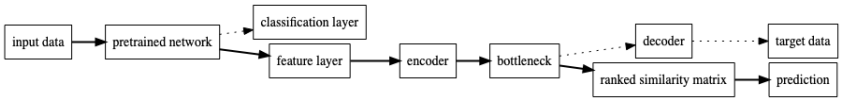


Figure 4: Outline of modeling strategy

4 Data

This section provides an overview of the data that has been utilized in training of models - and data that pretrained models have been trained on.

4.1 Danish Real Estate 2019 Data (DRE19)

This dataset was collected and catalogued by the author using webcrawlers targeting various danish real estate websites. Every one of the collecetd 21.000 images are hashed and linked to a database entry, allowing for backtracking of images to an actual sales posting.
On average an apartment/house has 21 unique images - on average 5 of these are maps, aerial, outdoor, public facilities and non-related images. Of the images that are of the property, apartment or house some are difficult to put into one single category or are simply not representative images. Therefore, in order to simplify the labeling task, a significant portion of the images were disregarded.
Describing formally which images are acceptable, and which are not, is difficult due to many one-offs and outliers.

(a) A particular object in focus **(b)** Sparsely populated class; basement

(c) Unrelated to aesthetic of home **(d)** Ambiguous class

Figure 5: Examples of irregular images that have been excluded

Examples of images that have been discarded are presented in **??**.

Prior to labeling, data was shuffled and divided into 20 different folders, see **??** Ultimately 6 distinct labels were chosen to describe the images - any images that do not fall into these categories have not been considered for this project. These labels appear to capture the majority of interior home images. In total this yielded **5458** unique, labeled images split into a 60-20-20 train-test-val split.

	kitchen	living_room	bed_room	bath_room	dining_room	entre	**total**
Train	605	569	545	515	492	484	**3310**
Test	202	190	182	172	164	161	**1071**
Validation	203	191	183	173	165	162	**1077**
Total	**1010**	**950**	**910**	**860**	**821**	**807**	**5458**

Table 1: Labeled images from various danish real-estate pages

4.2 ImageNet

`ImageNet`**[imagenet_cvpr09]**is an ongoing project, which aims to map images to features by means of crowd-sourcing the effort to label the dataset. The dataset consists of over 14M distinct images across 1000 main categories and additional subcategories/hierarchies.

Figure 6: Example of ImageNet data. Image source: [**imagenetex**]

ImageNet is a commonly used dataset for large-scale training of deep neural networks and serves as a benchmark in many computer vision publications. Milestones in computer vision reached on the ImageNet Challenge(ILSVRC) involve work such as AlexNet in 2012, VGGNet[**NIPS2012_4824**] in 2014 and ResNet[**ResNet2015**] in 2015.
Models trained on the ImageNet data may perform well in transfer learning tasks due to the large source domain, $\mathcal{D}_S$ - which should produce generalizable features. It may also be the case that such a model picks up on too general features and may not produce the features needed to distinguish more minute details in rooms such as door-frames or washing machines.

4.3 Places

The Places[**Places365**] dataset is intended for scene understanding and scene recognition tasks. It is an MIT Big Data Initiative, which has gathered and catalogued more than 10M images comprising 400+ unique scene categories. The categories include highways, pantries, fire stations, landfills, rainforests and much more.

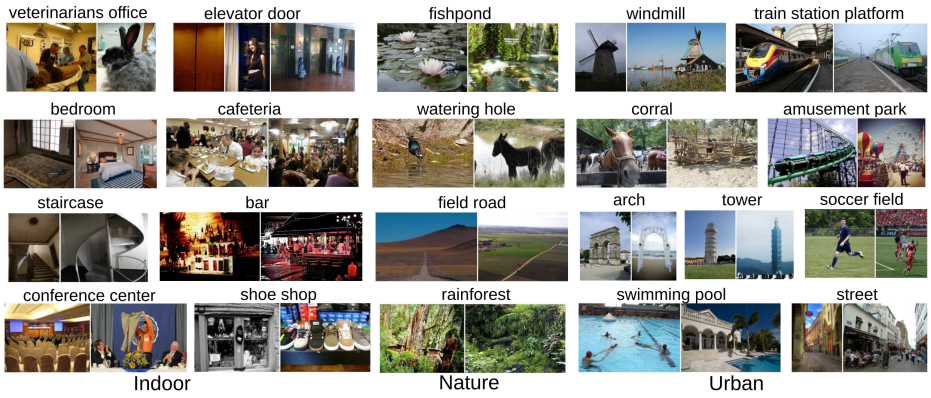


Figure 7: Places macro-classes. Image source: [**Places365**]

A model trained on places should learn significantly different features from a model trained on ImageNet due to the more focused objective of the underlying data. In particular the features it could learn from the indoor scenes may be much more adaptable to the danish housing domain.

Figure 8: Places indoor scene categories. Image source: [**Places365**]

4.4 Data Preprocessing

The constructed **DRE19** dataset consists of images of varying resolutions; all images are full color. In order for the images to be passed into pretrained models, all images were rescaled to a (224, 224, 3) resolution - which appears to be the standard format.
In determining the most model-performant image augmentations, all models were trained and evaluated with both a simple preprocessing procedure and a more advanced image preprocessing procedure.

4.4.0.1 Simple Image Augmentation

The default image augmentation scheme involves two image preprocessing steps; rescaling the image to conform with the (224, 244, 3) format and normalizing pixel values by dividing the image array by 255. The pixel normalization ensures that all values in the image array are constrained to the range $[0, 1]$. Pixel normalization is a common transformation which achieves lower combinatorial span and thus reduces memory required for model training.

4.4.0.2 ImageNet Image Augmentation Configuration

The ResNet50 image augmentation scheme operates slightly differently depending on the input type and `keras.backend` configuration - this will not be discussed further. Utilizing `keras.applications.resnet50.preprocess_input` subtracts the mean RGB-channel values of the entire `imagenet` dataset from the provided input. The RGB-channel means for imagenet are $[103.939, 116.779, 123.68]$[2].

4.4.0.3 Places365 Image Augmentation Configuration

VGG16 pretrained on `places365`[**gkallia2017keras_places365**] uses a similar strategy to **??**, but centers the RBG values around the mean RGB-channel values for the places365 dataset. The RGB-channel means for places365 are $[104.006, 116.669, 122.679]$ - implying that colors in the red and blue spectrum are slightly more prevalent in the places365 dataset compared to imagenet.

4.4.1 Comparing augmentation schemes

From **??**, it is clear, that the choice of image augmentation scheme has significant repercussions for what features a model is inclined to learn.

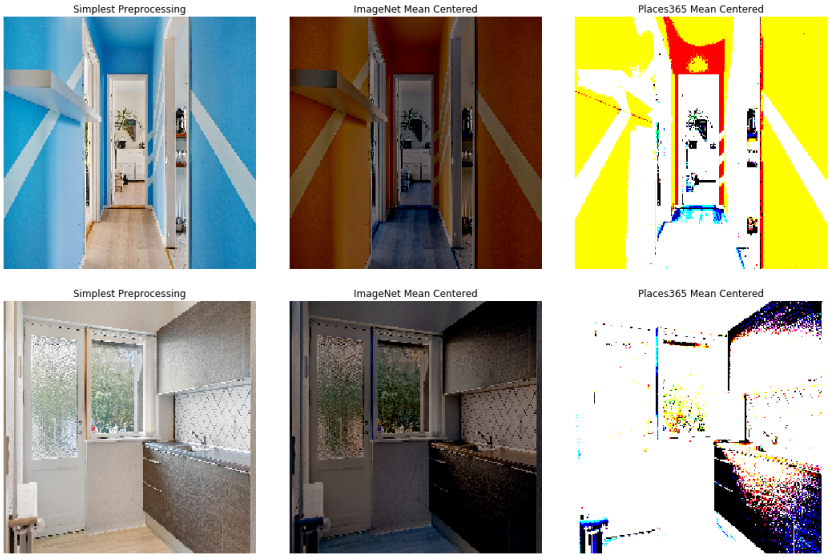


Figure 9: The 3 Preprocessing Schemes

[2] keras.applications source code

It appears, that the places365 image augmentation picks up lighting in rooms in a more pronounced way in both sample images - this could be a contributing factor for detecting commonalities in human-percieved aesthetics.

5 Experiments

Model development is conducted by determining a reasonable baseline to compare candidate models with. The baseline is constructed by first establishing a measure with which to propose a baseline performance.

5.1 Model Performance Metrics

5.1.1 Top-1 Accuracy

For each row in the cosine similarity matrix the column-index of the largest similarity value is found. This is the model prediction for most similar item to the query item, m, is denoted $\widehat{(y_m*)}$

$$\widehat{y_m*} = argmin_n(cdist_{mn})$$

The top-1 classification accuracy is then computed as the fraction of candidate predictions that are correct out of all predictions made.

5.1.2 Top-5 Accuracy

As suggested in the imagenet challenge[**ILSVRC15**] a reasonable metric for asserting accuracy in image-classification tasks is to consider if any of the top-5 predictions match the source image class. This metric is arguably slightly generous for this particular task, seeing that there are only 6 classes as compared to the 1000 classes in the imagenet challenge, but may well reveal if models are underperforming in particular settings.

5.1.3 Training Optimizer

Due to time and resource constraints, experimenting with different training optimizers was not feasible, thus a choice was made to use the `ADADELTA`[**Zeiler2012**] optimizer, a variation on gradient descent. Throughout experimentation, the choice of training optimizer has been held constant, thus this factor will not be reported below. ADADELTA was chosen due to its configuration simplicity as it has an adaptive learning rate - simplifying hyperparameter search. Another optimizer, `ADAM`, was also considered for these experiments, as it adopts an exponentially decaying learning rate adaptation. `ADAM` may have yielded faster convergence in deeper model choices[**Kingma2015**], but was discarded due to its more complex nature.

5.1.4 Model Loss

Model loss is kept constant as well. Mean Squared Error (MSE), see **??**, is chosen due to its wide usage for autoencoder models throughout the literature. Loss-metrics are not reported in the following sections, as it varies significantly with choice of image preprocessing scheme and the introduction of noise in autoencoders. All models stated in the below section **??**

5.2 Baseline Models

This section presents the baseline model candidates and discusses the performance of each baseline candidate in turn.

5.2.1 ResNet50 Forward Pass to Maxpooling Layer

This model candidate is obtained by extracting the global maxpooling feature layer(`max_pooling`) from ResNet50 pretrained on imagenet and passing input through to that layer. This was done for all image preprocessing schemes:

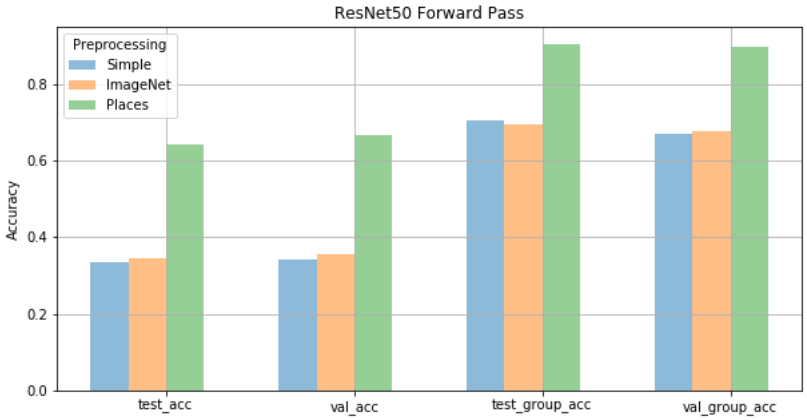


Figure 10: Baseline performance of ResNet50

5.2.2 VGG16 Forward Pass to Fully Connected Layer

A baseline candidate model was obtained by extracting the last fully connected(`fc_1`) layer from VGG16 pretrained on places and passing input to through to that layer:

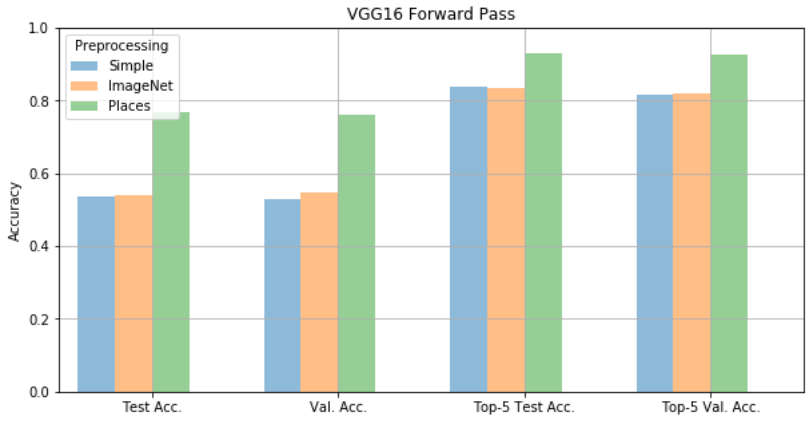


Figure 11: Baseline performance of VGG16

The most performant of the baseline candidate is the VGG16 Forward Pass to `fc_1`, achieving the significantly higher accuracy in all configurations than ResNet50 MaxPool Forward Pass.

Model	Preprocessing	Test Acc.	Val. Acc.	Top-5 Test Acc.	Top-5 Val. Acc.
ResNet50 (maxpool)	places	0.6438	0.666	0.904	0.898
VGG16 (fc_1)	places	**0.769**	0.760	**0.929**	0.927

It holds that for all candidate baseline models, the Places image preprocessing boosts accuracy by a substantial amount regardless of pretraining data. One possible explanation for this is that the domain data of Places is significantly more in line with the target domain, that is the case for ImageNet, suggesting that subtracting the mean Places RGB-values in the target dataset provide a better input image representation for DRE19**??**.

6 Results

6.1 Candidate Models

Given the baseline performance of the pretrained VGG16 with places image preprocessing, a set of candidate models are proposed with the aim of 1) improving accuracy and 2) reducing dimensionality of feature vectors allowing for faster computation of similarity matrix. The results from **??** would suggest, that out-of-the-box performance for autoencoder arhcitectures may perform better in producing well-defined encodings from VGG16's fc_1-layer using Places image preprocessing.

6.1.0.1 Pretrained Model with Denoising Autoencoder Several configurations of pretrained models with denoising autoencoders have been attempted throughout experimentation. Generally autoencoders in this setting tend to

perform better when the first encoder layer is of similar dimensions to the input feature vector. Broader bottlenecks also tend to retain more information during training.

Pretrained	AE Layers	Preprocessing	Test Acc.	Val. Acc.	Top-5 Test Acc.	Top-5 Val. Acc.
ResNet50 (max_pool)	2048, 512	Simple	0.337	0.338	0.693	0.697
ResNet50 (max_pool)	2048, 512	ImageNet	0.341	0.328	0.704	0.671
ResNet50 (max_pool)	2048, 512	Places	0.650	0.661	**0.927**	0.911
VGG16 (fc_1)	4096, 1028	Simple	0.528	0.529	0.818	0.820
VGG16 (fc_1)	4096, 1028	Places	**0.778**	**0.783**	0.915	**0.928**

Initial experiments with denoising autoencoders, noise-factor = 0.3

It would appear, that models trained on input preprocessed with Simple or ImageNet prepocessing schemes achieve significantly lower accuracies compared to models trained on Places preprocessed data. Furthermore, models with Places pretrained weights perform better across the board, all else equal.

6.1.0.2 Pretrained Model with Deep Autoencoder Initial experiments with deep autoencoders show no significant boosts to performance in using either ResNet50 or VGG16 compared to the denoising counterparts.

Pretrained	AE Layers	Preprocessing	Test Acc.	Val. Acc.	Top-5 Test Acc.	Top-5 Val. Acc.
ResNet50	2048, 1028, 512	Simple	0.324	0.345	0.698	0.664
ResNet50	2048, 1028, 512	Places	0.659	0.654	**0.926**	0.929
VGG16	4096, 2048, 1028	Simple	0.541	0.516	0.830	0.820
VGG16	4096, 2048, 1028	Places	**0.775**	**0.764**	0.914	**0.934**
VGG16	6000, 4000, 3000, 2000, 800	Places	0.745	0.749	0.9	0.915

Initial experiments with deep autoencoders

Deep autoencoders can be said to perform on-par with narrower denoising autoencoders. They do however take longer to converge; see **??**.

6.1.0.3 Pretrained Model with Deep Denoising Autoencoder Adding noise to deep autoencoders yields slightly worse results in terms of accuracy compared to any of the previous approaches.

Pretrained	Architecture	Noise	Preprocessing	Test Acc.	Val. Acc.	Top-5 Test Acc.	Top-5 Val. Acc.
VGG16	4096, 2048, 1028	0.3	Places	**0.764**	**0.753**	**0.919**	**0.913**
VGG16	4096, 2048, 1028	0.5	Places	0.74	0.735	0.908	0.904

More results for DDAE's with pretrained layer alternation can be found in **??**.

6.2 Model Evaluation

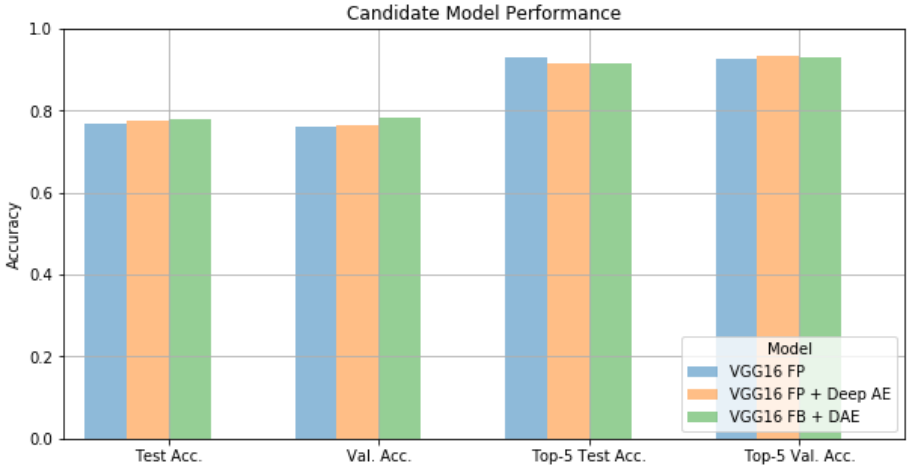


Figure 12: Candidate Model Accuracies v. Baseline

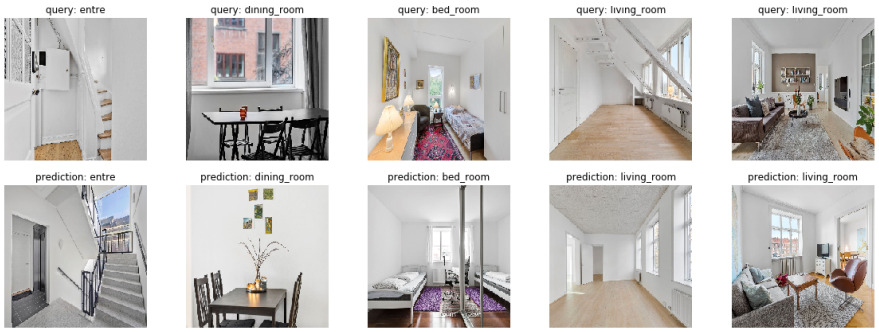
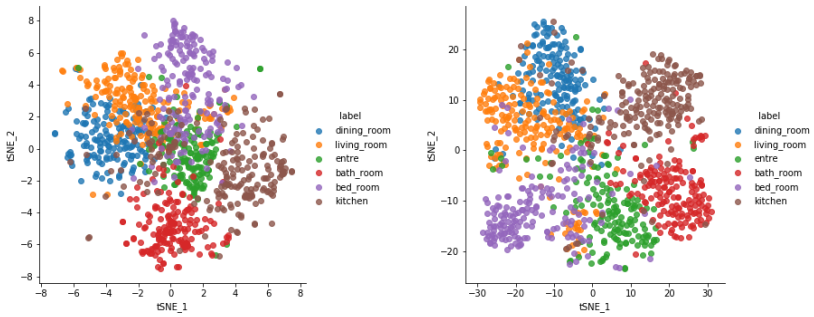


Figure 13: Predictions (test): VGG16 FP

Figure 14: Predictions (test): VGG16 + DAE

In order to get and understanding of the models it useful to map the feature vectors of a model into a 2-dimensional space using `t-SNE`[**Chu2017**][**HintonScience2006**]. As suggested in [**wattenberg2016how**], several combinations of hyperparameters were tested on train, test, and validation data; see **??**. For all configurations, the DAE produces a lower KL-divergence, indicating that the DAE has learned a representation better at separating images that would otherwise be clumped together.



(a) t-SNE: VGG16 Forward Pass, p=50 **(b)** t-SNE: VGG16 + Autoencoder, p=50

Figure 15: t-SNE feature space comparison

7 Future Work

This section proposes possible improvements to models, data and further experiments to conduct.

7.1 Use unlabeled data for training

8 Conclusion

Appendix

Appendix A: Data Labeling

In order to reduce the number of images to label, all images that could be easily determined to be primarily white/black images were discarded using mean-pixel values computations. See `datatools/utils/classify_greyscale.py` for implementation.

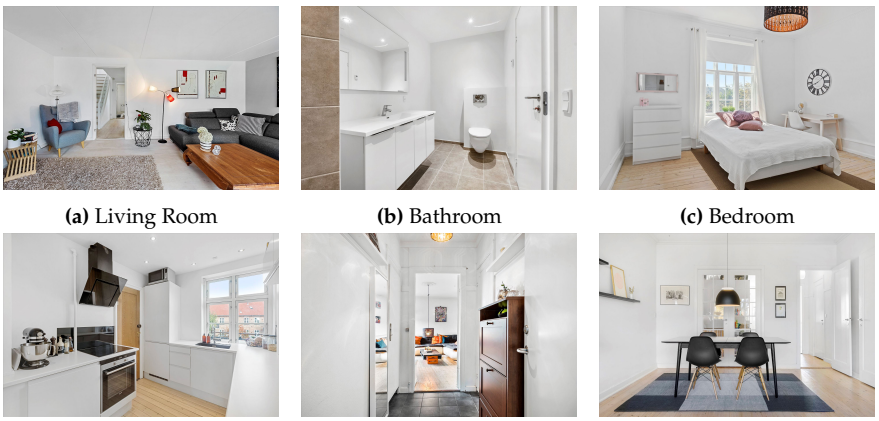
Opening a folder with 22.000 images in an attempt to label them is taxing on hardware, and in order to alleviate the issue of a crashing file-manager GUI, data was split randomly into 20 subfolders using a shell script:

```
for i in {1..20}
do
    echo "Moving $1 images into $i"
    ls -Q images/ | head -$1 | xargs -i mv images/{} segments/$i
done
```

Finally the labeled data was split into train-test-val split using `datatolls/utils/split_data.py`.

Appendix B: DRE19 Examples

Presented below, in **??**, representative examples of the 6 different classes in the DRE19 dataset.

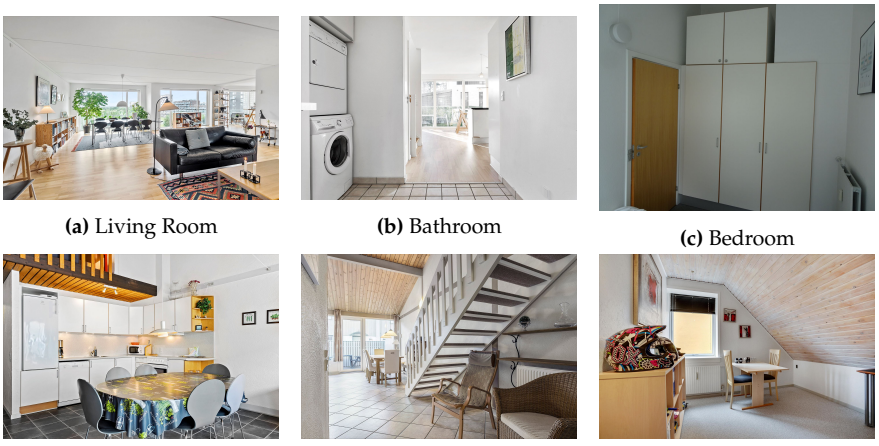


(a) Living Room **(b)** Bathroom **(c)** Bedroom

(d) Kitchen **(e)** Entre **(f)** Dining Room

Figure 16: Room Archetypes

Less representative images have also been added to every class. For these images it holds, that labeling by hand was not as straightforward as in the examples above. This can often be attributed to the photo angle being odd or that the image contents spans multiple room classes. It has been the authors labeling strategy to try and infer the room type based on the images most prevalent features.



(a) Living Room **(b)** Bathroom **(c)** Bedroom

(d) Kitchen **(e)** Entre **(f)** Dining Room

Figure 17: Difficult-to-label rooms

Appendix C: DDAEs with pretrained layer variation

In experimenting with deeper architectures and noise, no models appear to outperform the deep autoencoders or denoising autoencoders. Many variations were attempted - shown below are some examples of results and configurations.

Pretrained	Architecture	Noise	Preprocessing	Test Acc.	Val. Acc.	Top-5 Test Acc.	Top-5 Val. Acc.
VGG16(flatten)	6272, 3136, 1568	0.3	Places	0.68	0.654	0.891	0.896
VGG16(flatten)	25088, 12544, 6272, 3136	0.3	Places	0.163	0.172	0.568	0.605
VGG16(fc_2)	4096, 2048, 1028	0.5	Places	0.762	0.734	0.935	0.932
VGG16(fc_2)	5000, 3000, 1600	0.3	Places	0.764	0.723	0.934	0.933

Table 2: Experimentation with DDAEs

Appendix D: Training Convergence of Autoencoders

Appendix E: t-SNE testing

In determining the appropriateness of stating that the t-SNE visualizations imply better data distribution learning for VGG+DAE compared to VGG alone, a thorough grid assessment was performed. This can be replicated by running `model_catalogue/w_image_augmentation/tsne_comparison.py` found on the project repository. Running this script also outputs a plot per comparison - similar to **??**

model	vgg+dae	vgg	vgg+dae	vgg	vgg+dae	vgg
data	train	train	train	train	train	train
perplexity	5	5	20	20	50	50
KL-divergence	1.85	3.22	1.80	2.99	1.56	2.55

Bibliography