Cat Coat Colour Clustering: Searching for Phenotypes with Unsupervised Learning

Jacob Suchardt
LT2326 H24 Maskininlärning för statistisk datalingvistik: avancerad
Department of Philosophy, Linguistics, and Theory of Science, University of Gothenburg
gussucju@student.gu.se

Abstract

This project attempts to cluster images based on the colours exhibited by a specific object that occurs across items. The approach includes the computer vision tasks of object detection and image segmentation, and deals with feature extraction and subsequent dimensionality reduction. Lastly, performance, limitations, alternatives, and possible improvements of the implemented methods and models are discussed.

1. Introduction

The primary aim is to take exploratory steps towards unsupervised machine learning to gain more familiarity with methods which may be applicable to problems without labelled data. The focus therein is refined to working on image data and extracting colour based features from image regions which belong to the object category 'cat'. Due to the scope of this paper there is no further segmentation within the object area. Instead the entire object area is taken as the animal's coat, for which a well-defined taxonomy has been described (for an accessible and comprehensive overview see Hartwell, n.d.). Groups as created by centroid clustering ought to then be compared to this taxonomy based on human judgement to gleam insights into the effect of training data on the clustering algorithm and how these reflect back on design choices made during data preparation and processing.

Thematically related work has been done in the direction of categorising cat breeds with supervised methods (e.g. ma7555, 2019; beleidy, 2017) and on clustering images by colour in general. Firstly, this project is not concerned with breeds but instead with coat colours and patterns, which are found across breeds and independent of shape-based features that may characterise the former. Work towards the second direction lacks the input restriction and requirements proposed here: It uses the entirety of an input image, sometimes because of the nature of the subject, e.g. landscape photography, or simply by having no background, e.g. stock photos (Rafidinarivo, 2022). For this project however, a specific, and often times minor, region of the input image must be isolated prior to feature extraction and clustering. Another factor is that due to the natural restriction of the object category to two pigments (eumelanin 'black', phaemelanin 'red'), the lack thereof (white), and their variations (dilutes, intensities) allows for very fine grained distinctions to be made, not to mention the possibility of detailed coat patterns (tabby, point) and pigment combinations (tortoiseshell, calico). This amount of detail confined to the colour spectrum created by natural pigment restrictions contrasts with work that clusters inputs whose main objects across images belong to different categories and thus may diverge more strongly across the colour spectrum (Devashree, 2022). Additionally, working on photographic images also includes a lot of noise in colour quality, a challenge that is not present when working with cell-shaded illustrations for example (Axel et al., 2021).

2. Data resources

The source dataset is originally from Zhang et al. (2008) and replicated on the Internet Archive (bnewbold, 2017), from which it is available and was accessed as 'Cat Dataset' on Kaggle (Crawford, 2017). It consists of a total of 9997 photos in .jpg format, each of which depicts at least one cat. For every .jpg photo, the folders also contain a corresponding .cat file that denotes information about the position of various feline facial features within the image. The dataset was chosen over other available options as due to its original purpose, cat head detection, it comprises images where the cat-object is generally easily visible and of sufficient size to extract colours from. Position, size, lighting, as well as image size and resolution were still varied enough to represent realistic and not overly simplified input.

For this project only the .jpg files in the folders CAT_00 (n=1706) and CAT_01 (n=1618) were considered. Two subsets of CAT_00, namely CAT_00_mixed/'mixed' (n=431) and CAT_00_solid/'solid' (n=96) were created manually to function as approximation for gold data sets for fitting models (see 3.1; set member lists can be found in gold_mixed_file_refs.txt and gold_solid_file_refs.txt). The solid set is a sub-

set of the mixed set and restricted to cats with the solid fur type (uniform colour/one pigment).

3. Methods

The initial goal of input preparation was to identify and locate the cat-object within a given image and separate its ROI from the background. Following this, a colour-sensitive feature representation was created which is required to be comparable with the other inputs representations to enable clustering. Below, this process is divided into three areas below: ROI isolation (3.1.), feature extraction (3.2.), and clustering (3.3.). As a preparatory processing step, all images were resized to a quarter of the average image size in their origin folder (see 5.).

3.1. Region of interest isolation

To extract a ROI, first the relevant object – a cat in this case – must be identified and located within the image. Next, the non-relevant background around this object has to be removed to prevent it from introducing noise during further processing.

To accomplish the first step, object detection was performed on each input image using OpenCV's detect_common_objects function with the YOLOv3 model (Bochkovskiy et al., 2020; Arunponnusamy, 2018). This function yields a list of object labels, bounding boxes, and confidence scores; one such list per found object within an image. Any list pertaining to non-cat objects was dropped. If either zero or more than one object in an image received the label 'cat' the image was excluded from further analysis.

For the second step, image segmentation, only a two-way distinction between semantic foreground and background is required, making the GrabCut algorithm a suitable candidate (Prost, 2022). Inputs consisted of the images remaining after object detection and their single associated bounding box. Bounding boxes denote the coordinates for a rectangle enclosing the cat-object within the image. Each such image-bounding box pair was passed through the GrabCut algorithm implementation of OpenCV (Preet, 2024) with two iterations. Ideally, GrabCut replaces any background area with transparent pixels (=RGB (0, 0, 0)) and leaves the ROI intact. Based on manual visual evaluation of the success of the GrabCut algorithm on a given image (fig.1), the data subset CAT_00_mixed, and from it another subset, CAT_00_solid, were created.

3.2. Feature extraction

The second major step was to reduce and unify the colour palette across inputs. As colour is the sole feature for clustering, reducing variety across inputs to a common range is necessary to render them comparable and to substantially reduce the highly dimensional nature of full colour photography. This 'colour compression' was achieved with a KMeans clustering model that groups similar colours together and replaces them with a centroid. It was fitted on 10k random samples from the concatenated image matrix of the chosen fitting dataset to unify colours across images and not just within single images, as would be common for colour quantisation (Layton et al., n.d.). The number of colour centroidgs to compress to was set to 32 (not including transparent pixels).

From the resulting compressed ROIs, a colour profile was extracted for each array. This profile consisted of a dictionary mapping each of the model's colour centroids to its relative frequency in the given array, including frequencies of zero. During this step, transparent pixels

Figure 1. An image with subjectively 'sufficient' GrabCut success (l.), and an image (r.) where due to the remaining non-ROI area's size, image segmentation was not deemed sufficient enough for the hand-picked data subsets.

were excluded both from being a centroid key and from contributing to the array size computation during frequencies normalisation. Lastly, each profile was sorted by the centroids' RGB values and the keys removed to yield a feature vector whose dimensions corresponded to the same colour centroid across input images. This step also abstracts from the original ROI shape, preventing possible non colour-based effects downstream.

3.3. Clustering

The vectorised colour profiles were the basis for a new KMeans centroid clustering model which should predict clusters that group input images based on the cats' coat colours (mohit, 2023). The number of clusters to predict was set to a minimum of k=4 for the control condition, and increased empirically for test conditions according to the subjective cluster quality and guidelines to aim for similar cluster sizes, when applicable. To facilitate plotting, the test items for cluster predictions were limited as needed to the amount necessary to form representative clusters. The data sets for fitting and predicting were alternated to examine effects of preprocessing, training data quality and expected task difficulty.

For this paper, clustering will be discussed using five configurations (table 1). The first two are intended to test whether a simplified version (uniform 'solid' coats only) of the task at hand is possible with the models and features implemented here. While the first configuration predicts and plots clusters based purely on the non-compressed ROI arrays, the second one uses the vectorised colour profiles from each compressed ROI. This pair illustrates the effect of the implemented feature representation. Configuration 3 uses the mixed set to explore if and how the addition of non-solid cats affects clustering. Configurations 4 and 5 test the transfer of models trained on the solid and mixed dataset to the unseen CAT_01 dataset, which has not been filtered based on GrabCut success either. For visual inspection, plots display images' extracted ROI arrays prior to colour compression in truncated clusters.

	Colour compression	Colour clustering		Fig.
	Fit (k=32)	Fit	Predict	
1	ROIs, solid	ROIs, solid	ROIs, solid	4.1
2	ROIs, solid	Embedddings, solid	Embedddings, solid,	4.2
3	ROIs, mixed	Embeddings, mixed	Embedddings, mixed	4.3
4	ROIs, solid	Embeddings, solid	Embeddings, CAT_01	4.4
5	ROIs, mixed	Embeddings, CAT_01	Embeddings, CAT_01	4.5

Table 1. Fitting and clustering configurations ('solid'=CAT_00_solid; 'mixed'=CAT_00_mixed) with respective figure references.

4. Results

Results of the described methods are briefly detailed here and discussed in section 5.

4.1. Object detection

Success of object detection was measured by the relative return rate of input images together with two categories to quantify causes for exclusion: object detection failure (ODF) and multiple object detection (MOD). YOLOv3, the basis for project development, exhibited high success rates of 92,85% for CAT_00 and 92.4% for CAT_01 (table 2: YOLOv3, input image resizing: 25% of average). The rate of ODF to MOD errors was about 3:1 with only slightly more errors emerging in CAT_00 (+12) than in the original size condition. It might seem intuitive that a rise in ODF is a result of the image ratio distortion during resizing, which in turn might impede object identification for multi-cat images too, thus explaining the drop in MOD. However, when compared with average resized conditions, ODF errors were in fact lower for averaged resizing in both folders (ODFs: 90, 90 vs 110, 109). Moreover, MOD rates were nearly the same for both of these conditions (±1 item). Since some items also behaved contrary to that supposed intuition, in that they moved from successful detection to the MOD category (fig.2), the exact nature of the tradeoff remains uncertain. This mentioned example further indicates that not all MOD cases were truthful. This is why evaluation speaks of a 'success rate' rather than accuracy, as it is unclear how many of the MOD cases would be true negatives (multiple cats pictured) or false negatives (only one semantic cat object, but multiple bonding boxes) for the single-cat object detection task.

4.2. Image segmentation

Building on a high success rate during object detection, the performance of GrabCut was the crucial factor in limiting the amount and quality of suitable items for further processing. Unfortunately, complete segmentation (such that little to no background area remained in the ROI array) was not common and motivated the creation of the CAT_00 subsets. As an example for a rough estimate of GrabCut performance, take the amount of sample images from CAT_00 for

		CAT_00 (n=1706)			CAT_01 (n=1618)		
		Input image resizing			Input image resizing		
		Original	25% of original	25% of average	Original	25% of original	25% of average
YOLOv3	Success rate (ODF, MOD)	93.55% (76, 34)	91.62% (110, 33)	92,85% (90, 32)	92.4% (83, 40)	91.29% (109, 32)	92.4% (90, 33)
	Iter/second Total runtime	2.74 it/s 10:21min	2.96it/s 09:36min	2.88it/s 09:51min	2.95it/s 09:08min	3.11it/s 08:40min	2.89it/s 09:19min
YOLOv3-tiny	Success rate (ODF, MOD)	6.33% (1598, 0)	4.4% (1629, 2)	4.63% (1627, 0)	6.49% (1507, 6)	4.45% (1542, 4)	4.89% (1530, 9)
	Iter/second Total runtime	26.02it/s 01:05min	24.02it/s 01:10min	23.31it/s 01:13min	24.16it/s 01:06min	22.51it/s 01:11min	22.75it/S 01:11min
YOLOv4	Success rate (ODF, MOD)	93.79% (77, 29)	93.02% (92, 27)	93.14% (87, 30)	92.89% (80, 35)	92.46% (90, 32)	92.21% (96, 30)
	Iter/second Total runtime	2.27it/s 12:31min	2.30it/s 12:22min	2.23it/s 12:45min	2.34it/s 11:30min	2.31it/s 11:40min	2.37it/s 11:24min
YOLOv4-tiny	Success rate (ODF, MOD)	55.45% (751, 9)	28.14% (1219,7)	36.58% (1074, 8)	52.6% (760, 7)	25.9% (1195, 4)	33.19% (1077, 4)
	Iter/second Total runtime	23.07it/s 01:13min	21.98it/s 01:17min	22.90it/s 01:14min	24.94it/s 01:04min	21.17it/s 01:16min	22.79it/s 01:10min

Table 2. Object detection success rate (in percent) and failure causes (total count of: object detection failures (ODF), multiple object detection errors (MOD)) of various YOLO models; with input resizing effects, and associated processing speed (iterations per second, total runtime in minutes).

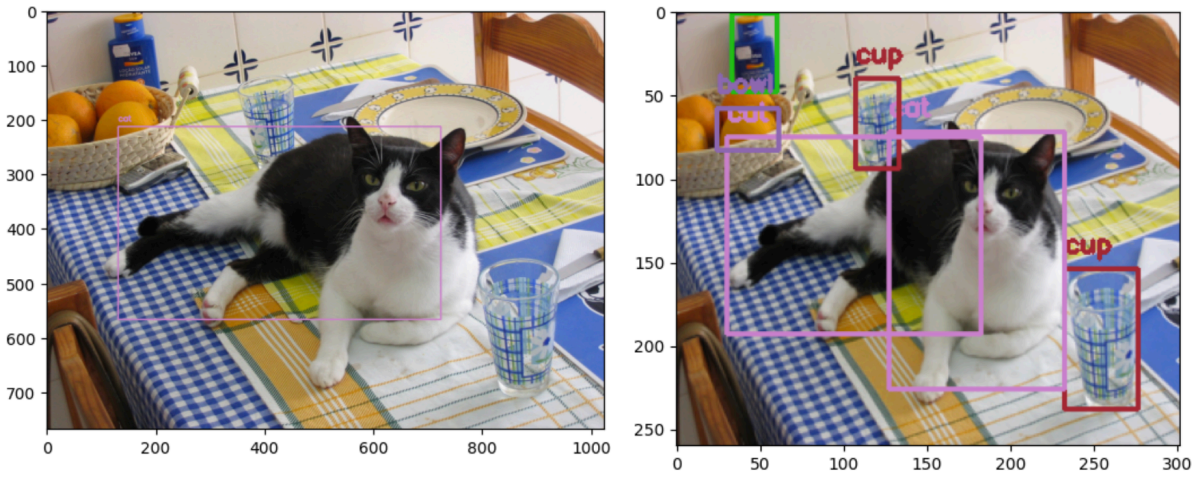

Figure 2. Image passing cat-object detection in its original format (l.), but causing MOD (two flamingo 'cat' boxes) when resized to 25% of the average folder size (r.).

which object detection was successful (n=1589, 93.14%). Compare this to the number of images in CAT_00_mixed (n=431) which consists of only those candidates for which the GrabCut output quality was somewhat sufficiently successful (fig.1, p.2). Due to the scope of this project, there was no time to employ a metric to automatically evaluate the success of image segmentation or to redesign this processing step fundamentally.

4.3. Colour compression and vectorisation

As a consequence of these GrabCut results, extracted ROI arrays may contain considerable amounts of noise affecting processing steps downstream. The colour compression model was therefore fitted on the selected high quality data sets to prevent non-ROI colours from affecting the centroid creation. During the compression there was no attempt at distinguishing

between colours from the ROI and remaining non-ROI area. Non-ROI colours were also still replaced by one of the palette centroids during compression. Relative colour quality and image readability was preservable with k being as small as 32. Further up- and downscaling of k in- and decreased colour quality accordingly, but was not included as a factor in further analysis.

Shown in figure 3 is a demonstration of colour compression (k=32) on a ROI array including the full possible colour centroid palette, and the resulting colour profile.

During the vectorisation process, the exclusion of transparent pixels allowed for the removal of items where GrabCut failed in a way such that the entirety of the input image had been classified as background area. In these cases, the returned array consisted of nothing but transparent pixels which lead to a colour profile with values summing to zero which were simply be dropped from the samples.

4.4. KMeans colour clustering

The following observations are to be made from the configurations (cf. table 1): Definitive colour groupings only emerged when prediction was based on the feature vectors (conf.2, fig. 4.2): 'Light'/cream, black/brown, red, and grey seem to be the associated archetypes which, except for the first, align with coat colour types. The control condition (conf.1, fig. 4.1) has a similar black/brown grouping yet this cluster also includes unexpected outliers, most notably the white cats in column one, rows five and seven. In this condition, the model appears to struggle to form four proper clusters in general, as indicated by the comparatively small second cluster. Increasing the number of clusters for configuration 2 to k=5 or k=6 leads to the 'light' and the red clusters being split into roughly equally sized halves based on brightness level. However, these grouping do not constitute more salient coat categories.

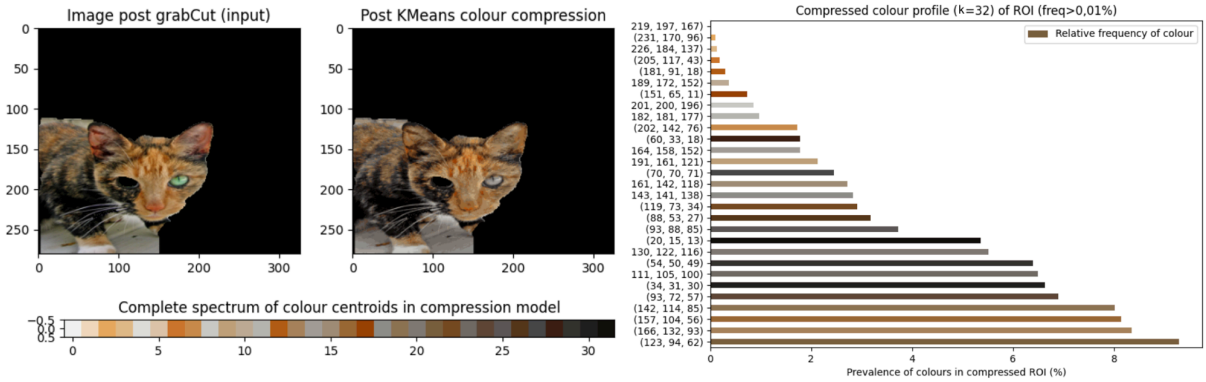

Figure 3. Plotted ROI array displaying mediocre GrabCut success before and after k=32 colour compression (l.) with colour hue normalisation post colour compression visible in reduction of green (eye) and red hues (inner ear, nose). Corresponding colour profile (r.).

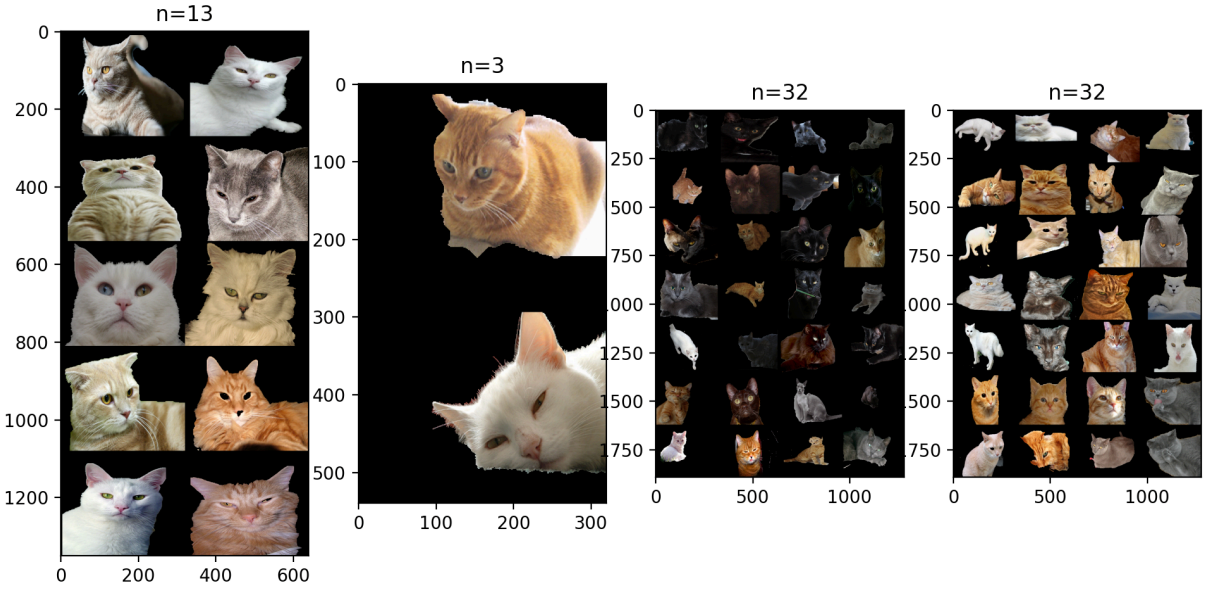

Figure 4.1. KMeans (k=4) colour clustering on CAT_00_solid (lim=80) based on the flattened ROI arrays (k=259200) from CAT_00_solid (n=96).

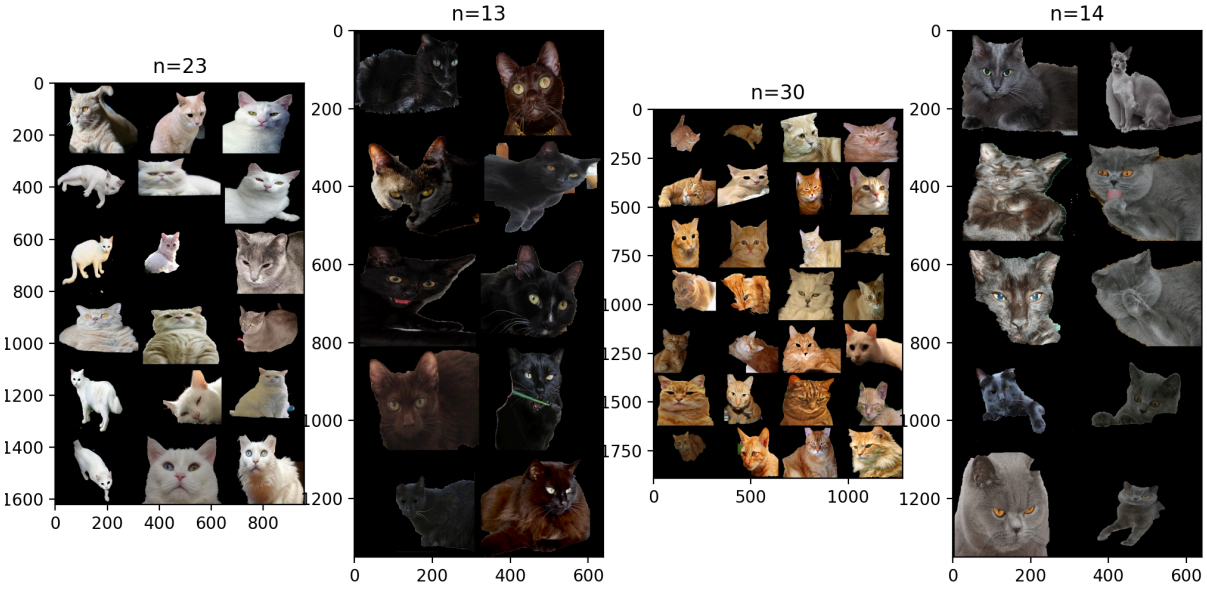

Figure 4.2. KMeans (k=4) colour clustering on CAT_00_solid (lim=80) based on the embeddings (k=32) from CAT_00_solid (n=96).

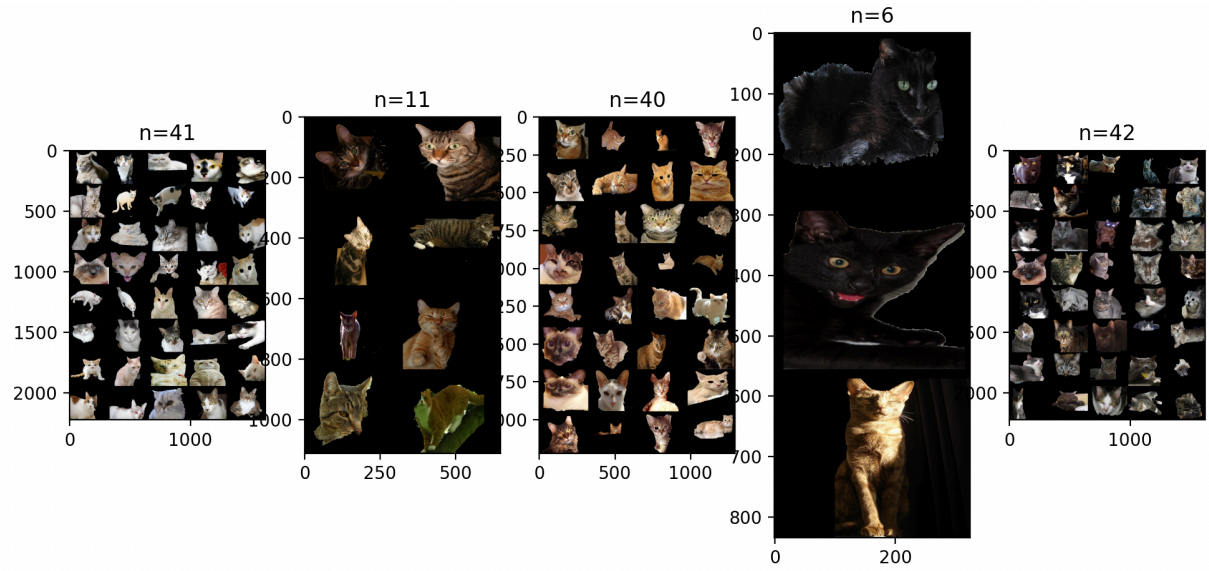

Figure 4.3. KMeans (k=5) colour clustering on CAT_00_mixed (lim=110) based on the embeddings (k=32) from CAT_00_mixed.

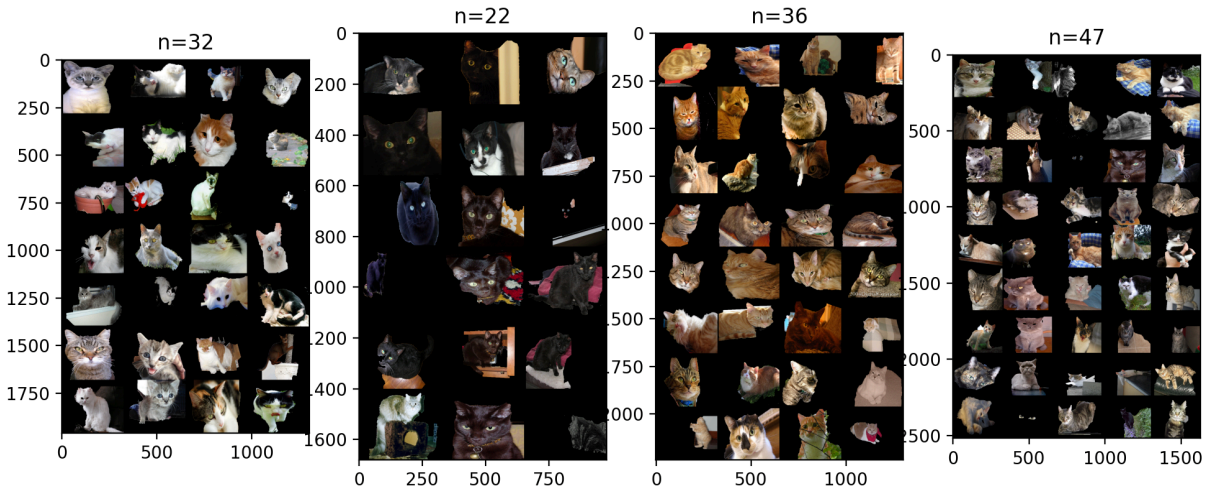

Figure 4.4. KMeans (k=4) colour clustering on CAT_01 embeddings (k=32, lim=140) based on embeddings (k=32) from CAT_00_solid.

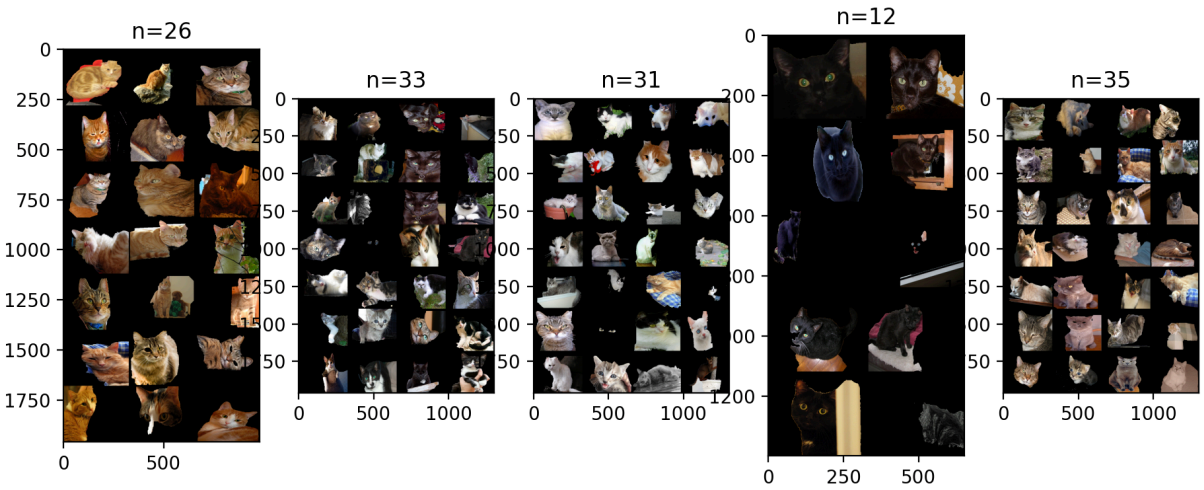

Figure 4.5. KMeans (k=5) colour clustering on CAT_01 embeddigns (k=32, lim=150) based on embeddings (k=32) from CAT_01_mixed.

Configuration 3 (fig. 4.3) used the mixed set for fitting and prediction, with k for the latter being raised by one to allow searching for a new cluster archetype. The number of items to use for prediction (n) was also increased accordingly to limit the chance of a disadvantageous item order blocking proper clusters. Cluster salience did not improve, but rather degraded in this configuration, independent of changes to k or in either direction.

Configurations 4 and 5 (figs 4.4, 4.5) exhibit similar patterns as 2 and 3, based on the set that their clustering models were fitted on. They also behaved the same when upscaling k, configuration 4 clustering breaking down and resulting in an n=5 cluster for k=6.

5. Analysis & discussion

Object detection This task was initially performed on the non-resized input images using YOLOv3 (Redmon & Farhadi, 2018) and the GrabCut algorithm. The runtime with this setup was excessive (~80mins/folder) and the slowest step in the whole processing pipeline.

During the following search for the cause of this time cost, two factors emerged which could easily be manipulated while preserving the existing code, and also impacted the processing speed: The input image size and the specific YOLO model used to perform object detection, as OpenCV offers support for multiple versions. Reducing the image size or switching to one of the faster YOLO-tiny models were feasible courses of action. Yet though both may improve speed, they can also harm the success rate of object detection (cf. table 2). The performance-time tradeoff relations based on these two factors, plus the subsequent inclusion of GrabCut are shown in table 3 and elaborated. Differentiating results based on this inclusion will in the end be demonstrated to be integral in illustrating the time cost bottleneck. To improve the current architecture, a combination which reduces runtime and preserves a high object detection success rate is desirable. To refer back to the results from table 2 first: The full size models, YOLOv3 and YOLOv4, achieve high success rates (>90%) for both image folders and across all three input sizes[1], while the tiny models perform substantially worse. YOLOv4-tiny still reaches an object detection success rate >50% for both original size conditions, however, YOLOv3-tiny barely passes a 6% rate. On the other hand, the strong performance of the full models is accompanied by about ten times as much processing time as their respective tiny counterpart. Input item rescaling does not appear to affect the processing speed of object detection in a meaningful way, but it does exert a sizeable effect on the performance of YOLOv4-tiny, seen in an average decrease of the success rate by about 23 percentage points for each dataset when resizing to the folder average. YOLOv3 and YOLOv4 appear to be largely unaffected by this parameter, success rate dropping by no more than -1,93% for YOLOv3 and only -0,77% for YOLOv4.

Regardless, the runtimes presented so far do not yet account for the aforementioned runtime of over an hour per image folder. The source of this increment is illustrated in table 3 with the +GrabCut rows. For the data in this table, the same code as for table 2 was run on just the first 100 files of each folder, and then succeeded by the GrabCut algorithm. Intermediary timestamps

[1] For the original size condition, preprocessing with YOLOv4 causes a unique (-125: Assertion failed) cv2.error during GrabCut but only for item 57 of CAT_00. I have been unable to find out why.

			CAT_00 (n=100) Input image resizing			CAT_01 (n=100) Input image resizing		
			Original	25% of original	25% of average	Original	25% of original	25% of average
YOLOv3	Success rate (ODF, MOD)		90% (7, 3)	91% (7, 2)	89% (8, 3)	91% (5, 4)	92% (6, 2)	90% (7, 3)
	Speed	Object detection	2.85it/s 0:35min	2.76it/s 0:36min	2.83it/s 0:35min	3.18it/s 0:31min	3.24it/s 0:30min	3.16it/s 0:31min
		+ GrabCut	0.22it/s 07:37min	1.06it/s 01:34min	1.44it/s 01:09min	0.26it/s 06:23min	1.17it/s 01:25min	1.50it/s 01:06min
YOLOv3-tiny	Success rate (ODF, MOD)		5% (95, 0)	6% (93, 1)	5% (95, 0)	3% (95, 2)	1% (98, 1)	3% (94, 3)
	Speed	Object detection	24.53it/s 00:04min	23.82it/s 00:04min	21.72it/s 00:04min	23.49it/s 00:04min	21.83it/s 00:04min	27.91it/s 00:03min
		+ GrabCut	5.77it/s 00:17min	15.12it/s 00:06min	20.11it/s 00:04min	14.09it/s 00:07min	23.33it/s 00:04min	19.39it/s 00:05min
YOLOv4	Success rate (ODF, MOD)		92% (4, 4)	92% (6, 2)	**93%** (4, 3)	94 % (3, 3)	95% (2, 3)	**95%** (3, 2)
	Speed	Object detection	2.50it/s 00:40min	2.45it/s 00:40min	2.59it/s 00:38min	2.41it/s 00:41min	2.51it/s 00:39min	2.55it/s 00:39min
		+ GrabCut	0.26it/s 06:28min	0.95it/s 01:45min	1.22it/s 01:21min	0.29it/s 05:45min	1.15it/s 01:26min	1.37it/s 01:12min
YOLOv4-tiny	Success rate (ODF, MOD)		55% (44, 1)	31% (69, 0)	42% (57, 1)	54% (45, 1)	28% (71, 1)	30% (69, 1)
	Speed	Object detection	25.14it/s 00:03min	27.03it/s 00:03min	25.93it/s 00:03min	24.39it/s 00:04min	25.28it/s 00:03min	24.68it/s 00:04min
		+ GrabCut	0.31it/s 05:24min	2.65it/s 00:37min	3.79it/s 00:26min	0.43it/s 03:49min	3.37it/s 00:29min	4.38it/s 00:22min

Table 3. Effects of input resizing on different YOLO models' performance in terms of object detection success (return rate in percent, count of object detection failures (ODF) and multiple object detection errors (MOD), processing speed (iterations/second, total runtime), as well as processing speed with subsequent addition of GrabCut algorithm. Highest success rate per input image folder is marked in bold.

for object detection on those 100 files are also noted for relative comparison. It becomes visible here that the long processing time stems from GrabCut, accounting for up to 90% of the total runtime when combined with the already slower, full size models. GrabCut's time cost is especially striking in the case of YOLOv4-tiny,the fastest model here: In the original size condition for CAT_00, object detection took only three seconds, but runtime jumped to over five minutes when including GrabCut.

Fortunately, GrabCut runtime is strongly correlated to the input image size and either resizing condition reduces runtime immensely. The latter being favourable because of its association with a better object return rate (cf. section 4.1).

These post-hoc results indicate that using YOLOv4 for object detection would have been the most efficient option for this task[2]. Integration into the pipeline was complicated by the fact that GrabCut turned out to be sensitive to the exact coordinates of the bounding to an extent where even slight differences may strongly affect its performance (fig.5). To ensure that the files chosen for the gold sets do in fact lead to clean ROIs no model switch was made, attempts at replication of this paper should be made with YOLOv3.

[2] Why YOLOv4 is slower than YOLOv3 in the pure object identification condition, but faster in the original size condition with subsequent GrabCut is unclear to me.

Figure 5. Performance of GrabCut when initialised with a bounding box from YOLOv3 (top) compared with the result when using a bounding box from YOLOv4 (bottom).

To improve efficiency, a hybrid use of two YOLO models could be experimented with, utilising the quicker YOLOv4-tiny as a default and only falling back on YOLOv4 for the images that the tiny version fails on (not to mention the abilities of more advanced YOLO models; Canu 2024). Other than YOLO, Single Shot Multibox Detector (SSD; Liu et al. 2016) or faster R-CNN (Ren, 2017) might be viable alternatives.

GrabCut became one, if not the, key limiting factor of the project. Not only forces it the inclusion of measures to reduce time cost, but the unstable quality and lack of an evaluation metric thereof cause extremely noisy and unreliable input data. Increasing the number of algorithm iterations did not affect the output quality either. Until this issue is addressed, further developing the feature extraction process, which is also desirable to improve the cluster quality, seems futile as well.

Possible approaches to improving image segmentation are combining GrabCut with neural networks (Rosebrock, 2023), examining the option of using different algorithms like Watershed (Bhatia, 2023), or resorting to networks like the noted ResNet101 or YOLOv8 (Rosebrock, 2018; Canu, 2024) which has built-in segmentation functions. As a more simplistic approach to removing noisy data, another colour compression could be fitted and run on each input ROI array individually. If less than a certain minimum threshold of unique colours are preserved, colour compression will raise a convergence warning, meaning GrabCut has likely destroyed most of the object and the image must be discarded.

Colour compression has proven to be an effective way to extract a base level colour feature representation. Fitting on a high quality ROI subset also succeeded in purging 'exotic' colours. Although this still had the adverse effect of non-ROI colours being forcibly integrated into the colour profile, it at least prevented the formation of some distinctly noise-based clusters (e.g. blades/patches of grass, that also often overlapped past ROI contours, causing their own cluster formation). Fitting a unique colour compression model on each image on its own prior to GrabCut to reduce the amount of colours still lead to the same clusters as in configuration 1. Fitting a model over all inputs was thus necessary.

Colour profile. Exclusion of transparent pixels was another vital step, as clusters would otherwise be largely based on the size of the non-ROI area, instead of the other way around. This feature extraction method enabled clustering of the solid cats based on the main colour component, but is likely not sophisticated enough to distinguish clusters beyond this basis. Improvements in clustering were neither visible by up- or downscaling colour palette sizes, nor by substituting the relative frequency measure by a binary encoding. Colour co-occurence was not encoded here but could also be a salient feature. Ultimately, resorting to using a pretrained network like VGG (Simonyan & Zisserman, 2014) for higher level feature extraction may yield better results.

Clustering. The results from configuration 1 and 2 have shown that an essential degree of success could be demonstrated by the architecture. Yet, once the complexity of the task is

raised above the solid condition, functionality begins to break down. Another concern is visible in configuration 3: Objects that belong to the same real-world class and even have similar visual features are not treated consistently (see colourpoint coats in figure 4.3 [cluster, row, column]: [1,4,1], [3,3,1], [3,4,1], and [5,4,1]). This too is an indication that improving the feature representation is necessary. Configurations 4 and 5 suffer from the same issues, here the adverse effect of the noisy background becomes visible, skewing cluster identities (figure 4.4 [3,6,1], distinctly non-black/brown cat in otherwise dark cat cluster due to exhibiting a sizeable dark background area). It would also be desirable to include heuristics that deal with the effect of lighting. Otherwise, distinctions between white coats in low lighting and grey coats in bright lighting are hardly possible.

Even though the evaluation of colour clustering here is not entirely subjective, the lack of a formal evaluation metric for this dataset is another limitation that makes it difficult to draw conclusions, especially for differences in performance between configuration 4 and 5.

Lastly, future work may want to consider rebuilding training data controlled for balance between expected cluster groups. Branching out to evaluate other clustering algorithms for this task is another possibility (Priy, 2024).

6. Conclusion

This project has laid the groundwork for an architecture to cluster images based on the main colour component of a common region of interest. As post-hoc analysis has shown, room for improvement can be found at many steps. A large factor in raising runtime and limiting overall potential for performance was the GrabCut algorithm. The runtime issue was partially mitigated by resizing the input, the second issue however had signifiant untethered impact. A naive and efficient method for feature extraction in the form of vectorised relative colour profiles enabled the baseline functionality of the approach, but ought to be expanded upon or even replaced in order to tackle the clustering of more complexly coloured ROIs.

Ethical concerns. The author does not endorse any extensions of the project that aim to cluster human subjects based on their skin colour components and have malicious intent or pseudo-scientific race theory in mind.

7. References

Arunponnusamy (2018, September 18). *Cvlib, v0.2.6*. GitHub. https://github.com/arunponnusamy/cvlib.

Axel, Zoe - Save the data dump, & Setchell, M. (2021, March 16). *How to ignore a color or alpha when using clusters*. Stack Overflow. https://stackoverflow.com/questions/66657077/how-to-ignore-a-color-or-alpha-when-using-clusters.

Beleidy. (2017, December 17). *Unsupervised Image Clustering*. GitHub. https://github.com/beleidy/unsupervised-image-clustering/blob/master/capstone.ipynb.

bnewbold. (2017, May 30). *Cat dataset : Weiwei Zhang, Jian Sun, and Xiaoou Tang : Free Download, borrow, and streaming*. Internet Archive. https://archive.org/details/CAT_DATASET.

Bochkovskiy, A., Wang, C. Y., & Liao, H. Y. M. (2020). Yolov4: Optimal speed and accuracy of object detection. https://doi.org/10.48550/arXiv.2004.10934.

Bhatia, J. (2023, July 15). *Exploring image segmentation techniques: Watershed algorithm using OpenCV*. Medium. https://medium.com/@jaskaranbhatia/exploring-image-segmentation-techniques-watershed-algorithm-using-opencv-9f73d2bc7c5a.

Canu, S. (2024, September 20). *Instance segmentation yolo V8: Opencv with python tutorial*. Pysource. https://pysource.com/2023/02/21/instance-segmentation-yolo-v8-opencv-with-python-tutorial/.

Crawford, C. (2017, May). *Cat Dataset*. Version 2. https://www.kaggle.com/datasets/crawford/cat-dataset/data.

Devashree. (2022, September 1). *Unsupervised learning in image classification - everything to know*. Documents Processing Solution. https://www.amygb.ai/blog/unsupervised-learning-in-image-classification.

Hartwell, S. (n.d.). *Colour and pattern charts*. Messybeast cats. http://messybeast.com/colour-charts.htm.

ma7555. (2019, December 12). *Cat breeds dataset*. Kaggle. https://www.kaggle.com/datasets/ma7555/cat-breeds-dataset.

mohit gupta_omg :). (2023, December 4). *Unsupervised learning*. GeeksforGeeks. https://www.geeksforgeeks.org/ml-types-learning-part-2/.

Layton, R., Grisel, O., & Blondel, M. (n.d.). *Color quantization using K-means*. scikit. https://scikit-learn.org/1.5/auto_examples/cluster/plot_color_quantization.html.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., & Berg, A. C. (2016). SSD: Single shot multibox detector. *Lecture Notes in Computer Science*, 21–37. https://doi.org/10.1007/978-3-319-46448-0_2.

Preet, A. (2024, October 16). *Image Segmentation using OpenCV.* Analytics Vidhy. https://www.analyticsvidhya.com/blog/2022/03/image-segmentation-using-opencv/.

Priy, S. (2024, March 20). *Clustering in machine learning*. GeeksforGeeks. https://www.geeksforgeeks.org/clustering-in-machine-learning/.

Prost, J. (2022, April 8). *GrabCut for automatic image segmentation [opencv tutorial]*. Theodo Data & AI. https://data-ai.theodo.com/blog-technique/grabcut-for-automatic-image-segmentation-opencv-tutorial.

Rafidinarivo, I. (2022, March 22). *6000+ store items images classified by color*. Kaggle. https://www.kaggle.com/code/itokianarafidinarivo/6000-store-items-images-classified-by-color.

Redmon, J., & Farhadi, A. (2018). YOLOv3: An Incremental Improvement. *ArXiv*. https://doi.org/10.48550/arXiv.1804.02767.

Ren, S., He, K., Girshick, R., & Sun, J. (2017). Faster R-CNN: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *39*(6), 1137–1149. https://doi.org/10.1109/tpami.2016.2577031.

Rosebrock, A. (2023, February 20). *OpenCV grabcut: Foreground segmentation and extraction*. PyImageSearch. https://pyimagesearch.com/2020/07/27/opencv-grabcut-foreground-segmentation-and-extraction/.

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. https://doi.org/10.48550/arXiv.1409.1556.

Torchvision semantic segmentation - pytorch for beginners. *LearnOpenCV*. (2024, July 30). Https://learnopencv.Com/pytorch-for-beginners-semantic-segmentation-using-torchvision/.

Zhang, W., Sun, J., & Tang, X. (2008). CAT head detection - how to effectively exploit shape and texture features. *Lecture Notes in Computer Science*, 802–816. https://doi.org/10.1007/978-3-540-88693-8_59.