MŰEGYETEM 1782

Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Telecommunications and Media Informatics

Selection of training data for deep learning methods

Scientific Students' Association Report

Author
Brunó Bence Englert

Advisor
Dr. Csaba Zainkó

November 1, 2022

Contents

Kivonat

A mai napig időigényes és drága folyamat új tanító adathalmaz létrehozása neurális hálózatokhoz. Emiatt szeretnénk, ha címkézés során a legrelevánsabb adatpontokat dolgoznánk fel. Erre a problémára szeretnénk egy megoldást találni úgy, hogy egy, még nem címkézett adathalmazból megkeressük a legértékesebb adatpontokat. Az adatpontokat bevett eljárásokkal n-dimenziós vektorokká alakítom. A vektor reprezentáció a hasonló adatpontokat egymáshoz közel, az egymástól különböző adatpontokat pedig egymástól távol helyezi. Ekkor, ha az egymástól legtávolabbi vektorokat választom, akkor a hozzájuk tartozó adatpontok ugyancsak különbözők lesznek és így létre tudok hozni egy diverz részhalmazt az eredeti halmazból.

Ezen munka során megvizsgáltam, milyen vektor reprezentációs eljárásokat érdemes használni. Ahhoz, hogy kezelhető legyen a feladat bonyolultsága és mérete, úgy döntöttem, csak kép és videó adathalmazokat használok. A viszonyítási értékünk a random adatpont mintavételezés. Ezzel ellenőriztem, hogy a lehetséges megoldások hogyan teljesítenek. A legjobb adathalmaz mintavételezést úgy lehet elérni, hogy a címkézetlen adatokat betanítom egy - a jelenleg korszerűbb - felügyelet nélküli klasszifikáló módszerrel. Ebből a betanított neurális hálóból már tudok a címkézetlen képekből egy vektor reprezentációt csinálni. Végül a címkézetlen képek közül kiválasztom azokat, amik egymástól a lehető legtávolabb vannak a vektor reprezentációban.

Az eljárás egyik különlegessége, hogy nem szükséges címke az adathoz, hiszen felügyelet nélküli tanítást használok. Emiatt olyan doméneneken is használható, amihez eddig nem készült adathalmaz. Emiatt ez az módszer eltér a hagyományos aktív tanulási eljárásoktól, mely során már van valamennyi címkézett adat.

A javasolt megoldásokat leteszteltem képosztályozáson, illetve szubjektíven megvizsgáltam autók fedélzeti kamera felvételein. Ezek mellett megvizsgáltam a módszer ipari felhasználásának lehetőségeit is.

Abstract

Creating new training datasets for neural networks is still an expensive process due to labeling. This creates a need to only label datapoints with the greatest added value. The proposed solution addresses exactly this problem by exploring various methods to select potentially valuable datapoints. The method starts by transforming all datapoints to n-dimensional vectors, using already tried-and-tested techniques. This vector representation places the similar datapoints close to each other and places dissimilar points far from each other. By only choosing the furthest vectors, I can create a subset of datapoints with the greatest diversity.

In this paper I explore ways to create vector representations of datapoints for subsampling. To make the situation manageable, I narrowed down the problem set to image and video data only. The proposed methods are compared to a baseline method, which is randomly downsampling the images. The best result was obtained by training a classifier neural network on the unlabeled dataset using a SOTA unsupervised method. After this classifier neural network is trained, I can obtain the vector representation of the images. The unlabeled images are downsampled by choosing only the furthest vectors.

One added bonus to this method is that it only uses unsupervised methods for creating the vector representation. Thus it can be used on unexplored domains too, where previously created training data is not present. This setup deviates from classical active learning where we already have some labeled datapoints.

The proposed method is evaluated on image classification and also subjectively evaluated on dashcam videos. Furthermore, the industrial use cases is also explored.

Chapter 1

Introduction

Today's machine learning methods are using bigger and bigger models, that needs more and more data to be trained on. This creates a clearly visible problem: training for days or weeks and the need to label massive amounts of data makes SOTA solutions almost unreachable for businesses and universities. For example labeling 1000 images for classification is only 35$ [1], but for segmentation it is 870$ [1]. Smaller datasets start at around 50,000 images, which would translate to 43,500$ for image segmentation. The reason behind this is simple: data collection and labeling is still mostly done by hand. To make things even more difficult, even if the data is already labeled it's placed behind a pay wall. The consequence of this is that one still needs the ability to create completely new datasets cheaply for unexplored domains. If only manual work is the issue that drives up the cost, is it possible to automate some parts of the data creation process? When new data is collected (eg. from a video stream) a subset is created by downsampling the datapoints (eg. images). The hypothesis is that not all data are equal in value. For example, if there is a video stream, consecutive frames are very similar. If all the images in this video stream would be labeled, then most of the images would be similar to each other and the overall value/cost would be small. So, subsampling is needed to avoid too similar datapoints and to have a more diverse dataset. The downsampling is usually done randomly or with heuristics. For example, a heuristical way of downsampling in case of self driving cars, one can log the position of each recorded image via GPS. Images which are too close to each other are filtered out. Such heuristics are not perfect, they can miss difficult scenarios (wild animals, road accidents), which should be included in training/testing data. Research areas like active learning deals with a very similar problem: it's a special case of machine learning where the algorithm starts with an already labeled dataset and then selects from a set of unlabeled datapoints. The selected unlabeled datapoints are then labeled and added to the training data of the active learning algorithm. However, it can't be used in every case: active learning assumes that you already have labeled datapoints. When we create a completely new dataset of an unexplored domain we don't have the luxury to have labeled data. In this paper I explore the possibilities of how to select relevant datapoints for labeling, when no training data is present.

I only focus on a small subset of issues [21], which is data selection. I also narrowed down the problem set to image and video data only. So the question rises: how can data points that hold the most value be selected? Image to feature vector representation is a well accepted[30] method to differentiate images and at this point it is a textbook method[11]. Based on this I want to somehow represent the images as n-dimensional vectors where visually similar images are close to each other in this vector space. However, the definition of what is "visually similar" is hard to define. Even for the human eye, this is a subjective concept. Creating a method that captures the idea of "visual similarity" is non trivial and can mean many things. For this reason and the fact that the user doesn't necessary have already labeled data, I will only use off-the-shelf unsupervised methods to capture the meaning of visual similarity. I use these models to generate the vectors representing the images and evaluate various sampling methods on this vector space. Since I only use unsupervised methods, no labeled data is needed compared to active learning.

In this paper I show that it is possible to do better than random sampling. I evaluated the proposed method on image classification and also subjectively evaluated on dash cam videos. The dash cam videos are not processed as videos, rather I use the image frames. I also showed that using labels is not necessarily good. If I use the feature vectors from classifiers trained on image labels, they can perform poorly compared to my proposed method. I also explore how to deploy such a system and examined the problems related to it.

Chapter 2

Related work

2.1 Artificial neural networks

A good chunk of human inventions just like artificial neural networks were inspired by nature. As we zoom the brain at the microscopic level we find neurons. These neurons are connected by synapses and we can group these synapse as inputs and outputs for a neuron. We observe that as the input synapses fire more frequently, our neuron will also fire more frequently. Based on these observations, we can create our own single neuron and later neural networks. A basic form of neuron is weighing to inputs with a learnable parameter and a learnable bias: $output = \underline{W} * \underline{inputs} + b$. We also call this a linear model. However, something is still missing. To model more complex data, such as non linear data, we have to introduce a non linearity in our neuron model: $output = h(\underline{W} * \underline{inputs} + b)$. We call this h non linear function as the activation function and this function is somewhat analogue to a real neuron's threshold potential. This is how neurons are defined in all deep learning methods. The single neuron model, called perceptrons, were developed in the 1950s and 1960s by scientist Frank Rosenblatt, inspired by earlier work of Warren McCulloch and Walter Pitts. However, we are still far away from modelling complex problems such as image classification. To do that, we use deep neural networks. In these models, we use more neurons, where neurons are layered on top of each other. The learnable parameters are tuned by training the network on a data set. For each input we determine a desired output. We compare the desired output to the network's output, which defines an error on the networks prediction. If the out network is fully differentiable, we can propagate the gradient from the error to each of the learnable parameters. This gradient describes the slope towards the optimal parameter value. So if we modify our learnable parameters along the gradients, we can tune our parameters to reduce the error present at the output.

This lacks details, since not all inputs are as important. Some inputs are even inhibitory, meaning they lower the probability our neuron will fire. To model this, we want to add weight to inputs as learnable parameters and a bias:

2.2 Learning approaches

We differentiate multiple learning methods for deep neural networks, but we will only focus on two: supervised and unsupervised learning. The difference is that supervised leaning requires labelled data (usually done by hand), while unsupervised learning doesn't require labeled data. When we want to do for example classification, class labels are human defined, meaning they are arbitrary chosen. In such scenarios we have to have labeled data. However, labeling data is expensive so the vast amount of data is still unlabeled. Since unlabeled data is easy and cheap to acquire, we still want to make use of it. During unsupervised training, we have to use substantially different methods compared to supervised training to still learn useful structures from the unlabeled data. In the past few years, the renaissance of unsupervised learning sprung to action in the form foundation models [5], such as the language model GPT-3[7] or the image generating model DALL-E[20]. Unsupervised image feature learning is also prevalent, like: autoencoders[2], Swapping Assignments between Views(SwAV)[8].

2.3 Data diversity

To have a common ground, we want to define what it means that a data point holds "more information". The intuition is that if a data point is more "diverse" compared to the already selected points, then it holds more information. We define that a "diverse" point is somehow different from the already selected points. We could also say that a diverse dataset covers a larger domain. This larger domain is what we want to achieve. A common issue is observed in practice: if the training and testing datasets are similar we can achieve a very good accuracy on the test set. But these models usually perform poorly on unseen domains[28][3][4][15][22][26]. We can also somewhat prove this point by using augmentations. Using augmentations on the training data artificially increases the dataset variety and results in a more robust test performance [19] [23].

2.4 Autoencoder

Autoencoders are neural networks, where the input and output is an image and the expected output image should be the same as the input. The network architecture is constructed in way that the amount of information that's able to propagate through the layers is restricted. The restricting part is called the bottleneck and it is usually implemented by significantly restricting the vectors size compared to the input image size. This restricted feature vector representations is created by the encoder network. Then the so called decoder network tries to reconstruct the input image from this restricted feature vector. The goal of this setup is to learn a more compressed representation of the input image. The encoder can be later used for transfer learning[6] or the feature vector present at the bottleneck can be used for image clustering. We usually use l_2 loss to force the model to reconstruct the input image.

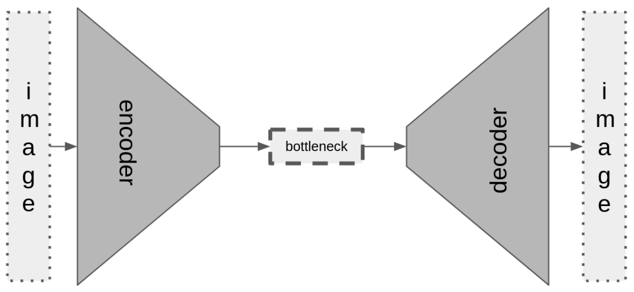


Figure 2.1: Illustration of autoencoder model architecture.

2.5 Swapping Assignments between Views

Another approach used for unsupervised learning is contrastive learning. In this setup, there is only an encoder compared to autoencoder, meaning the images are only transformed to feature vectors. On these feature vectors we define the following restrictions: similar images should have a similar vector compared to dissimilar images. The similar images are called positive pairs, while the dissimilar images are called negative pairs. Using this definition we can create the so called Multi-Class N-pair loss[25]:

$$-\frac{1}{N}\sum_{i=1}^{N}\log\frac{\exp x_i^T v^+/\tau}{\exp x_i^T v^+/\tau + \sum_{v\in\{v^-\}}\exp x_i^T v/\tau}$$

, where positive pairs are denoted by v^+ and negative pairs are denoted by v^-. However, calculating all the pairwise comparisons during loss is impractical on large datasets so various tricks and solution have been tried to solve this, like NPID[29], MoCo[13], SimCLR[9] and SwaV (Swapping Assignments between Views)[8].

Since contrastive learning methods are unsupervised, it's not clear how to determine the positive and negative pairs. To overcome this, a common practice is to take a single image, create different view from this image via augmentations (cropping, rotating, etc.) which creates the positive pairs. The negative pairs are randomly selected from the dataset. If the dataset is large enough, it is unlikely a collision will happen, when randomly selecting the negative pairs. In SwAV, instead of using all the images as negative pairs, they create learnable clusters in the feature vector space. These learnable clusters are called prototypes and denoted by c on figure 2.2. The original loss function is reformulated: the feature vectors from the input images created by the encoder are assigned to the clusters and they use this assignment as the target. They also swap the cluster assignment for positive pairs to learn consistent feature vectors.

The clustering is done online by simply getting the cosine similarity between the feature vectors and the clusters. However, this creates a trivial solution, where all the images are assigned to only one cluster. To overcome this, the authors restrict the batched assignments to be equipartitioned along all the clusters. They define the problem as an optimal transport problem, which can be solved with linear programming solvers. In this case, they relax the problem by introducing an entropy term, so it can be solved with the iterative Sinkhorn-Knopp algorithm[10]. This makes it much faster and thus a viable solution to the contrastive learning problem.

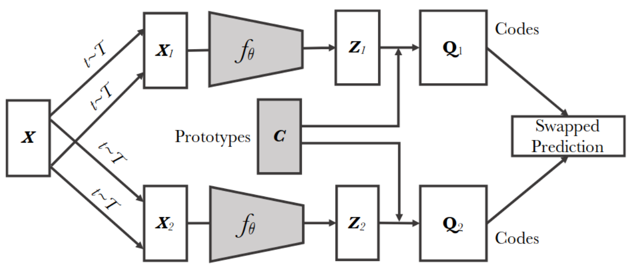


Figure 2.2: Illustration of the SwaV training setup. [8]

2.6 Sampling with the furthest neighbor algorithm

If we would select the optimal furthest points by checking every possible subset, then the step complexity would be $O(\binom{N}{k})$. This is a big issue, for larger sets, such step complexity is unacceptable. For this reason we have to use a method which is fast enough, but still outputs a good approximation. One solution is to use a greedy algorithm:

1. Choose a random initial point.
2. Choose the next point furthest from the already chosen points.
3. Repeat step 2. until we have the required amount of points.

This algorithm however still has an issue. It we do the trivial solution to get the next furthest point which is to calculate all the distance between the already selected points and all the points, then the memory complexity at the jth step will be $O(j * n)$. If there is a million points and we chose 500K points, then the memory alone would take up 500GB, which in most cases is not feasible.

To further reduce complexity, we instead do a slightly different way of calculating distances. Instead of calculating the distance between all the points and the already selected points, we maintain a one dimensional array called $point2pointMinimum$. In this array we store a distance value for each point, where each stored distance is the closest distance to an already selected point. We choose the next point, which index is $furthestPointIdx$, by selecting the largest distance value found in the maintained array. After the point with $furthestPointIdx$ index is selected, we have to update the $point2pointMinimum$ array. To do the update, we first calculate the distances between all the points and the point with $furthestPointIdx$ index. Then we take the element-wise min between these distances and the $point2pointMinimum$ array. The result is then stored in the $point2pointMinimum$ array. This algorithm has a much favorable memory use, which is $O(n + k)$ and a time complexity of $O(n * k)$. If k is small enough, this will be a fast enough algorithm. The full algorithm is presented on Algorithm 1. As an example of what this method looks like, we made a dummy point cloud from a noisy sine wave, where the selected point are denoted with red, shown on a Figure 2.3. We choose this point cloud to showcase how this algorithm behaves on dense and sparse parse parts too.

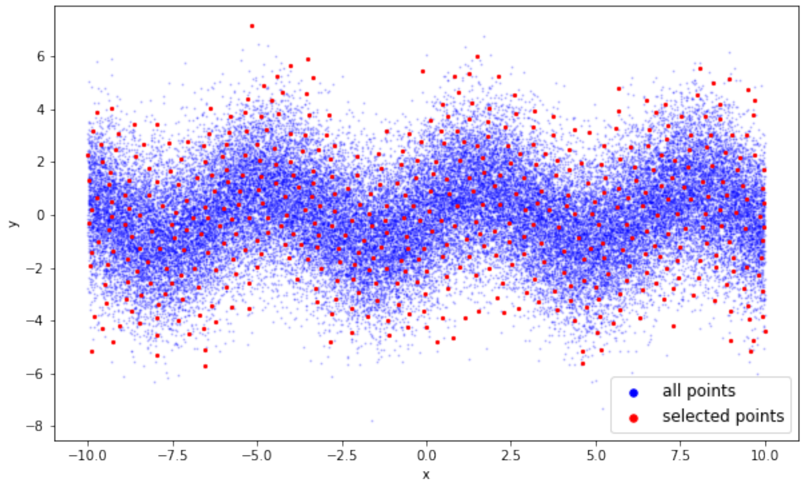


Figure 2.3: A noisy sine wave downsampled

Algorithm 1: The greedy algorithm for selecting the furthest points.

```
1  function farthestNeighbors($allPoints, k$);
   Input  : A list of n dimensional points $allPoints$ and a number of points to be
            selected $k$.
   Output: A list of selected points.
2  selectedPoints = [];
3  numberOfPoints = length(allPoints);
4  point2pointMinimum = array(size=(numberOfPoints, 1), fillValue=inf);
5  for $i = 0$; $i < k - 1$; $i = i + 1$ do
6      distances = getEuclideanDistance(selectedPoints[i], allPoints);
7      point2pointMinimum = min(point2pointMinimum, distance);
8      furthestPointIdx = argmax(point2pointMinimum);
9      selectedPoints.add(allPoints[furthestPointIdx]);
10 end
11 return selectedPoints;
```

Chapter 3

Proposed method

The goal is to create a subset of the most valuable datapoints, but how to do that is not clear. Since data selection in itself is still way too broad to investigate, I only investigated a small subset of this topic. I narrowed it down to the following problem set: the dataset is images only and a classification task. This way the only variable remaining is how I create a subset of images of the dataset.

3.1 Method and implementation

Using the assumption that "diverse" images are better, I investigate different methods to measure visual similarity/dissimilarity. The idea is to somehow turn the images into a feature vector representation. If we do this transformation to all the images, then such a vector representation would create a point cloud. I want this transformation to place visually similar points closer to each other. If I can obtain such a representation, then I just have to choose the furthest points from each other. This is equivalent to choosing the most dissimilar image pairs. Thus the overall information in the selected images should be higher compared to just randomly sampling.

I assume that most collected datasets are not uniform, meaning there are many repetitions and similar images in video streams. However, if the dataset is already somewhat uniform, meaning the datapoints are as dissimilar as possible, then we shouldn't see any improvement compared to random uniform sampling. When we use tried-and-tested datasets (such as CIFAR10), we expect the datapoints are already uniformly distributed. So even if I can come up with a good algorithm for selecting datapoints, there shouldn't be a big accuracy difference compared to the baseline method.

In summary, I want to test various unsupervised methods and select which one creates the best feature vectors for this unsupervised active learning scenario. I test two possible methods: create the vector representation with an autoencoder and a SWAV described in section 2.4 and section 2.5. After the images are transformed, the images are selected in the point cloud with the furthest neighbor algorithm described at subsection 2.6. The summary of the examined methods for sampling are present at Table 3.1

Method	Image to feature vector	Postprocessing	Sampling
baseline	-	-	random selecting
pretrained classifier	pretrained classifier	UMAP	furthest neighbor
proposed method 1	autoencoder	UMAP	furthest neighbor
proposed method 2	SWAV	UMAP	furthest neighbor

Table 3.1: Methods examined for image sampling.

3.2 Feature vector generation

One of the requirement's was to be able to run as many experiments as possible. To do that I choose a small backbone for all unsupervised methods, which is the resnet18[12]. This resnet18 was slightly modified to run well on 32x32 images. This change was modifying the very first convolution to have a 3x3 kernel size with 1 stride and no maxpooling after the first layer.

3.2.1 Using a pretrained classifier

As a reference, I trained a classifier on CIFAR10 with the given labels. Usually, this is the standard way of getting feature vectors from datasets. While this is not an unsupervised method, I wanted to see how the standard method behaves compared to the proposed methods. I only use this methods as a reference.

The model used for this is the same resnet18 described above. During traning I use random horizontal flipping and random cropping as augmentations. I use the feature vectors from the penultimate layer.

3.2.2 Using autoencoder

I used 5 layers of deconvolutions for the decoder to reconstruct a 32x32 image. The encoder's output vector (or in other words: the autoencoder's bottleneck) is chosen to be as small as possible compared to usual autoencoders. When the bottleneck is super small, the model is forced to separate different images and place similar images close to each other. Otherwise the feature space for an autoencoder would be too unregulated. After hyperparameter optimization, the chosen vector size is 8 dimensional. Then the vectors are transformed from 8 to 8 dimensions using umap[18]. After each image is transformed into an 8-dimensional vector, I downsample them with the furthest neighbor algorithm. I use euclidean distance on the points. During training, no augmentations are used as it would lead to augmentation leakage [16].

3.2.3 Using SWAV

Using an autoencoder has some drawbacks. I found that increasing the number of images requires a larger bottleneck. However, using a larger bottleneck ("curse of dimensionality") resulted in a lower accuracy on the evaluation tasks. This is somewhat similar to what is described in Sinha et al. 2019[24]. Another issue with autoencoders, is that they need to reconstruct the image. This reconstruction is an extra computation cost and I would like to avoid it, thus achieving a lower training time.

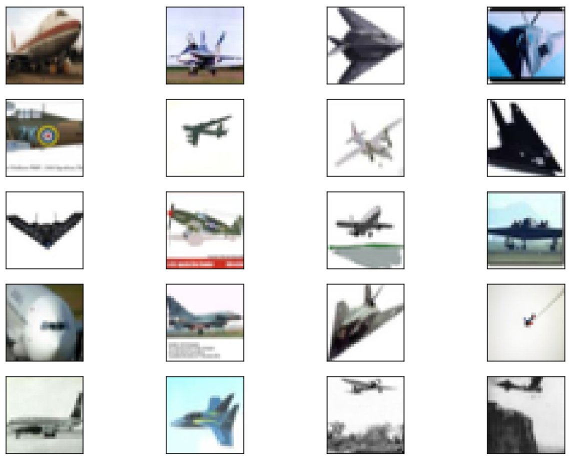


Figure 3.1: The images selected when directly sampling with the furthest neighbor algorithm. In this scenario the sampler finds outlier images and thus unfit for my purposes.

To resolve all of these issues, I choose an off-the-shelf unsupervised learning solution used for classification: SWAV. The setup is almost exactly the same as for the autoencoder method: I only train one SWAV model on all of the 50,000 training images. The model outputs 2048 dimensional vectors and then I reduced them to 20 dimensional vectors using umap[18]. For the different views, I use random horizontal flipping, random color distortions, random gaussian blur and random cropping augmentations. Overall, this method produced the best results compared to all the tested methods.

One downside of using SWAV instead of an autoencoder is that SWAV's output is only a vector. While the autoencoder comes with a decoder, which can convert the vectors back to images, the vector created by a SWAV model is unreadable by a human. Because of this, there is no feedback (other then the loss) on how well the model performs.

3.3 Normalizing the feature vector space

And the third issue is, even though a super small bottleneck somewhat regularizes the feature space, it is still highly nonlinear. This results in big distance differences between similar images.

In both the autoencoder and SWAV method the feature vector space is unregularized and some feature vectors can be extremely far from other feature vectors. This is shown on fig 3.1. To perform well, an extra step was needed. Since furthest neighbors is used for sampling, these outlier-like feature vectors will always be selected. To overcome this issue, a transformation is needed to remove global structure from the point cloud, while maintaining local structure. Dimensionality reduction algorithms like t-sne[27] and umap[18] do exactly this transformation. So after the images are turned into feature vectors, the feature vectors are transformer by umap. I choose umap because it is significantly faster compared to t-sne. Since I only want to remove the global structure, even transforming to the same dimensions improves the accuracy.

3.4 From images to videos

Creating image datasets from videos are a common practice for example in autonomous driving. Since this is an important part and possible future use of a sampling process, I wanted to try my best sampling method on images created from video sources.

Videos sources are significantly larger than the CIFAR10 dataset, a proper evaluation would take weeks. For this reason, I choose not to go down this route and decided to only do subjective measurements on videos and this section doesn't focus on objective measurements, like accuracy. I only wanted to investigate if the selected images are subjectively acceptable and to see if the selected images are relevant.

One of the features of video data is that consecutive frames are very similar to each other. I can use this similarity to my advantage to redefine the loss for SWAV described in section **??**. In the original paper, the different views are obtained by augmenting the image. In this case, different views will be obtained by choosing temporally close frames. Since SWAV tries to equipartition the codes for each images in the batch, I need the other images in the batch to be sampled as far from each other as possible in the temporal dimension. This way I don't try to distinctively code frames which are temporally close. This temporally distant sampling is done with a greedy algorithm: I select a minimum distance (eg. 10 second) and I randomly select frames. If the new frame doesn't collide with the already selected frames, the new frame is accepted, otherwise the new frame is dropped. Although this is a very naive way of selecting frames, it's fast enough in this case.

To reduce the space the images take up on disk and speed up the training time I made several changes. First, the video resolution was reduced significantly. The aspect ratio is kept but the width of the images is lowered to only 100 pixels. The frame rate is also reduced to 10FPS.

Chapter 4

Experiments and analysis

4.1 Testing methodology

All sampling methods are evaluated in the following way:

1.1 Define a set of hyper parameters: classifier model architecture, epoch size, batch size, etc. We call this the **probe** model. From now on I will not change any parameters of the **probe** model.

1.2 Subsample the dataset with method **X**. Let's call the subsampled dataset **R**.

1.3 We only want to evaluate unsupervised methods. Subsampling method **X** cannot use any information other than the images itself.

1.4 We train the **probe** model on the subsampled dataset **R**.

1.5 Evaluate the **probe** model on a predetermined test dataset.

A **baseline method** is defined by selecting images with random uniform subsampling. The subsampler method should perform better compared to this **baseline method**. To account for the noisy nature of the sampling and the training, each subsampling method (including the baseline method) are executed multiple times to obtain an average accuracy. From now on, I refer to this number of iterations as **sampling iterations**. For each evaluation, the accuracy of the probe model is aggregated to determine a mean accuracy and a standard deviation. These criteria define a system shown on Fig4.1.

4.2 Evaluating the proposed methods

The dataset I am using is CIFAR10[17], because it is a relatively small dataset. The smallness of the dataset chosen for evaluation really matters here. The evaluation consists of sampling the dataset and then training a probe model multiple times, even slightly large dataset with a few gigabytes of training data would take weeks. By choosing CIFAR10, I ensured that the time to evaluate an algorithm would only take a few days and thus makes it easier to reproduce results. It has 50,000 training images and 10,000 testing images with a resolution of 32x32 and an RGB color space. It defines 10 classes, like *cat*, *bird* and *truck*. The downsampled data is created by sampling from the 50,000 images. I decided to downsample by a factor of 10, so 5,000 images are chosen for each sampling

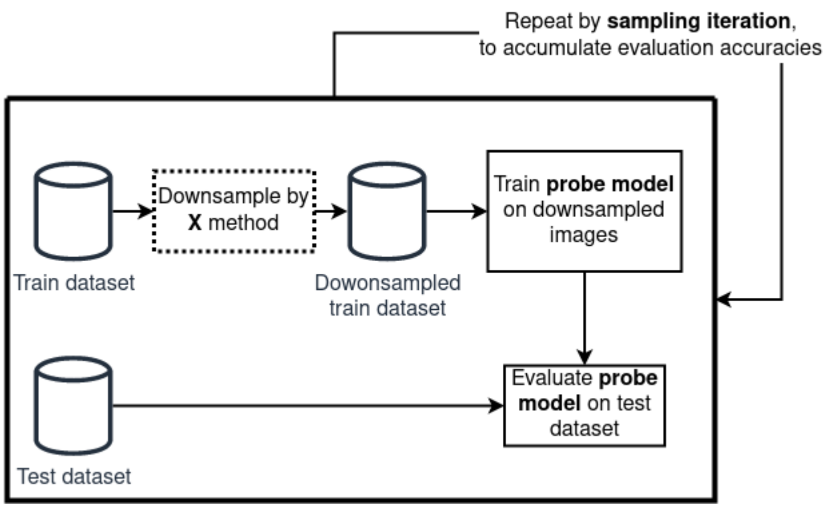

Repeat by **sampling iteration**, to accumulate evaluation accuracies
Train dataset
Downsample by **X** method
Dowonsampled train dataset
Train **probe model** on downsampled images
Test dataset
Evaluate **probe model** on test dataset

Figure 4.1: The system for testing the subsampling methods. The only changing variable is the sampling method denoted by a dotted rectangle.

iterations. I choose the sampling iteration to be 150, based on the standard deviation of the accuracy from previous evaluations. The probe model is the same resnet18 [12] used for the unsupervised methods. I deliberately choose the same model, because it turned out to be fast enough, which is important for evaluation. I use random horizontal flipping and random cropping as augmentations for the probe model. The probe model runs at maximum 100 epoch with early stopping. I use mixed precision training to speed up the process, and vanilla SGD for optimization.

Table 4.1 shows that in all cases we achieve higher accuracy compared to the baseline. It is important to note, that since we don't use labels, the selected classes can be unbalanced. This makes it much harder to beat the baseline method, which naturally creates balanced classes, if the original dataset is already balanced.

While the autoencoder method is better than the baseline methods I found it to be very sensitive to the bottleneck size. To get the best accuracy, I had to do a hyperparameter optimization on the bottleneck size, which is not possible without the labels. Meanwhile, the SWAV methods way significantly more robust. It worked out of the box, with no required modifications. This makes the SWAV method usable on real life problems and I recommend it over the autoencoder method.

Downsample size	**Baseline**	**Autoencoder**	**SWAV**
5000	77.727%	77.968%	**78.144**%
10000	84.650%	84.959%	**85.036**%

Table 4.1: Mean accuracy of the baseline method and our method. The sampling iteration is 150.

I also examined what are the effects of the sampling size. I found, that as the sampling size increase, the gained accuracy compared the baseline method decreases. This is shown of Fig 4.2.

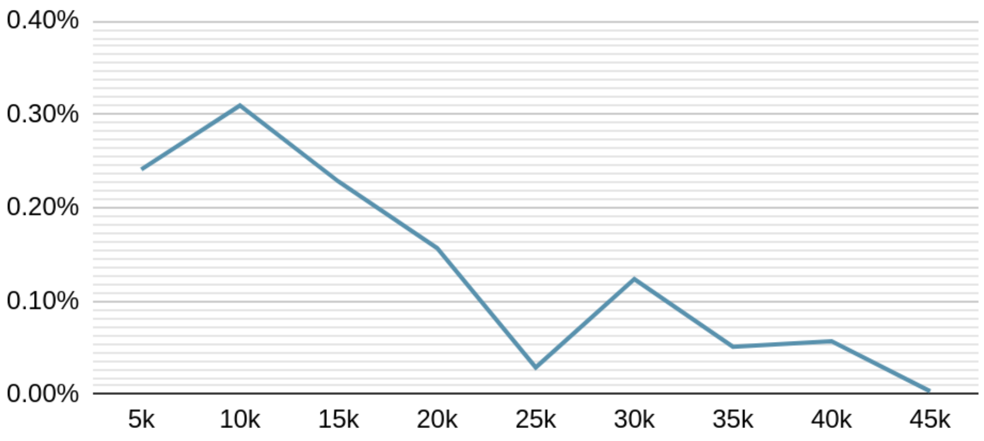


Figure 4.2: Mean accuracy difference between the baseline method and our autoencoder method. X-axis shows the downsample size, Y-axis shows the percentage difference. The baseline is the reference.

4.3 Comparing supervised and unsupervised methods

To check how big is the difference between supervised and unsupervised methods, I also evaluate a resnet18 classifier trained on CIFAR10. I use the feature vectors from the penultimate layer. Table 4.2 shows that the representation and autoencoder methods both outperformed the baseline method, however, the autoencoder method was better compared to the representation method. This is quite surprising, since using a pretrained classifier's feature vectors are commonly used in recommendation systems or for image similarity problems. Although I didn't test it, my hypothesis is that when training is done using labels, the model doesn't care about image similarity anymore, but to only separate the images based on the classes. Not only that but using labels can introduce biases. This is because the label classes are created arbitrary and it can be influenced by human biases. To add some foundation to this hypothesis, I visualized the closest image pairs for the trained classifier and for an autoencoder shown on 4.3 and 4.4. As you can see on these figures, the trained classifier returns images from the small class, while the autoencoder returns images extremely similar compared to the reference image.

4.4 Evaluation on videos

I trained the SWAV model on a mix of diverse dash cam videos from Budapest, Copenhagen New York and Sweden. The videos accumulate to 6.5 hours at around 150Gb with no video compression (35Gb if compressed). After the resolution and fps decrease all the videos take up 5.3Gb.

To see how SWAV behaves, I transformed various clips into feature vectors. These vectors were then reduced to 2 dimension with umap. You can see the result of one of these point clouds produced with this methods on Fig. 4.5. One interesting distinctness of this figure is that the frames create a somewhat continuous curve. This indicates that the model places consecutive frames next to each other, which seems promising.

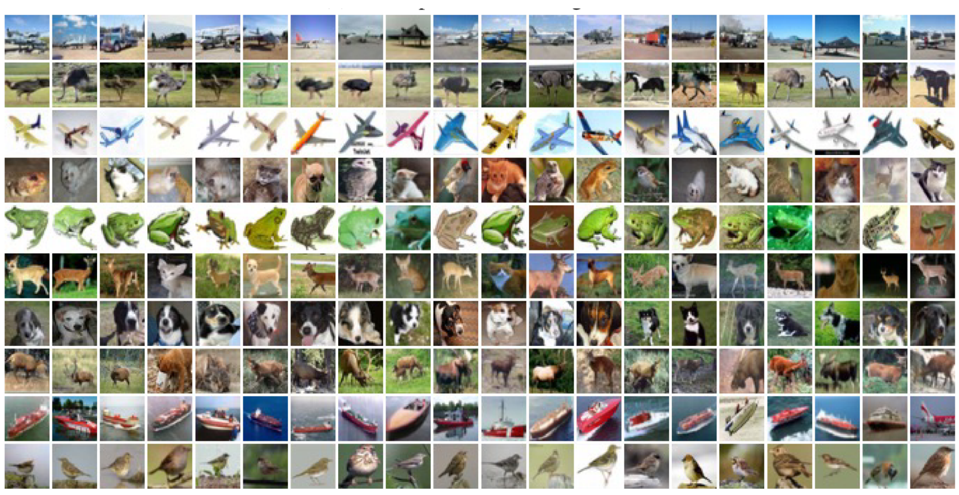


Figure 4.3: A trained inception classifier's feature space with nearest neighbors.[14]

Class	Baseline	Representation	Autoencoder
all	74.17%	74.46%	**74.84%**
plane	77.02%	**77.97%**	77.86%
car	84.60%	**86.52%**	85.98%
bird	64.01%	63.46%	**64.89%**
cat	56.02%	**56.70%**	55.84%
deer	69.70%	69.95%	**72.24%**
dog	65.45%	64.82%	**65.49%**
frog	80.62%	80.36%	**81.05%**
horse	77.00%	**77.74%**	76.52%
ship	84.15%	85.00%	**85.50%**
truck	**83.16%**	82.04%	83.00%

Table 4.2: Accuracy of the baseline, pretrained classifier and autoencoder method with downsample size 3000.

4.5 Summary

The go-to solution to create good feature vectors from images is by using a pretrained classifier. I showed that, while this methods gives class relevant images, the images are not visually similar. The pretrained classifier could be great for recommendation systems, but in this specific scenario using an unsupervised method is much better.

I also showed that the proposed method where feature vector are generated by SWAV was 0.417% better than the baseline method when the downsampling size is at 5000 on the CIFAR10 dataset. The proposed method is also fully unsupervised, this makes it a viable solution for unlabeled image selection.

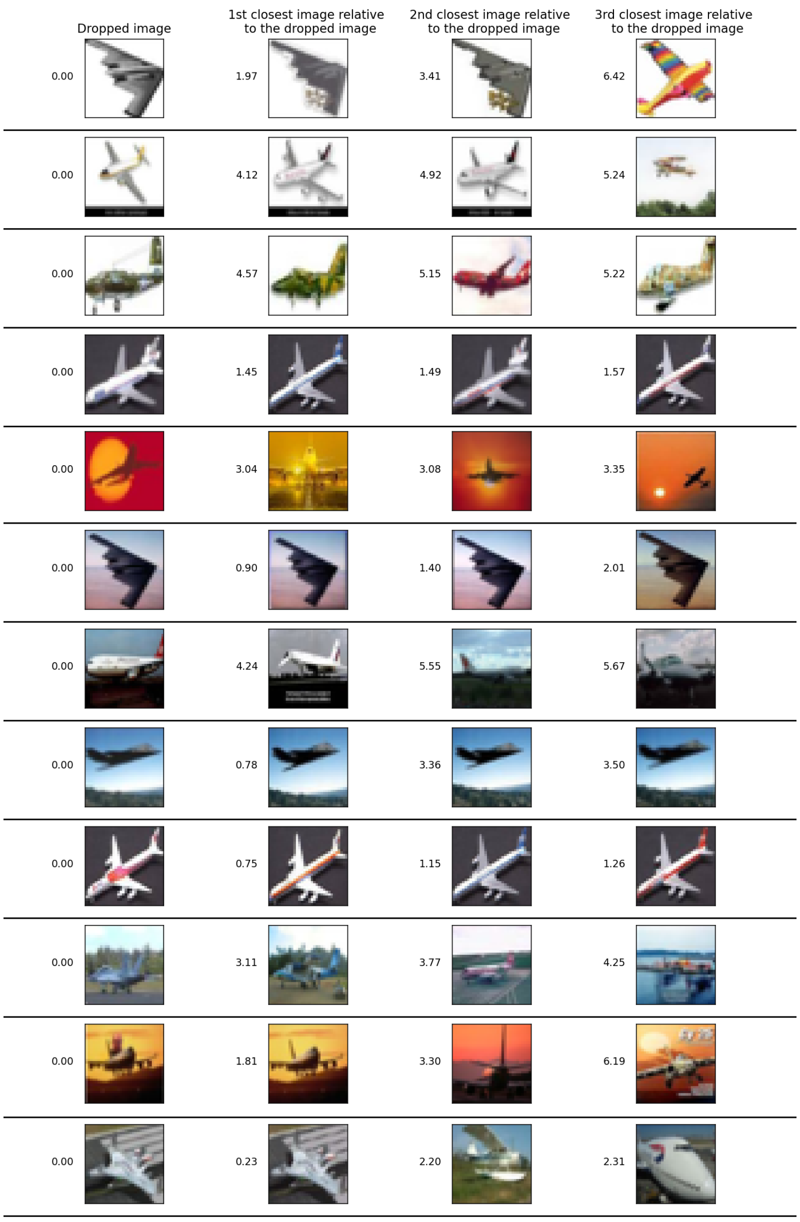


Figure 4.4: An autoencoder's feature space with nearest neighbors. The first column are the reference images. The other columns show the images closest to the reference image. The numbers indicate the euclidean distance relative to the reference image.

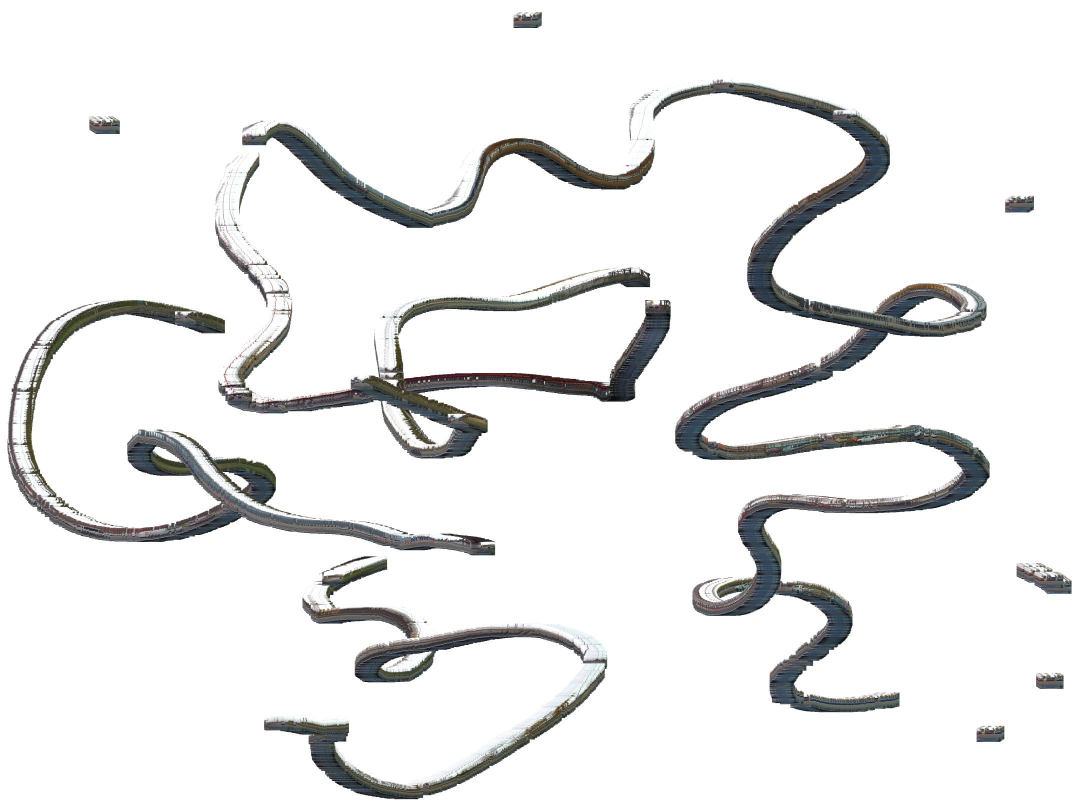


Figure 4.5: A video clip's frames converted to a point cloud with SWAV.

Chapter 5

Evaluating consumer applicability

I wanted to explore how to deploy such a sampling solution. Since the proposed method uses neural networks, most web frameworks don't work. They are not suitable for running neural networks or use accelerated hardware like GPUs. I decided to use a fully custom made solution, where the REST request are handled by Flask and the worker nodes containing the GPU is handled by Celery. This solution is scalable, since Celery is able to manage multiple nodes.

5.1 Web application

5.1.1 Physical structure

I decided to use python based web frameworks because it was convenient: most machine learning frameworks only support python. For this reason I am using Flask for the web framework. Since the bottleneck in a video processing system is usually computational power, we don't have the privilege to instantly process an uploaded video. For this reason, when a user uploads a video, it is immediately saved to a physical storage instead of storing it in memory. This helps us maintain a low memory usage even if users upload many videos. After that, the videos are queued up until there is a free worker node. The queue system is done by a Redis server, where arguments for a task (eg. video ids, downsample size) are kept. I use Celery for managing the worker nodes. Celery communicates with the Redis server to get the next task arguments and starts it on a worker node. Another reason I went with python (even though it can be slower compared to pure cpp frameworks), is that higher abstraction allows us to easily switch between types of hardware acceleration. This way, worker nodes can use GPU or CPU to run a model.

5.1.2 Logical structure

To run inference on a video, the videos are converted into images. During this conversion, even a small 1GB video can turn into 20GB of images due to not having video compression. For this very reason, the conversion is done frame by frame in memory. The frames are resized to have a width of 100 pixels while maintaining the same aspect ratio. The video frame rate is decreased to only 10FPS. After a frame is extracted from the video feed, an inference step is instantly run on the current frame. The inference is done by a pretrained model. The only thing stored in memory is the vector created during inference. After all the frames are converted to vectors, we can select the relevant points by the furthest

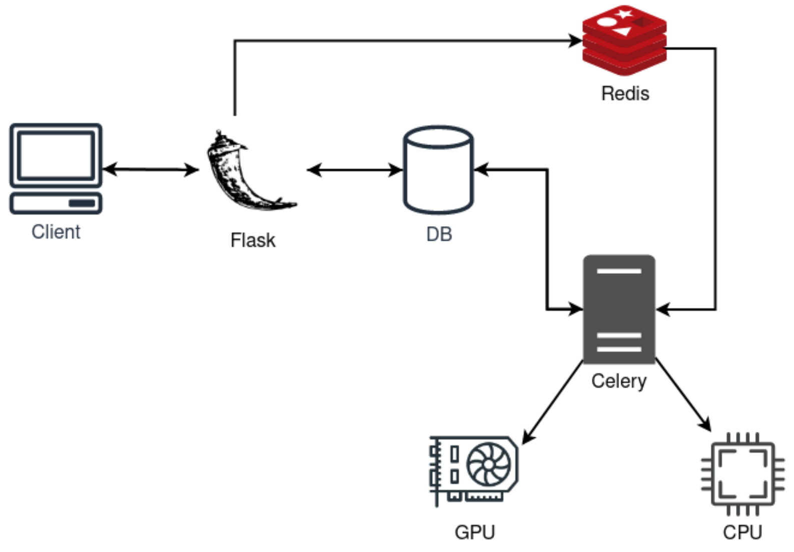


Figure 5.1: Physical structure of the web app.

neighbor algorithm. After the frame has been selected, we go through the video feed again and only save the relevant frames.

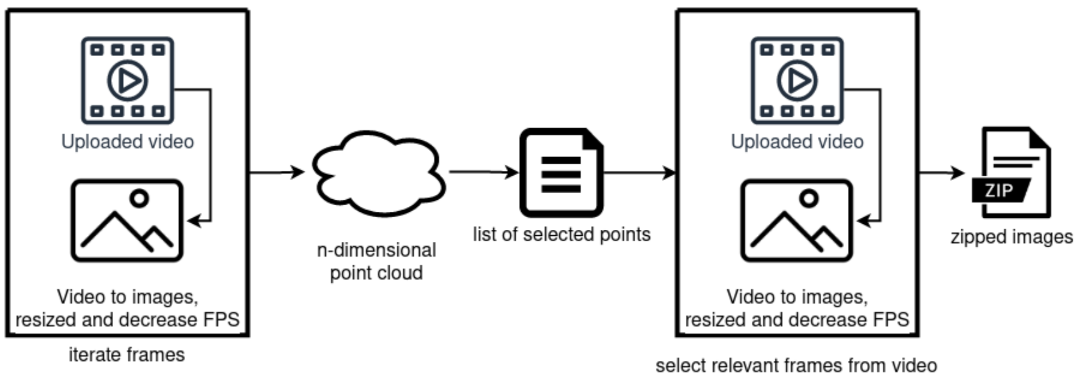

Uploaded video
Video to images, resized and decrease FPS
iterate frames
n-dimensional point cloud
list of selected points
Uploaded video
Video to images, resized and decrease FPS
select relevant frames from video
ZIP
zipped images

Figure 5.2: Logical structure during inference.

To train a model on the videos different approach is needed compared to inference. The first step is to create a training dataset from the video. The video is resized the same way as in inference and frame rate is also limited to 10FPS. After this, the whole video is stored on disk as a numpy array. Usually this process converts a 1GB video to 900MB-1400MB numpy arrays. After a model is trained on this dataset, we can run the same inference described above, but with the recently trained model.

5.2 Run time

The time it takes to run the full data selection pipeline is important for web applications. I measured how much time it takes to sample 1000 frames from a 71 minute video. The video is downsized to 100x56 resolution at 10FPS. The SWAV method was used with a resnet18 model and a furthest neighbors sampler. The measurement was done on a GTX 1080ti. These results are show on Table 5.1 and it turns out that even running on CPU

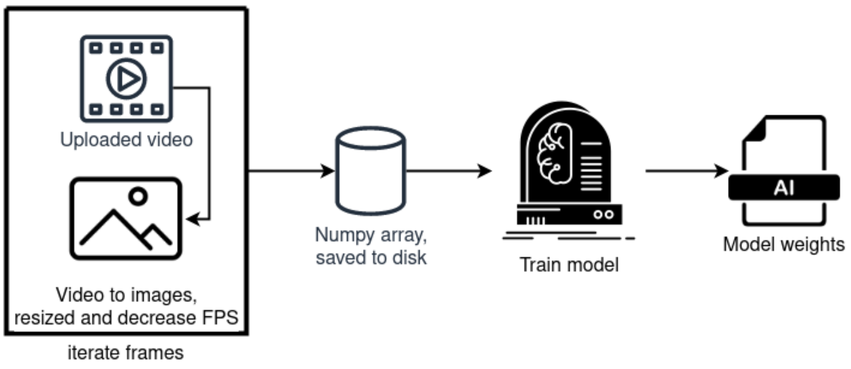

Uploaded video
Video to images, resized and decrease FPS
iterate frames
Numpy array, saved to disk
Train model
AI
Model weights

Figure 5.3: Logical structure during training.

is 2.86X faster than real time. This makes our method a great choice for low cost web applications.

Method	**Run time in minutes**
CPU	24.8
GPU	5.45

Table 5.1: Average inference time of subsampling a 71 minute video to a 1000 frames.

Chapter 6

Conclusion

In this research, I explored if it's possible to create a method for selecting datapoints which hold more information compared to randomly selected datapoints. To do so, feature vectors were created from images and sampled the feature vectors by only choosing the furthest points. Two solutions were explored for generating feature vectors: with an autoencoder and with SWAV. After the feature vectors are created from these two methods, the feature vectors are regularized with umap. Creating a fully unsupervised solution is possible and one method was particularly working well:

1. Use SWAV to generate the feature vectors from the images.
2. Transforming the feature vectors with umap.
3. Selecting the furthest points in this hyperspace.

The SWAV method was 0.417% better than the baseline method when the downsampling size is at 5000 on the CIFAR10 dataset. Since CIFAR10 is a balanced dataset, this is a great result.

I also explored how to deploy such a system for industrial uses. Requirement was to make the system able to handle videos and multiple requests. To handle multiple requests, a queue is used, where multiple worker nodes process the requests. Videos are notoriously difficult to handle with neural networks due to significantly higher data rates. To make processing the videos manageable, an aggressive image resizing and FPS rate reduction is applied. The videos are also handled in memory and processed frame by frame to bypass disk bandwidth bottlenecks.

Acknowledgments

This work was supported by NVIDIA Corporation with the donation of the Titan Xp GPU used for this research.

I wish to express my sincere gratitude to my supervisor Dr. Csaba Zainkó, for his suggestions, guidance and helpful comments to my work. His broad knowledge helped me build my own understanding of image processing, deep neural networks and unsupervised learning.

My work needed many GPU hours to run and this was possible to the Joker system managed by Dr. Imre Szeberényi. I'm grateful for his work and quick responses, when a problem arrived, which greatly helped that my experiments finished in a timely manner.

And finally, I would like to thank Anna Cserny for her patience to proofreading my work.

List of Figures

List of Tables

Bibliography

[1] Labeling costs. https://cloud.google.com/ai-platform/data-labeling/pricing#labeling_costs. Accessed: 2021-05-04.

[2] Dana H. Ballard. Modular learning in neural networks. In K. Forbus and H. Shrobe, editors, *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 279–284. San Francisco, CA: Morgan Kaufmann, 1987. URL http://www.mpi-sb.mpg.de/services/library/proceedings/contents/aaai87.html.

[3] Shai Ben-David, John Blitzer, Koby Crammer, and Fernando Pereira. Analysis of representations for domain adaptation. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems*, volume 19. MIT Press, 2007. URL https://proceedings.neurips.cc/paper/2006/file/b1b0432ceafb0ce714426e9114852ac7-Paper.pdf.

[4] John Blitzer, Ryan McDonald, and Fernando Pereira. Domain adaptation with structural correspondence learning. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 120–128, Sydney, Australia, July 2006. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/W06-1615.

[5] Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, Shyamal Buch, Dallas Card, Rodrigo Castellon, Niladri S. Chatterji, Annie S. Chen, Kathleen Creel, Jared Quincy Davis, Dorottya Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren Gillespie, Karan Goel, Noah D. Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, Omar Khattab, Pang Wei Koh, Mark S. Krass, Ranjay Krishna, Rohith Kuditipudi, and et al. On the opportunities and risks of foundation models. *CoRR*, abs/2108.07258, 2021. URL https://arxiv.org/abs/2108.07258.

[6] Stevo Bozinovski and A Fulgosi. The influence of pattern similarity and transfer of learning upontraining of a base perceptron b2. , . (original in croatian:utjecaj slicnosti likova i transfera ucenja na obucavanje baznog perceptrona b2). *Proc. Symp.Informatica 3-121-5, Bled*, 1976.

[7] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher

Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *CoRR*, abs/2005.14165, 2020. URL `https://arxiv.org/abs/2005.14165`.

[8] Mathilde Caron, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin. Unsupervised learning of visual features by contrasting cluster assignments. *CoRR*, abs/2006.09882, 2020. URL `https://arxiv.org/abs/2006.09882`.

[9] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey E. Hinton. A simple framework for contrastive learning of visual representations. *CoRR*, abs/2002.05709, 2020. URL `https://arxiv.org/abs/2002.05709`.

[10] Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013. URL `https://proceedings.neurips.cc/paper/2013/file/af21d0c97db2e27e13572cbf59eb343d-Paper.pdf`.

[11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. 2016. URL `http://www.deeplearningbook.org`.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL `http://arxiv.org/abs/1512.03385`.

[13] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross B. Girshick. Momentum contrast for unsupervised visual representation learning. *CoRR*, abs/1911.05722, 2019. URL `http://arxiv.org/abs/1911.05722`.

[14] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *CoRR*, abs/2006.11239, 2020. URL `https://arxiv.org/abs/2006.11239`.

[15] Hal Daumé III and Daniel Marcu. Domain adaptation for statistical classifiers. *CoRR*, abs/1109.6341, 2011. URL `http://arxiv.org/abs/1109.6341`.

[16] Tero Karras, Miika Aittala, Janne Hellsten, Samuli Laine, Jaakko Lehtinen, and Timo Aila. Training generative adversarial networks with limited data. *CoRR*, abs/2006.06676, 2020. URL `https://arxiv.org/abs/2006.06676`.

[17] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[18] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction, 2020.

[19] Luis Perez and Jason Wang. The effectiveness of data augmentation in image classification using deep learning. *CoRR*, abs/1712.04621, 2017. URL `http://arxiv.org/abs/1712.04621`.

[20] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. *CoRR*, abs/2102.12092, 2021. URL `https://arxiv.org/abs/2102.12092`.

[21] Yuji Roh, Geon Heo, and Steven Euijong Whang. A survey on data collection for machine learning: a big data - AI integration perspective. *CoRR*, abs/1811.03402, 2018. URL `http://arxiv.org/abs/1811.03402`.

[22] Kate Saenko, Brian Kulis, Mario Fritz, and Trevor Darrell. Adapting visual category models to new domains. In Kostas Daniilidis, Petros Maragos, and Nikos Paragios, editors, *Computer Vision – ECCV 2010*, pages 213–226, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. ISBN 978-3-642-15561-1.

[23] Connor Shorten and Taghi Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6, 07 2019. DOI: `10.1186/s40537-019-0197-0`.

[24] Samarth Sinha, Sayna Ebrahimi, and Trevor Darrell. Variational adversarial active learning. *CoRR*, abs/1904.00370, 2019. URL `http://arxiv.org/abs/1904.00370`.

[25] Kihyuk Sohn. Improved deep metric learning with multi-class n-pair loss objective. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL `https://proceedings.neurips.cc/paper/2016/file/6b180037abbebea991d8b1232f8a8ca9-Paper.pdf`.

[26] Antonio Torralba and Alexei A. Efros. Unbiased look at dataset bias. In *CVPR 2011*, pages 1521–1528, 2011. DOI: `10.1109/CVPR.2011.5995347`.

[27] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008. URL `http://jmlr.org/papers/v9/vandermaaten08a.html`.

[28] Riccardo Volpi, Hongseok Namkoong, Ozan Sener, John C. Duchi, Vittorio Murino, and Silvio Savarese. Generalizing to unseen domains via adversarial data augmentation. *CoRR*, abs/1805.12018, 2018. URL `http://arxiv.org/abs/1805.12018`.

[29] Zhirong Wu, Yuanjun Xiong, Stella X. Yu, and Dahua Lin. Unsupervised feature learning via non-parametric instance-level discrimination. *CoRR*, abs/1805.01978, 2018. URL `http://arxiv.org/abs/1805.01978`.

[30] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. *CoRR*, abs/1801.03924, 2018. URL `http://arxiv.org/abs/1801.03924`.

Appendix

A.1 Visualizing the n-dimensional space created by the autoencoder.

When creating the autoencoder solution for the sampling solutions, one main debugging tool for us was visualizing how and where the autoencoder places the images. During this visualization it turned out how well the model gathers similar images to the same place. While the primary color places a big role to situate the image shown on Fig A.1.3, the planes are also separated by shapes seen on Fig A.1.2.

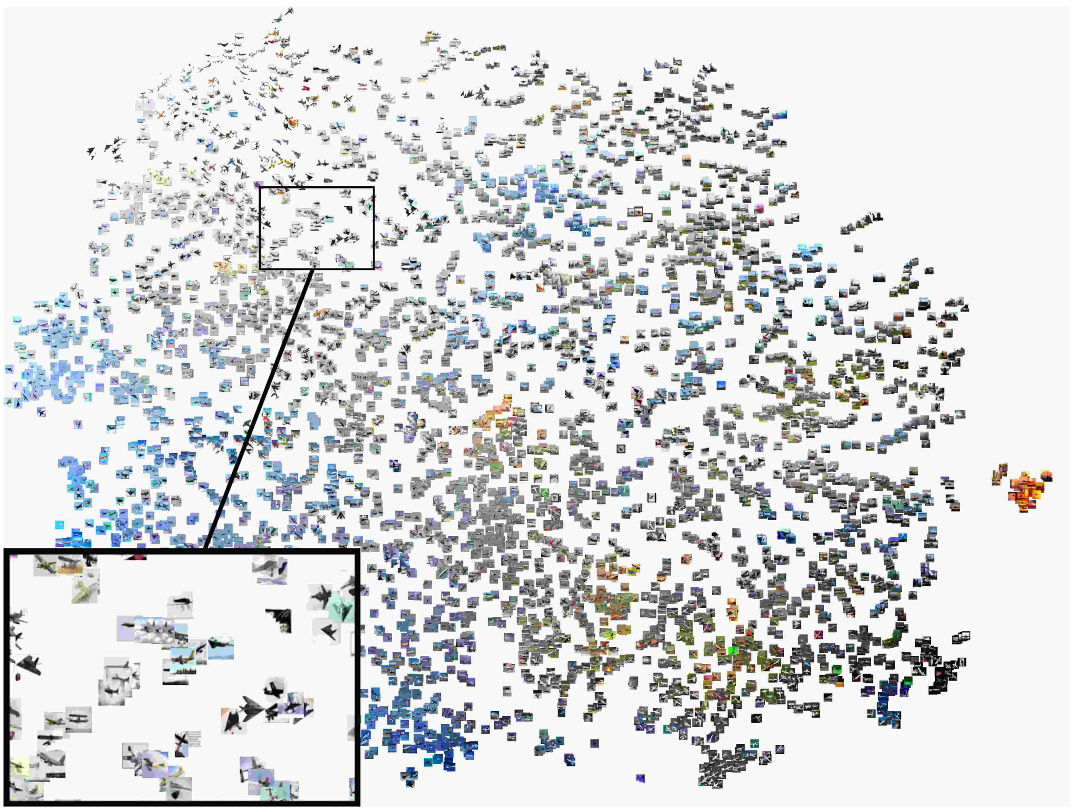
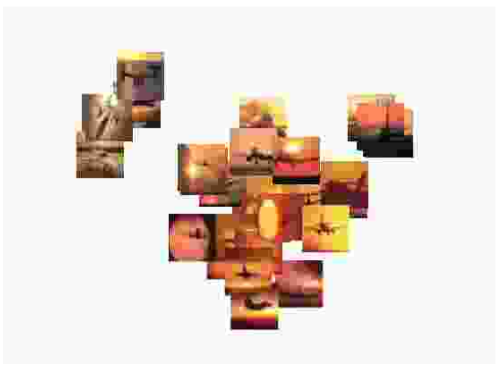
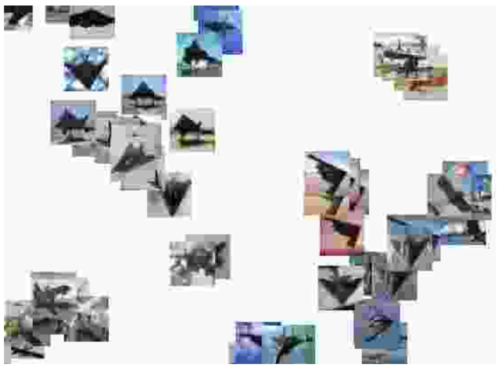


Figure A.1.1: The *plane* class placed in the hyperspace created by obtaining a representation vector from the autoencoder and then reduced to 2 dimensions with t-SNE.


(a) Planes in sunset


(b) Planes with deltoid shape

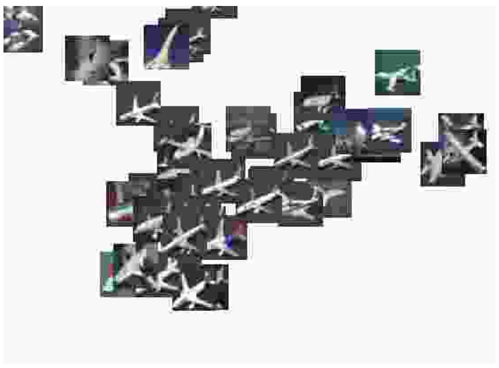

(c) Planes landed

Figure A.1.2: Autoencoder places similar images close to each other.

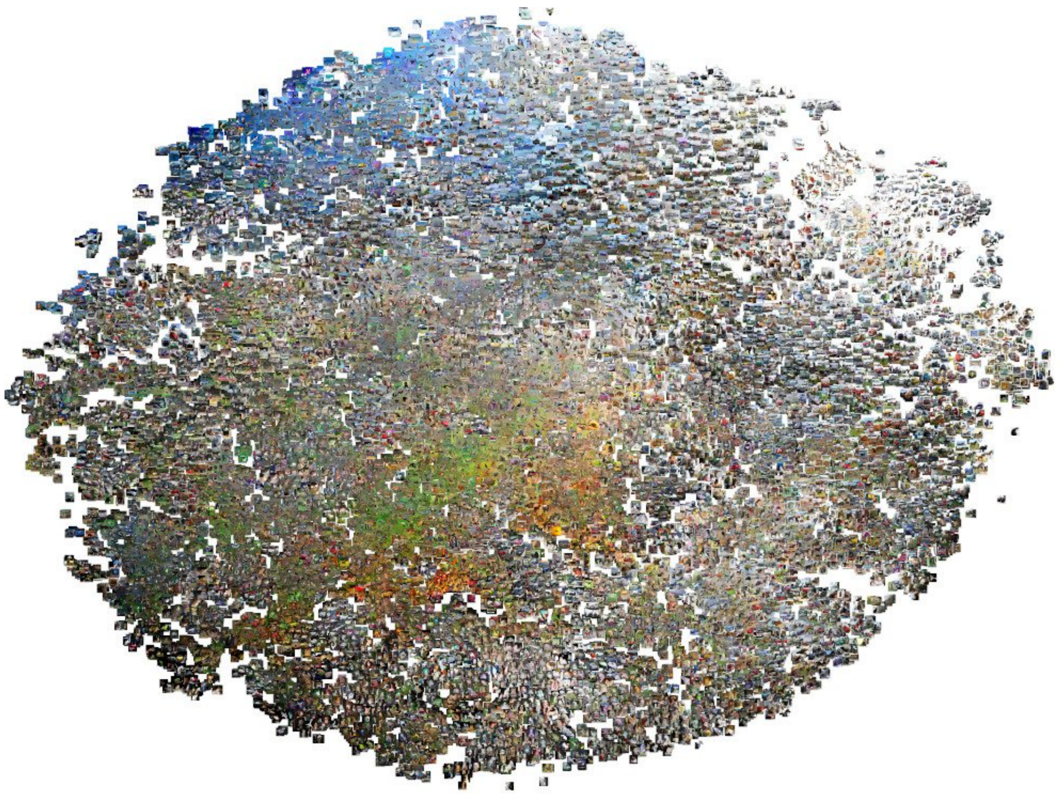


Figure A.1.3: All the 50000 training images placed in the hyperspace. The 2d representation is created by reducing the latent vector from the autoencoder and then reduced to 2 dimensions with t-SNE.[2]

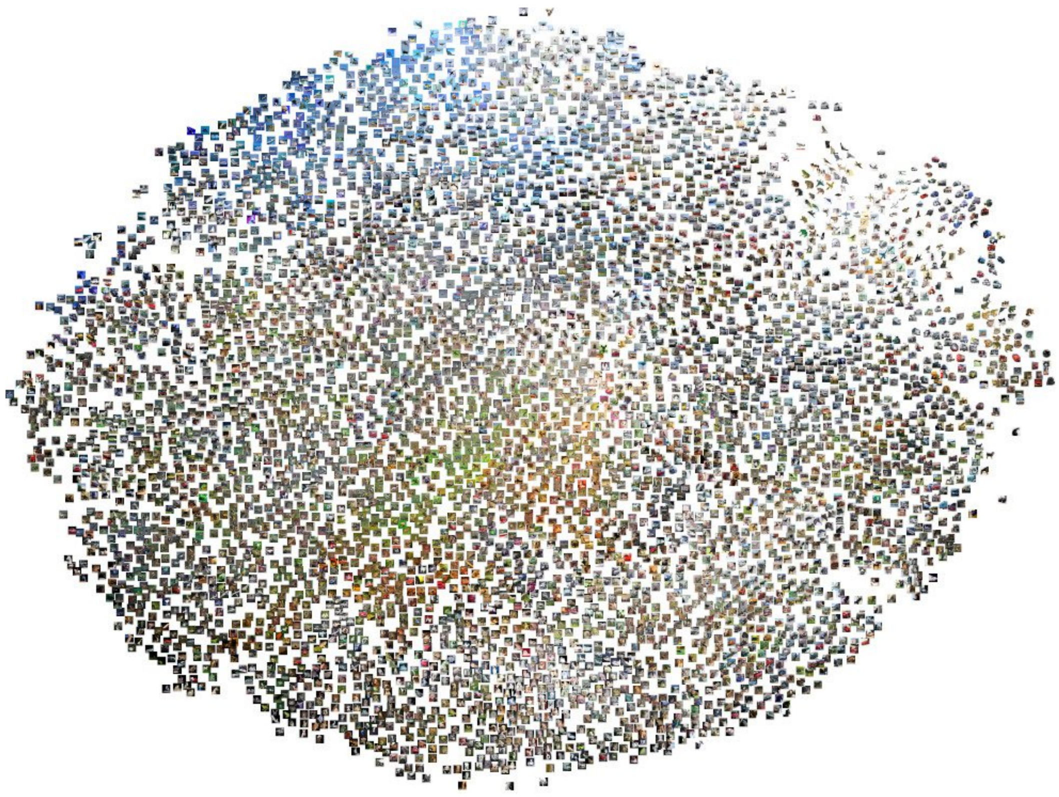


Figure A.1.4: The selected 5000 images by the furthest neighbours algorithm.

A.2 Point clouds in 2d

During visualizing the feature vector point clouds from the autoencoder and the pretrained classifier, it was quite conspicuous how different are the representations. The pretrained classifier's feature vectors are very structured, while the autoencoder's feature vectors are more evenly distributed.

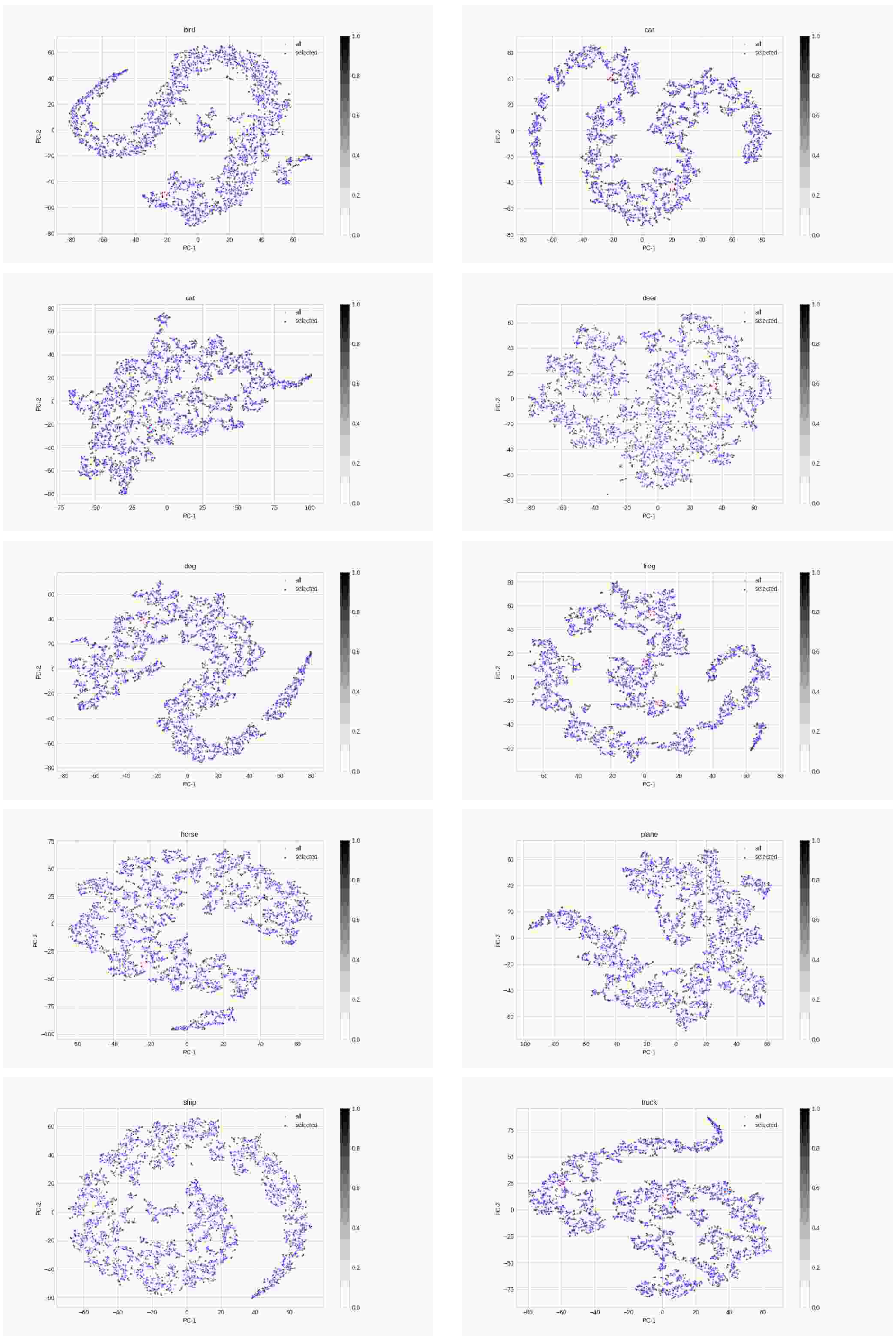


Figure A.2.1: Feature vectors from a trained classifier and reduced dimensions with t-SNE.

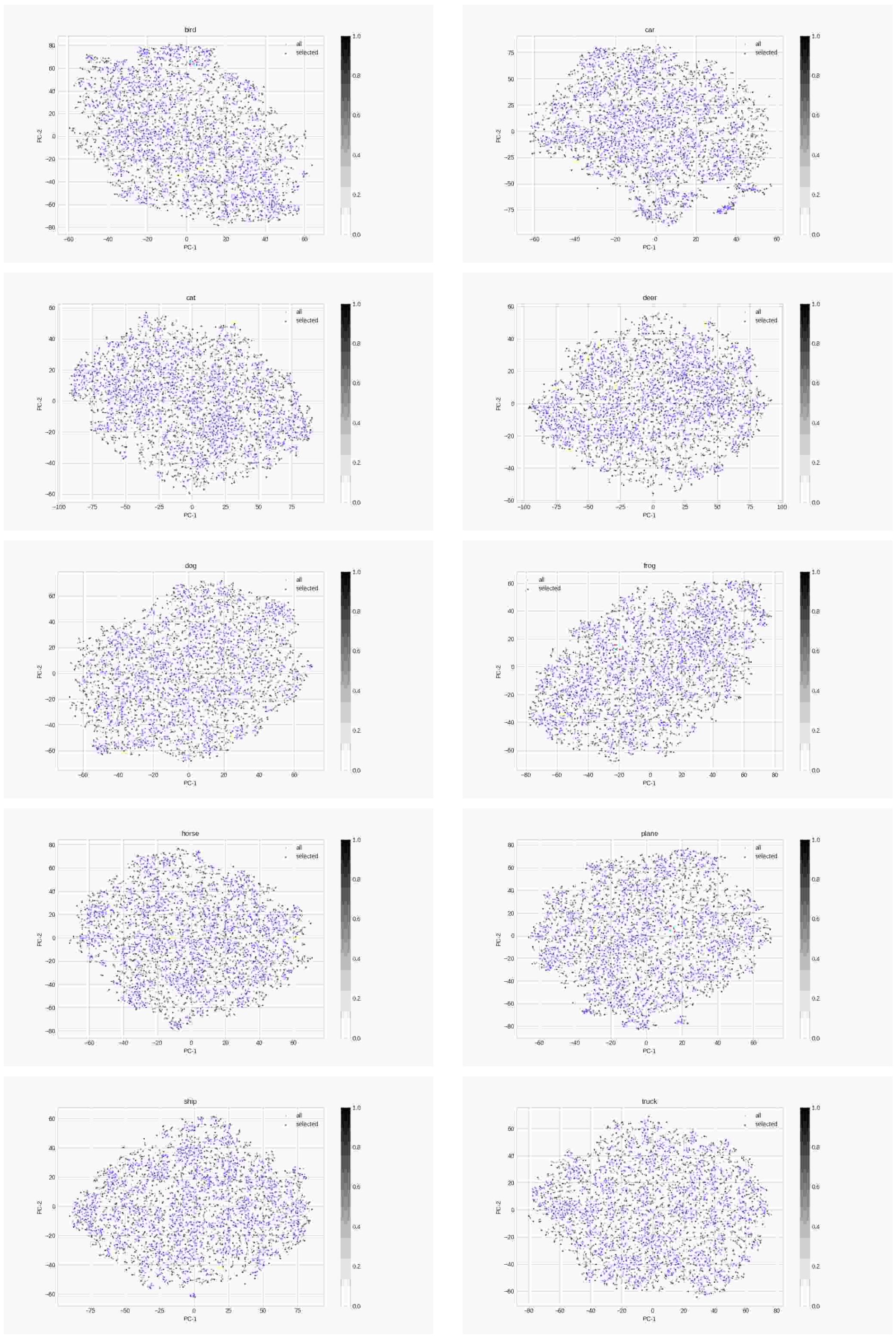


Figure A.2.2: Obtained latent vector at the bottleneck of the autoencoder and reduced dimensions with t-SNE.

A.3 Additional links

The source code for the project is available at:

`https://gitlab.com/bruno.englert/dissimilarity_sampling`

Video made with the SWAV method is available at:

`https://www.youtube.com/watch?v=s2kzjI_v5gQ`

`https://www.youtube.com/watch?v=vSml3RE7p5w`