Active Learning for Semi-Supervised image classification

Author:
Diego Coello de Portugal
312838

Supervisors:
Thorben Werner
Lars Schimdt-Thieme

31st October 2023

Thesis submited for
Master of Science in Data Analytics

Wirtschaftsinformatik und Maschinelles Lernen
Stiftung Universität Hildesheim
Universitatsplätz 1, 31141 Hildesheim

Statement as to the sole authorship of the thesis:

Active Learning for Semi-Supervised image classification. I hereby certify that the master's thesis named above was solely written by me and that no assistance was used other than that cited. The passages in this thesis that were taken verbatim or with the same sense as that of other works have been identified in each individual case by the citation of the source or the origin, including the secondary sources used. This also applies for drawings. sketches, illustration as well as internet sources and other collections of electronic texts or data, etc. The submitted thesis has not been previously used for the fulfilment of a degree requirements and has not been published in English or any other language. I am aware of the fact that false declarations will be treated as fraud.

Date, City Signature

Abstract

Machine learning models are algorithms highly dependent on the amount and quality of data they are given. In real world scenarios, it's difficult to get quality and quantity of labeled data. Nevertheless, unlabeled data is much easier to obtain and with a suitable usage, it could help to improve the models. In this sense, Active Learning and Semi-Supervised learning focus on how to use efficiently the unlabeled data. Active Learning aims to find the unlabeled examples that would improve the most the model when given labeled for training, while Semi-supervised learning focuses on using unlabeled data to increase the model performance in an unsupervised manner.

In this Master Thesis we aim to combine ideas from both settings in order to obtain a more general way to train the image classifiers and to label new instances. In particular, we will use a pre-trained encoder to improve the classifier accuracy and study the different Active Learning strategies with the idea of generating a surrogate model that is able to predict the expected increase in performance after labeling a given instance.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

This project aims to define a surrogate model to sample new instances in the Active Learning regime. This surrogate model will query new instances that will be labeled before using them to improve the classifier performance. Additionally, different strategies from Semi-supervised Learning will be applied in order to improve the classifier performance on the labeled dataset.

There are several studies on the usage of a loss prediction model in order to do the sampling in the Active Learning regime. Nevertheless, they focus on predicting the loss value of a specific instance for a given model with fixed weights. In our case, we aim to predict the classifier accuracy's increase on the validation set after retraining it with that new instance, which is the true objective the Active Learning problem aims to solve. Notice that values with a high loss might be outliers, which could hinder the performance of the model when added to the labeled dataset. Some previous studies have found that using a feature space representation helps the model to generalize better. This feature extraction could be done with autoencoders or pretext tasks. However, these features are mainly used for the classifier in order to reduce the problem complexity. In this project we will also use those features in the query selection part of the Active Learning procedure.

Lastly, we will experiment with different training procedures. In particular, we will use a closed-form approximation for the softmax regression as a classification model. The idea is to reduce the variance that different part of the standard gradient based training has, which could lead to a more robust learning and less noisy problem for the surrogate model.

Chapter 2

Related Work

2.1 Active Learning

Active Learning is a subfield of Machine Learning focused on selecting new instances to be labeled and added to the dataset in order to increase the performance of the model. We can distinguish three main strategies:

- **Membership Query Synthesis**: Also known as *membership queries* [Angluin, 1988], is a setting where the model may request labels for any unlabeled instance in the input space, including (and typically assuming) queries that the learner generates de novo, rather than those sampled from some underlying natural distribution. Nevertheless, an issue that may arise is that some of the queried instances generated by the model not correspond to any output (class). For example, an image that doesn't correspond to any classes.

- **Stream-Based Selective Sampling**: Also known as *selective sampling* [Atlas et al., 1989], is based in the assumption that obtaining an unlabeled instance is free (or inexpensive), so it can first be sampled from the actual distribution, and then the model can decide whether or not to request its label.

- **Pool based sampling**: This method [Lewis and Gale, 1994] assumes that there is a small set of labeled data L and a large pool of unlabeled data U available. Queries are selectively drawn from the pool, which is usually assumed to be closed (i.e., static or non-changing), although this is not strictly necessary.

In this thesis we will focus on *Pool-based Active Learning*, where a labeled and an unlabeled pool of data has been collected before the experiments. The Active Learning procedure will select the instances (x_u) from the unlabeled pool to label. Some of the main strategies used in Active Learning are:

- Random [Settles, 2009]: Randomly pick an instance to label. It is used to have a baseline to compare against, since *Active Learning* procedures are not always giving an improvement when compared against it.

  $argmax_{x_u} Random\_Value(x_u) \sim Uniform()$

- Entropy [Settles, 2009]: It tries to find instances near the model decision boundary to sample using Shannon Entropy.

  $argmax_{x_u} \sum_c \hat{y}_c(x_u) log(\hat{y}_c(x_u))$

- Least Confidence [Settles, 2009]: It samples the instance where the predicted class has the lowest probability assigned.

  $argmin_{x_u} \max_c \hat{y}_c(x_u)$

- Margin [Settles, 2009]: Aims for instances where the highest probability class and the second highest probability class have the lowest difference.

  $argmin_{x_u} \max_{c_1} \hat{y}_{c_1}(x_u) - \max_{c_2, c_2 \neq c_1} \hat{y}_{c_2}(x_u)$

- Coreset [Sener and Savarese, 2018]: Looks for instances that are the furthest away from all the labeled instances (x_l) given a specific metric ($\triangle$), normally an L_p distance metric. An embedding space of a hidden layer from the model is often used as the representation for each instance instead of the raw data.

  $argmax_{x_u} \min_{x_l} \triangle(x_l, x_u)$

- Learned loss [Yoo and Kweon, 2019, Shukla and Ahmed, 2021]: A surrogated model ($\hat{Loss}$) is used to predict the expected loss of a given instance. It is slightly modified to predict a rank of the instances losses instead of the actual loss value.

  $argmax_{x_u} \hat{Loss}(\hat{y}, x_u)$

- Badge [Ash et al., 2020]: Calculates the gradient of the cross entropy loss for the last layer weights (θ_{out}) using the most confident label as the true label. For batch Active Learning, the method continues using KMeans++ [Arthur, 2007] as a way to select the remaining sampled instances:

  $argmax_{x_u} \frac{\delta}{\delta \theta_{out}} CE(\hat{y}(x_u), max_c \hat{y}_c(x_u))|_{\theta=\theta_t}$

2.2 Semi-Supervised and Self-Supervised Learning

Semi-Supervised Learning [Ouali et al., 2020] is halfway between *Supervised* and *Unsupervised Learning*. In addition to unlabeled data, the algorithm is provided with some supervision information but not necessarily for all examples. In particular, this thesis will be focused on *Self-Supervised Learning* [Balestriero et al., 2023], which defines a pretext task based on unlabeled inputs to produce descriptive and intelligible representations. Some of this pretext tasks are based on splitting the image into tiles to then solve a jigsaw puzzle of the image fragments [Noroozi and Favaro, 2017] or coloring a black and white image [Zhang et al., 2016].

In more recent times, this pre-text tasks tend to be more generic and to be focused around augmentations. The main idea is to try to match the representation of related/augmented inputs with *Contrastive Learning* [Bachman et al., 2014]. The most used variation is *Consistency regularization* [Chen et al., 2020, Kim et al., 2021], which aims to reduce the difference between augmentations of the same input (Eq. 2.1).

$$l_{i,j} = -log\frac{exp(sim(z_i, z_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{I}[k \neq i] exp(sim(z_i, z_k)/\tau)} \tag{2.1}$$

In Equation 2.1 z_i is the embeded representation of input x_i, $sim(\cdot, \cdot)$ is a similarity function between representations (example: cosine similarity) and $\mathbb{I}[\cdot]$ is a function that outputs 1 if the condition given is true and 0 otherwise.

2.3 Semi-Supervised Active Learning

In the pool-based Active Learning scenario, there is no reason why one shouldn't use the unlabeled data in a semi-supervised way to improve the model's performance.

One idea would be to use this unlabeled data to improve the Active Learning procedure by generating an embedding with the usage of the unlabeled data. Having an embedding allows to have a more meaningful representation of the data that holds more information, reducing noise and the end task complexity. Some methods are:

- SSALConsistency [Gao et al., 2020]: Aside from adding a KL-divergence regularization term between augmentations (x,x^1,...,x^N) of the unlabeled data as a regularization term, this method selects the instances with the highest sum of classes variances prediction:
  $argmax_{x_u} \sum_{c=0}^{K} Var[\hat{y}_c(x_u), \hat{y}_c(x_u^1), ..., \hat{y}_c(x_u^N)]$

- Typiclust [Hacohen et al., 2022]: It clusters the instances in $|X_l|+B$ clusters, to then select 1 instance of each of the B clusters without any labeled instances. The sample selected is the one with the highest typicality in the cluster. Similarly to *coreset*, this is not done in the input space but in an embedded representation.
  $argmax_{x_u} typicality(x_u) = argmax_{x_u}(\frac{1}{K} \sum_{x_i \in KNN(x_u)} ||x_i - x_u||^2)^{-1}$

- Diff2AugKmeans [Song et al., 2019]: The authors propose to use margin or least confidence sampling (Sec. 2.1) and average the result over K different augmentation versions of the given instance. Additionally, in order to ensure diversity in batch sampling, they propose to cluster the data using K-means and sample from each cluster a number of examples proportional to the cluster size with a fixed number of clusters.

2.4 Image Classification models

ResNet

ResNet [He et al., 2015, He et al., 2016] is a convolutional architecture (see Section 3.1) conformed of multiple residual blocks, followed by a global average pooling layer and a dense layer. Each of these residual blocks receives an input which is then processed by a sequence of batch normalization layers, convolutional layers and activation functions to produce the output.

Additionally, the original input is also added before passing the output to the next block with a *skip connection* (Fig 3.2). The number of residual blocks is what gives name to the model (ResNet18, ResNet50, ResNet101, etc.). Lastly, the global averaging pooling layer is used to remove the spatial information before predicting the classes, i.e., it averages over the height and width dimensions, leaving only the channels as an input for the final dense layer.

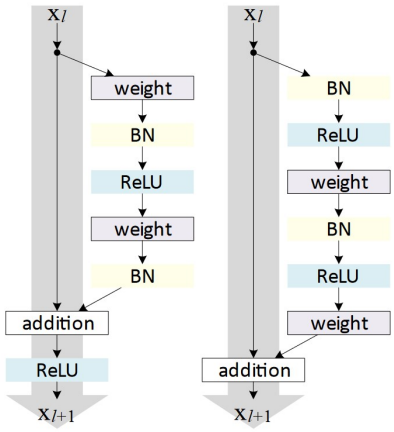


Figure 2.1: ResNet block architecture comparison. (a) is the original implementation *pre-activation*. (b) is a revision of the ResNet block architecture with better performance called *pos-activation*. [He et al., 2016]

DINO

DINO [Caron et al., 2021b, Oquab et al., 2023] is a transformer based model (see Section 3.2). The idea is to train a model to generate an embedding in a self-supervised fashion. This is done using a teacher-student setting with an exponential moving average of the teacher's weights based on the student weights.

More formally, the student network is updated with backpropagation on the entropy loss between its predictions and the teacher's prediction. Then the teacher model weights are updated as a weighted sum of the teacher's weights and the student's weights: $\theta_{teacher}^{(t+1)} = k * \theta_{teacher}^{(t)} + (1-k) * \theta_{student}^{(t+1)}$. The authors also add a temperature factor for the student and teacher's network before the softmax function (tps, tpt), as well as a center factor (C).

Additionally, they construct different distorted views or crops of an image with multicrop strategy [Caron et al., 2021a]. More precisely, for a given image a set V of different views is generated. This set contains two global views (x_g^1 and x_g^2) and several local views of smaller resolution. All the crops are passed through the student while only the global views are passed through the teacher, therefore encouraging 'local-to-global' correspondences.

Algorithm 1: DINO pseudocode w/o multi-crop. [Caron et al., 2021b]

```
# gs, gt: student and teacher networks
# C: center (K)
# tps, tpt: student and teacher temperatures
# l, m: network and center momentum rates
C = 0
gt.params = gs.params
for x in loader: # load a minibatch x with n samples
    x1, x2 = augment(x), augment(x) # random views

    s1, s2 = gs(x1), gs(x2) # student output n-by-K
    t1, t2 = gt(x1), gt(x2) # teacher output n-by-K

    loss = H(t1, s2, C)/2 + H(t2, s1, C)/2
    loss.backward() # back-propagate

    # student, teacher and center updates
    update(gs) # SGD
    gt.params = l*gt.params + (1-l)*gs.params
    C = m*C + (1-m)*cat([t1, t2]).mean(dim=0)

def H(t, s, C; tps, tpt): #tps, tpt: temperature hyperparmeters
    t = t.detach() # stop gradient
    s = softmax(s / tps, dim=1)
    t = softmax((t - C) / tpt, dim=1) # center + sharpen
    return - (t * log(s)).sum(dim=1).mean()
```

After that, a linear layer is added to this embedding to generate the classification (downstream task), using a freezed version of the embedding model.

On the second version of DINO model (DINOv2 [Oquab et al., 2023]), the authors also decided to untie the head weights between the loss objectives (Image-level loss vs Patch-level loss), use Sinkhorn-Knopp centering [Ruan et al., 2023], use KoLeo regularizer [Sablayrolles et al., 2019] and start with a lower resolution in the beginning of the training and increase it only in the end for lower training time [Touvron et al., 2022].

2.5 XGBoost

XGBoost [Chen and Guestrin, 2016] is a tool to optimize Boosting Decision Tree methods. Boosting Decision Tree methods are characterized by using a set of classification and regression decision trees (CART) in order to predict the output. It can be expressed as follows:

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), f_k \in \mathcal{F}$$

where K is the number of decision trees and $\mathcal{F}$ is the set of all CARTs. The loss function that is optimized is defined as:

$$\sum_{i=1}^{N} loss(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

where $loss(\cdot)$ refers to the training loss and $\Omega(\cdot)$ is the regularization term. In order to learn all the different K models, an iterative process is done where one CART is learned at a time:

$$\hat{y}_i^{(0)} = 0 \tag{2.2}$$
$$\hat{y}_i^{(1)} = f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i) \tag{2.3}$$
$$\hat{y}_i^{(2)} = f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i) \tag{2.4}$$
$$... \tag{2.5}$$
$$\hat{y}_i^{(K)} = f_{K-1}(x_i) + f_K(x_i) = \hat{y}_i^{(K-1)} + f_K(x_i) \tag{2.6}$$

Using this iterative calculation, the objective at each step t can be defined as follows:

$$obj^{(t)} = \sum_{i=1}^{N} loss(y_i, \hat{y}_i^{(t)}) + \sum_{k=1}^{t} \Omega(f_k) = \sum_{i=1}^{N} loss(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + \epsilon$$

This objective has a constant factor ϵ independent of the model. In order to solve this for any loss, a second order approximation is used with the Taylor expansion:

$$obj^{(t)} = \sum_{i=1}^{N} [\frac{\delta loss(y_i, \hat{y}_i^{(t-1)})}{\delta \hat{y}_i^{(t-1)}} f_t + \frac{1}{2} \frac{\delta^2 loss(y_i, \hat{y}_i^{(t-1)})}{\delta^2 \hat{y}_i^{(t-1)}} f_t^2] + \Omega(f_t) + \epsilon$$

With this any two times differentiable function can be used as the loss function. Lastly, the regularization term can be defined taking into account the definition of decision tree:

$$f_t(x_i) = w_{q(x_i)}, w \in R^T, q : R^M \rightarrow \{1, ..., T\}$$

where T is the number of leaves, R^M is the input domain, q is the function that assigns each input to a leave and w is the value assigned to each leaf. With this the regularization term can be defined as:

$$\Omega(f_t) = \gamma T + \frac{\lambda}{2} \sum_{i=1}^{T} w_i^2$$

where λ and γ are hyperparameters that define the trade-off between the training loss, the training step and the score values in each leaf.

Chapter 3

Background

3.1 Convolution based models for Image Classification

Convolutions

Convolutional layers [LeCun et al., 1989, Krizhevsky et al., 2012] are based on the idea of generating a transition equivariant layer, i.e., shifting the input 'image' slightly should return the same output shifted the same way.

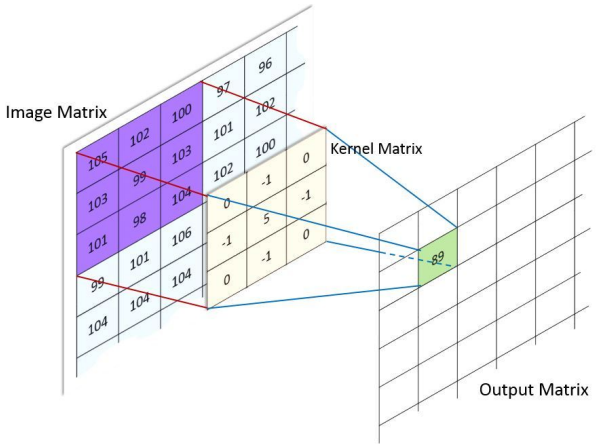


Figure 3.1: Convolutional layer example. [Nameer Hirschkind, 2023]

This is done by applying a kernel matrix over all the input image/matrix, shifting it horizontally and vertically. This shift is done 1 pixel at a time (stride 1), but it could be increased to reduce the output matrix size. Notice that for stride greater than 1 the convolutional layer looses the equivariant

properties for some shifts. In particular, it is no longer shift invariant for image shifts with non zero modulo for the kernel matrix shift.

Additionally, this layer also allows to have a bias term as their dense layer counterpart or to add padding (zero padding, reflect, replicate, etc.), i.e., to add a certain amount of columns and rows in the border (0 values for zero-padding) so that the output matrix has the same dimension as the input. Depending on the task and data, the padding could also reflect the input (reflect padding), replicate the last column or row (replicate padding), repeat the input with a shift (circular padding), etc.

Skip connection

The skip connection function adds the original input to the output of a layers' sequence or block before passing the output to the next block (Fig. 3.2). The main purpose of this skip connection is to allow a better flow of the gradients when updating the model weights with backpropagation. Without this skip connection, deep models tend to collapse due to gradient explosion/vanish. This leads to the model returning always the same output (*collapsing*) and not solving the task properly.

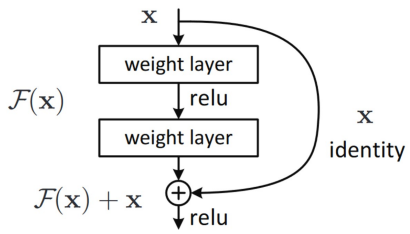


Figure 3.2: Skip connection architecture. [He et al., 2015]

3.2 Transformers for Image Classification

Transformers models ([Vaswani et al., 2017]) are based on attention layers. The basic version of the attention layer is known as the *self-attention* layer. It calculates the *key* (K), *query* (Q) and *value* (V) of the given input to then combine it and get the final output. Additionally, there exists also the *multihead-attention*, which has several self-attention layers that get their outputs linearly combined in order to get the output of the *multihead-attention* layer.

Attention layers

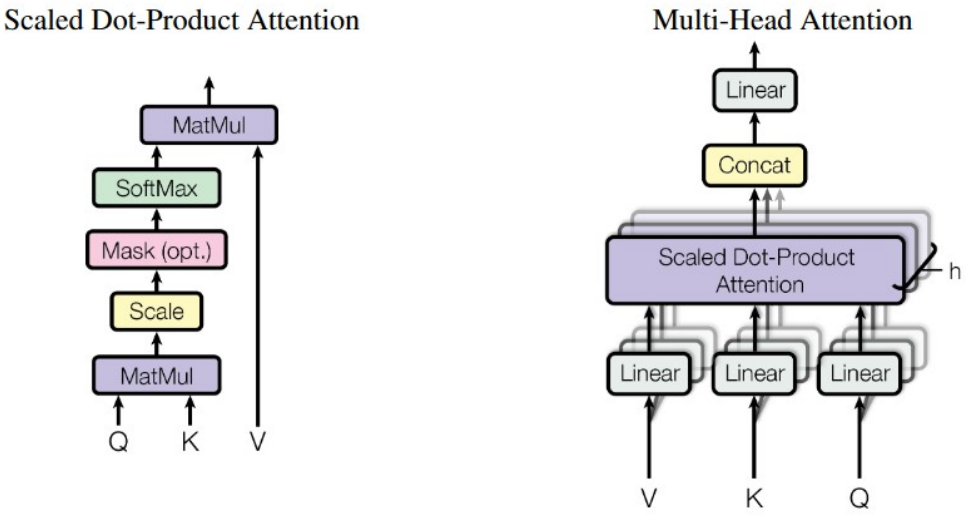


Figure 3.3: Self-attention layer structure on the left image and Multihead-attention layer structure on the right image. [Source: [Vaswani et al., 2017]]

In a more rigorous way, the self-attention layer (Eq. 3.1) calculates the similarity between the *keys* and the *queries*, to then pass the scaled result through a softmax operation and finally multiply it by the corresponding *value*. The multihead-attention layer (Eq. 3.2) concatenates the output of several self-attention layers to then apply a linear operation to it, i.e., it is a linear combination of the self-attention outputs.

$$\begin{aligned} selfAttention(X) &= softmax(\frac{QK^T}{\sqrt{d}})V = \\ &= softmax(\frac{(W_Q X)(W_K X)^T}{\sqrt{d}})(W_V X) \end{aligned} \tag{3.1}$$

$$multiheadAttention(X) = \sum_{i=1}^{H} W_i(selfAttention_i(X)) \tag{3.2}$$

Transformer

On the other hand, the transformers are complete models that receive the data and returns the prediction. Transformers are conformed of 2 main parts: the *encoder* and the *decoder*. Both of them are a concatenation of N times their corresponding block.

The *encoder* block always receives as input the output of the previous block or the input for the first block. Each block has the same structure composed of a multi-head attention or feed-forward layer with a skip connection followed by a fully connected layer with an *Add & Norm* block. The *Add & Norm* block consists in adding the original output (*skip connection*) and then do a *Layer Normalization* ([Ba et al., 2016]). More formally, the idea is to perform $LayerNorm(X + SubBlock(X))$ where the *SubBlock* can be a multihead-attention layer or a fully connected layer.

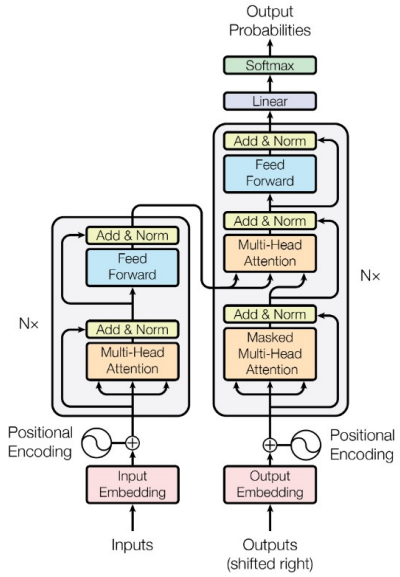


Figure 3.4: Transformer model architecture. The left part is known as the *encoder* while the right part is known as the *decoder*. [Source: [Vaswani et al., 2017]]

The *decoder* follows a similar idea as the *encoder*. Nevertheless, it receives not only as input the output of the previous *decoder block* but also the output of the *encoder* for the second block ('*encoding*'). There is an additional operation in the middle of the decoder block that receives the output of the previous sub-block to perform cross-attention, where the queries correspond to the previous result and the keys and values use the encoder result.

Lastly a *Add & Norm* layer is perform after adding the previous result (skip connection). This result is then added with the input without the encoding and then normalized, i.e., the operation performed is $LayerNorm(X + MultiCrossHeadAttention(encoding_{value}, encoding_{key}, X_{query}))$. The other 2 sub-blocks are performed analogously to their encoder counterparts.

The only part missing is the *Positional Encoding*, which is a value added in order to give information about the position in which the elements are given. The importance of this value lies in the equivariance property of the attention networks, i.e, changing the input order will only change the output order but not the output values. Therefore, in order to pass the information regarding the position a small value is added that takes into account the token position (pos) and the dimension in the token (i). The original positional encoding from [Vaswani et al., 2017] uses a sinuosoidal function defined as follows:

$$\begin{aligned} PE_{(pos,2i)} &= sin(pos/10000^{2i/d_{model}}) \\ PE_{(pos,2i+1)} &= cos(pos/10000^{2i/d_{model}}) \end{aligned} \tag{3.3}$$

where d_{model} refers to embedding dimensions. The idea of this function is to emulate a geometric progression from 2π to $10000 \cdot 2\pi$. This will allow the model to attend relative positions since PE_{pos+k} can be represented as a linear function of PE_{pos}.

Vision Transformers (ViT)

In order to apply transformers to the image classification task, *Vision Transformers* (ViT) [Dosovitskiy et al., 2021] architecture was proposed. This variation of transformers splits the image in small patches (16x16 pixels in the original ViT) to then consider each flatted patch as an individual element passed to the vision transformer. Additionally, an independent and learned patch ($clf/class$) is added, whose output is used for the class prediction. Also, only the transformer encoder is considered. This is due to the fact that the classification tasks converts the image to the label, instead of generating a new image patch or its analogous in the Natural Language Processing domain, a new word.

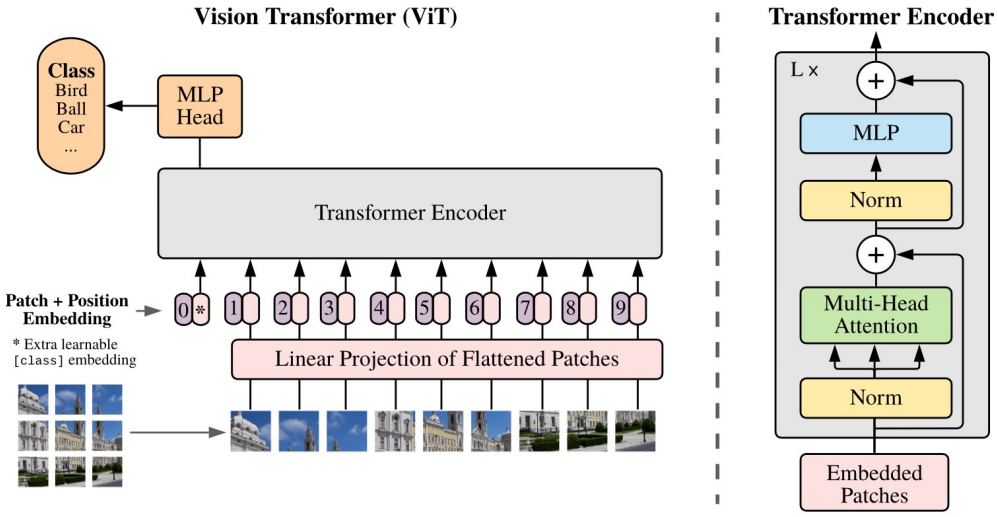


Figure 3.5: Vision Transformer architecture. [Dosovitskiy et al., 2021]

3.3 Closed form solution for linear regression

A linear regression is a linear model used to predict a continuous value. The loss function used is a Mean Squared Error loss. In this setting, there is an optimal set of parameters that can be calculated as follows.

$$\frac{\delta Loss}{\delta \theta} = \frac{\delta}{\delta \theta}\frac{1}{2}\sum_{i=1}^{N}||y_i - x_i\theta||_2^2 = \sum_{i=1}^{N} -x_i(y_i - x_i^T\theta) = -X(Y - X^T\theta) = 0 \Leftrightarrow$$

$$\Leftrightarrow X^TY = XX^T\theta \Leftrightarrow \theta = (XX^T)^{-1}X^TY$$

In general, we assume that XX^T is invertible, but this is not always the case, especially when the number of instances is much smaller than the input dimension. In order to solve this, we also consider the methods.

- **Solve** [Strang, 1980]: Instead of computing $x = A^{-1}b$, it computes $Ax = b$ as a linear system of equations to then calculate x. This allows to find solution when there is a linear system with 1 or more dimensions that defines the space of valid results for x, which can't be calculated by doing the inverse (since there will be some 0 eigenvalues). Even if this method is able to find more solutions in more cases than doing the inverse, it is still not able to solve every case.

- **Penrose Inverse** [Moore, 1920]: The key trick is to calculate a pseudo-inverse matrix. If we calculate the eigenvalue matrix of A by diagonalazing, i.e., $A = PDP^T$ some values of the diagonal of D might be zero. Only those values that are non zero in the diagonal are inverted and then multiplied by P and P^T to calculate the pseudo-inverse matrix or Penrose inverse matrix.

3.4 Principal Component Analysis

Principal Componenet Analysis or PCA [F.R.S., 1901, Hotelling, 1933] is a dimensionality reduction technique. The main idea is to select the dimension with the highest amount of variation in the input space. Subsequentially, a new dimension perpendicular to all the previous dimensions selected with the highest amount of variation. This way, we can change the reference dimensions of the input space in such a way that these dimensions are ordered by the amount of variation in the data. Therefore, in order to select the n dimensions with the highest amount of information available, the first n dimensions from the PCA procedure can be selected, reducing the dimensionality to n while keeping the most amount of data variation.

Some tasks have a huge amount of information/dimensions in the input space. In scenarios where the amount of instances is very small (such as Active Learning), the model might confuse some of the noise from this inputs with correlations to the output. In order to avoid this, we employ PCA which reduces the amount of inputs making it easier for the model to learn useful correlations. Additionally, it also helps for reducing the amount of scenarios where the closed-form solution can't be easily calculated because of the non invertibility of the data matrix (XX^T).

3.5 Data Foundation

This project will be focused on the image domain. We select the CIFAR10 and CIFAR100 datasets to work with [Krizhevsky, 2009].

CIFAR10

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

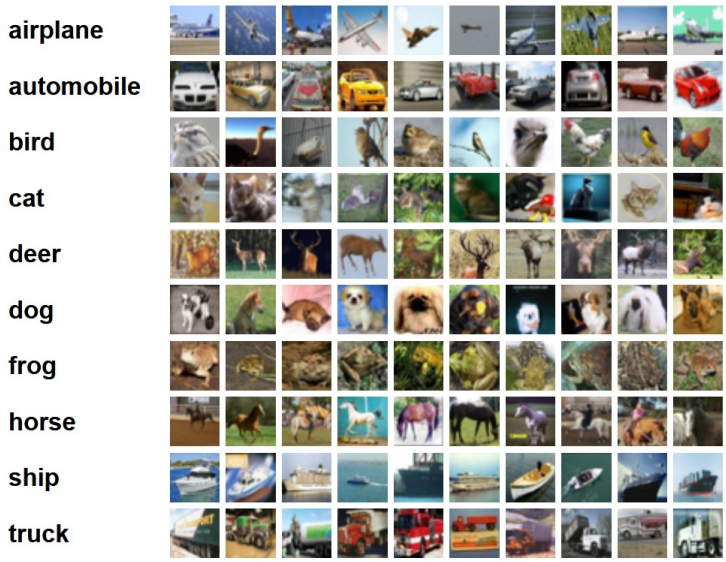


Figure 3.6: Cifar10 images examples [Krizhevsky, 2009]

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class. The classes are completely mutually exclusive.

CIFAR100

This dataset is just like the CIFAR-10, except it has 100 classes containing 600 images each. There are 500 training images and 100 testing images per class. The 100 classes in the CIFAR-100 are grouped into 20 superclasses. Each image comes with a 'fine' label (the class to which it belongs) and a 'coarse' label (the superclass to which it belongs). Here is the list of classes in the CIFAR-100:

Superclass	**Classes**
aquatic mammals	beaver, dolphin, otter, seal, whale
fish	aquarium fish, flatfish, ray, shark, trout
flowers	orchids, poppies, roses, sunflowers, tulips
food containers	bottles, bowls, cans, cups, plates
fruit and vegetables	apple, mushroom, orange, pear, sweet pepper
household electrical devices	clock, computer keyboard, lamp, telephone, tv.
household furniture	bed, chair, couch, table, wardrobe
insects	bee, beetle, butterfly, caterpillar, cockroach
large carnivores	bear, leopard, lion, tiger, wolf
large man-made outdoor things	bridge, castle, house, road, skyscraper
large natural outdoor scenes	cloud, forest, mountain, plain, sea
large omnivores and herbivores	camel, cattle, chimpanzee, elephant, kangaroo
medium-sized mammals	fox, porcupine, possum, raccoon, skunk
non-insect invertebrates	crab, lobster, snail, spider, worm
people	baby, boy, girl, man, woman
reptiles	crocodile, dinosaur, lizard, snake, turtle
small mammals	hamster, mouse, rabbit, shrew, squirrel
trees	maple, oak, palm, pine, willow
vehicles 1	bicycle, bus, motorcycle, pickup truck, train
vehicles 2	lawn-mower, rocket, streetcar, tank, tractor

Table 3.1: Superclasses and classes of CIFAR100 dataset. [Krizhevsky, 2009]

Chapter 4

Methodology

4.1 Problem setting

In this section, we will define the the Active Learning problem in the pool-based setting. There will be an emphasis on the classification scenario with C classes defined over a compact space X and a label space $Y = \{1, ..., C\}$. The loss function $l(\cdot, \cdot; w) : X \times Y \to \mathbb{R}$ is parameterized over the hypothesis class (w), e.g. parameters of the deep learning algorithm.

A collection of data points are sampled i.i.d. over the space $Z = X \times Y$ as $D = \{x_i, y_i\}_{i \in [n]} \sim p_{\mathbb{Z}}$ where $[n] = \{1, ..., n\}$. In this scenario there is an initial pool of data-points chosen uniformly at random as $s_0 = \{s_0(j) \in [n]\}_{j \in [m]}$. An active learning algorithm only has access to $\{x_i\}_{i \in [n]}$ and $\{y_{s(j)}\}_{j \in [m]}$. In other words, it can only see the labels of the points in the initial sub-sampled pool. It is also given a budget b of queries to ask an oracle, and a learning algorithm A_s which outputs a set of parameters w given a labelled set s. This way the initial sub-sampled pool will increase over the sampling iterations by adding the b queried instances. The active learning with a pool problem can simply be defined as:

$$\min_{s_1 : |s_1| \leq b} E_{x,y \sim p_Z}[l(x, y; A_{s_0 \cup s_1})] \tag{4.1}$$

In other words, an Active Learning algorithm can choose b extra points and get them labelled by an *oracle* (usually a human) to minimize the future expected loss. This thesis will consider the case in which the budget is 1 for every sample iteration (*Instance Active Learning*). We avoid $b > 1$ (*Batch Active learning*) since most of the Active Learning procedures don't have a theoretical reason to work better in this scenario compared to Instance Active Learning.

In general, the Active Learning procedure can be defined as follows:

Algorithm 2: Pseudocode for the Active Learning loop

```
# D, L, U: Full, labeled and unlabeled datasets.
# b: number of instances sampled every iteration
# s: instances sampled from the Active Learning method
# random_sample(D, n): sample uniformly n instances from D
# get_model(L): Get/Learn a model on the dataset L
# AL_method(): Active Learning procedure to select instances.
#    This function doesn't have access to the unknown labels in U.

L = random_sample(D, n)
U = D \ L
model = get_model(L)

# Repeat until there is no more budget to sample more instances
while budget():

    # Sample b elements from the unlabeled dataset
    s = AL_method(model, L, U, b)

    # Update datasets
    L = L U {s}
    U = U \ {s}

    # Update model
    model = get_model(L)
```

Notice that the optimal sample s is dependent on the learning algorithm A_s. Therefore, the optimal data selection for a specific learning algorithm may not transfer to other learning algorithms.

In general, we will denote $D = \{x_i, y_i\}_{i \in [n]}$as the full dataset, $L = \{x_{s(i)}, y_{s(i)}\}_{i \in [m]}$ as the labeled dataset and U=D\L as the unlabeled dataset. During the Active Learning loop, the Labeled dataset will increase its size by b every iteration while the unlabeled dataset will reduce its size in b every iteration.

4.2 Model architecture

In order to take advantage of the unlabeled pool (U), we will use an encoder $h : X \to X_E$ where X_E is the embedding space. The encoder will be pre-trained on a self-supervised task or on a different dataset. Once h is trained, we add a linear classifier $Clf : X_E \to Y$ to perform the downstream classification $\hat{y}_i = Clf(h(x_i))$. The Active Learning training algorithm will only fine-

tune the parameters of the Clf model on the labeled dataset $\{x_{s(i)}, y_{s(i)}\}_{i\in[m]}$.

In order to optimize the process, we will consider a 2 staged approach. First we perform a self-supervised training (SSL) with all the pool of images ($\{x_i\}_{i\in[n]}$) to train the encoder h. Then, we will freeze this model and add a linear layer to perform the classification on the labeled images ($\{x_{s(i)}, y_{s(i)}\}_{i\in[m]}$).

We consider a single linear layer to perform the classification. This combination of using an encoder and a linear classification model has been shown by different SSL methods to perform better than a fully trained model on the downstream task: DINO [Caron et al., 2021b], SwAV [Caron et al., 2021a], BYOL [Grill et al., 2020] or SCLR [Chen et al., 2020]. Additionally, we also consider the usage of pre-trained models on other image datasets such as ImageNet [Deng et al., 2009]

We also tried to train a full model without a pre-trained encoder instead of finetuning the last layer (linear classifier). However, the computation requires a week for one repetition in one specific setting. Notice that more than one repetition are needed for each setting since the variation is an important factor that could change the results of the experiments. It is true that the different settings and seeds could be parallelized, but due to hardware restrictions it is not reasonable to realize all these experiments in the time given for this Thesis. For comparison, the linear model takes around 8 hours (depending on the experiment setting). Additionally, this problem setting is compatible with the usage of a pre-trained encoder on the unlabeled dataset, which would reduce the computation time needed for the Active Learning experiment.

4.3 Backbone selection

The most popular architectures for the image classification task are CNNs and Transforemers, in particular, ResNet and ViT. Therefore, we decide to pick at least one of both cases.

We observe that ResNet50 is a popular choice in the Active Learning literature [Beluch et al., 2018] as well as ResNet18 [Ash et al., 2020, Hacohen et al., 2022]. For these models, we used pre-trained versions on ImageNet [Deng et al., 2009]. We remove the last dense layer of these models and add a new linear layer to perform a fine-tuning or transfer-learning on the Active Learning Image

classification task. This last layer is the only one that will be trained, while the rest of the model is freezed.

For the ViT, we decide to use DINO architecture since it has been trained in a self supervised task. In order to make a fair comparison with the ResNet backbones, we select DINOv2-S/14 that has 21 million parameters (ResNet50 has 25 million parameters). The next DINO model by size is DNOv2-L/14, which has 83 million parameters, i.e., more than 3 times the number of parameters of ResNet50.

4.4 Closed form solution for softmax regression

A closed-form solution for the linear regression and OLS loss can be obtained (see Sec. 3.3). Nevertheless, that is not the case for a sigmoid regression and neither for a softmax regression. In order to obtain an approximated closed-form solution for a softmax regression, we consider the approximation of solving C exponential linear regressions, where C is the number of classes and an exponential regressions refers to a linear regression exponenciated. In this approximation, the only difference is the absence of the normalizing factor when calculating the softmax's output.

```
def get_softmax_regression_weights(x, y, epsilon=10**-32, n_classes=10):
    y_processed = np.log(np.eye(n_classes)[y] + epsilon)
    inv = np.linalg.inv(x.T@x)      #Could be substituted by other methods
    weights = inv@x.T@y_processed  #(Solve, Pinv) to calculate the weights
    return weights
```

Notice that the matrix inverse calculation can be also be done with the *solve* method that tries to find x such that $Ax = b$ instead of inverting A, or with the *Penrose inverse* which inverts only the eigenvalues bigger than a specific threshold and zeros out the rest. Only, the *Penrose inverse* method is solvable in every setting (see Sec. 3.3). Due to this fact, we will use the *Penrose Inverse* in case a solution has not been found by the other methods. As we will see in the experiments, this can and tends to happen when the input dimension is bigger or equal than the number of labeled instances.

4.5 Metric selection

When comparing Active Learning procedures, normally the accuracy of the last step has been used as the preferred metric. Nevertheless, this metric doesn't take into account if a method is better in the earlier stages of the Active Learning procedure. Some might even argue that the Active learning should be stopped once a specific performance has been achieved or when the budget needed to increase the performance is no longer worth it.

To address this issue, we also use another metric that averages the accuracy over the Active learning step known as the *Area Under the Budget Curve* (AUBC). This can be defined as:

$$AUBC = \frac{1}{N} \sum_{i=1}^{N} Accuracy(X, Y; model^{(i)})$$

where $model^{(i)}$ refers to the model calculated in the i-th iteration from the Active learning loop.

4.6 Implementation details

Epsilon approximations for numerical stability

In order to avoid divisions by 0 or Nan values, we add a small ϵ value to avoid numerical collapse. In particular, for Entropy and BALD Active Learning strategies a logarithmic function is applied, which could lead to Nan values for input values close to 0.

Seed setting

In order to prove that a specific Active Learning algorithm is objectively better than others, we decide to use the same set of seeds for all the experiments. The main issue with using different seeds is that a good weight initialization or a good data distribution in the mini-batches can lead to better models. Some Active Learning procedures will then use these models to sample new instances, leading to better models sampling better data. This could lead to the experiments showing an increase in performance of some Active Learning sampling strategies even if the lift comes from a better weight initialization or training.

Inconsistent results with unregular batch sizes

One of the most interesting findings is that setting the *drop_last* parameter to False in the dataloader increases the robustness of the model substantially. We hypothesis that this is due to the fact that the last batch with fewer examples tends to shift the model weights too much, specially in the low data regime where the model converges with very few iterations. We observe that this behaviour is accentuated the smaller the batch size of the last batch is, generating a repeated pattern during training where there is a peak for cases where $|L| \ mod(batch \ size) = 1$. In the case of Figure 4.1, the batch size used is 32.

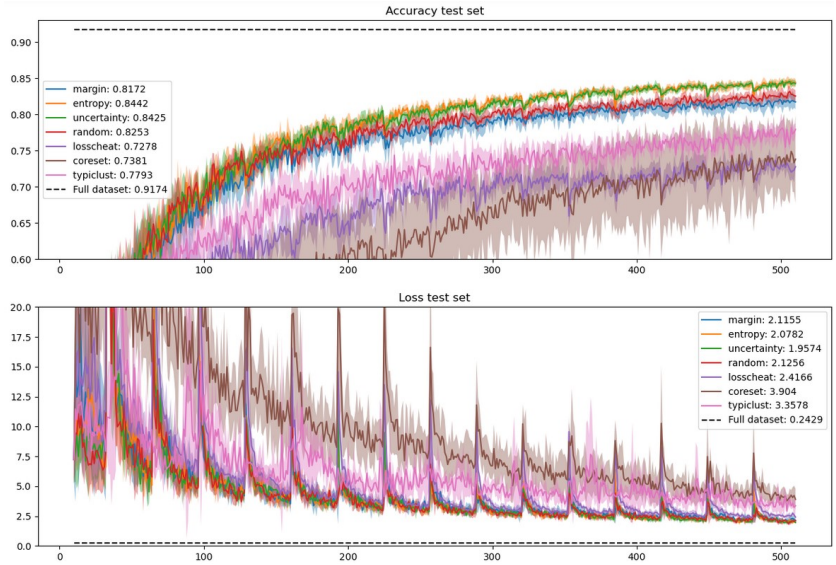


Figure 4.1: Test set performance with drop_last=False. Peaks appear every 32 batches, which correspond to the cases that the labeled dataset size modulo batch size is 1.

Hyperparameter tuning

We experiment on the full dataset to select the Hyperparameters. We acknowledge the fact that in a standard Active Learning loop, this is not possible. Nevertheless, we only consider the optimizer and learning rate selection, with the rest of the parameters set by default. This selection should be enough to have a good performance on the overall training pipeline, while not overfitting to the studied dataset. In a general Active Learning pipeline, the hyperparameters should be finetuned in every iteration, possibly using

several seeds and different partitions of the dataset to alleviate the randomness of the process. Nevertheless, due to time constrains on this project, we decided to select a specific training procedure to focus on the Active Learning strategies rather than on the training pipeline optimization.

Computation cost reduction

Due to the characteristics of the problem setting, computing a solution is highly costly. Not only for the time used computing, but also for the memory required in every iteration. In order to reduce the computation we utilize some tricks to alleviate this issue:

- **Precompute embeddings and augmentations**.

  Even if we don't push gradients through the hall network, doing inference on it is very costly in our setting. With a Resnet50 (Sec. 2.4) model where only the last dense layer is trained, we need 7 days to perform 1 Active Learning experiment. Due to the nature of Active Learning we would need to repeat this experiment to have a better estimation of the mean result, increasing the computation time too much for the thesis time given. We decide to precompute the embeddings of the images so that we only have to pass this embedding through a linear classifier at each training and inference step. Additionally, we also precompute the embeddings of augment versions of the original image such as *horizontal flip*, *sharpening*, *TrivialAugment wide* [Müller and Hutter, 2021] (for ResNet; Sec. 2.4, used for the pre-trained version used in this project [Vryniotis, 2021]) and *blur* (for DinoV2; Sec. 2.4, used in the training of the model weights used in this project [Oquab et al., 2023])

- **Reduce the unlabeled sample size to a random subset**.

  When an Active Learning procedure selects new instances, it will select it from the full unlabeled dataset. This could take a lot of compute time, especially for those methods that perform some type of clustering (Typiclust) or combinatorial calculation (Coreset). Therefore, we decide to randomly sample a subset of the unlabeled dataset and perform on that subset the Active learning procedure.

- **Usage of Mini-batch version of some algorithms**.

  Some algorithms require to cluster the data. In particular, we would like to highlight the use of mini-batch k-clustering use for *Typiclust* algorithm, since it has to be recomputed in every step of the Active

Learning loop. This is an implementation that was also used by the original authors of the *Typiclust* algorithm [Hacohen et al., 2022]. Notice that this may lead to different solutions compared to the original *k-Means* clustering. We also perform a mini-batch version of other algorithms such as *Coreset* when computing the distances of the labeled and unlabeled data.

- **Early stopping**.

  We notice that models tend to overfit specially fast (2-10 epochs) for the very low data regime (<50 examples). We decide to use an early stopping callback to speed up the training process. For consistency, once a number of epochs equal to the patience parameter have been done we restore the weights of the best iteration on the validation dataset.

4.7 Surrogate model

We decide to train a *surrogate model* that predicts the improvement in validation accuracy or validation loss when adding the instance given as input to the label set. We argue that predicting this increase from the raw data would be too difficult for the model. If a neural network is not able to solve the problem of classifying the data correctly, why would then a neural network be able to asses which instances are more important to select?

We observe that most Active Learning methods use a combination of preprocesed version of the output or some input augmented variation. Therefore, we decide to pass as inputs relevant metrics from the Active Learning field such as: entropy, least confidence, margin, typicality, coreset distance, cosine distance, embedding variation between augmentations and variation of the class prediction. Additionally, we also add some meta-features such as the current validation loss/accuracy and the current Active Learning step. This input data is most similar to the tabular domain, where *Decision Trees* have shown significant performance advantage when compared to Neural Networks [Grinsztajn et al., 2022]. In particular, we decide to use XGBoost [Chen and Guestrin, 2016] one of the best and most popular Decision Trees methods. Lastly, to avoid Out-of-Memory errors, we decide to reduce the sample size for the Active Learning procedure.

Chapter 5

Experiments

5.1 Encoder Backbone

We first want to select the backbone encoder. We had multiple pre-trained models to select from. We decide to include two of the most popular architectures in the computer vision domains: CNN (Sec. 3.1) and ViT (Sec. 3.2). For the CNN models, we decide to use ResNet18 and ResNet50 (Sec. 2.4), 2 variations of the same architecture to see how performance varies between different amount of parameters. Both models have been pre-trained on ImageNet and would be applied in a transfer learning setting for the Active Learning problem. On the other hand, we consider DinoV2 (Sec. 2.4) as a ViT example. One important characteristic from Dino is the self-supervised training recipe, which could be done on the unlabeled data pool.

For the selection of the specific sizes, ResNet50 is a popular choice in the Active Learning literature [Beluch et al., 2018] as well as ResNet18 [Ash et al., 2020, Hacohen et al., 2022]. Therefore, we decide to include both. For the Dino version, we selected a model with an amount of parameters comparable to the ResNet50 model in order to have a fair comparison between CNN-based models and ViTs.

As mentioned in the implementation details section (Sec. 4.6), we want to precompute the embeddings for the different backbones before doing the Active Learning task. In order to have a wider range of non-repeated images, we decide to store 3 additional embedded copies of the dataset with different augmentations. Horizontal flip and sharpening are applied for ResNet and Dino models, however we additionally stored a TrivialAugmentation wide [Müller and Hutter, 2021] for the ResNet models and a blurred image

for Dino. This difference between in the augmentation selection is due to augmentations used during the training of the backbones. We decide to only use augmentations that have been used during the training of the models. If the set of augmentations used in our experiments are different than the set of augmentations used during training, there could be a decrease in performance that we want to avoid. Notice that doing augmentations in the embedding space is not trivial at all, therefore we only apply augmentations on the raw data.

5.2 Classifier model

We argue that for the ResNet backbone, using a linear layer classifier is a very sensible choice since the original architecture also adds a linear layer after the convolutions. For the DinoV2 model, we have seen previously (Sec. 4.1) that Self-supervised methods report a high performance with only a linear layer for classification. Since the classifier used is a linear model, we apply a softmax regression to perform the classification.

To further prove our point, we perform a classification on the full dataset labeled to show the capabilities of each backbone (Table 5.2).

Hyperparameter selection

In the Active Learning setting, the hyperparameter selection should be done at every iteration step since the data is dynamically changing between iterations. Nevertheless, that would be a very compute intense process. In order to avoid this problem, we use the best hyperparameter obtained when training a linear model on the full dataset. The results for the learning rate for each optimizer are shown in Table 5.1.

We observe that there is no big difference in accuracy between the optimizers. The best rank in average is achieved by AdamW, which performs the best with a learning rate of 1e-3. We argue that using an adaptive optimizer is the best approach for avoiding to fine-tune the learning rate every iteration. We will use AdamW with learning rate 1e-3 for all the following experiments with the remaining arguments of the optimizer set by default from the PyTorch implementation. Additionally, AdamW has been shown to act as a better regularizer than L2 in Adam optimizer [Loshchilov and Hutter, 2019].

Method	ResNet18	ResNet50	DinoV2
SGD	0.8536 ± 0.0001	0.8972 ± 0.0002	0.9515 ± 0.0002
Adam	0.8536 ± 0.0001	0.8972 ± 0.0001	0.9512 ± 0.0001
AdamW	0.8537 ± 0.0001	0.8973 ± 0.0001	0.9513 ± 0.0002
NAdam	0.8535 ± 0.0001	0.8973 ± 0.0	0.9514 ± 0.0001

Table 5.1: Linear classifier accuracy with different backbones and optimizers on the full dataset. The result shown are the best results for a set of learning rates (1e-1, 1e-3, 1e-5). In all the cases the best learning rate is 1e-3.

This would help the model to generalize better and not overfit, especially in the low data regime.

5.3 Closed-form solution

Whenever a linear model is used, the idea of using a closed-form solution is very intuitive. Nevertheless, for a softmax regression there is not an exact closed-form solution so we use an approximation (Sec. 3.3).

We will first compare the closed-form approximation with the gradient based counterpart. Additionally, there are different methods to approximate the solution which may lead to different results in the low data regime. We will do a small study on their behaviour and how the could impact the Active Learning procedure.

Closed-form solution vs Gradient based methods

In order to compare the closed-form solution approximation with the gradient based method we do a comparison on the full dataset (Table 5.2).

Accuracy	ResNet18	ResNet50	DinoV2
AdamW	**0.8537 ± 0.0001**	0.8973 ± 0.0001	**0.9513 ± 0.0002**
Closed-form	0.8444 ± 0.0	**0.9104 ± 0.0**	0.9377 ± 0.0

Table 5.2: Linear classifier accuracy with different backbones on the full dataset. Notice that the closed-form solution are the same for *Standard*, *Solve* and *Penrose Inverse*.

We observe that depending on the backbone the closed-form solution might outperform the gradient based counterpart. However, we would like to highlight some issues that the gradient based methods have when compared to the closed form solution:

- **Training time**: Obtaining the closed-form solution is way faster than obtaining the solution with a gradient based method. A comparison has been done on the full dataset (Table 5.3) as well as on the Active Learning procedure (Table. 5.7).
- **Hyperparameter selection**: When doing a gradient based training there are a lot of hyperparameters that have to be defined (optimizer, learning rate, epochs and other optional implementation details such as drop_last=False (Sec. 4.6), early stopping, learning rate schedule, etc.). In general, this should be not only tuned for every training procedure but also for the different Active Learning steps. On the other hand, the closed-form solutions doesn't require these hyperparameters, making the solution more robust and even faster to compute in case you want to finetune the hyperparameters.
- **Reduction in variation**: When using a gradient based training, there are a lot of steps that have a certain random factor that can affect the final result (weights initialization, data shuffle, etc.). All this is not required in the closed-form solution, which for a given dataset returns always the same model. This reduction in variation is especially important in the Active Learning scenario where different acquisition functions tend to have small differences in the results.

	ResNet18	ResNet50	DinoV2
AdamW	10.1632 ± 0.8835	11.6511 ± 0.4369	10.8427 ± 1.2001
Closed-form	**0.1287 ± 0.0076**	**0.8994 ± 0.0113**	**0.1138 ± 0.0351**

Table 5.3: Linear classifier training time in seconds with different backbones on the full dataset. The different times on the closed-form solution for different backbones is due to the different dimensions of the backbones' outputs: ResNet18 - 512, DinoV2 - 728, ResNet50 - 2048.

Existing methods for Closed-form solution

For the full dataset or when the number of data instances are higher than the number of input dimensions, all the approximations return the same output (*Standard*, *Solve* and *Penrose Inverse*). Nevertheless, they obtain the same solution in different ways which would lead to different compute times. We compare the training times on the full dataset in Table 5.4.

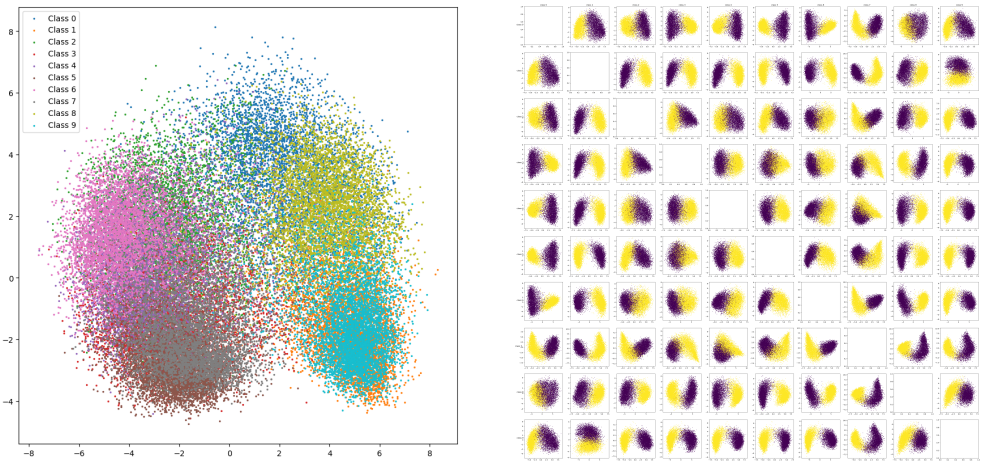
Training time	ResNet18	ResNet50	DinoV2
Std.	0.2792 ± 0.0342	2.6448 ± 0.0304	0.2233 ± 0.0253
Solve	**0.1287 ± 0.0076**	**0.8994 ± 0.0113**	**0.1138 ± 0.0351**
Pen. Inverse	0.4259 ± 0.0035	4.8105 ± 0.0455	0.2626 ± 0.0063

Table 5.4: Linear classifier training time in seconds with different backbones on the full dataset. For more information on the individual methods, go to Sec. 3.3. Notice that all these methods result on the same model for any given backbone.

Notice that the previously mentioned methods (Sec. 3.3), give the same result for the case where the matrix is invertible. Therefore, we will use *Solve* method for the following experiments unless stated otherwise.

Nevertheless, there are some minor implementation differences on what happens on the low data regime when the matrix might not be invertible. In order to study this behaviour, we randomly sample instances on which we calculate the closed form solution and check the test accuracy (Fig. 5.2). We also do a small ablation on the *Penrose Inverse* for the Active Learning procedure (sec. 5.8

Additionally, we also perform a visualization on how the cluster is structured and how the different classes can be linearly separated (Fig. 5.1). In order to visualize it in a 2 dimensional image we use PCA with 2 dimensions.For more visualizations on the different backbones we have additional information in the Appendix 7.2.



(a) Full data (b) Pairwise class comparisons.

Figure 5.1: ResNet50 embeded data visualization with PCA.

Problems with dataset size

We observe that for instances with less samples than the input space dimension (2048 for ResNet50), the result is almost random for *standard* and *solve* methods (Fig. 5.2).

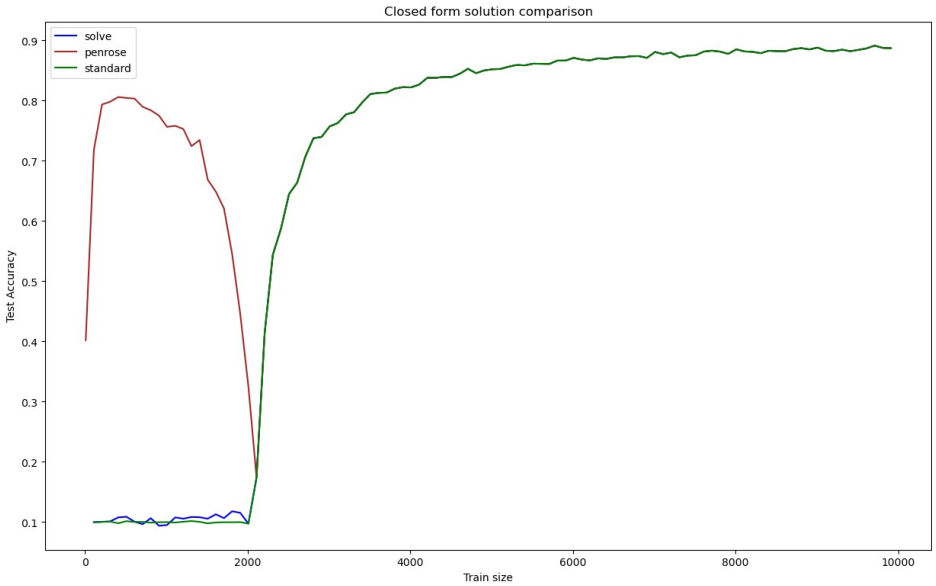


Figure 5.2: Accuracy comparison of different closed-form solution methods. The backbone used is ResNet50 with 2048 output dimensions.

In Figure 5.2, *Penrose Inverse* shows an unwanted behaviour of having at an intermediate step a huge drop in performance before recovering. We have the hypothesis that this is due to the *Penrose Inverse* actively deleting some of the dimensions (converting eigenvalues to 0) before performing the inverse transformation. As we will see in Section 5.4, reducing the dimension in the early stages of the active learning procedure is what gives this method a huge boost in performance compared to the other methods.

5.4 PCA

Due to problems with the dataset size compared to the input size for the closed solution (more on Sec. 5.3), a dimensionality reduction technique is needed. We decide to use PCA due to its simplicity and versatility.

PCA sizes comparison on full dataset

First, we want to make sure that applying the PCA method would not destroy the data structure and that the model will still be able to learn properly from the compress data. To test this, we apply PCA on the full dataset and show the accuracy of Solve method for different dimensions (Fig. 5.3).

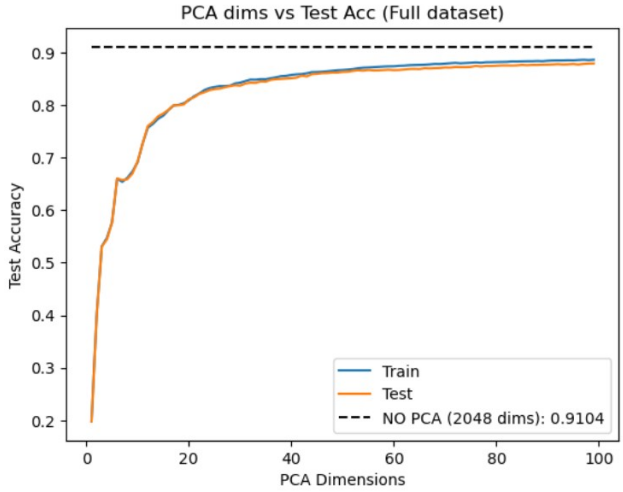


Figure 5.3: Accuracy of the solve method on the full dataset after applying PCA dimensionality reduction. ResNet50 is used as the backbone structure.

From Figure 5.3, we observe that with less than 20 dimensions the model is not capable of learning properly. However, the increase is minimal for more than 50 dimensions. We also report the results for the different backbones and for PCA with 20 and 50 dimensions (Table 5.5).

	ResNet18	ResNet50	DinoV2
NO PCA	0.8444 ± 0.0	0.9104 ± 0.0	0.9377 ± 0.0
PCA 50	0.793 ± 0.0	0.862 ± 0.0	0.9223 ± 0.0
PCA 20	0.7315 ± 0.0	0.8098 ± 0.0	0.9036 ± 0.0

Table 5.5: Accuracy comparison for the closed-form solution on the full dataset.

PCA sizes comparison on Active Learning setting

Using as a stepping stone the results from the previous section (Sec. 5.4), we decide to experiment with 20 and 50 dimensions on the Active Learning setting and compare how different PCA dimensions affect the performance (Fig. 5.4).

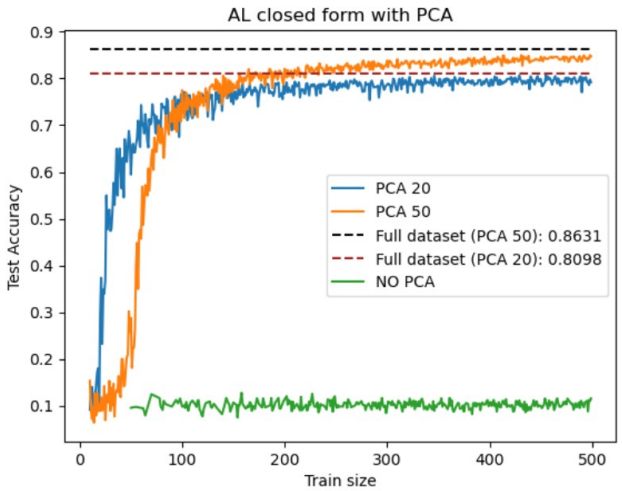
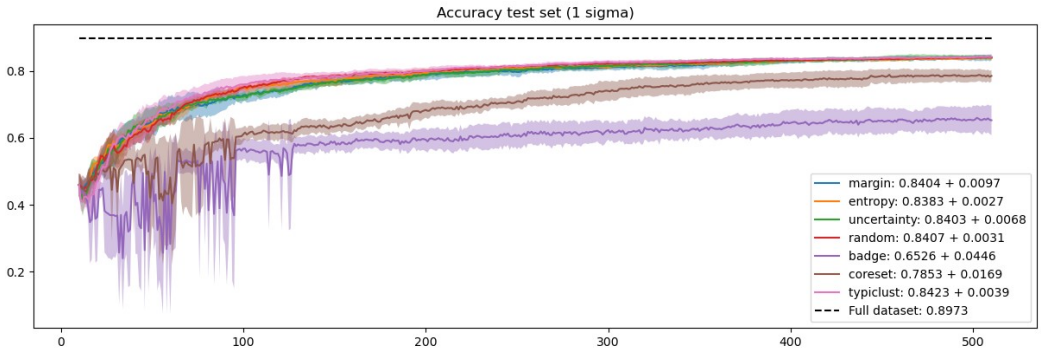


Figure 5.4: Accuracy comparison between different PCA dimension sizes for different training sizes. NO PCA starts to be plotted with 45 train size since all the cases before return singular matrix cases for the 2048 dimension inputs.

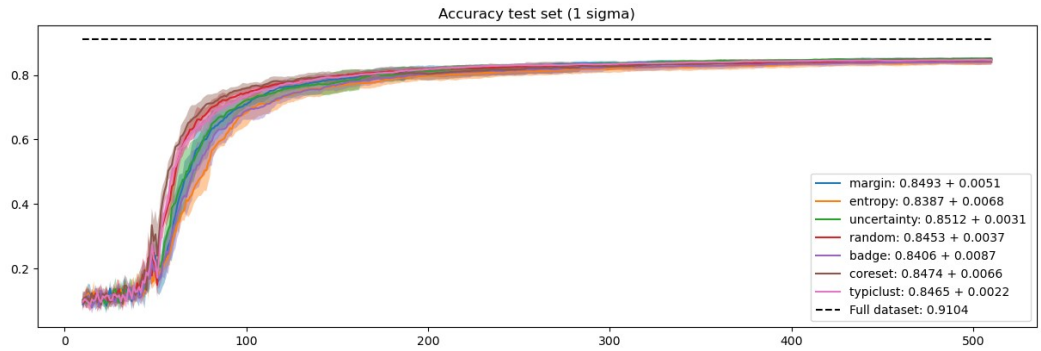
We observe that the smaller the PCA dimensions are, the less instances it needs to converge. Nevertheless, the performance platoes faster and the model stops to learn quicker. Therefore, we decide to use 50 dimensions in the following experiments.

5.5 Active Learning

After all this setup, we decide to perform the Active Learning procedure with both the closed-form solution and the AdamW optimizer to compare the results. We report the training curves in Table 5.5, where we plot the mean with a solid line and 1 standard deviation with the shadowed area.



(a) AdamW



(b) Solve (PCA 50)

Figure 5.5: Accuracy comparison between different Active Learning methods. The backbone used is ReNet50. Every experiment has been repeated 5 times. The legend reports the accuracy mean and standard deviation on the final step.

From Figure 5.5 we can deduce that:

- Using a closed-form solution with less instances than the input dimension is not useful for the Active Learning problem. We can observe that until we have 50 examples the model learned is almost random. This is consistant with our previous experiments on the closed-form solution performance (see Sec. 5.3).
- Gradient based methods such as AdamW are more prone to overfit to out-of-distribution instances. This can be observed with the *Coreset* method. Notice that the *Coreset* selection strategy is completely independent on the classifier and tends to select instances that are far away from any labeled instance. Therefore, the difference in performance must come from the training procedure itself.
- The closed-form solution training is more robust to the Active Learning sampling strategy selection. All models perform similarly in the later stages, while in the gradient based training *Coreset* and *BADGE* are never able to match the other methods.

Lastly, we also notice that methods that tend to cover the space such as *Coreset* tend to work better for the closed-form solution, while methods that select common examples such as *Typiclust* tend to be better in the gradient based scenario. In order to show this, we report the AUBC for the mentioned cases.

	ResNet18	ResNet50	DinoV2
Closed-form solution (Solve)			
Random	**0.6397 ± 0.0031**	0.7464 ± 0.0034	0.8141 ± 0.0058
Typiclust	0.6391 ± 0.0083	0.7467 ± 0.002	0.8194 ± 0.0042
Coreset	0.6113 ± 0.0164	**0.7497 ± 0.009**	**0.8282 ± 0.0076**
Gradient based solution (AdamW)			
Random	0.7722 ± 0.003	0.7825 ± 0.0035	0.8404 ± 0.002
Typiclust	**0.7775 ± 0.0059**	**0.7862 ± 0.0095**	**0.8574 ± 0.0011**
Coreset	0.715 ± 0.0256	0.6863 ± 0.0198	0.8182 ± 0.007

Table 5.6: Area Under the Budget Curve (AUBC) comparison for different settings.

From Table 5.6 we can confirm our previous statement that the optimal Active Learning strategy is dependent on the training procedure (sec. 4.1).

For visualizations of the other backbones go to Appendix 7.3.

Additionally, we also show a time comparison between the time required for an Active Learning experiment in average for the different settings (Table 5.7).

	ResNet18	ResNet50	DinoV2
AdamW	2542.30 ± 318.30	5254.46 ± 390.00	7466.71 ± 320.14
Solve	**65.80 ± 0.77**	**516.73 ± 4.07**	**78.46 ± 1.40**

Table 5.7: Time needed in seconds for an Active Learning experiment with a linear classifier starting with 10 labels and a budget of 500. PCA 50 has been applied on the closed-form data in order to avoid dimensionality problems (Sec. 5.3).

5.6 Surrogate model

The main objective of this project is to create a surrogate model that is able to work as an agent that selects which instance to label by predicting the increase in performance in the validation set. In order to train this model we first collect the data needed.

Data collection

In order to collect a wide enough dataset, we perform 5 iterations on a specific trainning procedure with a list of Active Learning strategies. Each of this strategies gives a metric (entropy, margin, least confidence, coreset distance, typicality, cosine distance, etc.) that we could use as an input for the surrogate model. Additionally, we also stored some meta-features such as the current validation loss/accuracy and the current Active Learning step (dataset size).

In order to have a big enough sample size in the dataset, we not only consider the different Active Learning steps. We also consider the different backbones (ResNet18, ResNet50 and DinoV2) and training procedures (gradient based, closed-form with *Solve*, closed-form with *Penrose Inverse*, etc).

After collecting the dataset, we want to also see if the increase of performance in the validation set is correlated to the increase in performance on the true test. Calculating the correlation in the collected datasets, we observe that there is an averaged correlation of 0.99. Therefore, we can confirm that properly predicting the increase in the validation performance would also lead to an increase in performance on the test dataset.

Training the surrogate model

Once we have the dataset collected, we use the first 4 repetitions of the experiments as train and the last one as test. Since the inputs are varied, we decide to consider different subsets and check the performance differences. Additionally, the model could also try to predict the expected increase in loss or accuracy. All this differences can be seen in a later ablation study (Sec. 5.8).

As mentioned in Section 5.5, different training procedures require different sampling strategies. Therefore, even if we could combine the collected data with different training procedures and backbones, we refrain from doing so and utilize exclusively the data that has been collected from different Active Learning sampling strategies with the same backbone and same training procedure for the classifier. We will later test the surrogate model in that same setting.

As mentioned in Section 4.7, we will train the model with Decision Trees since they are one of the best performing models in the tabular domain. However, we will need to finetune them for different hyperparameters (learning rate, maximum depth, objective function ratio between training loss and complexity loss (lambda), etc.). We will use the test partition from the collected dataset (last repetition from the Active learning experiment) to validate the hyperparameters selection.

Additionally, we also use early-stopping in order to let the models train for a huge amount of steps (50000) but make sure they don't overfit. In practice we observe that with a patience of 50, the models never end up using all the 50000 steps during training.

One of the most important claims from Typiclust [Hacohen et al., 2022] is that some methods can perform better in the initial stages of the Active Learning procedures while other are better in later stages. The argumenta-

tion is based on the idea that at first the model should view representative samples (or as they called them in the paper, 'typical') in order to understand the general data structure, while later stages should focus on difficult samples.

In order to tackle this, we add as a metafeature of the training procedure the current Active Learning step. This should indicate the model in which types of samples it should be focused. Additionally, we also consider the current loss and accuracy in the validation set as a metafeature. When studying the metadata, we observe that there is a correlation of 0.99 between the accuracy on the test set and the validation set, as well as between the validation loss and the test loss. This way, we can be sure that the models trained with the Active Learning heuristics are not overfitting to the validation set.

MSE of train vs. validation

Once the models have been trained, we would like to know which inputs are the best ones for the surrogate model. This could be considered an additional hyperparameter selection. Nevertheless, we would like to study this selection more in depth before deciding on the inputs. We will report the average rank and average MSE loss for different inputs over the different backbones and training scenarios in Table 5.8.

	Standard	+AL step	+Valid loss/acc
Train set			
Average MSE	6.6255e-06	2.6057e-06	2.6565e-06
Average rank	1.467	1.133	0.4
Test set			
Average MSE	6.371e-06	2.6213e-06	2.6865e-06
Average rank	1.467	1.133	0.4

Table 5.8: Average rank for MSE error on the increase in accuracy prediction for the train/test partition in the collected dataset. The column indicates the inputs that the model receives. Subsequent columns have the mentioned input and the inputs from the previous columns. AL step indicates the Active Learing step normalized, i.e., the percentage of samples in the budget that have been added to the labeled dataset. Valid loss/acc refers to the accuracy and cross entropy loss of the classifier on the validation set.

From this table we will like to point out that the average ranks are the same in the train and test partition. Also, the average MSE is comparable between the test and train partitions. Therefore we conclude that the models have been properly regularized and don't overfit the training partition.

Model selection

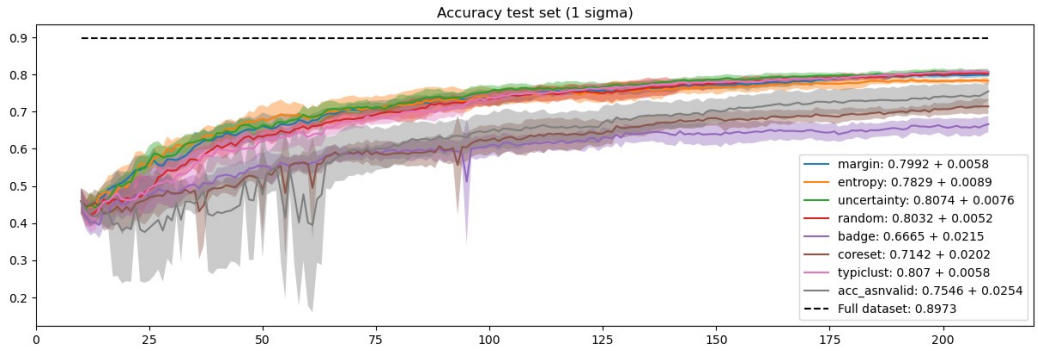
In order to pick one of the versions of the surrogate model, we could run all the different versions and select the one that performs the best. Nevertheless, then we would be cherry picking. In order to properly select one model version, we calculate the Mean Squared Error on the test partition from the collected dataset and report the average rank over a wide variety of settings. We report the results in table 5.8.

We notice that the overall ranking increases when adding new inputs to the surrogate model. Nevertheless, the average MSE error is comparable with and without the validation loss and accuracy. This could be a signal that in some cases this additional information makes the model overfit instead of generalize better. However, since the average rank is better, we decide to use it for our experiments.

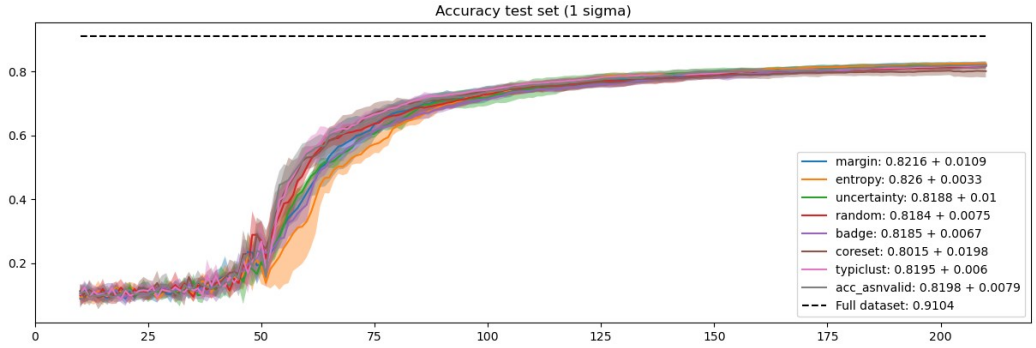
Test surrogate model for Active Learning sampling

Once the surrogate model is selected, we perform the Active Learning procedure using this model as the agent. One important detail is that in order to use the surrogate model a lot of different metrics need to be calculated (coreset distance, input variation, typicality, etc.). In order to alleviate the compute requirements, we decide to reduce the subsampling size in every iteration from 10000 to 500, i.e., the surrogate model will select the instance not from the full unlabeled dataset but a smaller random subset with size 500. We compare the performance with the other sampling methods.

From Figure 5.6, we observe that the surrogate model is in the best only comparable. This has only been achieved for the closed-form solution with ResNet50 backbone. Any other configuration it performs worse than most of the baselines. Go to Appendix 7.5 to see the visualizations with the other backbones.



(a) Surrogate model with a gradient based training for the classifier.



(b) Surrogate model with a closed-form solution for the classifier.

Figure 5.6: Accuracy comparison between the surrogate model (*acc_asnvalid*) and the other Active Learning strategies. The backbone used is ResNet50. Every experiment has been repeated 5 times.

5.7 Discussion

Main experiment is a failure

As shown in Figure 5.6, the results when using the surrogate model are not satisfactory. We would like to have a small discussion on what are the points that may have lead to the experiment being unsuccessful. Some of the reasons we argue why it has not perform as expected could be:

- **No Reinforcemnt Learning training**:

  The surrogate model has been train to predict the increase in performance for other strategies. Nevertheless, this an issue that may lead to inconsistent results when sampling.

- **Freezed backbone**:

  One big issue is that if the encoder is not able to properly represent the data in the embedding space, some instances might be wrongly represented. If any instance is wrongly represented, then no matter how good the classifier is it won't be able to predict correctly those instances. This could also explain why the sampling strategies don't show better performance than random.

- **Small subsample size**:

  We want to highlight that using a smaller subsample size is a decision that can reduce the training time. Nevertheless, it will lead to bad behaviour for some Active Learning strategies. Not only are the less samples to select but some strategies require the full unlabeled dataset to work properly, for example *Typiclust* clusters the labeled and unlabeled data before doing the selection.

5.8 Ablations

Sampling size

One of they key points we want to tackle is whether or not the subsample size could have a huge impact. In order to test this, we will compare the different heuristics with different sample sizes and see the impact it has. We will use AUBC since it is more representative of the intermediate steps an the overall model performance. The results are shown in Table 5.9.

Subsample size	Random	Entropy	Margin	Least Confidence	Typiclust	Coreset	BADGE
Gradient based							
500	0.7803 ± 0.005	0.7756 ± 0.0086	**0.7827 ± 0.0074**	**0.7888 ± 0.0017**	0.7787 ± 0.0043	**0.6992 ± 0.0175**	**0.6636 ± 0.0132**
10000	**0.7825 ± 0.0035**	**0.7792 ± 0.0058**	0.7754 ± 0.0118	0.7764 ± 0.0095	**0.7862 ± 0.0095**	0.6863 ± 0.0198	0.5827 ± 0.0192
Closed-form approximation (*Solve*)							
500	0.7452 ± 0.0075	**0.7424 ± 0.0039**	**0.7446 ± 0.0067**	**0.7455 ± 0.0056**	**0.7486 ± 0.0026**	0.7402 ± 0.0099	**0.7409 ± 0.0021**
10000	**0.7464 ± 0.0034**	0.7195 ± 0.0099	0.7381 ± 0.008	0.7392 ± 0.0094	0.7467 ± 0.002	**0.7497 ± 0.009**	0.7242 ± 0.0042

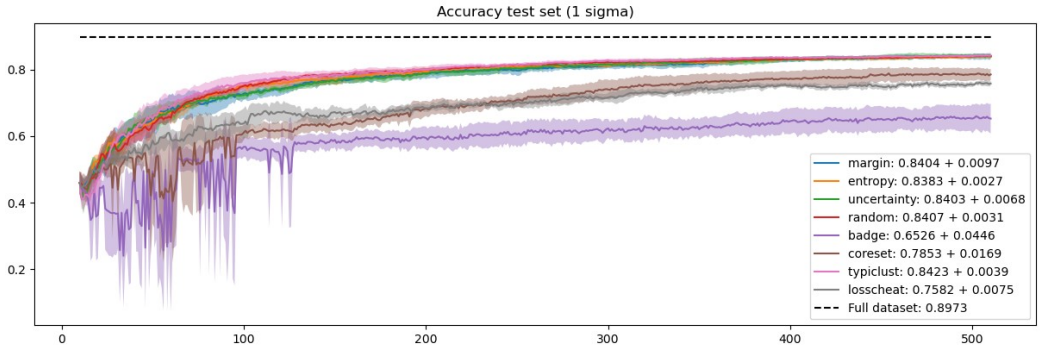
Table 5.9: AUBC comparison for different subsample sizes. Backbone used is ResNet50. All the experiments have been repeated 5 times.

Theoretically, if the sampling strategy is useful for the model training (better than random), having more instances to select from should increase the performance. In general, the differences between the subsample sizes is within 1 or 2 standard deviation. Notice that random should have the same performance independently of the subsample size. We would argue that since the Active Learning strategies have a worse or comparable performance to

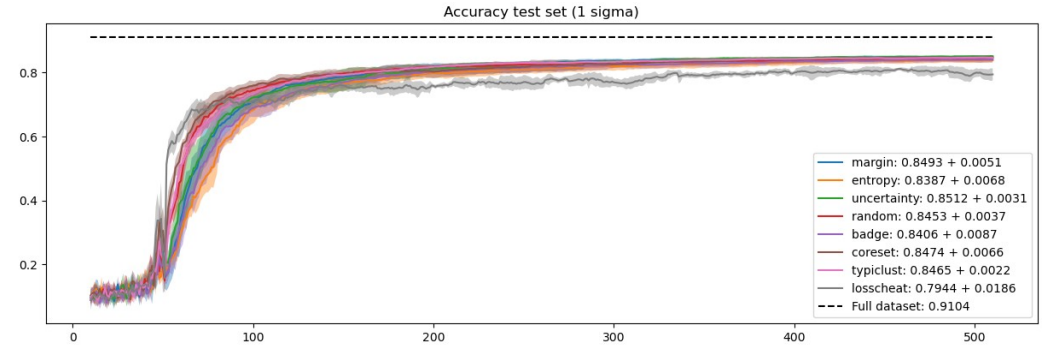
random, we can not derive a solid conclusion for these results.

Loss based Active Learning methods

We notice that some methods aim to use an approximation of the cross entropy loss as the Active Learning selection strategy [Yoo and Kweon, 2019, Shukla and Ahmed, 2021]. These methods are highly dependent on the architecture used and would require a lot of tuning. We believe that using the loss as the selection strategy might lead to select outliers more frequently and would lead to a worse performance. In order to prove this, we decide to use the true loss and show that not even with access to the true loss this strategy works.



(a) AdamW



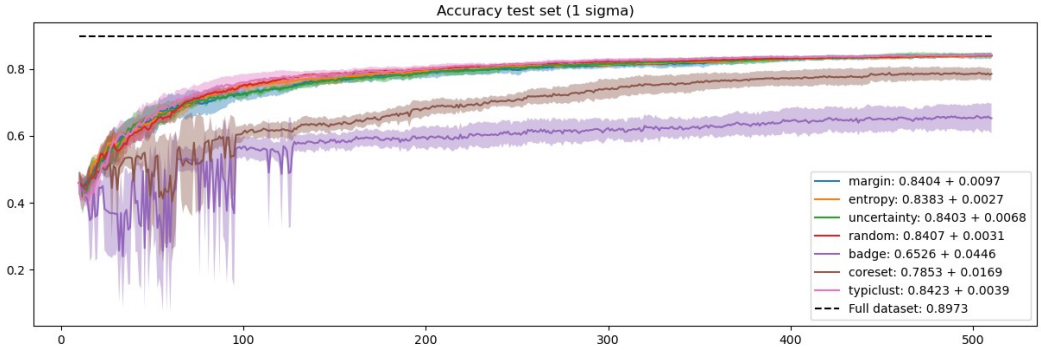
(b) Solve

Figure 5.7: Accuracy comparison between different classifiers. The backbone used is ResNet50. Every experiment has been repeated 5 times.

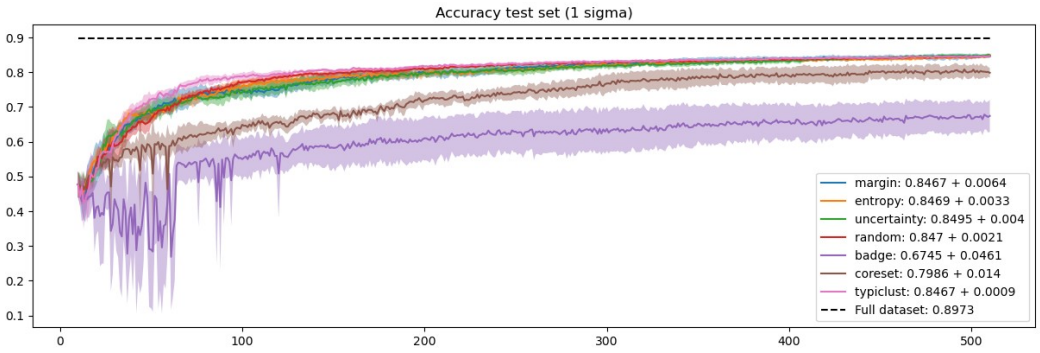
In Figure 5.7 we observe that using the loss as a sampling strategy is in the long term a bad procedure. Nevertheless, in the early stages of the closed-form solution it is unexpectedly good. We believe that this is due to the selection of instances that are missclassified. In the later stages, it continues selecting instances with a high loss but this might be just instances that are not representative of the underlying data structure.

Linear classifier vs MLP

One reasonable criticism is whether a more powerful classifier could be able to solve some of the issues that have appeared during this thesis. In particular, one might argue that the problems lies in the classifier not being able to understand the underlying data structure. In order to tackle this issue we repeat some of the previous experiments with an MLP.



(a) Linear classifier



(b) MLP classifier

Figure 5.8: Accuracy comparison between different classifiers. The backbone used is ResNet50. Every experiment has been repeated 5 times.

From Fig. 5.8, we conclude that even if the MLP gives better performance, it stills has the same issues as the linear classifier (training time, hyperparameter tuning, etc). Additionally, we also observe a bad performance from

Different targets for the surrogate model

Another experiment we did is to use different targets for the surrogate model. One idea would be to predict the decrease in the cross-entropy loss when a given instance is added to the labeled dataset. Comparing surrogate models on the loss and accuracy is difficult since the output given models different values. Therefore, we run all the combinations and report the results in Figure 5.9.

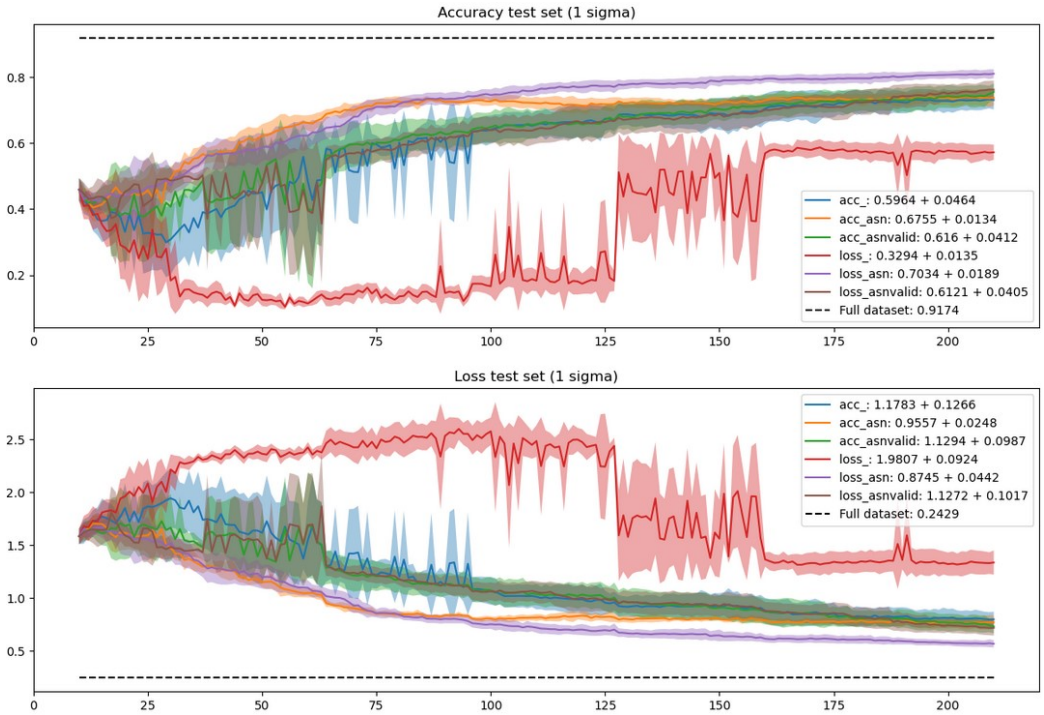


Figure 5.9: Surrogated models performance comparison. The backbone used is ResNet50. The training procedure is gradient based (AdamW). The experiments have been repeated 5 times.

One of the most intriguing findings is that the models with the validation loss and accuracy tend to not perform too well. This something that was hinted by the results from Table 5.8, where the asnvalid had a higher

ranking than the case without the validation loss and accuracy but had the same average performance.

With this finding we would like to see whether or not using a better surrogate model would lead to better results. For this experiment we decide to cherry-pick and select the best surrogate model for each case and compare it to the other heuristics in Table 5.10.

Training	Random	Entropy	Margin	Least Confidence	Typiclust	Coreset	BADGE	SM
ResNet18								
Gradient based	0.6293 ± 0.0103	0.5819 ± 0.0203	0.6117 ± 0.0123	0.6005 ± 0.0128	**0.6296 ± 0.015**	0.5007 ± 0.0244	0.5287 ± 0.0209	0.6089 ± 0.0157
Closed-form	0.5307 ± 0.0079	0.5023 ± 0.006	0.516 ± 0.0139	0.5172 ± 0.0105	**0.534 ± 0.0085**	0.5182 ± 0.0183	0.5318 ± 0.0158	0.5113 + 0.0063
ResNet50								
Gradient based	0.7229 ± 0.0095	0.7273 ± 0.0149	0.7299 ± 0.0097	**0.7389 ± 0.0064**	0.7211 ± 0.0095	0.6284 ± 0.0216	0.6075 ± 0.0191	0.7034 ± 0.0189
Closed-form	0.6433 ± 0.0114	0.6337 ± 0.0089	0.6399 ± 0.0117	0.6383 ± 0.0103	**0.6506 ± 0.0048**	0.6427 ± 0.012	0.633 ± 0.0052	0.6211 ± 0.0074
DinoV2								
Gradient based	0.7879 ± 0.0067	0.7876 ± 0.0062	0.7924 ± 0.0048	0.792 ± 0.0075	**0.7989 ± 0.0058**	0.7584 ± 0.0103	0.5981 ± 0.0308	0.773 ± 0.0091
Closed-form	0.7176 ± 0.0055	0.7034 ± 0.0068	0.7049 ± 0.0091	0.7097 ± 0.0077	0.7206 ± 0.006	**0.7395 ± 0.0116**	0.7135 ± 0.005	0.7005 ± 0.0042

Table 5.10: AUBC comparison for different methods with SM the best surrogate model (cherry-picked) for that specific setting. All the experiments have been repeated 5 times.

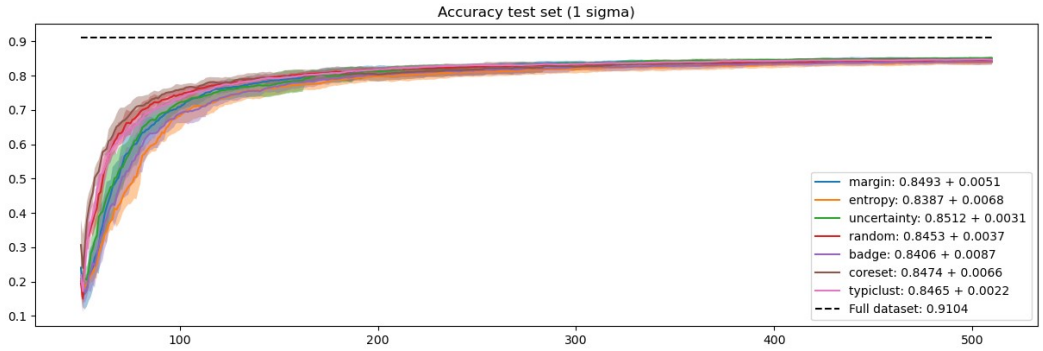
We observe that cherry-picking is not enough to win against the other baselines. However, we would like to point out the best models where: loss_asn, acc, acc, loss, acc and acc_asn. The suffixes indicates what additional data the surrogate model receives as an input (asn: Active Learning step normalized, asn_valid: Active Learning step normalized and the validation accuracy and loss). This shows that adding the validation loss and accuracy don't help the surrogate model at all. Another point we would like to highlight is that in Table 5.10 the cherry-picked surrogate model is in every case worse than the random baseline. This could also explain why the best performing surrogate models are the ones that have the least amount of information (less inputs) and therefore have more random behaviour.

Closed-form solution ablation

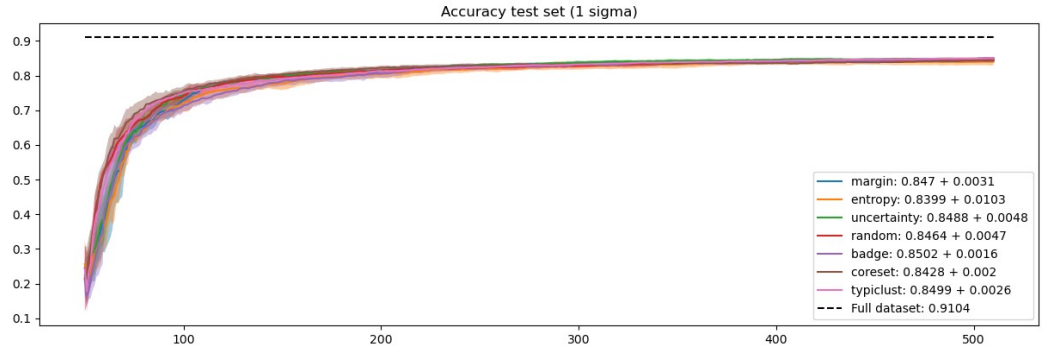
Another small study we did is on the importance of the first samples for the closed-form solution. Notice that the model is almost random before the number of instances are the same as the number of input dimensions (see Figure 5.5 b).

Importance of the initial samples

Those Active Learning methods that rely on the classifier should so no difference between sampling the first instances randomly or with that specific strategy. However, other strategies that don't rely on the classifier should be affected by this change. In particular, *Coreset* shows a better performance than random for this specific scenario. We perform an experiment where the first 50 samples are obtained at random and then compare it to the original set up.



(a) Solve starting the sampling at 10. Plot starts at 50 for a better visualization and comparison with figure (b).



(b) Solve starting the sampling at 50.

Figure 5.10: Accuracy comparison between following a certain sampling strategy from the 10th to the 40th sample or doing it randomly. The backbone used is ResNet50. Every experiment has been repeated 5 times.

The visualization from figure 5.10, we can not derive any solid conclusion. In order to further investigate this, we will use the AUBC metric. In particular, we will focus on *Coreset* since it is a method independent on the classifier. Therefore, the first 50 steps should be helpfull when compared to

other methods that require a classifier.

	ResNet18	ResNet50	DinoV2
Coreset 10	0.6538 ± 0.0178	0.8029 ± 0.0095	**0.887 ± 0.0081**
Coreset 50	**0.6729 ± 0.0144**	**0.8035 ± 0.0062**	0.882 ± 0.0047

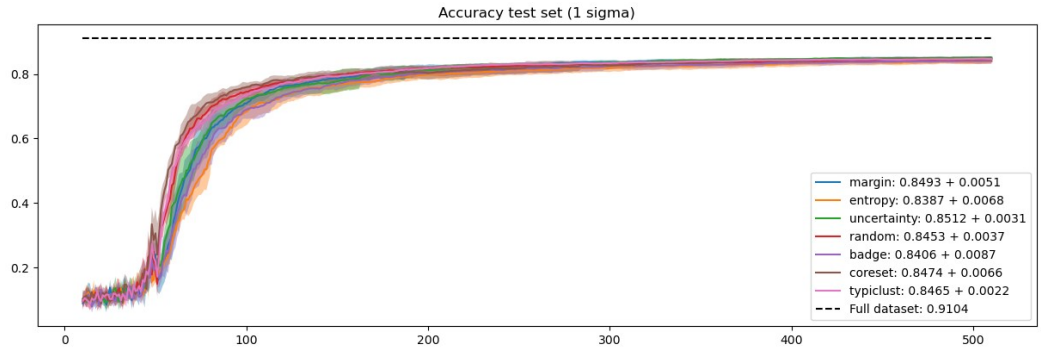
Table 5.11: AUBC comparison between starting to use Coreset strategy with the 10th sample or 50th sample. For a fair comparison, the AUBC considers only the accuracy from step 50 onwards, even if the sampling started on the 10th instance.

From Table 5.11, we can not state any end conclusion since the values are within one standard deviation for ResNet50 and DinoV2, and within 2 standard deviation for ResNet18.

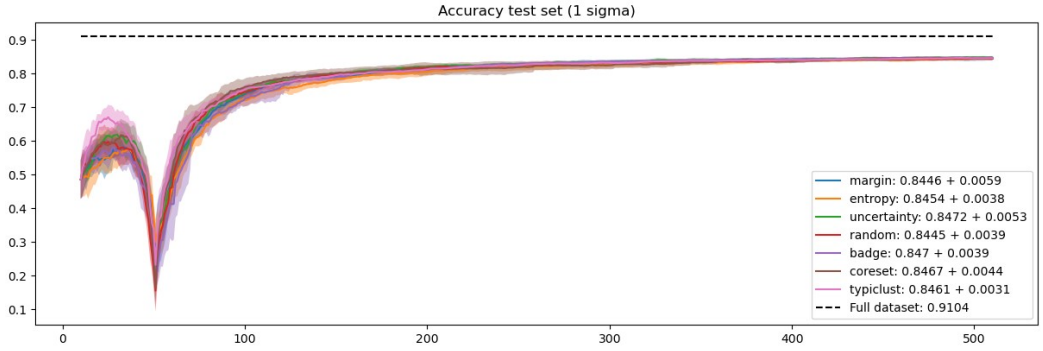
Penrose Inverse

Another idea would be to use the *Penrose Inverse* even if it has the unwanted behaviour of reducing its performance when the number of instances matches the number of input dimensions (see Figure 5.2). This is due to its dynamic behaviour of dynamically deleting some dimensions of the input. As shown in Figure 5.4, changing the amount of input dimensions can change the model's performance. We argue that in the early stages it is beneficial to have less dimensions since the model is more prone to catch on noise due to the little amount of instances. In later stages, more dimensions should be given since the model has more examples to learn from, which allows it to discern noise from correlations. We argue that *Penrose Inverse* benefits in the early stages from having less input dimensions but it increases the number of dimensions too fast and that's why it has the drop in performance.

We decide to perform an experiment of the Active Learning problem using *Penrose Inverse* as the learning method for the model (see Figure 5.11). We observe that the unwanted behaviour of collapsing when the number of inputs gets close to the number of input dimensions is still present. However, this way the methods that rely on the classifier for selecting instances have a better opportunity compared to using the solve method. We additionally report the AUBC for both cases in Table 5.12. In order to do a more sensible comparison we will consider the AUBC after the 50th iteration, since the results of the first 50 iterations is random for the solve method. Therefore,



(a) Solve method.



(b) Penrose Inverse method.

Figure 5.11: Accuracy comparison between the Penrose Inverse and Solve methods for the closed-form approximation. The backbone used is ResNet50. Every experiment has been repeated 5 times.

the full AUBC is biased and not representative of the impact that the first 50 samples from the *Penrose Inverse* have compared to the 50 pseudo-random samples from the solve method

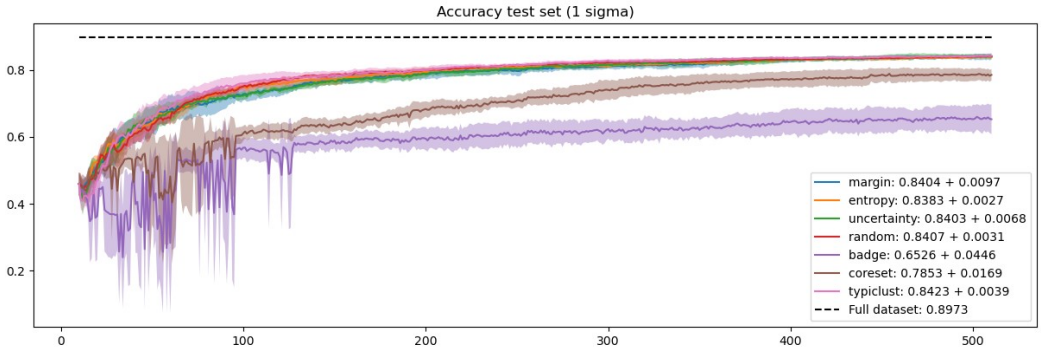
From Table 5.12, we would like to focus on the *Margin*, *Entropy* and *Least Confidence* columns since these methods are the ones that rely on the classifier to sample new instances. In only 1 of the 9 experiments we report a better performance with the *Penrose Inverse* compared to solve method. Therefore, we conclude that the first 50 samples selected with the *Penrose Inverse* don't improve the classifier compared to the ones selected using solve method.

Training	Random	Entropy	Margin	Least Confidence	Typiclust	Coreset	BADGE
ResNet18							
Solve	0.756 ± 0.0038	**0.7585 ± 0.0062**	**0.7572 ± 0.0106**	**0.7586 ± 0.0098**	0.7534 ± 0.0058	**0.7341 ± 0.0129**	0.7489 ± 0.0032
Penrose Inv.	**0.7561 ± 0.0047**	0.7424 ± 0.0103	0.7547 ± 0.0053	0.7563 ± 0.0041	**0.7559 ± 0.0028**	0.726 ± 0.0155	**0.7513 ± 0.0076**
ResNet50							
Solve	**0.8453 ± 0.0037**	0.8387 ± 0.0068	**0.8493 ± 0.0051**	**0.8512 ± 0.0031**	**0.8465 ± 0.0022**	**0.8474 ± 0.0066**	0.8406 ± 0.0087
Penrose Inv.	0.8445 ± 0.0039	**0.8454 ± 0.0038**	0.8446 ± 0.0059	0.8472 ± 0.0053	0.8461 ± 0.0031	0.8467 ± 0.0044	**0.847 ± 0.0039**
DinoV2							
Solve	**0.9101 ± 0.0028**	**0.9092 ± 0.0048**	**0.9151 ± 0.002**	**0.9192 ± 0.0032**	**0.9156 ± 0.0016**	0.9138 ± 0.0034	0.9149 ± 0.0033
Penrose Inv.	**0.9101 ± 0.004**	0.9089 ± 0.0059	0.9138 ± 0.0054	0.9158 ± 0.0025	0.9133 ± 0.0017	**0.9139 ± 0.0033**	**0.9163 ± 0.0034**

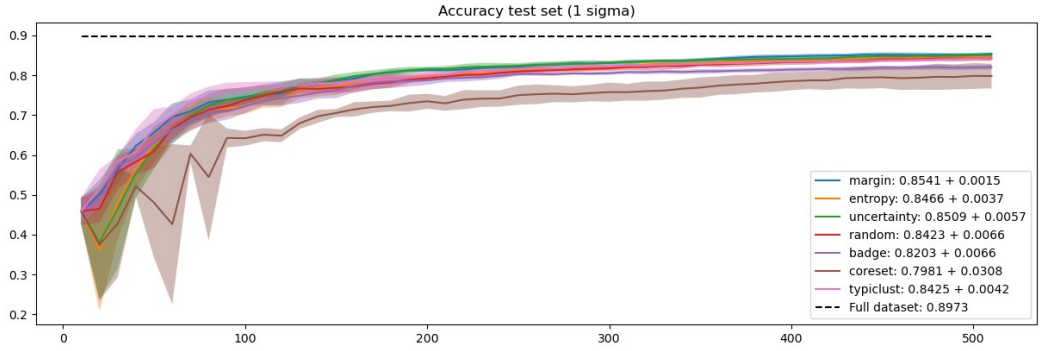
Table 5.12: AUBC after the 50th step for different methods and different-closed form approximations. All the experiments have been repeated 5 times.

Query size: Instance vs Batch Active Learning

A typical discussion in Active Learning is whether or not it is reasonable to sample more than one instance at a time. Most heuristics don't have a theoretical/practical reason to sample more than one instance at a time.



(a) Instance sampling.



(b) Batch sampling.

Figure 5.12: Accuracy comparison between batch and instance sampling with a gradient based training. The backbone used is ResNet50. Every experiment has been repeated 5 times.

Methods that rely on the classifier (*Entropy*, *Margin*, *Least Confidence*, etc.) should technically work better with more accurate classifier. Methods that establish some relations between the labeled data and unlabeled data should also perform better knowing what are the instances in the batch that have been sampled to label, which doesn't happen in most methods (*Coreset*, *Typiclust*, etc.). Nevertheless, some methods are explicitly created for the batch setting (*BADGE*) with the instance and batch sampling following different procedures.

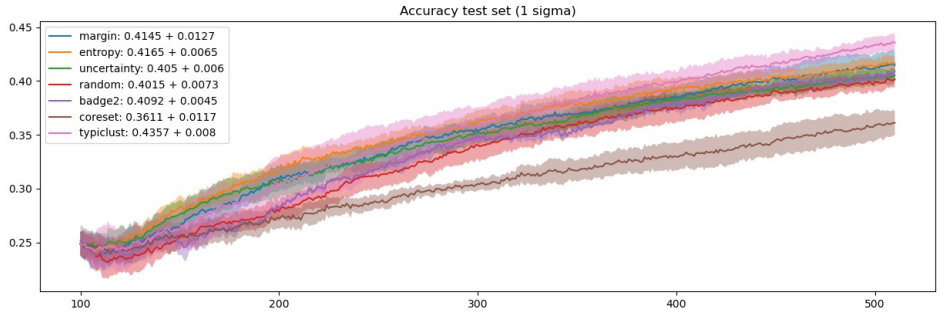
In this section we would like to test the performance differences between the batch setting (sample more than one instance at a time) and the instance setting (sample one instance at a time). We report the results in Figure 5.12.

We observe in Figure 5.12 a huge boost in performance for *BADGE* sampling strategy. Nevertheless, the batch version of BADGE adds a lot of randomness to the sampling that doesn't happen in the instance setting. We will refrain to argue if this improvement is due to the randomness of the batch process that makes the performance closer to the random baseline or if it is due to the underlying batch strategy. For the rest of the sampling strategies, no noticeable differences are observed.

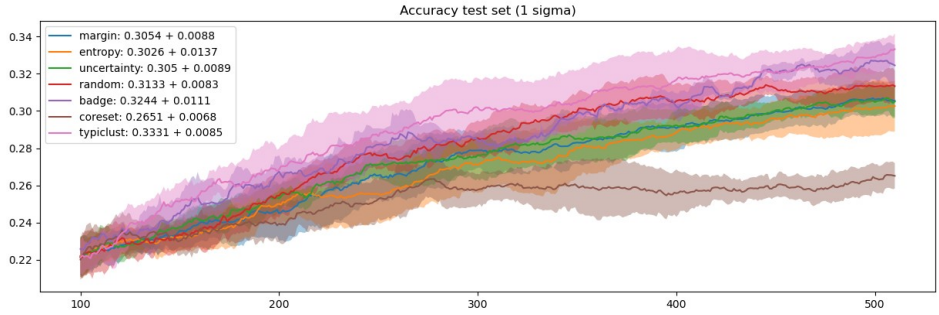
Dataset ablation: CIFAR100

Lastly, we will perform a small study on CIFAR100. The idea is to show the main results shown for CIFAR10 in other datasets. In order to have a sensible starting point, we will initialize the labeled set with 100 instances (1 of each class). The budget size will still be the same. We report the results in Figure 5.13.

In Figure 5.13 we observe a much bigger difference between the different Active Learning strategies compared to CIFAR10 results (Fig. 5.5). In particular, *Entropy*, *Margin*, *Least confidence* and *Typiclust* are consistently outperforming random in the gradient based setting. Additionally, *Typiclust* is also consistently beating the rest of the methods in the closed-form solution setting. This contradicts our previous findings about *Coreset* being better for the closed-form solution and *Typiclust* being better for the gradient based method (Sec. 5.5).



(a) Gradient based training.



(b) Closed form solution.

Figure 5.13: Accuracy comparison for CIFAR100. The backbone used is ResNet50. Every experiment has been repeated 5 times.

One important distinction we can see here is that the closed-form solution is performing way worse than the gradient based method, around a 10% worse in the test accuracy. We hypothesize that the number of PCA dimensions used are not enough to convey the underlying data structure.

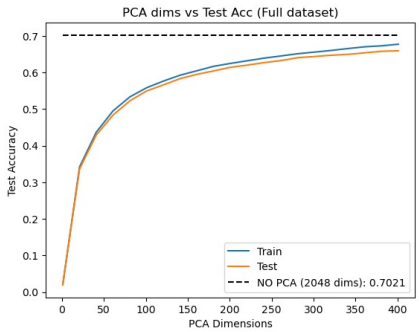


Figure 5.14: Impact of the number to the PCA dimensions in the accuracy for CIFAR100 dataset. The backbone used is ResNet50.

We test this hypothesis in Figure 5.14. From the experiment we conclude that 50 dimensions is not enough for the model to be able to understand the underlying data. In order to tackle this we repeat the closed form solution experiment using 150 dimensions for the PCA compression algorithm. We select 150 as a compromise between the accuracy that the model can achieve in the full dataset and the number of selected samples. We still want to have a good performance with very few labels. We report the results in Figure 5.15.

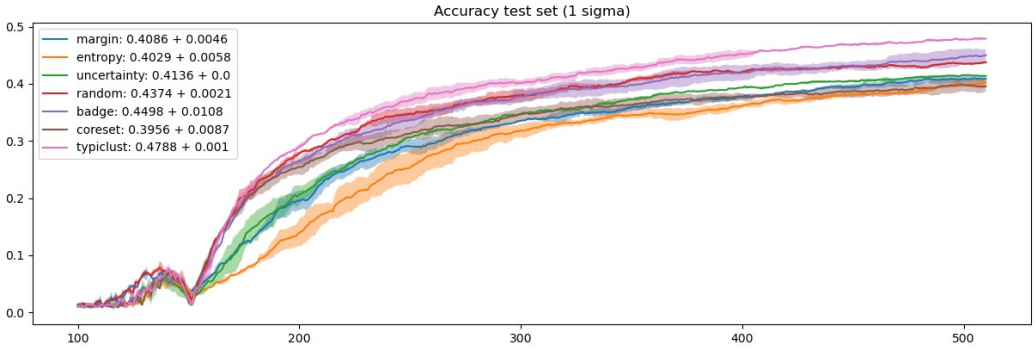


Figure 5.15: Accuracy of different Active learning strategies using the closed-form solution with 150 PCA dimensions. Backbone used is ResNet50. Only 2 repetitions have been done for this experiments due to time constrains.

From Figure 5.15 we observe that:

- There was a huge boost in performance due to a better selection of the number of PCA dimensions.
- We discard the hypothesis that *Coreset* is better than *Typiclust* in the closed-form solution regime by stating that it is comparable on the early stages, but drop in performance later on.

After observing these changes between CIFAR10 and CIFAR100, we arrive to the conclusion that one of the reasons why the Active Learning methods might not be showing good results could be that most of the heavy lifting is done by the encoder, needing very few and not representative samples to get to a reasonably good model. Nevertheless, in CIFAR100 with 10 times more classes a lot more instances are needed for having a good model. This could explain why some of the results are suboptimal and not much improvement could be achieved over the *Random* baseline.

Chapter 6

Conclusion

The main findings from this thesis are:

- Closed-form approximations can solve the Active Learning problem with comparable performance to gradient based methods in a faster way and less hyperparameter tuning (for a linear classifier).
- Closed-form approximations are more robust to the sampling strategy used compared to gradient based training.
- We show empirically that different Active Learning sampling strategies perform better/worse depending on the classifier training procedure and dataset.
- Active Learning strategies that use an approximation of the cross entropy loss do not perform well for a gradient based model. Closed-form approximation might improve in the early stages, but it degrades pretty quickly with worse performance than other methods on the long run.
- Too simple scenarios for linear models could lead to most Active Learning baselines reporting a result comparable to *Random* baseline.

6.1 Future work

Some ideas that can expand on this thesis are:

- Learning a surrogated model in a more complex environment than CIFAR10 with a pre-trained encoder. As shown before, to easy scenarios don't allow to compare the different Active Learning sampling strategies.
- Use regularization for the closed-form approximation (OLS, L1 norm, elastic net, etc.)
- Use online hyperparameter tuning. This could be especially useful to increase the PCA dimensions along with the sampled instances. In particular, a slower version of the dynamical reduction of the *Penrose Inverse* should produce notable lifts in performance when the number of samples is close to the number of original input dimensions.
- Use the closed-form approximation as a weight initialization for the gradient based methods.
- Train a surrogate model on several targets, for example the increase in loss and accuracy at the same time. Then use a combination of those predictions from the surrogate model to sample new instances.

Bibliography

[Angluin, 1988] Angluin, D. (1988). Queries and concept learning. *Machine Learning*, 2:319–342.

[Arthur, 2007] Arthur, D.; Vassilvitskii, S. (2007). k-means++: the advantages of careful seeding. *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms.*, page 1027–1035.

[Ash et al., 2020] Ash, J. T., Zhang, C., Krishnamurthy, A., Langford, J., and Agarwal, A. (2020). Deep batch active learning by diverse, uncertain gradient lower bounds.

[Atlas et al., 1989] Atlas, L. E., Cohn, D. A., and Ladner, R. E. (1989). Training connectionist networks with queries and selective sampling. In *NIPS*.

[Ba et al., 2016] Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization.

[Bachman et al., 2014] Bachman, P., Alsharif, O., and Precup, D. (2014). Learning with pseudo-ensembles.

[Balestriero et al., 2023] Balestriero, R., Ibrahim, M., Sobal, V., Morcos, A., Shekhar, S., Goldstein, T., Bordes, F., Bardes, A., Mialon, G., Tian, Y., Schwarzschild, A., Wilson, A. G., Geiping, J., Garrido, Q., Fernandez, P., Bar, A., Pirsiavash, H., LeCun, Y., and Goldblum, M. (2023). A cookbook of self-supervised learning.

[Beluch et al., 2018] Beluch, W. H., Genewein, T., Nürnberger, A., and Köhler, J. M. (2018). The power of ensembles for active learning in image classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Cao and Tsang, 2021] Cao, X. and Tsang, I. W. (2021). Bayesian active learning by disagreements: A geometric perspective.

[Caron et al., 2021a] Caron, M., Misra, I., Mairal, J., Goyal, P., Bojanowski, P., and Joulin, A. (2021a). Unsupervised learning of visual features by contrasting cluster assignments.

[Caron et al., 2021b] Caron, M., Touvron, H., Misra, I., Jégou, H., Mairal, J., Bojanowski, P., and Joulin, A. (2021b). Emerging properties in self-supervised vision transformers.

[Chen and Guestrin, 2016] Chen, T. and Guestrin, C. (2016). XGBoost. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM.

[Chen et al., 2020] Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. (2020). A simple framework for contrastive learning of visual representations.

[Deng et al., 2009] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*.

[Dosovitskiy et al., 2021] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. (2021). An image is worth 16x16 words: Transformers for image recognition at scale.

[F.R.S., 1901] F.R.S., K. P. (1901). Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, 2:11, 559-572*.

[Gao et al., 2020] Gao, M., Zhang, Z., Yu, G., Arik, S. O., Davis, L. S., and Pfister, T. (2020). Consistency-based semi-supervised active learning: Towards minimizing labeling cost.

[Grill et al., 2020] Grill, J.-B., Strub, F., Altché, F., Tallec, C., Richemond, P. H., Buchatskaya, E., Doersch, C., Pires, B. A., Guo, Z. D., Azar, M. G., Piot, B., Kavukcuoglu, K., Munos, R., and Valko, M. (2020). Bootstrap your own latent: A new approach to self-supervised learning.

[Grinsztajn et al., 2022] Grinsztajn, L., Oyallon, E., and Varoquaux, G. (2022). Why do tree-based models still outperform deep learning on tabular data?

[Hacohen et al., 2022] Hacohen, G., Dekel, A., and Weinshall, D. (2022). Active learning on a budget: Opposite strategies suit high and low budgets.

[He et al., 2015] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition.

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Identity mappings in deep residual networks.

[Hotelling, 1933] Hotelling, H. (1933). Analysis of a complex of statistical variables into principal components. *TJournal of Educational Psychology, 24(6), 417–441.*

[Kim et al., 2021] Kim, B., Choo, J., Kwon, Y.-D., Joe, S., Min, S., and Gwon, Y. (2021). Selfmatch: Combining contrastive self-supervision and consistency for semi-supervised learning.

[Krizhevsky, 2009] Krizhevsky, A. (2009). Learning multiple layers of features from tiny images.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C., Bottou, L., and Weinberger, K., editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc.

[LeCun et al., 1989] LeCun, Y., Boser, B. E., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. E., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1:541–551.

[Lewis and Gale, 1994] Lewis, D. D. and Gale, W. A. (1994). A sequential algorithm for training text classifiers. *CoRR*, abs/cmp-lg/9407020.

[Loshchilov and Hutter, 2019] Loshchilov, I. and Hutter, F. (2019). Decoupled weight decay regularization.

[Moore, 1920] Moore, E. H. (1920). On the reciprocal of the general algebraic matrix. *Bulletin of the American Mathematical Society*, 26(9):394–395.

[Müller and Hutter, 2021] Müller, S. G. and Hutter, F. (2021). Trivialaugment: Tuning-free yet state-of-the-art data augmentation. *CoRR*, abs/2103.10158.

[Nameer Hirschkind, 2023] Nameer Hirschkind, Saruque Mollick, A. F. B. M. . J. K. (2023).

[Noroozi and Favaro, 2017] Noroozi, M. and Favaro, P. (2017). Unsupervised learning of visual representations by solving jigsaw puzzles.

[Oquab et al., 2023] Oquab, M., Darcet, T., Moutakanni, T., Vo, H., Szafraniec, M., Khalidov, V., Fernandez, P., Haziza, D., Massa, F., El-Nouby, A., Assran, M., Ballas, N., Galuba, W., Howes, R., Huang, P.-Y., Li, S.-W., Misra, I., Rabbat, M., Sharma, V., Synnaeve, G., Xu, H., Jegou, H., Mairal, J., Labatut, P., Joulin, A., and Bojanowski, P. (2023). Dinov2: Learning robust visual features without supervision.

[Ouali et al., 2020] Ouali, Y., Hudelot, C., and Tami, M. (2020). An overview of deep semi-supervised learning.

[Ruan et al., 2023] Ruan, Y., Singh, S., Morningstar, W., Alemi, A. A., Ioffe, S., Fischer, I., and Dillon, J. V. (2023). Weighted ensemble self-supervised learning.

[Sablayrolles et al., 2019] Sablayrolles, A., Douze, M., Schmid, C., and Jégou, H. (2019). Spreading vectors for similarity search.

[Sener and Savarese, 2018] Sener, O. and Savarese, S. (2018). Active learning for convolutional neural networks: A core-set approach.

[Settles, 2009] Settles, B. (2009). Active learning literature survey.

[Shukla and Ahmed, 2021] Shukla, M. and Ahmed, S. (2021). A mathematical analysis of learning loss for active learning in regression. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE.

[Song et al., 2019] Song, S., Berthelot, D., and Rostamizadeh, A. (2019). Combining mixmatch and active learning for better accuracy with fewer labels.

[Strang, 1980] Strang, G. (1980). Linear algebra and its applications. *Orlando, FL, Academic Press, Inc.*

[Touvron et al., 2022] Touvron, H., Vedaldi, A., Douze, M., and Jégou, H. (2022). Fixing the train-test resolution discrepancy.

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. *CoRR*, abs/1706.03762.

[Vryniotis, 2021] Vryniotis, V. (2021). Pytorch : How to train state-of-the-art models using torchvision's latest primitives.

[Yoo and Kweon, 2019] Yoo, D. and Kweon, I. S. (2019). Learning loss for active learning.

[Zhang et al., 2016] Zhang, R., Isola, P., and Efros, A. A. (2016). Colorful image colorization.

Chapter 7

Appendix

7.1 Active Learning

In this thesis we acknowledge the fact that there are somer *Active Learning* methods that have not been used. Nevertheless, most of them require a specific model architecture that didn't satisfy our specific case. In particular, they either require a dropout layer or to use an ensemble. Some of those methods are:

- Monte-Carlo [Beluch et al., 2018]: Given a specific *Active Learning* metric, calculate the average after passing the input trough the model with dropout active or on 'train' mode.

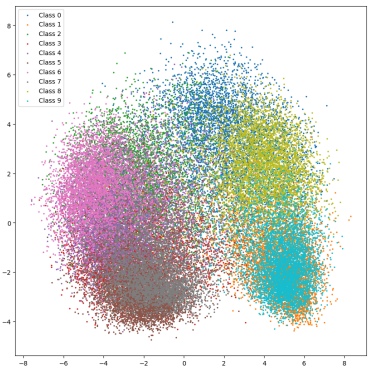
  $argmax/min_{x_u} \frac{1}{K} \sum_k metric(\hat{y}(x_u; dropout))$

- Ensembles [Beluch et al., 2018]: Given a specific *Active Learning* metric, calculate the average after passing the input trough the different models.

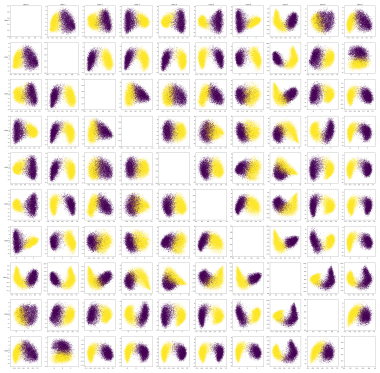
  $argmax/min_{x_u} \frac{1}{K} \sum_k metric(\hat{y}(x_u; \theta_k))$

- BALD [Cao and Tsang, 2021]: Similarly to Monte-Carlo, but it specific for the entropy and checks the difference between the entropy of the averaged prediction compared to the averaged entropy. It tries to separate the instance noise and the model noise.

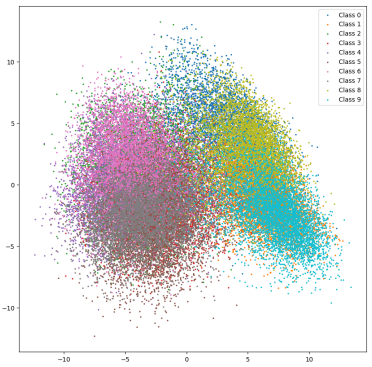
  $argmax_{x_u} Entropy(\frac{1}{K} \sum_k \hat{y}(x_u; dropout)) - (\frac{1}{K} \sum_k Entropy(\hat{y}(x_u; dropout)))$



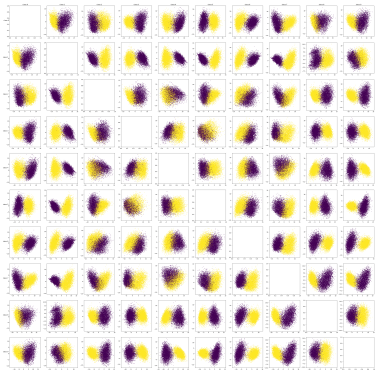
(a) Resnet50 full dataset



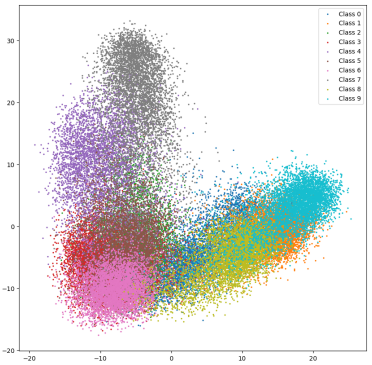
(b) ResNet50 class comparison



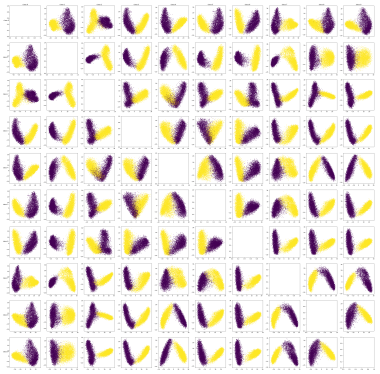
(c) Resnet18 full dataset



(d) ResNet18 class comparison



(e) DinoV2 full dataset



(f) DinoV2 class comparison

Figure 7.1: Backbones comparison. The left column images are the full datasets. The right column images are the pairwise classes comparisons.

7.2 Backbones visual comparison

In order to prove the capabilities of the different backbones, we visualize if any 2 classes of the given embeded data are linearly separable. In order to do this, we consider the points of any 2 classes and plot them (with PCA) to show if there is a clear boundary between both sets. The results are shown in Figure 7.1.

In those images, we can observe that even if in the pairwise comparison all the backbones tend to have a good split between classes. DinoV2 is the one with less amount of outliers/noise in all the different classes comparison. This could explain why the results with the DinoV2 backbone are better than the ResNet backbones. A similar trend can be observed between ResNet50 and ResNet18, where ResNet50 has less overlap between the different classes.

Please notice that even if the splits are somewhat noisy, the PCA dimensionality reduction selects the 2 dimensions with the highest amount of variations. Therefore, it is possible to find other dimensions where the data has less variation but have a better split between the different classes. Due to this fact, it is important to also prove this hypothesis using a linear classifier with all the different backbones (Table 5.2).

7.3 Active Learning performance

In this section we show the test curves for the Active Learnin procedure with different backbones, the closed-form and gradient based methods.

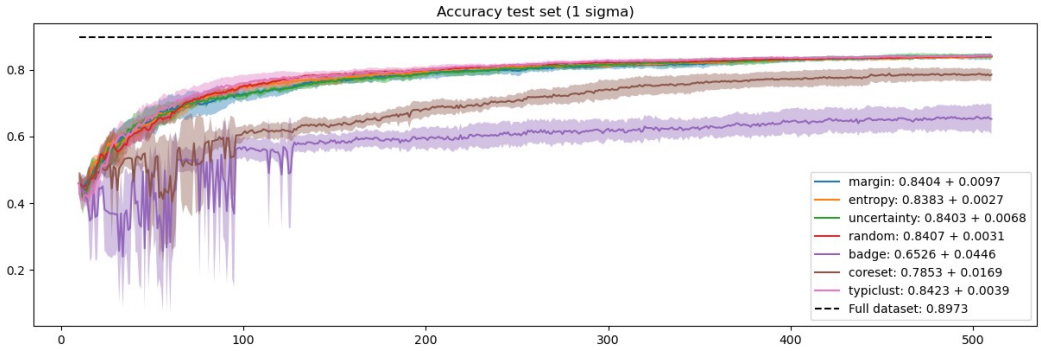


Figure 7.2: ResNet50 - AdamW

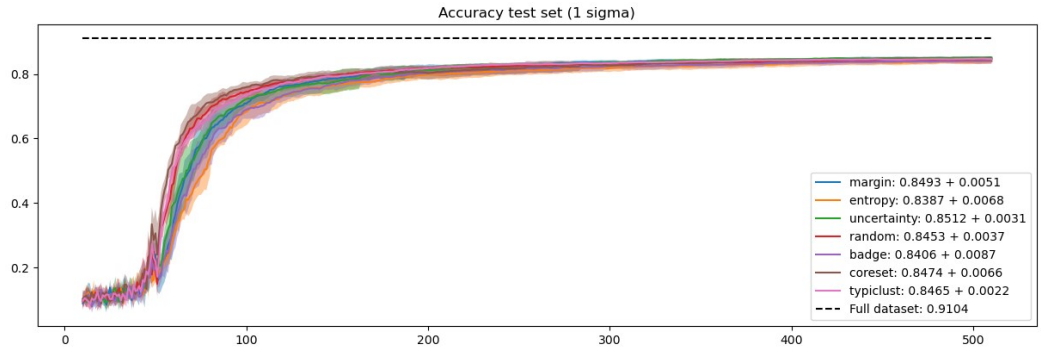


Figure 7.3: ResNet50 - Solve (PCA 50)

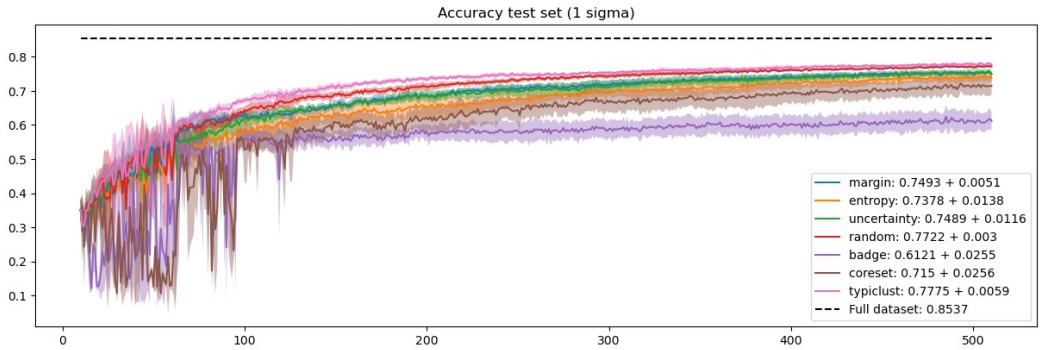


Figure 7.4: ResNet18 - AdamW

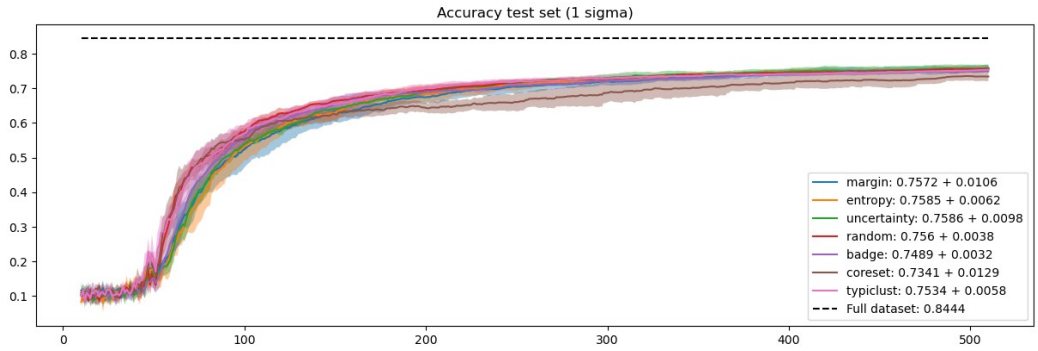


Figure 7.5: ResNet18 - Solve (PCA 50)

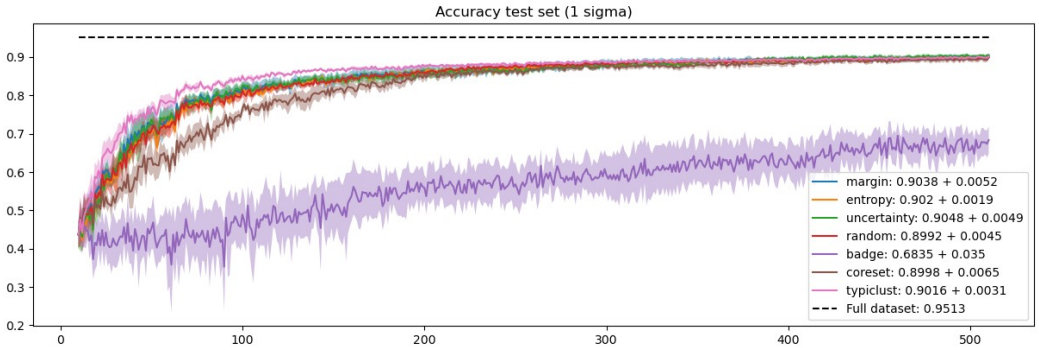


Figure 7.6: DinoV2 - AdamW

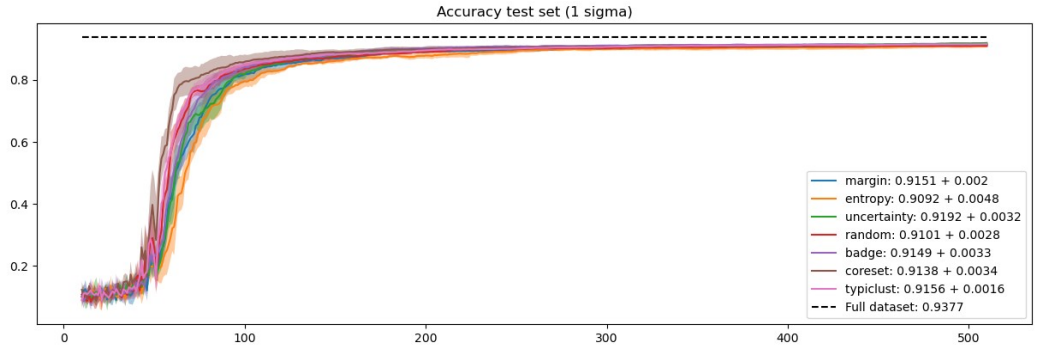


Figure 7.7: DinoV2 - Solve (PCA 50)

7.4 MLP ablation

Here we show the results for the MLP classifier with different backbones.

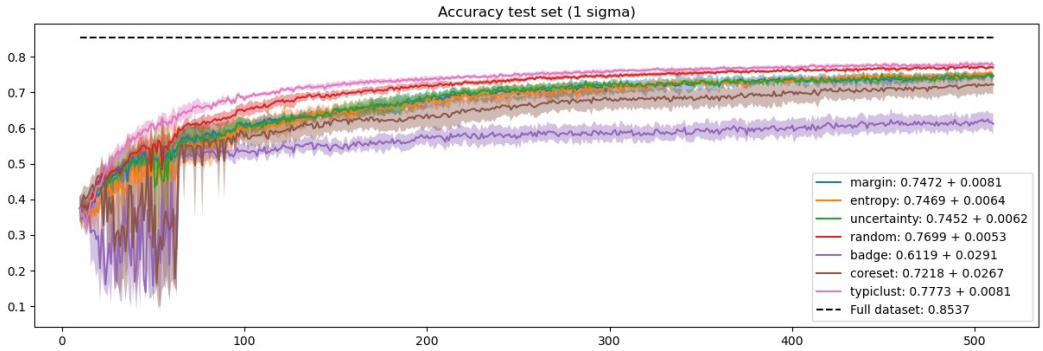


Figure 7.8: ResNet18 - MLP

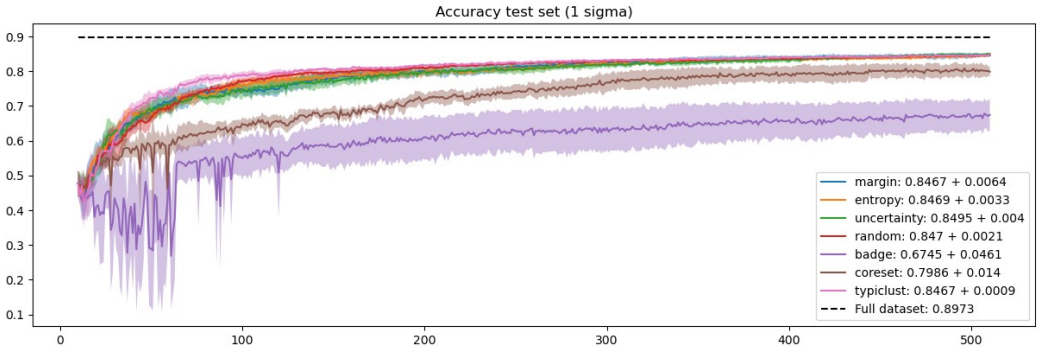


Figure 7.9: ResNet50 - MLP

7.5 Surrogate model : Accuracy + AL step + Valid loss/acc

In this section we show the performance of the selected surrogate model for the ResNet18 and DinoV2 backbones .

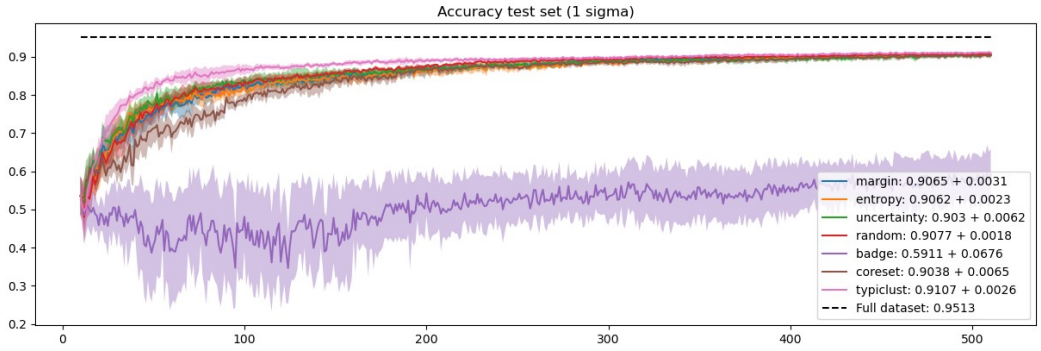


Figure 7.10: DinoV2 - MLP

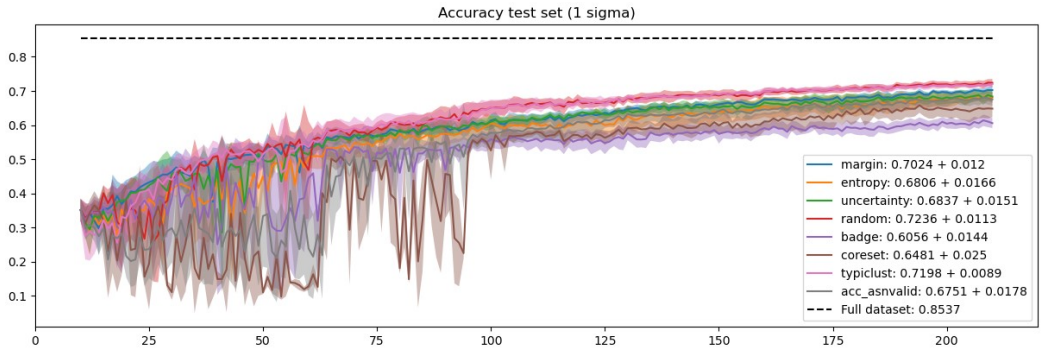


Figure 7.11: ResNet18 - AdamW

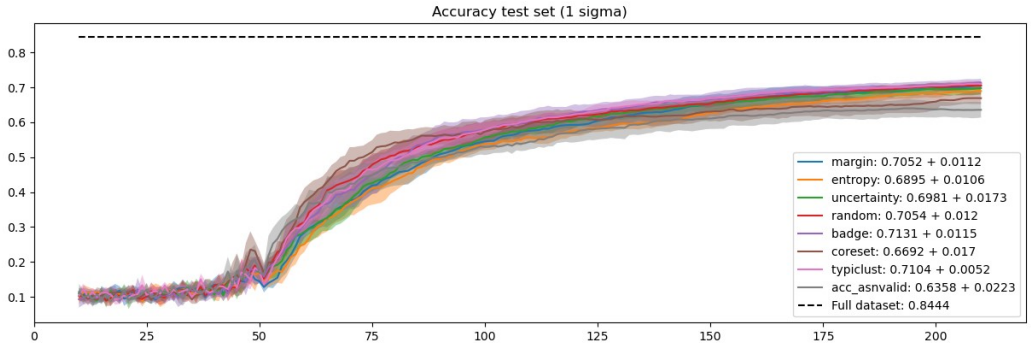


Figure 7.12: ResNet18 - Solve

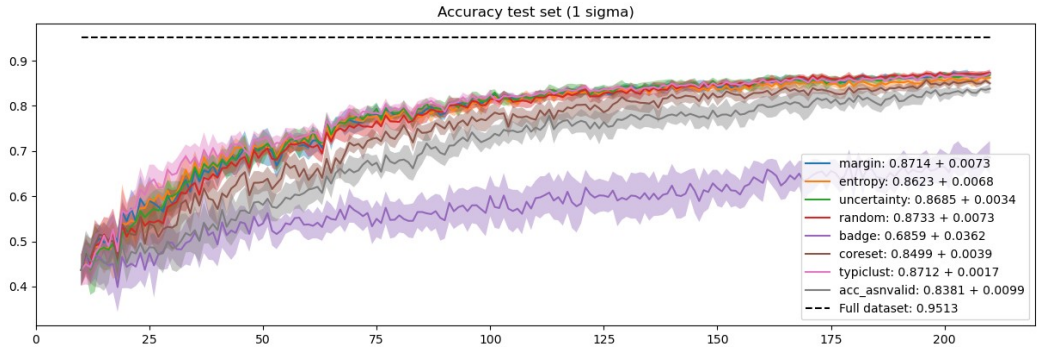


Figure 7.13: DinoV2 - AdamW

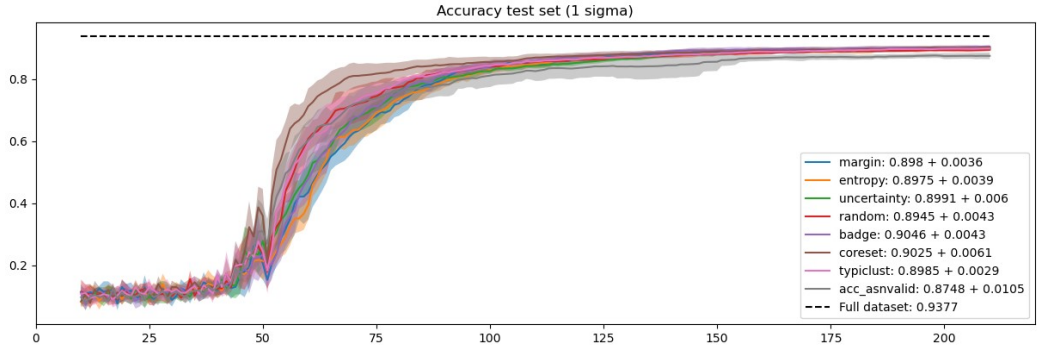


Figure 7.14: DinoV2 - Solve