

# Idea

Our sampling approach can be considered as a dropout measure. During training, it is useful as a regularization, during testing, it is useful for saving time/space.

Now that NN-kNN is working, we can apply it to three things: an image data set, CBIS-DDSM, a language data set, and maybe a task involving adaptation (regression task?) to show case that NN-kNN is a powerful CBR method that utilize the benefits of deep learning for modern AI data sets.

Let’s see what’s the drosophila of these domains, computer vision, natural natural language processing.

1. DONE (low-hanging) local global feature weighting, where each case has a vector representing what set of feature weighting it should use. Not sure how useful this will be though. I am not sure what kind of problem this can solve. Update: maybe use for one-shot learning? See idea number 3.

Could be a dense layer from the surface feature (problem description) towards a feature weight vector.

1. Weight Choice Vector + sets of Feature Weight Vector.
2. Generate Feature Weight directly.

**DONE. Optimize the layer calculation so it does not depend on the number of cases.**

Each case should be a network module by itself (named as “case network”). The final model will be working by sampling a batch of case networks to use. Effectively, given a query, it is comparing the query with a batch of cases.

By doing this, we allow easy addition/delete of the cases. This will be beneficial for case base maintenance later.

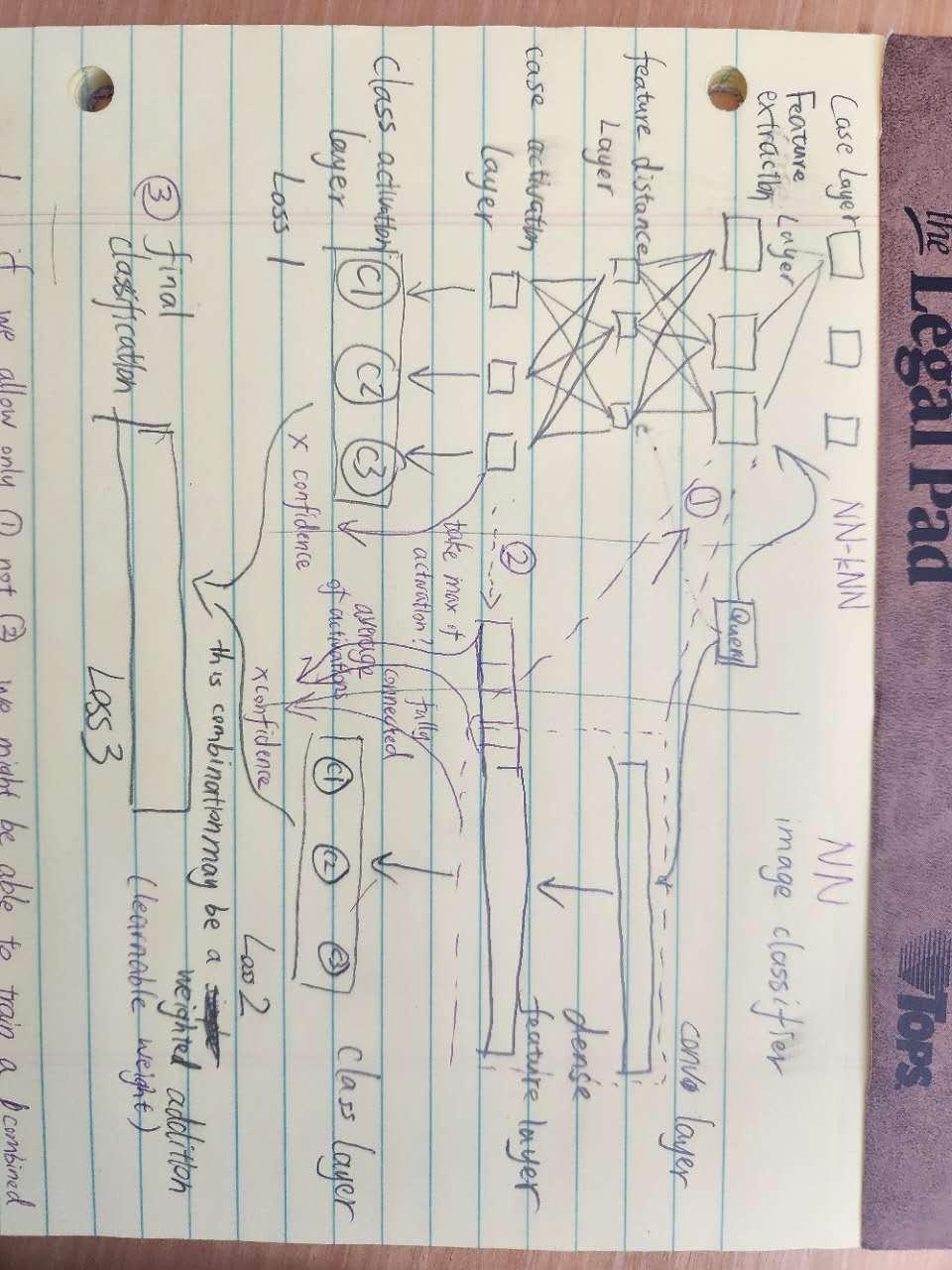
Each case network should keep track of provenance or utility of the case. How often it is activated, how often it is sampled, how often it correctly classifies a query.

There will be some overhead for this approach (I am not sure myself, does anybody know?), an NN-kNN could potentially speed things up by choosing a few case networks and consolidating itself into an old style NN-kNN.

On the other hand, using case network might allow parallel/distributed computation easily. **They can even be trained in distributed manner.**

The case networks share certain components, for example, they share feature extraction layers (they may have their glocal weightings).

1. (high-in-the-air) Paralleling NN-kNN with a standard neural network, allow their information to flow both ways. This way maybe we can make a neural network to reason based on cases too. Can we make it play Chess?
2. One shot learning? inspired by [**Matching Networks for One Shot Learning**](https://arxiv.org/pdf/1606.04080)**.** Since our model is k-nn, we should naturally be good for one shot learning. We could reuse the same experiment in that paper. We will have to use global weighting, or the new glocal weighting, because if we one-shot a class, we will not have class related weighting for that one-shot class. We will have to use a global weighting that applies to all classes, or use a glocal weighting.
3. Combining ideas 2 and 3



information flow 1. feature extraction for NN is used for feature extraction in kNN (short for NN-kNN here) Similar to [NOC CBR](https://www.researchgate.net/publication/328152888_Novel_Object_Discovery_Using_Case-Based_Reasoning_and_Convolutional_Neural_Networks_26th_International_Conference_ICCBR_2018_Stockholm_Sweden_July_9-12_2018_Proceedings)

Flow 1 may use multiple layers as the feature extraction, or just early layers, (more interpretable).

information flow 2. Case activation will tell kNN how confident it should be (based on highest activated case) and how confident NN should be (based on average of activations, or maybe this should be done by a dense layer on the features extracted?). The reasoning is based on the outcome of the model, if kNN predicts right but NN predicts wrong, then kNN shouldn’t be punished, so its feature distance, case activation and class activation layer would not be punished, but NN should be. This will drive down the average case activation, drive up the max case activation, which is intended. If kNN predicts wrong but NN predicts right, then the max activation will be driven down (it shouldn’t have been the nearest neighbor), but NN’s feature extraction process shouldn’t be punished. If both predict wrong or both predict right, they will be rewarded/punished together.

information flow 3. the classification from the two models are combined using a weight, the weight is based on confidence, which is based on max activation and average activation (or a confidence based on the features extracted), respectively. This is to ensure information flow 2 works and can be properly trained.

A couple of note:

Confidence 2 may not depend on NN-kNN at all. Maybe sometimes hallucination/imagination of the NN model is desired and we don’t want NN-kNN to interfere with that. In this case, maybe confidence 2 is a learned parameter from the features (dense layer from features -> confidence2).In other words, information flow 2 may be undesirable.

Confidence 2 might be a dense layer from case activations. (It might be able to learn by itself into a threshold mechanism, if it’s just a single dense layer)

This whole model can be ablated for ablation comparison, we can train for loss1, loss2, loss3 or some combinations of the three. We might force one of the confidence to be 0, so we train only the kNN or the NN.

Benefit for NN, can now potentially work with outliers, one-shot or few-shot scenarios.

Benefit for NN-kNN, can now store a minimum number of cases and leave the intuition part to NN.

To best let them complement each other, NN-kNN would be best if it stores samples on the class boundary.

Maybe during training, if NN is confident and correct, we can drive down the case weights of activated cases in NN-kNN. If they are below a certain threshold, they can be safely deleted?

ONLINE LEARNING?? Maybe as the model gets more and more cases, they are first stored in NN-kNN. As the model progress, cases in NN-kNN are assimilated into the parameters of NN, and k-NN can safely delete them?

The big issue right now is, how should case maintenance work?

We shouldn’t delete a case just because NN is doing well enough for a query. Because if we do, we lose explainability of NN-kNN for that query. But we can delete some cases which serve the same purpose (answers the same query).

DONE I think we need to do glocal feature weighting, so that adding/deleting case is easy. This also means revamping the NN-kNN module right now, so that the matrix calculation does not depend on the number of cases, as they may increase or decrease.

Confidence1 and confidence2 should be related. Because being confident in 2, means that we don’t need to rely on 1. Rebute: this idea is wrong, we could be confident in both models at the same time.

There could be a loss measuring the discrepancy between model1 and model2.

**My main question, in other ML works, how do they align multiple models? Any general framework we can follow?**

In other words, how do we coordinate the two systems. We want NN-kNN to delete cases and become more light weight, but we don’t want it to delete all cases so that it loses explainability. We want NN to absorb as many cases as possible, but maybe not boundary cases to maintain overall accuracy?

Idea: we can give each case its own provenance. During the forward pass of the NN-kNN, updates the provenance/utility, the provenance should reflect the importance of the case, if it’s the sole/top case for a correct classification, compared to how often it is activated but not the most important, or wrongly activated.

# TODO

[case based reasoning aiplay chess - Search (bing.com)](https://www.bing.com/search?q=case+based+reasoning+aiplay+chess&qs=n&form=QBRE&sp=-1&lq=0&pq=case+based+reasoning+aiplay+chess&sc=6-33&sk=&cvid=E65FA17CCCBD4345A2897869991F1891&ghsh=0&ghacc=0&ghpl=)

[Information based explanation methods for deep learning agents—with applications on large open-source chess models | Scientific Reports (nature.com)](https://www.nature.com/articles/s41598-024-70701-2)

# DOING

[Improving Generalization via Scalable Neighborhood Component Analysis](https://arxiv.org/abs/1808.04699)

Applying NCA to image net, large image data set.

The two below are for textual

k[-Nearest Neighbor Augmented Neural Networks for Text Classification](https://arxiv.org/abs/1708.07863)

[kNN-LM](https://arxiv.org/pdf/2210.15859)

https://openreview.net/forum?id=HklBjCEKvH

Relevant, we may use wikitext-2m for our experiment later.

[Case Retrieval Reuse Net (CR2N): An Architecture for Reuse of Textual Solutions | SpringerLink](https://link.springer.com/chapter/10.1007/978-3-642-02998-1_3)

**Meta-learning with memory-augmented neural networks**

[**Neural Nearest Neighbors Networks (neurips.cc)**](https://proceedings.neurips.cc/paper_files/paper/2018/file/f0e52b27a7a5d6a1a87373dffa53dbe5-Paper.pdf)

**Distance Metric Learning**:

* Methods like **Large Margin Nearest Neighbor (LMNN)** and **Neighbourhood Components Analysis (NCA)** attempt to learn a global distance metric that optimizes k-NN classification. These are global weighting methods, but they can adapt to local patterns if the learned metric reflects important local structures.

**Locally Adaptive Distance Metrics**:

* **Local Learning-Based Feature Weighting (LLFW)** modifies the feature weights for each query point, making the method closer to local weighting. This type of method can blend global and local weighting by applying global metrics but adjusting weights based on the locality of the query point.

**Hybrid Methods**:

* Some methods aim to strike a balance between global and local weighting. For example, **Simultaneous Global and Local Learning Algorithms (SGLA)** aim to combine global feature importance with local adaptations. These methods start with global feature weights and then adjust the importance of features based on the locality of the samples.

**Adaptive k-NN**:

* In adaptive versions of k-NN, feature weights can be tuned depending on local neighborhoods but still maintain a global understanding. These methods adapt the feature weighting strategy depending on the data distribution in different regions of the feature space, which brings a balance between global and local information.

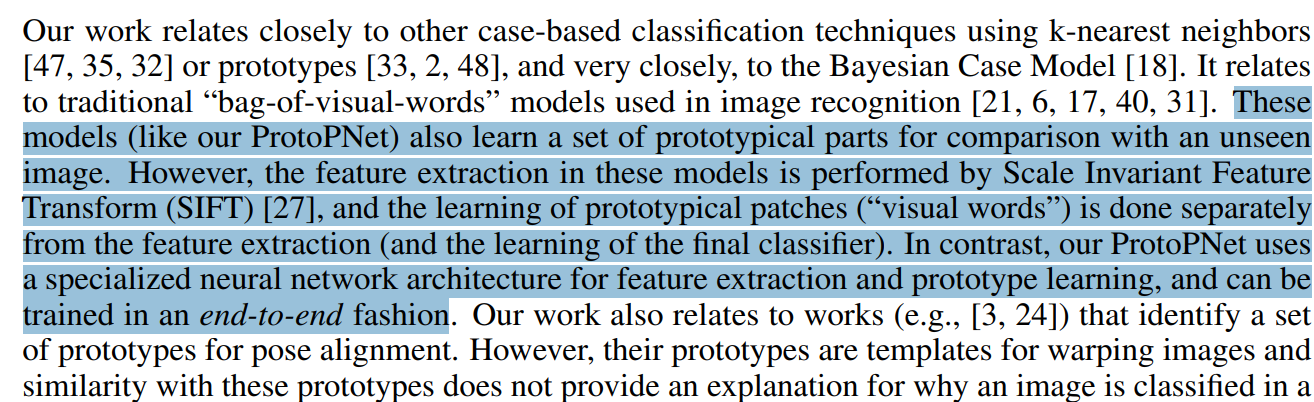
# DONE

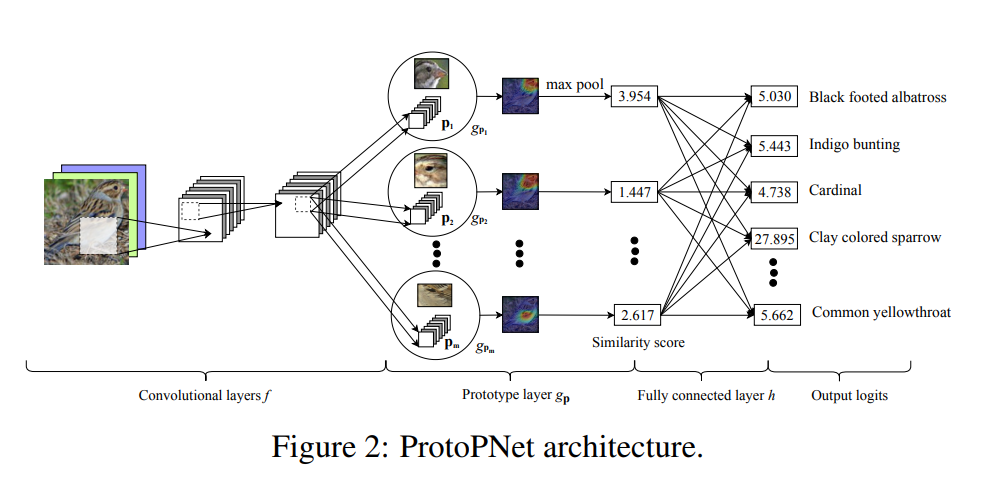
[This Looks Like That: Deep Learning for Interpretable Image Recognition](https://scholar.google.com/citations?view_op=view_citation&hl=en&user=rtpoh5wAAAAJ&citation_for_view=rtpoh5wAAAAJ:u-x6o8ySG0sC)

They use the bird data set CUB-200-2011

This extends [Deep learning for case-based reasoning through prototypes: a neural network that explains its predictions](https://dl.acm.org/doi/abs/10.5555/3504035.3504467)

From ProtoNet to prototypical part network (ProtoPNet)



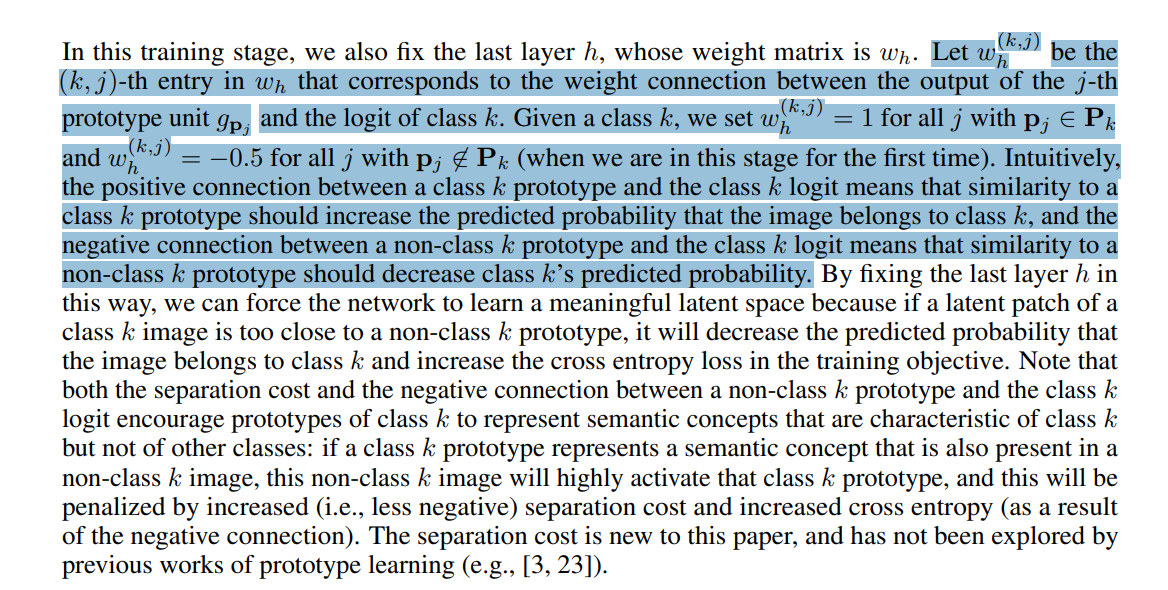


Very similar to ours.

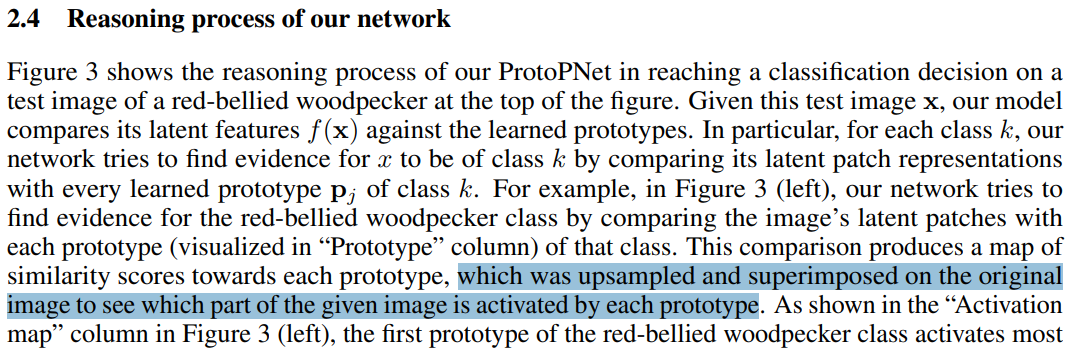
Their prototype layer is essentially our case layer

Their fully connected layer and output logits are mapping to our case activation + class activation layer.

The difference is that, for the later layers, theirs are more neural, ours more CBR symbolic.



This highlighted part is similar to how we set our class activation layer

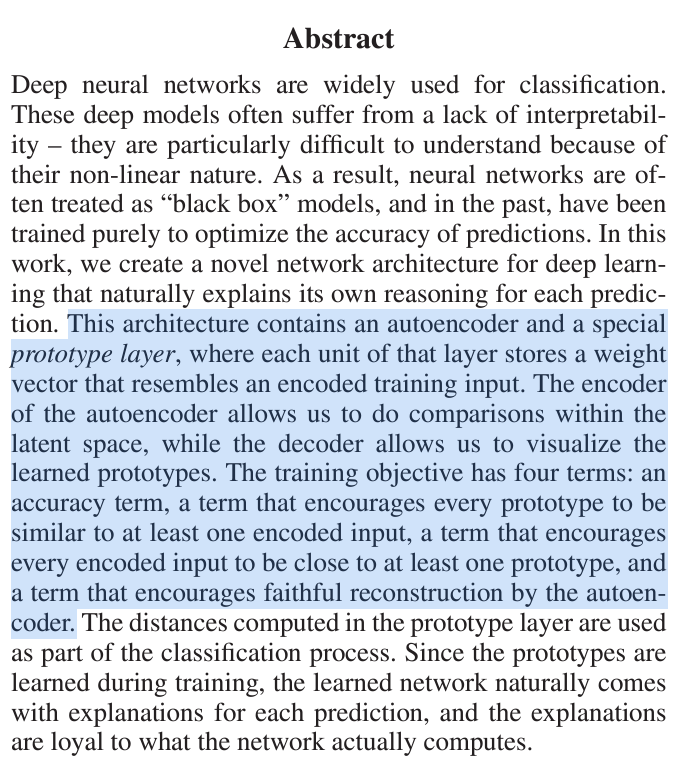


The highlighted part is something that we haven’t considered but could adopt as well. We may want to upsample and superimpose the most important features to explain on the input level (pixels highlighted)

[Deep learning for case-based reasoning through prototypes: a neural network that explains its predictions](https://dl.acm.org/doi/abs/10.5555/3504035.3504467)

Used on MNIST, cars, FashionMNIST

Very relevant, reading this again



[Deep k-Nearest Neighbors: Towards Confident, Interpretable and Robust Deep Learning](https://arxiv.org/abs/1803.04765)

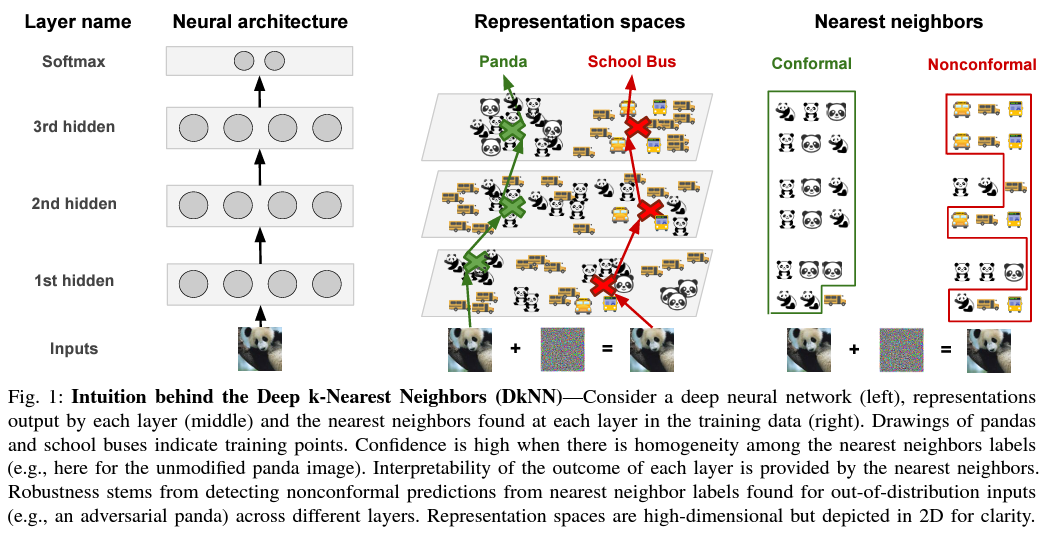
Data sets: MNIST, SVHN, GTSRB

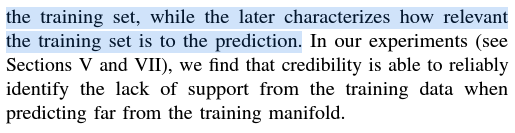
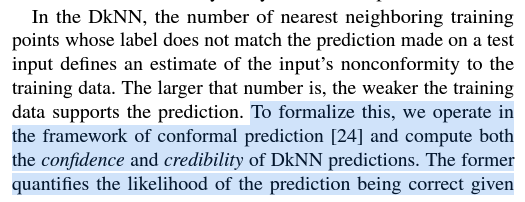
DkNN, very relevant. lol even the name is similar.

“For each layer in the DNN, the DkNN performs a nearest neighbor search to find training points for which the layer’s output is closest to the layer’s output on the test input of interest. We then analyze the label of these neighboring training points to ensure that the intermediate computations performed by each layer remain conformal with the final model’s prediction”

“Rather than nurturing model integrity by attempting to correctly classify all legitimate and malicious inputs, we ensure the integrity of the model by creating a novel characterization of confidence, called credibility, that spans the hierarchy of representations within of a DNN: any credible classification must be supported by evidence from the training data.”

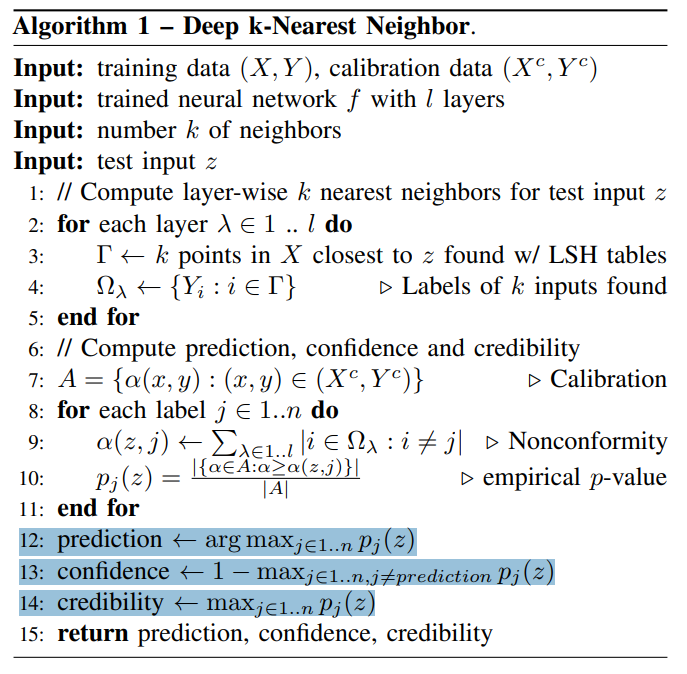
WE CAN DO THE SAME THING. We can added a credibility mechanism into our model.





“The confidence, interpretability and robustness of the DkNN are empirically evaluated respectively in Sections V, VI and VII”

Somewhat interesting, somewhat relevant, they are using conformal prediction to do this, basically, it checks the trend of a holdout set of samples to see how many their neighbors, according to each layer embedding, is of the same class. If the test sample show better conformality, less samples are of a different class than the trend of the holdout set, then the test sample is of high credibility/confidence. See pseudocode below.



In other words, the measures of confidence and credibility are based on past trend in the calibration set (the holdout set). I think it can be understood as a statistics-based method. This part honestly is not so relevant to our work.

The rest of the paper talks about adversarial attacks, which is also not directly related to our goal.

Neighbourhood Components Analysis

Jacob Goldberger, Sam Roweis, Geoff Hinton, Ruslan Salakhutdinov

This can be super relevant. NCA is “optimize a neural feature extractor explicitly for a kNN classifier”

“The algorithm directly maximizes a stochastic variant of the leave-one-out KNN score on the training set”



[Neighbourhood Components Analysis](https://proceedings.neurips.cc/paper/2004/file/42fe880812925e520249e808937738d2-Paper.pdf)

DONE

Linear transformation of feature spaces to lower the leave-one-out training loss. Relevant, see my screenshot of my thoughts above.

[Mahalanobis distance - Wikipedia](https://en.wikipedia.org/wiki/Mahalanobis_distance)

I read this. This is the background for NCA.

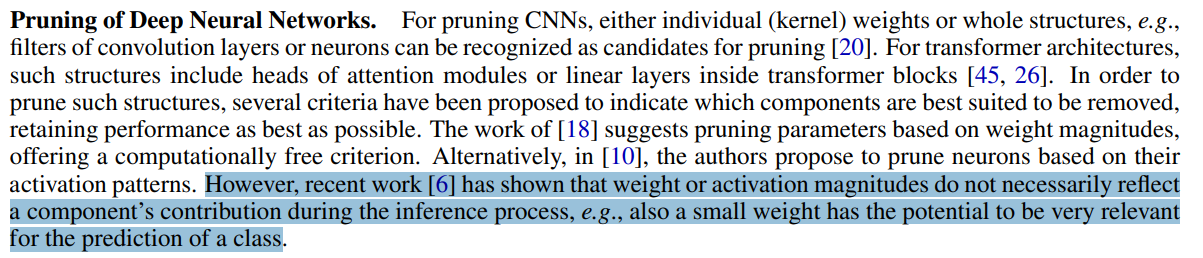
[On Convergence of Nearest Neighbor Classifiers over Feature Transformations (neurips.cc)](https://proceedings.neurips.cc/paper/2020/hash/93d9033636450402d67cd55e60b3f926-Abstract.html)

It is a lot of math that may be delayed for later for my own study.

Not too relevant, because the k-NN they are studying is the vanilla k-NN, used on top of feature extractors.

[Optimizing Attribution Methods to Prune CNNs and Transformers](https://arxiv.org/pdf/2408.12568)

This paper talks about using LRP (layer-wise relevance propagation) to determine the importance of neurons/weights, and use that to prune a network. It can be very relevant to our case-base maintenance idea. The following paragraph is very notable.



Note that, we actually don’t need to use LRP to know the importance of our cases in case nets (maybe we do?). We can just directly use the case weights and case bias in the case nets. However, we shouldn’t just prune a case because it has low case weights and low case bias. As the highlighted section above says. The case may still be very useful, its small bias means a small coverage of the case, its low class weight means a small contribution for a certain class. But maybe it does not need a big coverage, nor a big class contribution to make the right prediction.

This method can be useful later. I did not read the “Experiment” section

A novel breast cancer detection architecture based on a CNN-CBR system for

mammogram classification

2023

This is a very application oriented paper. The feature extraction and preprocessing is very complex. Maybe not as easily converted to how NN-kNN feature extractor can work. Maybe worth a try.

As expected, their feature extraction and the CBR part, or their CNN and CBR part are separated. CBR use the features from CNN, CBR then use mutual information to calculate a weight of the features.

Data is from here [CBIS-DDSM - The Cancer Imaging Archive (TCIA)](https://www.cancerimagingarchive.net/collection/cbis-ddsm/)

[CBIS-DDSM Dataset | Papers With Code](https://paperswithcode.com/dataset/cbis-ddsm)

The downside is that the CNN part is very complex. There are quite some preprocessing done. Maybe deviating too much to the application side of things, not as core/fundamental as we wanted.

[Understanding Gradient Clipping (and How It Can Fix Exploding Gradients Problem)](https://neptune.ai/blog/understanding-gradient-clipping-and-how-it-can-fix-exploding-gradients-problem)

Relevant to our problem of setting the learning rate

Some additional notes:

We are finally able to get good results on the Zebra data set, but somehow it sometimes lead to bad result? Where is the randomness coming from??

“after training

Parameter containing:

tensor([[1.5236, 0.1880]], device='cuda:0', requires\_grad=True)

[1.0, 0.6363636363636364, 0.9090909090909091, 1.0, 1.0, 0.6363636363636364, 0.9090909090909091, 0.9090909090909091, 1.0, 1.0]

Average accuracy:0.900”

I think there is some problem with the design. We currently have some parts shared, some parts not shared. But the learning rate applies to all of them. Problem is that, the learning rate (same number) will influence the global parts and local parts differently. The global parts are modified more often than the local parts. Not sure if this is a good thing.

optimizer = torch.optim.Adam([

{'params': shared\_params, 'lr': 1e-1}, # learning rate for shared parameters

{'params': case\_net\_params, 'lr': 1e-2} # learning rate for local parameters

], weight\_decay=1e-5)

This is doing really well for zebra(a)

Some notes for myself:

Each case should have a case weight, this case weight has to be on the case net, but cannot be on the case nets classifier.

The reasoning is this, technically, it makes sense to have the case net classifier (the manager) that keeps track of all case weights. But the problem is that, this cannot be a matrix, otherwise, the manager loses the ability to add/delete cases. So it will need to be a list, corresponding to the list of case nets, in other words, it is equivalent to have case nets manage their same weight. Instead of manage the case weights in a central location like the manager.

Because of this, every time a new case net is added to the manager, the manager is encouraged to “unfreeze” the existing case nets’ parameters and retrain everything. Of course, this is against the idea of lazy learning. We shouldn’t need to retrain, at least not retrain the whole thing.

So, the solution is that, instead of retraining all the case nets with the newly added case net, we can either just train the new case net to attune with the rest of the model. Or unfreeze just their case weights (but nothing else) and train case weights only.

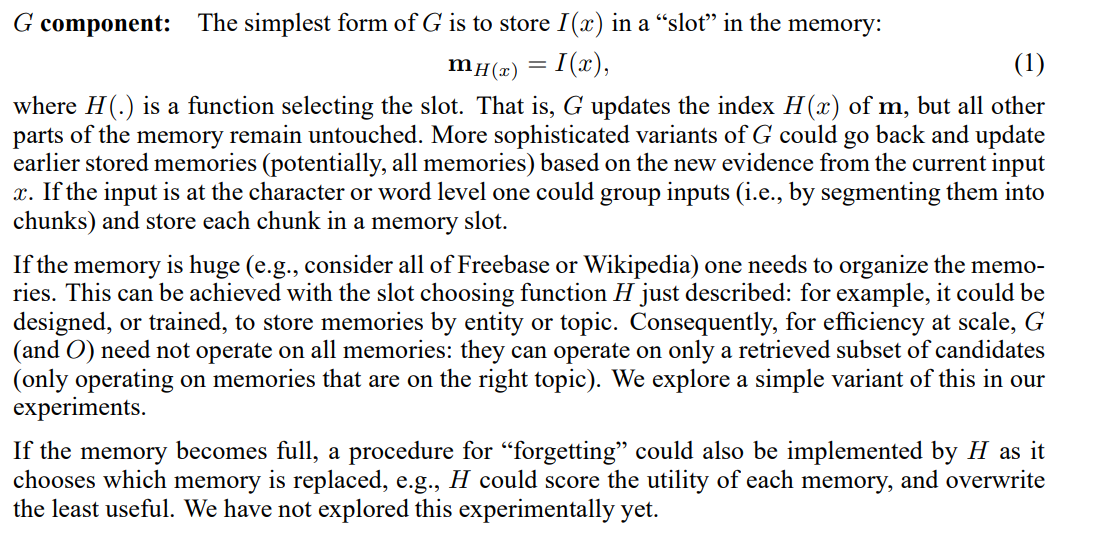
[Case retrieval nets: Basic ideas and extensions | SpringerLink](https://link.springer.com/chapter/10.1007/3-540-61708-6_63)

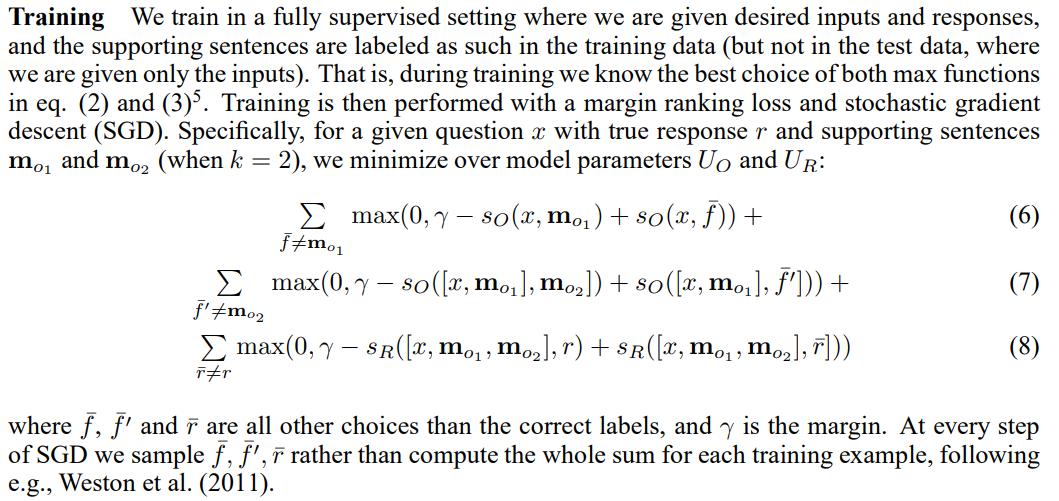
“we present Case Retrieval Nets (CRNs), a memory model that has recently been developed for this task. The main idea is to apply a spreading activation process to a net-like case memory in order to retrieve cases being similar to a posed query case”

Primitive model/results, not comparable, but similar motivation

[**MEMORY NETWORKS**](https://arxiv.org/pdf/1410.3916)

They have the following description which is very relevant to our case base maintenance in NN-kNN.

****

****

We can do the same by sampling during training!

Novel Object Discovery Using Case-Based Reasoning and Convolutional Neural Networks

They might have a way to calculate confidence score.

“We propose a case-based reasoning approach to detect the presence of novel object types and quickly learn from limited training data. Unlike CNNs, a CBR approach can learn using only a single training example and requires no training time. However, our approach does not propose to remove CNNs from the object classification process. Instead, our approach leverages the state-of-the-art performance of CNNs while providing capabilities that alleviate some of their limitations. ”

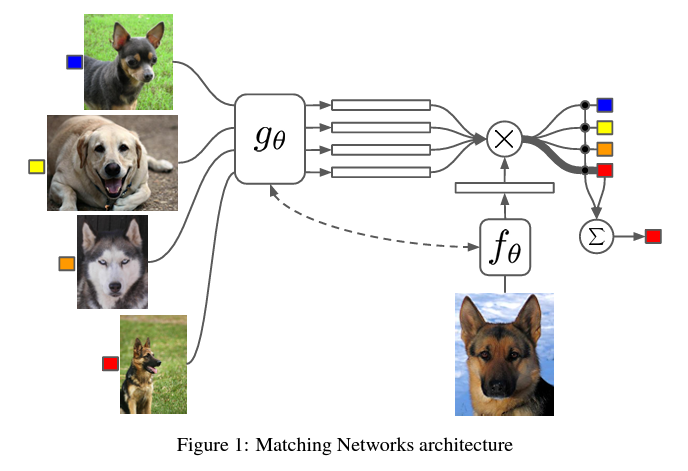
The paper proposed a measure for the performance in the task of novel class detection: class purity + class count divergence

“ Some systems may be able to perform outlier detection (e.g., when no similar cases are retrieved) but do not attempt to learn novel object types from these outliers”

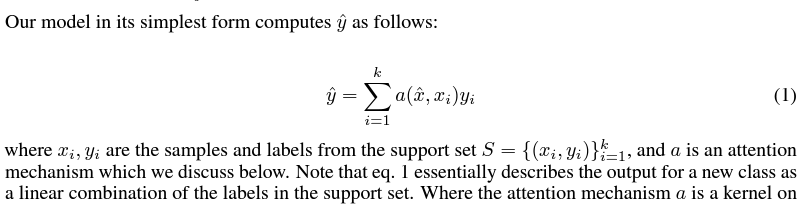
This may be a task where our combined model can do. Maybe a future direction to detect and classify unseen classes.

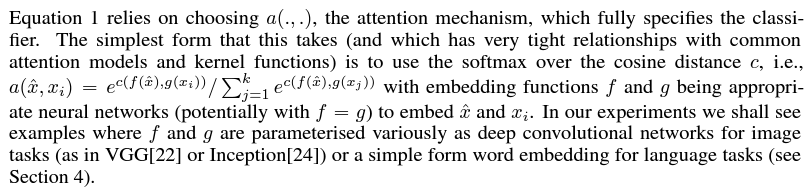
[**Matching Networks for One Shot Learning**](https://arxiv.org/pdf/1606.04080)

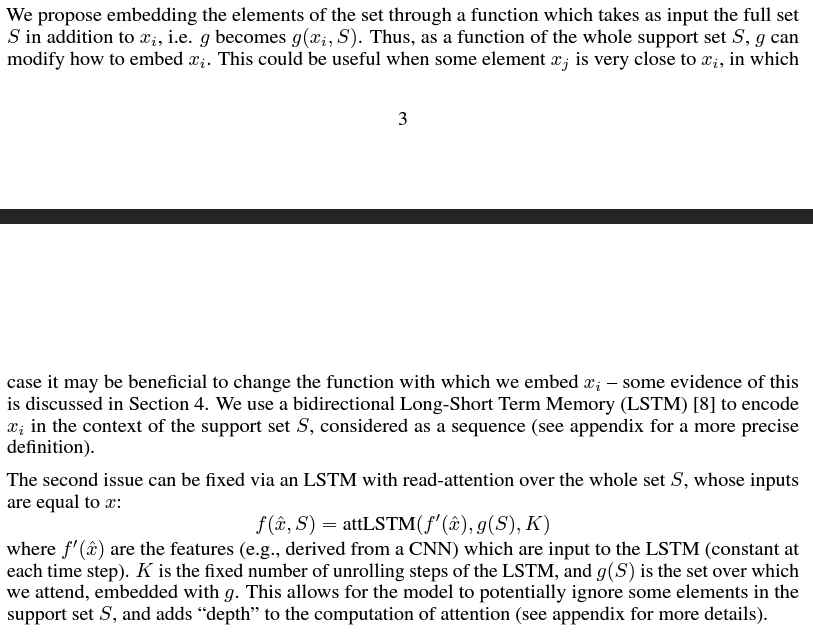
Neural networks with external memories for one-shot learning.

****

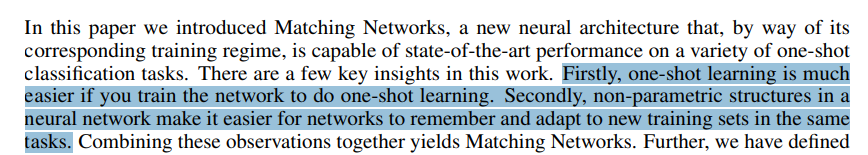
Given a support set, their method will come up with a classifier based on the support set.







Their conclusion is very relevant. Our NN-kNN can potentially be used for one-shot learning, and we are non-parametric in that sense, as we are k-NN



Their lesson is also relevant. They want to limit the support set, just as we want a small case base.

