Ruprecht-Karls-Universität Heidelberg

Fakultät für Mathematik und Informatik

Institut für Informatik

Masterarbeit

Bestärkendes Lernen für semantische Segmentierung von Bilddaten

Reinforcement Learning for semantic segmentation of image data

Name: Paul Hilt

Matrikelnummer: 3533310

Betreuer: Prof. Dr. rer. nat. Fred Hamprecht

Contents

List of Figures

0.1 Introduction

This is the template

Motivation

Preliminaries

2.1 Image segmentation

Image segmentation or image partitioning is defined as the process of dividing a digital image into sets of pixels also known as the objects in an image.
There are many variations of the segmentation method itself as well as of the goal that is to be achieved. With the rise of neural networks, nowadays these methods are usually fully or py parts learning based. That means, that some parameterized function is used to approximate a target distribution which can only be described by some sparse prior knowledge and/or by drawing samples from it. For the task of image segmentation these samples are of the form (x, y) which is a realization of a random variable (X, Y) with joint probability distribution $P_{X,Y}$. Samples respresent the raw input image that is to be segmented x and the desired segmentation also referred to as label or label image y. The goal is to learn a function $f(x)$ such that it approximates $\mathbb{E}[Y|X = x]$ at best. Since $P_{X,Y}$ is initially not known and only few samples and/or some sparse prior knowledge on its properties are available, the approximation can only be achieved by the empirical expectation from those samples and by dexterous use of the prior knowledge.
Some of the variations of learning based methods for image segmentation are distinguished by their level of supervision during the optimization. This is mainly defined by the amount of data samples and prior knowledge on the distribution $P_{X,Y}$ that is available.

- **supervised segmentation** is the highest level of supervision. Here only samples from $P_{X,Y}$ are available to the method. If enough samples are available such that all regions in the domain of the distributions are covered sufficiently, this is usually all one needs to arrive a a satisfactory result. However for most applications the set of available samples is only very limited.

- **unsupervised segmentation** is the lowest level of supervision. Here no samples from $P_{X,Y}$ are available to the method. The optimization method has to completely rely on prior knowledge on the underlying distribution. Realizations of X are usually still available and can be used for the approximation.

- **semi-supervised segmentation** is the transition between the previous two. Similar to supervised learning, semi-supervised learning uses samples from $P_{X,Y}$ but not only. There are also realizations of X available as well as some prior knowledge on $P_{X,Y}$ which is used during the optimization process as well.

- **self-supervised learning** is usually referred to when the method generates some kind of supervisory signal for itself. E.g. an automated labeling procedure to generate sample approximations $(x, \bar{y})$.

variations that focus more on the goal that should be achieved are

- **semantic segmentation** is the process of assigning class labels to each pixel in the image. Different objects of the same class are labeled equally. Usually only some objects of interest get a uique class label assigned to. All other objects receive the label background.
- **instance segmentation** is similar to semantic segmentation in the sense that each pixel in the image is assigned a label to. This label defines it either to background or to an instance of a object class. Therefore different objects of the same class are labelled differently. Here the label of an instance is also referred to as object id.
- **panoptic segmentation** is a fusion of the previous two. For all pixels belonging to instances of some defined set of classes, instance segmentation is performed. For the remaining pixels, semantic segmentation without is performed.

2.2 Reinforcement Learning (RL)

Please note that this is an aggressively shortened summary. For a deeper introduction please refer to [1]. The Reinforcement Learning problem originates from the idea of learning by interacting with an environment. The object that is learning is doing so by retrieving information from cause and effect. The causal model that is learned in such a way is updated with each change of the environment that can be related to some action. Therefore, the learned model fits the true model increasingly better with the number of induced causes and observed effects.
This type of learning problem can be modeled by ”Finite Markov Decision Processes”. Such processes usually need the following elements:

- **Environment** The environment is a dynamic model of some complex process.
- **State** The state s_t is generated by the environment. It changes over time according to the dynamics within the environment.
- **Action** An action a_t is a cause that might change the state of the environment. Actions are produced by the agent.
- **Reward** The reward r_t is a scalar value that is produced by the environment and received by the agent.
- **Agent** The agent is a instance which generates actions and observes the caused change of the state of the environment.
- **Policy** A policy $\pi(a_t|s_t)$ is a probability distribution over the set of possible actions at a time step. A agent is essentially defined by its policy as each action that is taken is sampled from that policy.

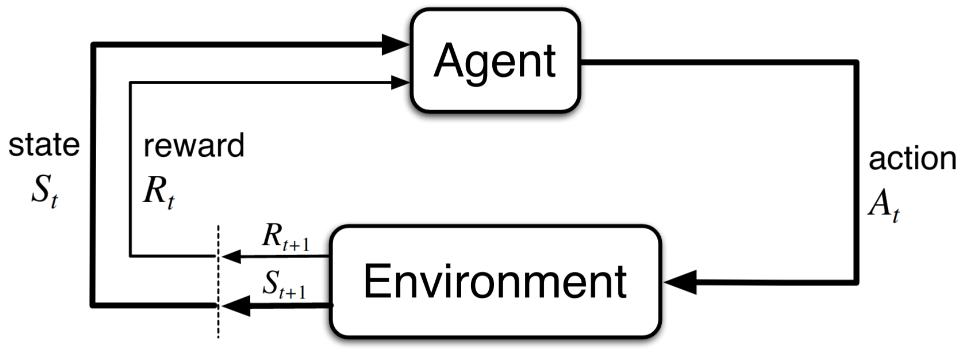

Figure 2.1: agent environment interaction [1]

The model with its signal flows is depicted in figure 2.1
All signals in this model are time dependent. Such a model satisfies the Markov Property if next s' and reward r only depend on the current action-state tuple (s, a). If assuming finite state and action spaces together with the Markov Property gives a Finite Markov Decision Process. The environment dynamics can therefore be represented by the bivariate probability distribution

$$p(s', r|s, a) = Pr\{r_{t+1} = r, s_{t+1} = s'|s_t = s, a_t = a\} \tag{2.1}$$

Further, if a_t is sampled from $\pi(a_t|s_t)$ the Markov Property induces conditional independence of (s_{t-1}, r_{t-1}) and (s_{t+1}, r_{t+1}) given s_t.

The agents task is to predict a policy that maximizes the expected future rewards. This objective is given by

$$\arg\max_{\pi} \mathop{\mathbb{E}}_{p_\pi} \left[\sum_{t=0}^{T} r_t | s_0 \right] \tag{2.2}$$

Here T marks the time limit of the process and p_π represents the environment dynamics following a action history sampled from π.

2.2.1 Value functions

Most methods of solving eq. (1.2) use so estimations of so called value functions. This are functions that provide a quality measure for an agent evaluating a state-action tuple.
Commonly three value functions are used. The state-value function, the action-value function and the advantage-value function. They all depend on the expected discounted future rewards

$$g_t = \sum_{k=0}^{T-t-1} \gamma^k r_{t+k+1}. \tag{2.3}$$

Here γ is referred to as the discount factor. Its value usually determines how prospective future rewards are weighted in the value function. E.g if $\gamma < 1$ rewards that are closer to t get a higher weight than those that are occuring at a later time. For $\gamma > 1$ the contrary holds.

The state-value function is defined by

$$V_\pi(s) = \mathop{\mathbb{E}}_{p_\pi}\left[g_t | s_t = s\right], \tag{2.4}$$

the action-value by

$$Q_\pi(s,a) = \mathop{\mathbb{E}}_{p_\pi}\left[g_t | s_t = s, a_t = a\right] \tag{2.5}$$

and finally the advantage-value by

$$A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s) \tag{2.6}$$

it follows

$$V_\pi(s) = \mathop{\mathbb{E}}_{a\sim\pi}\left[Q_\pi(s,a)\right] \tag{2.7}$$

Usually the objective is to maximize either one or both of the value functions. Note that, $\max_\pi V_\pi(s)$ and $\max_\pi Q_\pi(s,a)$ satisfy Bellman's principle of optimality. Hence they can be solved exactly by Dynamic Programming. This is referred to as the tabular solution. However for most problems this is not feasible and the value functions are approximated by neural networks.

2.2.2 Q-learning

Q-learning is a method to find the action-value maximizing policy by using temporal differences. This often the backbone of policy gradient algorithms. Let

$$\pi(\cdot|s) = \operatorname*{softmax}_a Q_\pi(s,a) \tag{2.8}$$

then the objective is to approximate Q_π which is achieved by the temporal difference loss

$$\mathcal{L}_{TD} = \frac{1}{2}\left(r_{t+1} + \gamma \max_a Q_\pi(s_{t+1},a) - Q_\pi(s_t,a_t)\right)^2 \tag{2.9}$$

this kind of approximation is usually referred to as one step TD method. The optimality follows directly and the convergence of the approximation under certain conditions has been proven in [5].
In contrast to one step TD methods there are Monte Carlo methods which collect the loss over whole episodes where one episode is defined by the history of temporal differences between a starting state and an end state. This methods are often solved by eligibility traces [6].
Optimizing eq (1.8) is referred to as on-policy policy optimization where the target policy π is the policy which is used when actions are sampled. This however is problematic as the target policy is defined by eq (1.7) depending on q_π which is not trustworthy as this is the function that is to be approximated. In order to have more control over the sampling of actions which is usually referred to as exploration a data collection policy $\mu(a|s)$ is used. Therefore in an off-policy setting eq (1.8) becomes

$$\mathcal{L}_{TD} = \frac{1}{2}\left(r_{t+1} + \gamma \max_a Q_\mu(s_{t+1},a) - Q_\mu(s_t,a_t)\right)^2. \tag{2.10}$$

and during inference

$$\bar{\pi}(\cdot|s) = \underset{a}{\text{softmax}}\, Q_\mu(s,a) \tag{2.11}$$

is used. There are many solutions to overcome the distribution mismatch between π and μ. Many use importance sampling or variance reduction techniques.[7]

2.2.3 Policy gradient methods

This class of methods optimizes the parameters θ that define the statistics of a policy $\pi_\theta(a|s)$. Let

$$\rho(\pi) = \sum_{t=1}^{\infty} \underset{\substack{s\sim d_\pi(s)\\ a\sim\pi(a|s)}}{\mathbb{E}} [r_t|s_0] \tag{2.12}$$

be the expected, discounted future reward per step and let

$$d_\pi(s) = \sum_{t=0}^{\infty} \gamma^t Pr\left\{s_t = s|s_0,\pi\right\} \tag{2.13}$$

be the discounted stationary distribution of states under π. Then

$$\frac{\partial\rho_\pi}{\partial\theta} = \sum_s d_\pi(s) \sum_a \frac{\partial\pi(a|s)}{\partial\theta} \bar{Q}_\pi(s,a) \tag{2.14}$$

Is the policy gradient with which gradient ascent on the policy can be performed in order to maximize ρ. A proof and a thorough discussion can be found in [8]. Note that the policy gradient is on-policy and that $\bar{Q}_\pi$ is an approximation of Q_π.
Since π is a probability distribution it follows $\sum_a \frac{\partial\pi(a|s)}{\partial\theta} = 0, \forall s \in S$. Therefore

$$\frac{\partial\rho_\pi}{\partial\theta} = \sum_s d_\pi(s) \sum_a \frac{\partial\pi(a|s)}{\partial\theta} \left[\bar{Q}_\pi(s,a) + b(s)\right], \qquad b: S \to \mathbb{R} \tag{2.15}$$

the function b is called a baseline and is often used to reduce variance and bias of the gradient ascent update step. Using $\frac{\nabla_\theta\pi(a|s)}{\pi(a|s)} = \nabla_\theta ln(\pi(a|s))$ and $\mathbb{E}_{x\sim p(x)}[f(x)] = \sum_x p(x)f(x)$, rewriting eq (1.13) yields

$$\frac{\partial\rho_\pi}{\partial\theta} = \underset{\substack{s\sim d_\pi(s)\\ a\sim\pi(a|s)}}{\mathbb{E}} \left[\nabla_\theta ln(\pi(a|s)) \left[\bar{Q}_\pi(s,a) + b(s)\right]\right] \tag{2.16}$$

In practice where there are large state and action spaces the expectations w.r.t s and a become infeasible to obtain. Using $\mathbb{E}_{x\sim p(x)}[f(x)] = \frac{1}{n}\sum_n f(x), n \to \infty, x \sim p(x)$, ample based learning uses enough samples of s and a in order to obtain a good enough approximation of the expectations. Therefore eq. (1.15) becomes

$$\frac{\partial\rho_\pi}{\partial\theta} = \frac{1}{n}\sum_n \nabla_\theta ln(\pi(a|s)) \left(\bar{Q}_\pi(s,a) + b(s)\right), \qquad s \sim d_\pi(s), \quad a \sim \pi(a|s), \quad n \to \infty \tag{2.17}$$

This leads to Actor Critic methods (A2C) where there are two instances that are updated in a turn based fashion. The critic is the value-function approximation and the actor is approximating the policy. Intuitively the critic evaluates the action taken by the actor who uses this evaluation to scale its gradient update (this is the role of $\bar{q}_\pi$ in eq (1.13)).

2.2.4 Maximum Entropy Reinforcement Learning

In off-policy settings it is common to use a data collection policy μ which has a large entropy in order to encourage exploration of the action and state spaces. The principle of maximum causal entropy has been introduced by [9] and has been elaborated on among others by [10]. The key idea is to incorporate an entropy term into the objective function acting like a regularizer.

$$\rho^{\mathcal{H}}(\pi) = \sum_{t=1}^{\infty} \mathop{\mathbb{E}}_{\substack{s_t \sim d_\pi(s_t) \\ a_t \sim \pi(a_t|s_t)}} [r_t + \alpha(t)\mathcal{H}(\pi(\cdot|s_t))|s_0] \tag{2.18}$$

Here α is a non-negative regularization weight which is usually monotonically decreasing with increasing t and $\mathcal{H}$ is some entropy measure. If α becomes 0 eq. (1.17) becomes equal to eq. (1.11), therefore, intuitively α should be high in the early phase of training the policy and the value function and converge to 0 as the policy gets closer to the perfect policy and therefore can afford to have more certainty in its prediction. This objective is on-policy and still gives control over the exploration behavior. Usually it does not bother if the policy has high entropy, since during inference the action where "π" has maximum probability is selected. This makes especially unimodal distributions attractive. They fixate on single actions and they imply few parameters only that need to be learned (e.g. mean and variance of normal distributions). However often more expressive multimodal distributions fit the true distribution which is approximated better. In [11] this idea has been extended by using normalizing flows. Normalizing flows [12] are based on the idea of transforming a probability density function by letting each sample undergo a transformation. If this transformation is a diffeomorphism the probability of the transformed sample can be determined. Let T be a diffeomorphism of a real vector u sampled from $p_u(u)$.

$$x = T(u) \quad \text{where} \quad u \sim p_u(u) \tag{2.19}$$

then

$$p_x(x) = p_u(T^{-1}(u))|detJ_T(u)|^{-1} \tag{2.20}$$

where $J_T(u)$ is the Jacobian matrix of T w.r.t. u. I practice an invertible neural network can be trained to transform a simplistic density function into a more expressive one.
E.g. let the agent predict mean and variance of a Normal distribution. Actions are the in the data collection process sampled from a transformed Normal distribution where the transform encourages entropy and maybe also multiple modes. Assuming the sampling happened using reparametrization the log probabilities and their gradient in eq (1.14) can still be calculated. This is a on-policy training with a expressive density function and the advantage is that easy reparameterization tricks can still be used since the sampling itself is happening from the base distribution.

2.2.5 Soft Actor-Critic (SAC)

This algorithm was introduced by [13]. They aim to maximize the objective in eq. (1.17). Particularly they focus on the selection of the weight factor α and show, that it can be seen as a learnable parameter which is trained jointly with actor and critic networks.
SAC is derived from the soft policy iteration where the temporal difference equation for the action value depends on the soft value function which is

$$V_\pi(s_t) = \mathop{\mathbb{E}}_{a \sim \pi(a|s_t)} [Q_\pi(s_t, a) - \alpha log(\pi(a|s_t))] \tag{2.21}$$

the negative log probabilities are the entropy measure in eq. (1.18). The action value function loss yields

$$\mathcal{L}_{critic} = \frac{1}{2}(Q_\pi(s_t, a_t) - (\gamma \mathop{\mathbb{E}}_{s_{t+1} \sim d_\pi(s)} [V_\pi(s_{t+1})] + r_t))^2 \tag{2.22}$$

For the policy improvement step the policy is updated such that it approximates softmax$_a(\frac{1}{\alpha}Q_\pi(s, a))$ where Q_π is the soft action value function, learned by minimizing eq (1.21). The loss for the policy then yields

$$\mathcal{L}_{actor} = DKL_a \left[\pi(a|s_t) \middle\| \frac{exp(\frac{1}{\alpha}Q_\pi(s_t, a))}{Z(s_t)} \right] \tag{2.23}$$

here, DKL_a is the Kullback Leibler Divergence over the actions. $Z(s_t)$ is the partition function of the distribution.

$$Z(s_t) = \sum_a Q_\pi(s_t, a) \tag{2.24}$$

It is usually too expensive to evaluate $Z(s_t)$ since it involves integrating/summing over the action value space which means many forward passes through the neural network that represents Q_π. Since $\mathcal{L}_{actor}$ is minimized by gradient descent methods only the gradient is needed. If one expands the KL-divergence term, $Z(s_t)$ becomes additive and therefore vanishes once the gradient is obtained.
Note that α gives control over the differences between action values and therefore over the entropy of the resulting distributution. *lemma 2* in [13] claims the improvement of Q_π with each optimization step of $\mathcal{L}_{actor}$.
The gradient of eq. (1.23) w.r.t. the parameters θ of π yields

$$\nabla_\theta \mathcal{L}_{actor} = \nabla_\theta \mathop{\mathbb{E}}_{\substack{s_t \sim d_\pi(s_t) \\ a_t \sim \pi(a_t|s_t)}} [\alpha log(\pi(a_t|s_t)) - Q_\pi(s_t, a_t)] \tag{2.25}$$

approximating eq. (1.24) by a sample based method yields

$$\nabla_\theta \bar{\mathcal{L}}_{actor} = \nabla_\theta [\alpha log(\pi(a_t|s_t)) - Q_\pi(s_t, a_t)] \tag{2.26}$$

minimizing this loss by a gradient descent method involves backpropping through a sampling procedure. This can be made differentiable by the reparameterization trick [14].
[13] also provides a method to determine the entropy adjustment α such that it takes the minimal value needed to maximize the maximum entropy objective eq. (1.17) assuming a fixed policy π.
In practice, reinforcement learning problems have high dimensional action spaces but only one dimensional rewards. Therefore the learned action value function of the critic is also one dimensional in contrast to the actor who predicts the statistics of the policiy for each action dimension. Then the joint probability is the product of probabilities over all actions. This results in summing the log probabilities in eq. (1.25)

2.2.6 Common optimization methods

There are numerous optimization methods for reinforcement learning problems. This is just a listing of only very few but important ones, reviewed and tested in [15].

- **Double Q-learning** [16] Conventional Q-learning is affected by an overestimation bias of action values. Decoupling the action selection from its evaluation by learning two action value networks independently resulting in the loss

$$\mathcal{L}_{TD} = \frac{1}{2}\left(r_{t+1} + \gamma Q_\pi^{(\bar{\phi})}\left(s_{t+1}, \operatorname*{arg\,max}_a Q_\pi^{(\phi)}(s_{t+1}, a)\right) - Q_\pi^{(\phi)}(s_t, a_t)\right)^2. \tag{2.27}$$

  Here ϕ and $\bar{\phi}$ are the parameters of the independently trained action value functions respectively. A similar method trains two action value functions independently and and takes for all evaluations the min value of the two network predictions. Both methods show a reduction in overestimation as shown in [16].

- **Prioritized replay** During data collection, the tuples $(s_t, a_t, r_{t+1}, s_{t+1})$ are stored in a replay buffer and during training phases then sampled uniformly from the buffer. In [17] the sampling is not uniform but rather with a probability p_t relative to the last encountered loss of that replay tuple.

$$p_t \propto \mathcal{L}_{TD}^{\omega}. \tag{2.28}$$

  Raising the loss to the power of the parameter ω determines the shape of the distribution. New transitions that did not produce a loss yet are always sampled with maximum priority.

- **Multi-step learning** Q-learning bootstraps from single step temporal difference losses. [5] introduced multi-step temporal differences which is the transition from Monte Carlo methods to single step temporal difference methods. The n-step return is defined as

$$r_t^{(n)} \equiv \sum_{k=0}^{n-1} \gamma_t^k r_{t+k+1} \tag{2.29}$$

  then the multi step TD loss yields,

$$\mathcal{L}_{TD} = \frac{1}{2}\left(r_{t+1}^{(n)} + \gamma^n \max_a Q_\pi(s_{t+n}, a) - Q_\pi(s_t, a_t)\right)^2 \tag{2.30}$$

optimizing multistep TD losses with n sampled uniformly from the interval $[1..T]$ results in faster learning as shown in [5].

- **Dueling networks** This is an optimization method based on the neural network atrchitecture of value functions in value based RL and was introduced by [18]. It features one state-value and one advantage-value stream of computation that both share a common state feature extractor network $f(s)$. This leads to this factorization of action-values

$$Q_\pi^{(\phi)}(s,a) = V_\pi^{(\eta)}(f^{(\xi)}(s)) + A_\pi^{(\psi)}(f^{(\xi)}(s), a) - \frac{\sum_{a'} A_\pi^{(\psi)}(f^{(\xi)}(s), a')}{N_{actions}} \tag{2.31}$$

Here η, ψ and ξ are the parameters of the state-value function, the advantage-value function and the state feature extractor respectively. ϕ is the concatenation of η, ψ and ξ.
The last term in eq. (1.31) approximates $\mathbb{E}_{a' \sim \pi} A_\pi^{(\psi)}(f^{(\xi)}(s), a') = 0$, this equality follows from combining eq (1.6) and eq(1.7). Then eq. (1.31) follows from eq (1.4), eq (1.5) and eq (1.6).
This method outperforms vanilla, value based RL methods on common RL benchmarks.

2.3 Geometric deep learning

Convolutional neural networks use the convolution operation to ”filter” a regular grid graph of certain dimension with a filter consisting of learnable parameters. Graph convolution, was introduced by [19] generalizes this notion of convolutional filtering to arbitrary graphs. Through that it is possible to learn functions on non eucledian, structured domains that have a notion of locality.
Since then the field developed rapidly, [20] provides a good overview of the research that was done so far.
Most of the research focuses on the application where graphs are constructed from descretizations of 2-dimensional manifolds, embedded in 3-dimensional eucledian space, usually called point clouds. However the principle can be used for arbitrary graphs that contain features in their nodes.
Typically there are two equivalent definitions of convolution on graphs. One is the spectral definition which suffers from large complexity in terms of memory and time. The other one is a spatial construction motivated from signal flows on graphs which is much faster as operating with sparse representations of the graph. The A quick summary of the latter as in [21] is given below.

Let a graph be represented by $\mathcal{G} = (X, (I, E))$ where $X \in \mathbb{R}^{Nxm}$ is a node feature matrix of N, m-dimensional node feature vectors, with nodes ecoded as $i \in [1..N]$ and (A, E) is a set of adjacency tuples where $A \in \mathbb{N}^{2x|E|}$ encodes the set of $|E|$ edges with edge features E.
The generalization of the convolutional operator locally expressed by means of the neighborhood $\mathcal{N}(i)$ around node i is

$$\vec{x}_i' = \gamma\left(\vec{x}_i, \boxplus_{j \in \mathcal{N}(i)} \phi\left(\vec{x}_i, \vec{x}_i, \vec{e}_{ij}\right)\right) \tag{2.32}$$

where $\boxplus$ is a differentiable and permutation invariant function such as the sum or the mean. γ and ϕ are differentiable functions represented by multi layer perceptrons. This convolution is also referred to as message passing scheme. All this covolution operations w.r.t. to each node in the graph allow parallel computation which makes this schemes fast.

2.4 Mutex watershed

The Mutex Watershed [2] is a image partitioning algorithm based on watersheding that is able to operate without a prior seeding.
Like most image partitioning algorithms it is defined on a graph $\mathcal{G} = (\mathcal{V}, E^+ \cup E^-, W^+ \cup W^-)$ with a set of vertices $\mathcal{V}$, a set of edges as the disjoint union of attractive edges E^+ and repulsive edges E^- and a set of corresponding edge weights $W^+ \cup W^-$. Each vertex in this graph represents uniquely a pixel of the corresponding image. The data term is based on the affinity between the incidental vertices of an edge. The affinity between two nodes i and j is the probability p_{ij} of the nodes belonging to the same partition in the posterior partitioning. These affinities can be predicted by e.g. a CNN.
The two variants of edges, attractive edges $e^+_{ij} \in \mathcal{E}$ with edge weights $w^+_{ij} \in \mathcal{W}$ where $w^+_{ij} = p_{ij}$ and repulsive edges $e^-_{ij} \in \mathcal{E}$ with edge weights $w^-_{ij} \in \mathcal{W}$ where $w^-_{ij} = 1 - p_{ij}$.
A partitioning on that graph is defined by the disjoint union of a set of attracive and a set of repulsive edges by the active set $A = A^+ \bigcup A^-$ that encode hard merges and mutual exclusions of vertices and partitions. To represent a valid partitioning, the set A has to satisfy cycle constraints. Defining the set $\mathcal{C}_i(A)$ with $A \subseteq E$ as the set of all cycles with exactly i active repulsive edges.

$$\mathcal{C}_i(A) := \big\{c \in cycles(\mathcal{G}) | c \subseteq A \text{ and } |c \cap E^-| = i\big\} \tag{2.33}$$

a valid partitioning can only be inferred from an active set A if $\mathcal{C}_1(A) = \emptyset$. If additionally $\mathcal{C}_0(A) = \emptyset$, the algorithm can be defined as the search for the minimal spanning tree in each partition.

```
Data: weighted graph $\mathcal{G} = (\mathcal{V}, E^+ \cup E^-, W^+ \cup W^-)$
Result: clusters defined by spanning forest $A^\star \cap E^+$
Initialization: $A = \emptyset$;
for $(i, j) = e \in (E^+ \cup E^-)$ in descending order of $W^+ \cup W^-$ do
    if $\mathcal{C}_0(A \cup \{e\}) = \emptyset$ and $\mathcal{C}_1(A \cup \{e\}) = \emptyset$ then
        $A \leftarrow A \cup e$ ;
    end
end
$A^\star \leftarrow A$ ;
return $A^\star$
```

Algorithm 1: Mutex Watershed [2]

An example walk through of the algorithm is depicted in figure 2.2.

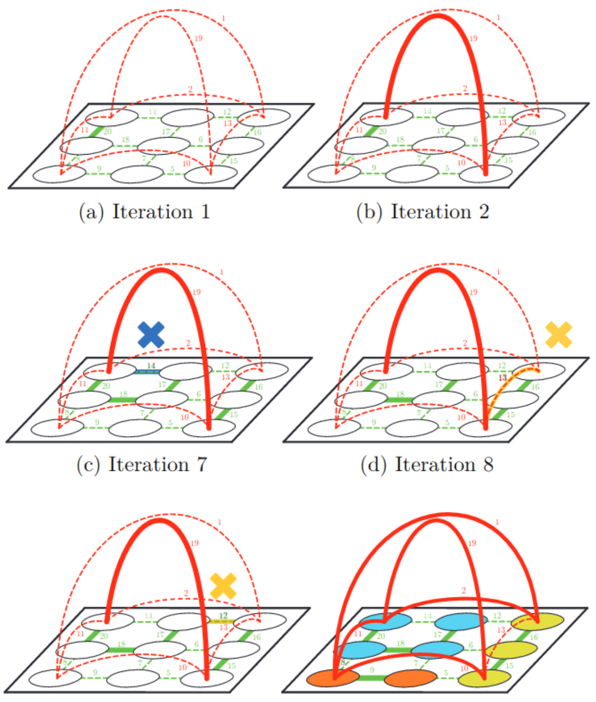

Figure 2.2: Some iterations of algorithm 1 applied to a graph with weighted attractive edges (green) and repulsive (ref) edges. Edges that are part of the active set A at each iteration are shown in bold. On termination (f), the connected components in $A \cap E^+$ represent the partitions of the final partitioning. Edges that are not added to A because of the violation of $\mathcal{C}_0$ or $\mathcal{C}_1$ are highlighted in blue and yellow respectively [2]

Algorithm 1 minimizes a energy functional that is defined by the active set A. This requires the following

Definition 1 ***Dominant power*** *[2]:*
Let $\mathcal{G} = (V, E, W)$ be an edge weighted graph, with unique edge weights $w_e \in \mathbb{R}_0^+$, $\forall e \in E$. Then $p \in \mathcal{R}^+$ is called a dominant power if:

$$w_e^p > \sum_{\substack{t \in E \\ w_t < w_e}} w_t^p \qquad , \forall e \in E \tag{2.34}$$

This allows the definition of the objective that is solved by algorithm 1

Definition 2 ***Mutex Watershed Objective*** *[2]:*
Let $\mathcal{G} = (V, E, W)$ be an edge weighted graph, with unique edge weights $w_e \in \mathbb{R}_0^+$, $\forall e \in E$ and $p \in \mathcal{R}^+$ a dominant power. Then the Mutex Watershed Objective is devined as the integer linear program:

$$\min_{a \in 0,1^{|E|}} -\sum_{e \in E} a_e w_e^p \tag{2.35}$$

$$\text{s.t.} \quad \mathcal{C}_0(A) = \mathcal{C}_\infty(A) = \emptyset, \tag{2.36}$$

$$\text{with} \quad A := \{e \in E | a_e = 1\} \tag{2.37}$$

2.5 Image partitioning by multicuts

The multicut problem is a graph cut problem that is in [22] redefined for the image partitioning task. The unsupervised pairwise image partitioning on a grid graph $G = (V, E)$ where each node $v \in V$ corresponds to a pixel in the image can be defined as the following minimization problem

$$\min_{x \in L^{|V|}} \sum_{uv \in E} \beta_{uv} I(x_u \neq x_v), \qquad L = \{1, ..., |V|\} \tag{2.38}$$

where I is a indicator function that maps a boolean expression to 1 if it is true and to 0 otherwise. L is the set of all possible labels, β_{uv} is the edge cost that is active if u has a different label than v and $x = (x_v)_{v \in V} \in L^{|V|}$ is a node labeling that defines a partitioning of V into subsets of nodes S_l assigned to class l such that $\bigcup_{l \in L} S_l = V$. Eq. (1.38) defines the unsupervised partitioning problem where there is no prior on the classes and the maximum number of classes in the final labeling is the number of nodes $|V|$. Therefore the coefficients β can depend on the data but are assumed not to depend on prototypical prior information about a fixed number of classes L.
A *multicut* on a graph $G = (V, E)$ with a partitioning $\bigcup_{l \in L} S_l = V$ is defined as

$$\delta(S_1, ..., S_k) := \{uv \in E | \exists i \neq j : u \in S_i \text{ and } v \in S_j\} \tag{2.39}$$

Where the sets $S_1, ..., S_k$ are called the *shores* of the multicut. To obtain a polyhedral representation of the set of multicuts on a graph on needs to define incidence vectors $\mathcal{X}(F) \in \mathbb{R}^{|E|}$ for each subset

$F \subseteq E$:

$$\mathcal{X}_e(F) = \begin{cases} 1, & \text{if } e \in F \\ 0, & \text{if } e \in E \backslash F \end{cases} \tag{2.40}$$

then the multicut polytope is given by

$$MC(G) := conv\,\{\mathcal{X}(\delta(S_1, ..., S_k)) | \delta(S_1, ..., S_k) \text{ is a multicut of } G\} \tag{2.41}$$

and the unsupervised image partitioning problem eq. (1.38) can be written as the equivalent multicut problem

$$\min_{y \in MC(G)} \sum_{uv \in E} \beta_{uv} y_{uv} \tag{2.42}$$

defining cycle constraints allows to rewrite eq. (1.42) as the integer linear program (ILP)

$$\min_{y \in [0,1]^{|E|}} \sum_{uv \in E} \beta_{uv} y_{uv} \tag{2.43}$$

$$\text{s.t.} \quad \sum_{uv \in C} y_{uv} \neq 1, \qquad \forall \text{ cycles } C \subseteq E \tag{2.44}$$

The cycle constraints in eq. (1.47) enforce that y lies inside the multicut polytope by guaranteeing that there are no active edges inside a shore. There are many solution methods for this problem. The one used in [22] is based on iteratively solving the ILP in eq. (1.43) then find violated constraints in the sense of eq. (1.44), add them to the ILP and reiterate until there are no more violated cycle constraints. Violated constraints can be found by projecting a obtained solution y to the multicut polytope and checking for differences in the solution and the projection y'. The projection is achieved by assigning a label to each connected component in $G = (V, \{uv | y_{uv} = 0\})$ which produces a valid partition of which the multicut and therefore y' can be obtained easily. If there exists an active edge uv inside the solution that is not active within the projection then this is an edge inside a shore and one of the inducing cycle constraints is obtained by computing the shortest path between u and v inside the shore and adding the active edge uv to that path yielding a cycle.

2.6 Principal component analysis

The principal components of a collection of data points can be thought of as the directions in which the variance of the data points is the highest. The magnitude of the variance in a specific direction is referred to as the scores of the respective principal component. All principal component vectors form a orthonormal basis.

Consider a data matrix $X \in \mathbb{R}^{nxp}$ of n, p-dimensional samples from a arbitrary distribution. The first principal components is the arg-maximizer

$$w_{(1)} = \underset{\|w\|=1}{\text{arg max}} \|Xw\|^2 \tag{2.45}$$

Since this is a convex optimization problem, the solution can be found by finding the stationary points of the Lagrange function

$$\mathcal{L}(w, \lambda) = w^T C w - \lambda(w^T w - 1) \tag{2.46}$$
$$\text{with} \quad C = X^T X \tag{2.47}$$

Note, that C is hermetian. The partial derivatives yield

$$\nabla_w \mathcal{L}(w, \lambda) = 2Cw - 2\lambda w \tag{2.48}$$
$$\nabla_\lambda \mathcal{L}(w, \lambda) = -(w^T w - 1) \tag{2.49}$$

setting eq. (1.41) to 0 yields

$$0 = Cw - \lambda w \tag{2.50}$$
$$Cw = \lambda w \tag{2.51}$$

it strikes that eq. (1.43) is a eigenvalue problem. Since C is hermetian its eigenvectors form a orthonormal basis, therefore setting eq. (1.42) to 0 holds if w is a eigenvector. To see that the solution to eq. (1.38) is in fact the largest eigenvalue, one has to substitute eq. (1.44) into eq. (1.40).

$$w^T C w - \lambda(w^T w - 1) = w^T C w = \lambda w^T w = \lambda \tag{2.52}$$

Since eq. (1.40) is a maximization problem, λ has to be the largest eigenvalue. Therefore the first principal component is the eigenvector $w_{(1)}$ with the largest eigenvalue $\lambda_{(1)}$ of C. $\lambda_{(1)}$ is also referred to as the score of the principal component $w_{(1)}$.

The remaining principal components are given by the remaining eigenvectors sorted by their eigenvalues.

2.7 Loss functions

This section reviews some important loss functions.

2.7.1 Contrasive loss

This loss, as defined in [3], applies to the task of instance segmentation in images. A differentiable function predicts points in a feature space that are embedded in a n-dimensional eucledian space. Each predicted point in the embedding space corresponds to a pixel within the image. Ideally the n-dimensional embedding vectors for each pixel are close to each other in the embedding space if the

corresponding pixels belong to the same instance and distant to each other if not. Then a final clustering algorithm can assign an instance/cluster to each embedding vector.
A loss that penalizes wrong predictions can be thought of as a force that pulls pixel embeddings of the same instance together and pushes those of different instances apart. This loss as defined in eq.(4) in [3] consists of three additive parts. Assuming C is the number of instances in an image (each label id in the ground truth image represents an instance). N_C is the number of elements in cluster $c \in [1..C]$, x_i is a embedding vector where i denotes a pixel. μ_c the mean embedding of cluster c, $||\cdot||$ is the $L1$ or $L2$ distance and $[x]_+ = \max(0, x)$. δ_v and δ_d margins that are used to hinge the pull and push forces. That is, the forces are only exerted if the distance that is under consideriation is larger than δ_v for the intra cluster pulling forces, and smaller than $2\delta_d$ for the inter cluster pushing forces. This allows to learn a more expressive embedding space since the points are allowed to move freely if there are no exerted forces on them.

- **variance term** this exerts a intra-cluster pull force that draws pixel embeddings towards the cluster center of their respective instance if the distance to the center is larger than δ_v.

$$\mathcal{L}_{var} = \frac{1}{C}\sum_{c=1}^{C}\frac{1}{N_c}\sum_{i=1}^{N_c}[||\mu_c - x_i|| - \delta_v]_+^2 \tag{2.53}$$

- **distance term** this exerts a inter cluster push force that pushes clusters away from each other by penalozing distances between cluster centers that are smaller than $2\delta_d$.

$$\mathcal{L}_{dist} = \frac{1}{C(C-1)}\sum_{c_A=1}^{C}\sum_{\substack{c_B=1\\c_B\neq c_A}}^{C}[2\delta_d - ||\mu_{c_A} - \mu_{c_B}||]_+^2 \tag{2.54}$$

- **regularization term** this is a small pull-force that keeps the predicted embedding vectors bounded by pulling them towards the origin.

$$\mathcal{L}_{reg} = \frac{1}{C}\sum_{c=1}^{C}||\mu_c|| \tag{2.55}$$

Then the final loss yields

$$\mathcal{L} = \alpha\mathcal{L}_{var} + \beta\mathcal{L}_{dist} + \gamma\mathcal{L}_{reg} \tag{2.56}$$

where α, β and γ are weights for the respective terms.

The behaviors of the different terms in the loss are sketched in figure 2.3

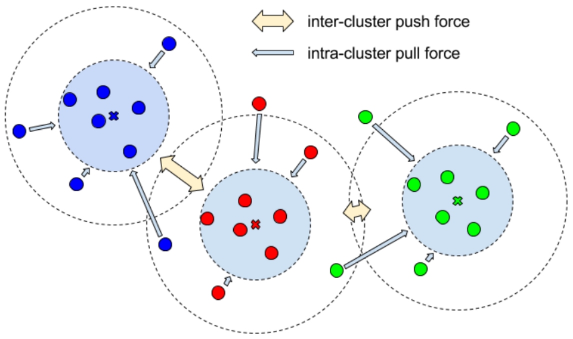

Figure 2.3: As defined by the loss, this are the hinged inter-pulling and intra-pushing forces acting on points in the embedding space [3]

2.7.2 Triplet loss

This loss, as defined in [4] is related to the contrastive loss in section 2.7.1 in the sense that it is also defined over data points in an embedding space where the resulting embeddings form ideally per instance clusters. The loss function expects three embedding vectors as an input. An anchor x_i^a, a embedding that is of the same class x_i^p and one that is of a different class x_i^n. Then the loss yields

$$\mathcal{L}_{trpl} = \sum_i^N \left[\|x_i^a - x_i^p\|^2 - \|x_i^a - x_i^n\|^2 + \alpha \right]_+ . \tag{2.57}$$

Again $[x]_+ = \max(0, x)$, N is the number of all possible triplets $(x_i^a, x_i^p, x_i^n) \in \mathcal{T}$ in the embeddings, $\|\cdot\|$ is the $L2$ norm and α is a enforced margin between positive and negative pairs. The training behavior using $\mathcal{L}_{trpl}$ as a loss is depicted in figure 2.4.

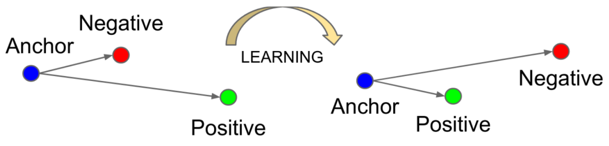

Figure 2.4: Position of triplets in the embedding space before and after training [4]

Often times N is very large and the loss requires too many resources to calculate. Therefore it is crucial to select triples that are active, namely $\left\{ (x_i^a, x_i^p, x_i^n) \middle| \left[\|x_i^a - x_i^p\|^2 - \|x_i^a - x_i^n\|^2 + \alpha \right]_+ \neq 0 \right\}$. However different selection strategies end in very different results regarding training time and convergence to satisfactory optima. E.g. picking only triplets that produce a high loss value can result in a training converging quickly into bad local optima. To prevent the embeddings from diverging a regularizer term is added in order to constrain the embedding vectors to the unit hypersphere surface.

$$\mathcal{L}_{reg} = \sum_{i}^{M} \frac{1}{2}(\|x_i\| - 1)^2 \tag{2.58}$$

Here M is the number of pixels in the image. The final loss therefore yields

$$\mathcal{L} = \mathcal{L}_{trpl} + \beta \mathcal{L}_{reg} \tag{2.59}$$

where β is a scalar weight for the regularizer term.

Methods

3.1 Using RL for the image segmentation task

The task of image segmentation fit generally into the RL framework. The agent takes the role of predicting action distribution where sampled actions perform some kind of change to the input of some clustering algorithm and the RL loss is calculated from the result of the cluster algorithm which is part of the environment.
The main advantage here is that the clustering algorithm does not need to be differentiable in order to optimize the agents parameters.
Common RL problems are usually problems that are easy to evaluate like determining the winner of a board game or evaluating a robots position relative to a target position. Also often intermediate results are not rewarded at all or given a small negative reward in order to put pressure on the fast arrival of the final state.
Typically in RL there is a phyiscal environment that can be explored by sensory input a reward evaluation can take place on that. The equivalent for the image segmentation task would be projecting the labels from the image plane into the 3d world and check if it labels all objects correctly. Of course this is highly infeasible therefore the reward generation has to be modeled by a virtual model of the environment. A segmentation can only be evaluated if enough information on the objects within the input image is known a priory. E.g number of object instances per object class, rough position, texture, shape etc..
Given such an evaluation and given that the function that is optimized by the agent is capable of capturing the underlying probability distribution of data ground-truth tuples, the described RL setting is theoretically able to learn the task of segmentation in a fully unsupervised way.

3.2 Using RL for predicting affinities

This work started with the idea of an RL setting where an agent manipulates affinities of pairs of pixels which form an input to the mutex wateshed algorithm 2.4. The edge set that mutex watershed typically receives is short range attractive edges and long lange repulsive edges where the length of the repulsive edges is dependent on the object sizes in the image. That means that there has to be at least one attractive edge for every neighboring pairs of pixels and some long range repulsive edge. Manipulating affinities is a hard task since there are simply so many. RL methods learns from rewards based on actions and an initial state. To achieve some convergence initially there need to be some rewards of a high value, that means that the actions taken lead to a fairly good segmentation. However when starting learning a networ from noise it outputs completely arbitrary actions that unlikely lead to a good results because the actions space is simply too large. The RL-typical bootstrapping works only if a high reward is likely even for random actions. It is not uncommon in RL to have many actions to predict while the reward is a single scalar value for a set of actions. If it would be possible to calculate more meaningful rewards, say per subregion in the image, one could compute a RL loss term for each such region only considering the actions that manipulated affinities within this region.

To downsize this problem, it is common in computer vision to work with superpixels [22] rather than with pixels. An a superpixel segmentation or oversegmentation us usually achieved by watersheding algorithms. Using mutex watershed on can easily globally weaken attractive edges in order to arrive at an oversegmentation. Starting from such an oversegmentation and assuming that the ground-truth segmentation is a partitioning of the superpixels, one can focus solely on merging and unsmerging superpixels. For mutex watershed a merge of two superpixels would be done by turning all the repulsive edges between them into attractive ones and vice versa for unmerging. Therefore one needs a decision/action for every pair of superpixels. However there are two problems with this, first one can only perform hard merges and unmerges which leads likely to contradictions for example consider 3 adjacent superpixels and there are 2 merges and 1 unmerge predicted (see figure **??**).
The second issue is, that for regular convolutional neural networks it is difficult to make predictions on affinities between adjacent superpixels.

3.3 Overview over the pipeline

To overcome the two problems stated in the previous section, the following was done. Making hard decisions on merges and unmerges between superpixels is on the one hand bad, because one has to do a hard thresholding of a networks predictions which robs us from the information that lies within the uncertainty in the predictions. On the other hand contradictions can arise in the final segmentation. Both can be overcome by predicting pseudo probabilities for the edges between superpixels which can be transformed into costs and a multicut 2.5 of the superpixel graph can be computed. This leverages the uncertainty of the network predictions and overcomes contradictions in the segmentation.
The second issue that arises from the irregularity of the superpixel graph can be overcome by using a graph neural network that predicts edge weights between superpixels by performing graph convolutions 2.3. However this generates the problem that there have to be regular representations for superpixels. Of course one solution would be to have two gcns, one for intra superpixel convolution and one for inter superpixel convolution. the former would take every pixel within a superpixel as a node with the scalar raw pixel vlaue as node feature and edges for all neighboring pixels. Doing some graph convolutions, pooling and moving all the information in the graph into the channels until there is only one node left. This nodes features can then be used as the features for the next inter superpixel gcn, predicting edge weights for the following multicut. The problem here is the intra superpixel gcn, doing graph covolution on raw images makes now sense, since it is not able to capute any spatial information like object shapes. There are some proposals [23] that introduce spactial dependant terms to graph cnvolutions but considering the power of CNNs it makes more sense to use those in favor ov GCNNs for convolutions on regular images.
So one can use a embedding network that predict pixel embeddings by minimizing a contrastive loss of

the form **??**. Still assuming, that the ground truth segmentation is within a partitioning of the superpixels it is safe to assume very similar pixel embeddings in terms of the distance used in the loss. Therefore one can average over all pixel embeddings within a superpixel in order to arrive at a singe feature vector that can be used as the node feature vector in the following GCN.

Bibliography

[1] R. S. Sutton and A. G. Barto, “Reinforcement learning:an introduction,” 2015.

[2] S. Wolf, A. Bailoni, C. Pape, N. Rahaman, A. Kreshuk, U. Köthe, and F. A. Hamprecht, “The mutex watershed and its objective: Efficient, parameter-free image partitioning,” 2019.

[3] B. D. Brabandere, D. Neven, and L. V. Gool, “Semantic instance segmentation with a discriminative loss function,” 2017.

[4] F. Schroff, D. Kalenichenko, and J. Philbin, “Facenet: A unified embedding for face recognition and clustering,” *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2015. [Online]. Available: http://dx.doi.org/10.1109/CVPR.2015.7298682

[5] R. S. Sutton and A. G. Barto, “Reinforcement learning:an introduction,” 2015, pp. 74–121.

[6] ——, “Reinforcement learning:an introduction,” 2015, pp. 157–194.

[7] Q. Liu, L. Li, Z. Tang, and D. Zhou, “Breaking the curse of horizon: Infinite-horizon off-policy estimation,” 2018.

[8] R. Sutton, D. Mcallester, S. Singh, and Y. Mansour, “Policy gradient methods for reinforcement learning with function approximation,” *Adv. Neural Inf. Process. Syst*, vol. 12, 02 2000.

[9] J. B. a. K. D. BrianD. Ziebart, Andrew Maas, “Maximum entropy inverse reinforcement learning,” 2008. [Online]. Available: https://www.aaai.org/Papers/AAAI/2008/AAAI08-227.pdf

[10] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine, “Reinforcement learning with deep energy-based policies,” *CoRR*, vol. abs/1702.08165, 2017. [Online]. Available: http://arxiv.org/abs/1702.08165

[11] P. N. Ward, A. Smofsky, and A. J. Bose, “Improving exploration in soft-actor-critic with normalizing flows policies,” *CoRR*, vol. abs/1906.02771, 2019. [Online]. Available: http://arxiv.org/abs/1906.02771

[12] G. Papamakarios, E. Nalisnick, D. J. Rezende, S. Mohamed, and B. Lakshminarayanan, “Normalizing flows for probabilistic modeling and inference,” 2019.

[13] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine, “Soft actor-critic algorithms and applications,” 2018.

[14] D. P. Kingma and M. Welling, “Auto-encoding variational bayes,” 2013.

[15] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, “Rainbow: Combining improvements in deep reinforcement learning,” 2017.

[16] H. van Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double q-learning,” *CoRR*, vol. abs/1509.06461, 2015. [Online]. Available: http://arxiv.org/abs/1509.06461

[17] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, “Prioritized experience replay,” 2015.

[18] Z. Wang, N. de Freitas, and M. Lanctot, “Dueling network architectures for deep reinforcement learning,” *CoRR*, vol. abs/1511.06581, 2015. [Online]. Available: http://arxiv.org/abs/1511.06581

[19] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, “Spectral networks and locally connected networks on graphs,” 2013.

[20] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, “Geometric deep learning: Going beyond euclidean data,” *IEEE Signal Processing Magazine*, vol. 34, no. 4, p. 18–42, Jul 2017. [Online]. Available: http://dx.doi.org/10.1109/MSP.2017.2693418

[21] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, “Neural message passing for quantum chemistry,” 2017.

[22] J. H. Kappes, M. Speth, B. Andres, G. Reinelt, and C. Schn, “Globally optimal image partitioning by multicuts,” in *Energy Minimization Methods in Computer Vision and Pattern Recognition*, Y. Boykov, F. Kahl, V. Lempitsky, and F. R. Schmidt, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 31–44.

[23] F. Monti, D. Boscaini, J. Masci, E. Rodolà, J. Svoboda, and M. M. Bronstein, “Geometric deep learning on graphs and manifolds using mixture model cnns,” 2016.