COMP 3105 Assignment #4 Report

Zhihao Xu (101306742)
Matteo Guerra (101316858)

Question 1:
Due time constraint and some technical difficulties in conceptions, the current version of submitted A4codes.py is not exactly the procedures described below(differs in segmentation, and for thus reason the performance may not be ideal)

If one wish to see a more complete version, one may visit the repository(Segmentaion-based-CNN) we created, and check the README log of the repo

Question 2: Model Design

In order for the model to be able to learn cues to classify things, first it has to be able to distinguish the differences of images; then the model needs to be able to recognize patterns and similar features at various scales, so that similarity can be found and thus suitable collective decisions can be made than simply overfits the data.

I. Segmentation (Not yet finishing implementation)
In order to accomplish the separation while preserve structure inside the images, we have the idea to first perform a suitable colour based segmentation of the image, with the processed image stored as pixel value being its corresponding segmented part index, into a jpeg file

I have three rather reasonable segmentation scheme possible:

(1) colour proximity segmentation

it faces the problem that if the image itself is chaos, or if a region is chaos in the eye of colour difference threshold, there simply would be too many noise segments. To solve such problem, one has to use adapative threshold to handle the differentiating stochastiness

(2) graph laplacian spectral decomposition with graph weights given by colour proximity

it defines a graph where vertices are pixels and edges between neighboring pixels(verticies) are assigned with weight w_{vu} given by their pixel colour proximity. Then one may consider a diffusion equation based on graph weights: given a density function $\psi : V \to \mathbb{R}$ is defined on graph's vertices, one has a difference equation $\frac{1}{\Delta t}(\psi[\Delta t]_v - \psi_v) = (\widehat{L}\psi)_v = \sum_{u:V} \psi_u w_{uv} - \sum_{u:V} \psi_v w_{vu}$ (it transfers the density content to neighboring vertices through its outgoing edges by exactly the weight assigned to the edges). This operator $\widehat{L} : C^0(V) \to C^0(V)$ is called its graph-laplacian(a differential function of diffusion can be given by considering $\frac{\mathrm{d}}{\mathrm{d}t}\psi[t] = \widehat{L}\psi$), its eigen vector represents the stable configuration of diffusions under this diffusion. This eigenvectors responds to graph weights (thus local color proximity) crticially, as graph laplacian is defined by this in the first place. Its second eigenvector has a zero set locus dividing the whole graph, which serve as a good generic segmentation of the graph thus the original image. One can iteratively perform segmentation on cutted subgraph(simply remove weights near the eigen-vector's 0-locus), which then it constructs a binary hierachy of cuts.

This has a few problems as well: if one is working with a $W \times H$ image, its graph would consists $W^2 \times H^2 - W \times H$ edges, which produces such a giant weight matrices for large images, that eigen-vector decomposition would consume 5 seconds for a 67×67 pixel image even with 14th gen i7 CPU, and even just to get the largest 2 eigen vectors still takes 2 seconds, which will be too expensive to just convert everything with this segementation directly.

Another issue came with the graph's lcoal textures, that it will cause some heavy local colour proximity variations. Even though solution of graph laplacian is to some extent resistant to these variations, it can still make the segmentaion wild. For that, there are three solutions I found: first wto are means of using Cartoon-Texture Decomposition(CTD):

one is using a solution to a normal direction diffusion system of first-order partial differential equation. I don't want to bother creating a PDE solver in this assignment, beside this equation is still painstakingly complicated.

Another is one idea I have, that to use 2D wavelet transform to directly downcast the quality of the image(filtering the high frequency band information, but only in a local sense), in this way, after 2 times of downcasting, 4×4 pixels will be grouped together, and even though clear boundaries is going to be maintained, complicated textures will already be dissipated in wavelet coefficients, and valid information will be kept in scaling coefficients. This is done simply by importing the WaveTFFactory package's tensorflow keras layer and mindlessly apply Daubechies wavelet('db2') 3 times

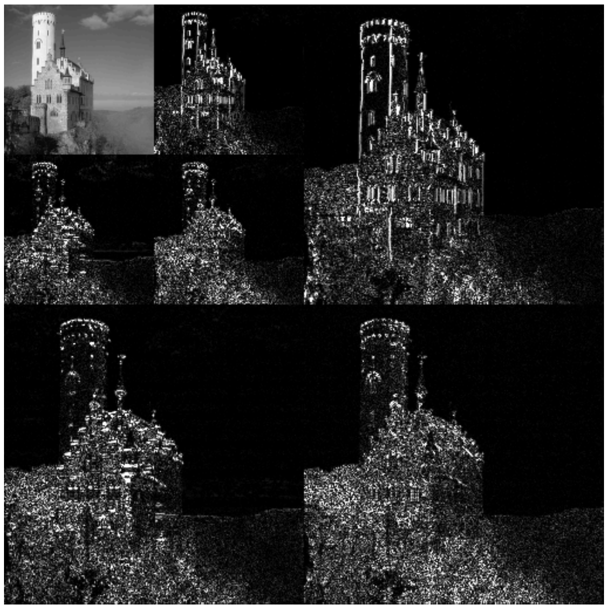

Figure 1: db2 wavelet transform result: the top-left corner is the scaling coefficients, rest are wavelet coefficients of different scale

However due time constraint, instead of implementing the graph lapalacian segmentation, I decided to just sequence the colour proximity direct segementation in (1) after wavelet inverse transformed scaling coefficient(so image will be of the original size, but with little texture)

(3) gradient minimizing spectral clustering segmentation (not considered)

The processed image then can be represented (more or less) by one hot $W_{\text{Img}} \times H_{\text{Img}} \times k$ tensor(it doesn't have to be one-hot so that it will even be capable of handling overlapping objects or trans), denoted ${P^{ij}}_K$(index i, j in the following of the article will be used to denote the position index of pixels of the image). Using this tensor, one can then attain many information with respect to each segment, now more or less isolated from its neighboring object's content created im-pureness:

The attained information has:

Colouring profile: (average colour: $\vec{C}_K = \mathbb{E}_{i,j}\left[\overrightarrow{Img}_{ij}{P^{ij}}_K\right]$, colour variance ...

Center position: $\vec{P}_K = \mathbb{E}_{i,j}\left[\vec{x}_{ij}{P^{ij}}_K\right]$

Size profile(inertia / positional variance): $\vec{\vec{P}}_K = \mathbb{E}_{i,j}\left[(\vec{x}_{ij} - \vec{P}_K) \otimes (\vec{x}_{ij} - \vec{P}_K){P^{ij}}_K\right]$

Position-Colour correlation: $\vec{\vec{P}}_K = \mathbb{E}_{i,j}\Big[(\vec{x}_{ij} - \vec{P}_K) \otimes (\overrightarrow{Img}_{ij} - \vec{C}_K) {P^{ij}}_K\Big]$

The eventual ${P^{ij}}_K$ will be merging accroding to similarity of colouring profile, in order to avoid the case where a shape is simply cut acrossed by some other object and then fails to be recognize by model's shape comparator. These colour profiles are ideally able to distinguish for example: physical images of object or symbolic images, as in the current training set's in/out domain

Then, one has to process the segmentation so that it they are given labels of abstract shapes. In order have model recongnize information of similar shapes, a size-agnostic comparison of eigen shapes will be applied to the whole graph:

Suppose the first layer weight consists of eigen-shapes (e.g. $\left\{{E^{ij}}_I\right\}_I$), then for each shape, each of its different size version ${m_\alpha(E)^{ij}}_I$ [1] (capped with a lower and upper bound to prevent meaningless Infrated divergence and Ultraviolet divergence of results) will be used to convolve with the segmented image data ${P^{ij}}_K$, and the result will be dividing by $\sqrt{||{P^{ij}}_K|| \cdot ||{E^{ij}}_I||}$ in order to exocist the differences caused by size through normalization, and thus the result after first layer will be:

$$X^{ij}_{(1)\,IK}(\alpha) = \frac{(m_\alpha(E)_I * P_K)^{ij}}{\alpha^{\dim Img}\sqrt{||{P^{\bullet\bullet}}_K|| \cdot ||{E^{\bullet\bullet}}_I||}}$$

In order to prevent needing for each scale α, image segment K and eigen-shape I a seprarte channel, since there will be only a few matches (local clusters where $X^{ij}_{(1)\,IK}(\alpha)$ is high), one instead here chooses to sparsely store only values of $X^{ij}_{(1)\,IK}(\alpha)$ when it passes certain threshold, along with a few extra parameter ($\alpha, I, K)$[2] and the information of its neighboring values(cluster configuration) the value is associated with. We store the first $k_{(1)}$<hyper-param> many large values, and denote its corresponding index $K_{(1)} : \{1, \cdots, k_{(1)}\}$, thus we have the actual final form after transform being:

$$X_{(1)\,K_{(1)}I},\ \vec{p}_{K_{(1)}}, \vec{\vec{p}}_{K_{(1)}}, \alpha_{K_{(1)}}, K_{K_{(1)}}$$

where $\vec{p}_{K_{(1)}}$ is the center pixel-position of the matching shape inside the image, and $\vec{\vec{p}}_{K_{(1)}}$ is the local covariance matrix along pixel-position and scale directions, where the "local" region is sketched by the unit

[1] due convolution convention choices, here the eigen-shapes tensor will be reflected true eigen-shapes

[2] instead of storaging, for each eigen-shape ${E^{\bullet\bullet}}_I$, its correspondance value, indexed by position i, j, I decided to use eigen-shape index I as one of the output indexing, and put positional information as additional information carried

sphere of the bilinear form $B_{K_{(1)}}$ given by the inverse of $\overset{\rightrightarrows}{p}_{K_{(1)}} + \text{stablizer} \cdot \overset{\rightrightarrows}{\mathrm{id}}$, with $\overset{\rightrightarrows}{p}_{K_{(1)}}$'s initial value being the Hessian matrix at local maximum.(In order to improve efficiency, the iteration might be saved, and Hessian may be used directly)

Then, the one substract ${X_{(1)}^{ij}}_{IK}(\alpha)$ by $-X_{(1)\,K_{(1)}I}\ \exp\left(-\frac{1}{2}B_{K_{(1)}} \cdot \left(\vec{x}^{ij} - \vec{p}_{K_{(1)}} \oplus \alpha_{K_{(1)}}\right)^{\otimes 2}\right)$, a negative masking normal distribution density given by this cluster position and covariance, and do the same local(global) maximum analysis on the new ${X_{(1)}^{ij}}_{IK}(\alpha)$, which would provide information of subsequent clusters (ordered by magnitude of ${X_{(1)}^{ij}}_{IK}(\alpha)$), we denote these info as $Info_{IK}$, for one hot index I of eigen-shape, and K being the index to first k significant clusters

(doing the sparse storage of significant value is very like PCA or having some 1×1 convolution layer followed by max-pooling layer after the original eigen-shape convolution)

Then, the second layer is supposed to inference from the relative positional and colouring relationships between the result of the first convolution (which converts segments to things represented by eigen-shapes). Due the fact that sparse storage $K_{(1)}$ index is unordered in terms of positions and many other perspectives, we really have to perform valid set operation on the remaining shapes. Such operation, conceptually, can be written down as:

$$
\begin{aligned}
&{X_{(1)}^{i_K j_K}}_{IK}(\alpha_K) \equiv S_{IK} \\
&{\omega^{IJ}}_T = \text{Compare}_T(Info_I, Info_J) \\
&\text{RelationLikelyhood}_T := S_{IK} S_{JK'}\ \text{Similarity}({\omega^{IJ}}_T, {\omega^{KK'}}_T)
\end{aligned}
$$

Where S_{IK} means similarity of Kth image shape data cluster with Ith eigen shape, the ${\omega^{IJ}}_T$ gives the relative relationships through information of shape I, J, where T is index to relation labels(channels). It can be relative position, or relative position modulos scale, or rotational relationships, adjacency, etc. It can be computed between eigen-shapes of the training images, or shape clusters of the same image, and Compare_T is going to be hard coded compare function. By using hard coded "Similarity" kernel(could be gaussian or linear, depends on how "Compare" function is designed), one gets the likelyhood of two shape cluster being with one of the archtypical eigen-shape relationships.

II. Fully Connected NN Layer

Till this step, shared visual patterns of objects and relationships between objects will all be encoded in $\text{RelationLikelyhood}_T$, and the precalculated colour and shape profile mean/variance etc., these data will be

Matteo Guerra (101316858)
Zhihao Xu (101306742)

feeding into Fully Connected NN to let it discern which is useful at what circumstances:

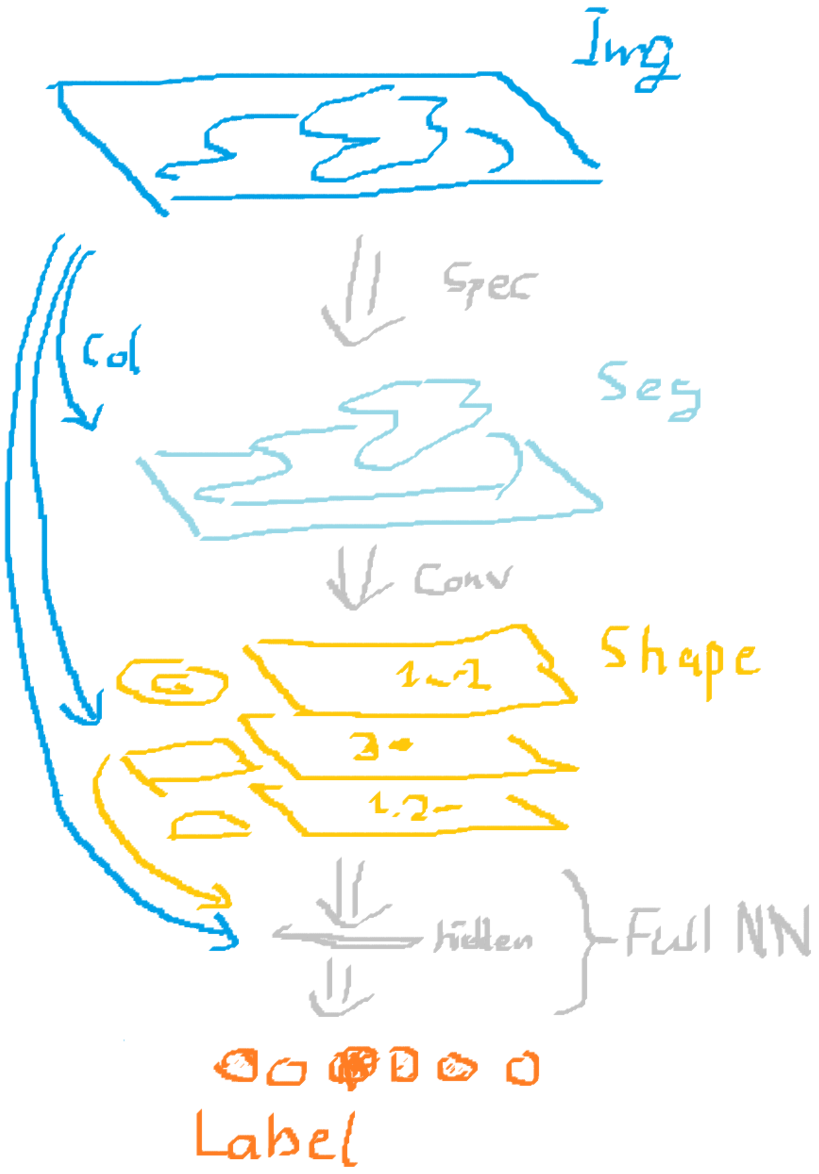

Figure 2: Sketch of Model's overall structure

III. The In/Out Domain and Decision Process

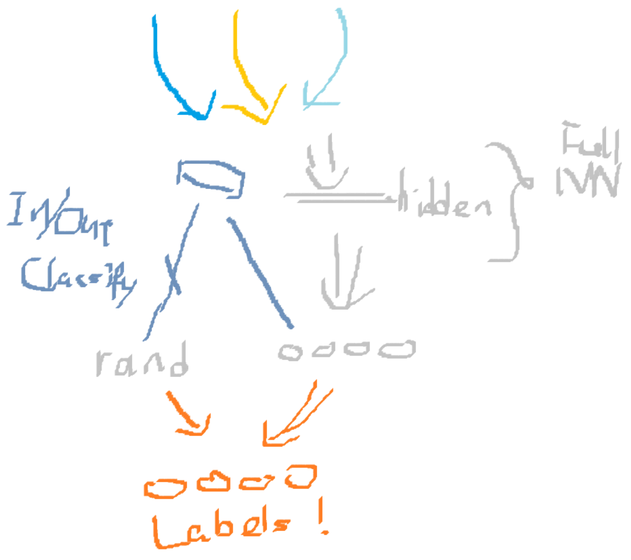

Figure 3: Model's Final Decision Process

The model will first train on labelled in-domain data to get weights for shape, then relational comparison, then fully connected Neural Net (gray) part. Then, while fixing all these part, train an in/out-domain classifier based on output of previous trained layers. This ensures relavant valid shape data is already capable to be outputted by previous layers, so that combined with colour and shape profiles, etc., the in/out-classifer has enough and valid evidences to learn to inference the image's in or out domain-ness. It also worth mention that the classification kernel would be gaussian, so that it ensures ANYthing statistically and featurally abnormal from the in-domain data would be termed out-domain, not just those in out-domain training data. Out-domain training set is merely checking the hyperparameters for gaussian can successfully separate in/out domain. (of course while not too overfit the in-domain training data)

Once the two parts are finished in training, the model will predict based on weights of the two components through the decision process given by Figure 3

(last step, if is classified to be out-domain, it will instead use random label, intentionally directly choose the label that has the lowest posibility (lowest predictor) amoung all labels; in theory, it may get out-domain accuracy even lesser than 10%)

IV. Some Extra Ideas (Not yet implemented)

Inside the optimizer, in addition to optimize by back-propagating gradient descent, it will also accumulate significance measure(in a way, this represnts apriori belief in statistical learning) on every eigen-shape and eigen-relationship, and in every complete epoch, it will seeks remove insignificant ones, merge similar ones (taking orthogonalization of eigen-shapes and redistribute weights to achive invariance), and inside removed eigen-shape/relationship, randomly find a segment / segement pair in pre-existing images as initialization.