COMP 3105 – Assignment 4 – Report

Raphael Mercier (101 306 865) and Patrick Wu (101 258 669)

1. Introduction

The goal of this assignment was to implement an AI which can solve a real world problem; attaining high accuracy on one dataset (in-domain), while getting low accuracy on another (out-domain). This problem aims to solve the issue of a malicious user wanting to use our model on their dataset.

Through trial and error, we have observed that as out-domain accuracy is lowered, in-domain accuracy is inevitably affected.

After having tested a few strategies such as intentionally training poorly for out-domain and determining if an image is in or out-domain before classifying it, we have determined that the best strategy is that of maximizing entropy for out-domain.

2. Methodology overview

The methods we have tested can be summarized into three categories

2 Stage model: first it trains to identify in from out-domain (the categorization model), then it trains to classify in-domain images (the classification model), finally during prediction, if the image is predicted as in-domain, use the model to predict its class, if its out-domain, intentionally guess the least likely class. This method is great for reducing out-domain accuracy, keeping it near 10%. On the other hand, the 2 stage method is slower since training to decipher in from out-domain takes more epochs, and it brings in-domain accuracy down to approximately 50% which is not great.

Adversarial Training: train in-domain and intentionally do bad for out-domain. To train poorly on the out-domain we had to dynamically create fake labels for the out-domain which were chosen based on the least likely class according to our model's prediction. This implementation is good for keeping in-domain accuracy high, it's quick since training out-domain is fast. However, it does not do a good job at reducing out-domain accuracy.

Entropy maximizing: we train both in and out at the same time, the trick is we minimize cross entropy for the in-domain while maximizing entropy for the out-domain by negating its loss (entropy). Mathematically the entropy for the out-domain is:, $H = -\sum_i(p_i log(p_i))$ where p is the probability distribution for mapping the image to a class. From there, we negate the entropy and then calculate the loss as follows: $L = L_{in} + \lambda H$ where λ is a parameter. In some ways, this method is similar to adversarial search except that instead of using the normal cross entropy loss function, where only the predicted class is considered for the loss, our function considers

every probability in our function, and as we maximize it the individual probabilities approach a uniform distribution. This strategy is great for keeping accuracy high all while having a decently sized gap between in and out-domain, It is also very fast. The down side is the out accuracy is not as low as it could be

3. Algorithm design and justification

3.1 Model architecture

Although the 2 step model had a much lower out-domain accuracy, we decided to stick with the entropy maximization implementation, since it is faster and has a better in-domain accuracy which we deemed as more important.

Entropy Maximizing Model Architecture:

- ResNet18 (randomly initialized, no pretrained weights)
- Custom classifier head: Dropout (0.5) → Linear layer (512 features → num_classes)
- Loss function: CrossEntropyLoss
- Optimizer: Adam (learning rate: 0.002, weight decay: 0.0001)
- Learning rate scheduler: StepLR (step_size: 8 to 16, gamma: 0.5)
- Batch size: 100
- Entropy weight (λ): 0.2
- Early stopping mechanism

We tried other models like ResNet34, EffecientNet etc… but they were either too slow or not as accurate, while ResNet18 struck a seemingly perfect balance having enough capacity to have 10 inputs but not too much that it's too slow.

We used PyTorch's sequential API to substitute the fully connected layer for a simpler version this avoided overfitting, and made sure to not overcomplicate

We used CrossEntropyLoss which is standard for classification models such as this one.

We implemented Adam optimizer because its faster than SGD, the parameters were determined empirically, we used weight decay to avoid overfitting

We also implemented a learning rate scheduler, this allows to avoid overfitting as the model reaches convergence, these were also determined empirically

Entropy weight is a multiplier that determines how much to consider the out-domain entropy in the loss function. This is expressed as $L = L_{in} + \lambda H$

Finally, we implemented an early stopping mechanism that checks if the accuracy has not improved for long enough or if it has reached a certain number like 99%, we assume we have reached convergence and we stop the training

Note that the model architecture did not change much across the different methodologies. Even within the 2 stage method, the categorization and the classification model architectures were very similar.

3.2 Entropy-based training strategy

Entropy is a measurement of how uncertain the model is. Mathematically, entropy is: $H = -\sum_i(p_i \log(p_i))$, and as we maximize that equation we find that p approaches $1/(\#\text{ of classes})$, for every i, in other words the probability distribution approaches uniformity.

The entire idea behind this model is that we maximize the entropy for the out-domain, all while minimizing the cross entropy loss function (using PyTorch) for the in-domain. The maximization is done simply by implementing it carefully into our total loss function: $L = L_{in} + \lambda H$, where λ is a parameter defining the entropy weight. This works because negating the entropy flips the function and tricks the backward pass to maximize entropy by minimizing the loss, furthermore the loss for the in function is added so that it can learn about both terms and achieve both goals at the same time.

The distribution of probabilities will therefore approach $[0.1, 0.1, ..., 0.1]$ for the out-domain, and $[0, 1, ..., 0]$ for the in-domain

3.3 Data preprocessing and augmentation

Data augmentation was vital to our success especially since the dataset size is very small, we used a combination of Resize, RandomCrop, RandomHorizontalFlip, RandomRotation and ColorJitter Transform functions from TorchVision. We also experimented with Normalize, and found that it was helpful to have a fraction of the data normalized.

Furthermore, the number of augmented versions we added to the dataset determined the complexity of the model, we found that after augmenting around 7 times the accuracy did not get better.

We also used data augmentation to balance out in and out-domain accuracies, meaning that they would have the same images and thus no bias for one or the other.

We also tested augmenting alongside the training, but it did not seem to be optimal

4. Experimental results and analysis

4.1 Model performance

Note that out and in-domain accuracy are closely correlated, meaning if one increases or decreases, the other will follow.

	in-domain accuracy (%)	out-domain accuracy (%)	approx Gap (%)
Only train in-domain	68 – 73	50 – 55	18
Adversarial train	60 – 65	40 – 45	20
Entropy maximization	68 – 63	35 – 40	28
2 Stage model	48 – 53	12 – 17	36

For the 2 stage model it is important to note that there are actually multiple different metrics we can look at:

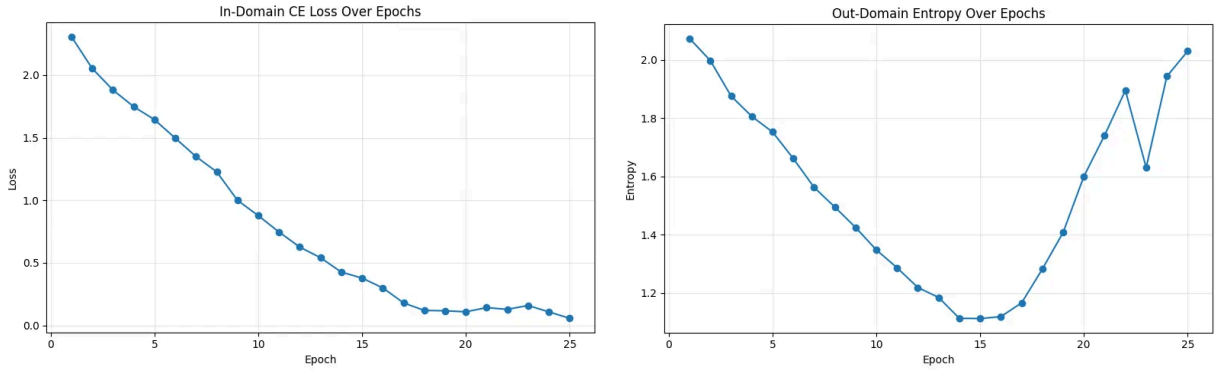
- Accuracy of categorization → 68-73%
- Accuracy of classifying in images → 68-73%
- Accuracy of classifying out images → 1-5% (This is so low because the model intentionally picks what it thinks is the least likely class)

Mathematically, we can express the in-domain accuracy with the 2 stage model as:

$$\text{Final in accuracy} = P(\text{in} \mid \text{in}) \cdot \text{InAcc} + P(\text{out} \mid \text{in}) \cdot \text{OutAcc}$$

$$\text{Final out accuracy} = P(\text{in} \mid \text{out}) \cdot \text{InAcc} + P(\text{out} \mid \text{out}) \cdot \text{OutAcc}$$

Entropy based solution:



Note the training accuracy increases smoothly, eventually stabilizing. This indicates our model is able to learn in-domain features effectively even while our out-domain entropy objective is being applied, meaning it does not interfere with our in-domain convergence.

For out-domain metrics, the entropy curve starts off high, then dips before rising and peaking again.This makes sense as during

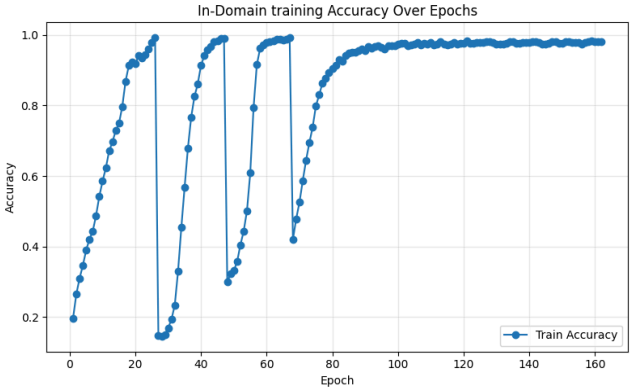
1. Early training, our model is untrained, thus SoftMax outputs/probabilities are random, which leads to high entropy. Essentially our classifier is not confident about classification as it hasn't learnt to classify anything accurately yet

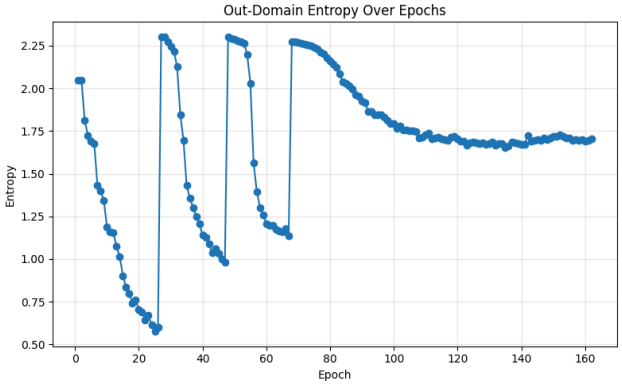
2. As our classifier improves on in-domain data, out-domain samples get pulled toward a more confident but probably incorrect in-domain prediction, lowering entropy and increasing confidence, explaining the steady decrease in entropy.

3. Once in-domain learning peaks and plateaus, the model learns to spread probabilities more evenly for out-domain inputs, decreasing confidence and conversely entropy, without disrupting our in-domain classification.

Thus our model settles into a balance where it is able to accurately classify in-domain data, while being unconfident with out-domain data.

Adversarial approach:





We can see the alternating pattern of the adversarial training and in-domain training approach directly in all 4 graphs. During adversarial phases, our model is pushed to misclassify out-domain samples by selecting the least likely class. The results are visible swings of increased entropy and softmax probability in the out-domain Entropy and confidence graphs.

When our training switches back to the in-domain classifier, our model recovers its in-domain training accuracy that was impacted by the adversarial training before stabilizing again. Over time, these cycles balance out where our model stays accurate on in-domain data and will become more unreliable on out-domain inputs. Similarly to our entropy approach to this problem, the plateau in accuracy shows this does not prevent the classifier from being able to predict in-domain samples accurately.

4.2 Training dynamics

For the 2 stage implementation, convergence takes approximately 50 to 60 epochs for the categorization model, and 35 to 45 epochs for the classification model it takes 15 to 20 minutes

to run in total depending on the parameters set, namely the number of times we augment the dataset.

The other methods tend to converge at around 20-30 epochs depending on different parameters, and generally run for approximately 7 to 12 minutes.

5. Design decisions and trade-offs

Entropy maximization has several practical advantages. Theoretically, it uses a single model, and is the most simple system (we don't need 2 separate models used in the 2 stage approach or 2 training loops with our adversarial approach). Making effective use of our unlabeled out-domain data, it directly optimizes uncertainty for unfamiliar data that is contained within the out-domain, without affecting in-domain accuracy. Empirical evidence shows that entropy on out-domain examples eventually begins to increase as training processes, while accuracy on in-domain data peaks and remains high.

Comparatively, our adversarial approach achieves a similar in-domain accuracy to our entropy maximization model, however with a higher out-domain accuracy (from accuracy table in section 4). With entropy maximization, the model is essentially collapsing out-domain features so our classifier cannot distinguish anything, resulting in what is essentially a random guess of its classification. Due to these uniform predictions, our model learns to get poor accuracy for anything not from the in-domain dataset.

On the other hand, the Adversarial model behaves differently. Instead, the model is learning feature representations of out-domain samples to pick the least likely class when encountering an out-domain sample. This makes the model less reliable and inaccurate on out-domain samples, but does not set predictions to uniform randomness. As a result, we found this approach seems to lead to slightly higher out-domain accuracy.

Compared to both our Adversarial model and Entropy maximization approach, the 2 stage approach demonstrates the lowest out-domain accuracy. This is expected given that one of our models learns to classify in-domain vs out-domain images, and the classifier only operates on samples considered in-domain. The consequence is that in-domain accuracy is significantly lower. Since the model needs to classify whether an image is in-domain or out-domain first, any misclassification would propagate and reduce the accuracy on in-domain data.

While the out-domain performance was much better with the 2 stage approach, we felt that having a significantly higher accuracy in the in-domain with a respectable gap between in-domain and out-domain accuracy would be a better compromise.

6. Improvements

The out-domain accuracy is still decently high with our solution. The entropy maximizing method was only discovered a few days before submission, had we had more time, we would have tested different ways to maximize entropy for out-domain. For instance, we could try a formula

other than $L = L_{in} + \lambda H$, to incorporate it into the loss function, or perhaps we could have tried having separate loss functions separating in and out-domains completely.

Moreover, the in-domain accuracy could have also been improved, although our solution only slightly hindered the in-domain accuracy when comparing to the control version where the out-domain was not considered during training. There could exist some strategy to improve its accuracy to 80%.

7. Conclusion

In conclusion, after exploring 3 different methods for trying to maximize in-domain accuracy while minimizing out-domain accuracy, we found empirical evidence supporting both main methods, the 2 stage model and single stage models. When looking for the absolute biggest difference in in vs out-domain accuracies, the 2 stage model yielded better results than the Adversarial and Entropy Maximizing methods, whereas the latter 2 have much higher in-domain classification accuracy. In our case, the Entropy maximization approach provided what we considered to be the better compromise, achieving high in-domain accuracy with a respectable out-domain accuracy, and thus was chosen to be our model.

Citations

Image Transform Documentation

PyTorch. *Transforms.* https://docs.pytorch.org/vision/main/transforms.html

ResNet-18 Documentation

PyTorch. *torchvision.models.resnet18.* https://docs.pytorch.org/vision/main/models/generated/torchvision.models.resnet18.html

PyTorch Core Documentation

PyTorch. *torch.* https://docs.pytorch.org/docs/stable/torch.html

TorchVision Documentation

PyTorch. *torchvision.* https://docs.pytorch.org/vision/stable/index.html

Consulted Individual

Student: Zhenxuan Ding — “Think about what maximizes out-of-domain entropy while minimizing in-domain cross-entropy.”