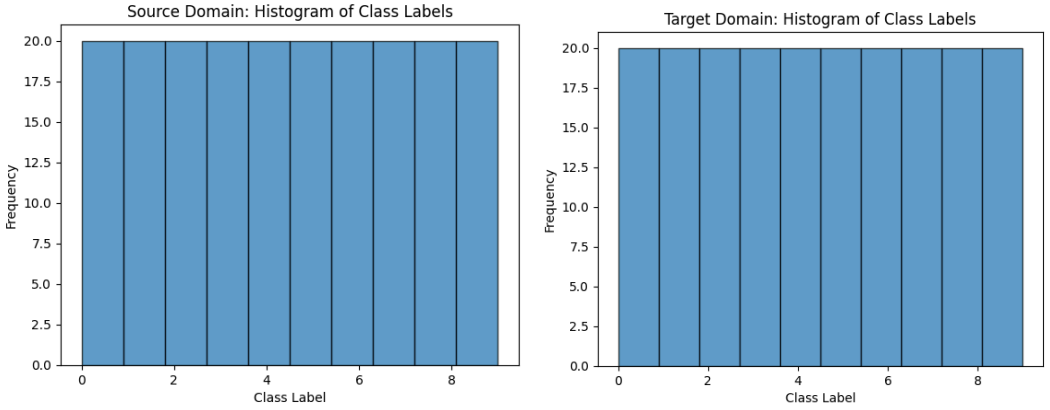
Sophea Bonne, 352901
Joshua Cohen-Dumani, 311105

ML for Predictive Maintenance Applications: Assignment 4

Methodology

Data

The provided data are FFT preprocessed vibration data. The source and target datasets both contain 200 samples with 512 time stamps each. The classes are perfectly balanced, with 20 samples across 10 classes. According to the exercise description, 9 classes belong to different faults.



Baseline Model:

Firstly, our baseline model. Our model architecture is as follows:

- A first convolutional layer taking in the 1d time series input and outputting to 10 channels
- Second and Third convolutional layers with 10 input and output channels.
- A batch norm layer is added after each convolutional layer, in order to standardize the outputs and hopefully stabilize training.
- A dropout layer which sets 10% of the output of each convolutional layer to 0. This adds regularization and hopefully reduces overfitting (especially useful since we will be performing a domain shift).
- We then flatten the output layer into what we can consider our learned features.
- We then feed these learned features to a classifier, composed of 2 fully connected layers with batch norm applied.

During training, we used the Adam optimizer and the negative log-likelihood (NLL) loss, as suggested by the boilerplate code provided.

Coral Loss Model:

The CORAL model builds on the baseline by introducing the CORAL loss to align feature distributions between the source and target domains. This loss minimizes the Frobenius norm of the difference between their covariance matrices, scaled by the feature dimension. The total loss combines the source domain classification loss and the CORAL loss, weighted by a coral_weight parameter.

This approach is effective when the domain shift involves differences in feature covariances, helping the model learn domain-invariant features that generalize better to the target domain. However, it may struggle with more complex domain shifts, such as differences in feature means or higher-order relationships, limiting its standalone effectiveness.

Adaptive Batch Normalization Model:

Adaptive Batch Normalization (AdaBN) enhances the baseline model by aligning the feature distributions between the source and target domains. In the baseline model, Batch Normalization layers normalize activations using statistics (mean and variance) computed from the source domain during training. However, when applying the model to the target domain, these source statistics may not accurately represent the target data, leading to decreased performance.

AdaBN addresses this issue by updating the Batch Normalization statistics to reflect the target domain. After training on the source data, we pass the target data through the model in training mode without updating the weights. This process recalibrates the running mean and variance in the Batch Norm layers to match the target domain's distribution.

By doing so, AdaBN helps the model adjust its normalization to the target data, improving its ability to generalize without retraining the entire model. This method is particularly effective when the domain shift primarily affects the feature distributions captured by Batch Norm layers (mean and variance).

However, AdaBN assumes that the main difference between domains is in the feature statistics. If the domain shift involves changes that Batch Norm cannot capture—like different feature representations or label distributions—AdaBN alone may not fully bridge the performance gap. In such cases, combining AdaBN with other domain adaptation techniques could provide better results.

Adversarial Model:

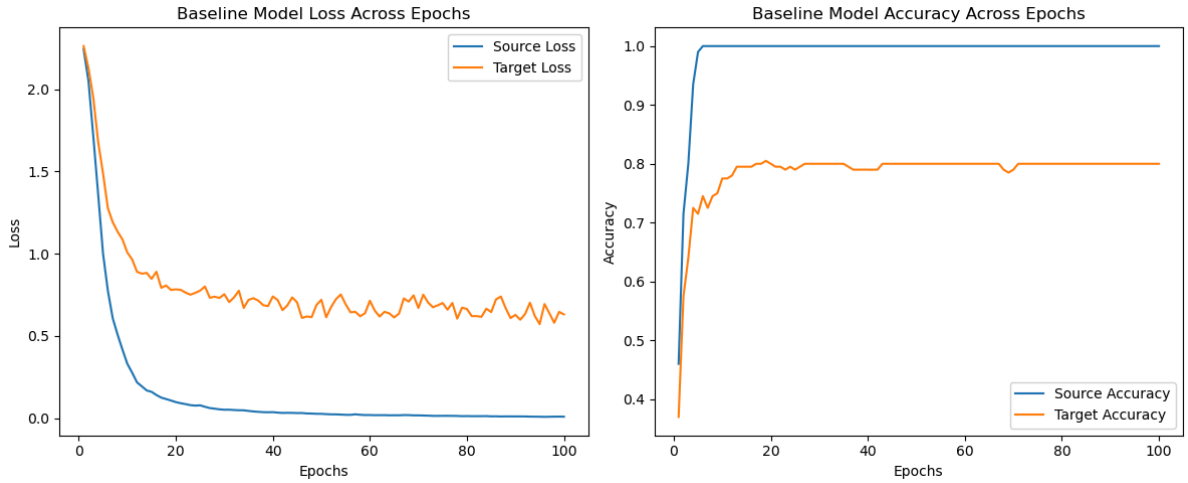
The adversarial model first trains the baseline model to get a feature extractor and a classifier. Then, we train a discriminator that classifies whether a given feature comes from the source domain or the target domain. This discriminator has two linear layers, both followed by batch norm and Relu activation, then a third linear layer. The discriminator's loss combines two cross-entropy losses: correctly classifying source features as source domain and target features as target domain. Then we train a generator that tries to confuse the discriminator. We do this by adding cross-entropy loss between source predictions and target, and target predictions and source. We scale this generator loss by adversarial_weight, which defaults to 1, and add to the classification loss.

This model works well if the source and target domain are similar but have a domain shift. This alternating “push and pull” of the discriminator correctly classifying source or domain, then the generator trying to confuse the discriminator, is what makes this model adversarial. This is advantageous for domain-invariant features that generalize well to new data distributions (target domain). However, there are limitations if the discriminator becomes too

strong – the generator may fail to create diverse, meaningful features and result in poor domain alignment.

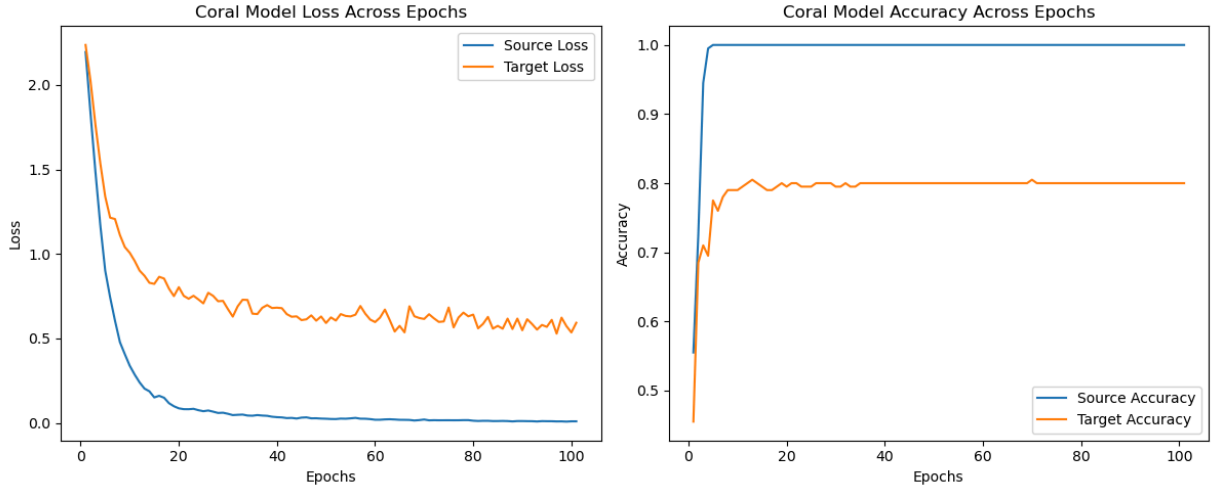
Evaluation and Discussion

Baseline Model:
The following figure shows the training curve for the baseline model. It includes the source and target loss, as well as source and target accuracy over 100 epochs.



We see a sharp rise in the early epochs, expected as the model quickly learns the dominant features in the training data. Indeed, we see that after a certain point the model has a 100% accuracy on the source domain. However, in the target domain, we see a saturation at around 80% accuracy. This can be due to several factors: we could be overfitting to the source domain (as shown by the 100% accuracy on the source domain quite quickly). To address this we could try increasing the dropout rate. In addition, this could be due to a large shift in distribution between the source and target domains, that our baseline model is unable to capture.

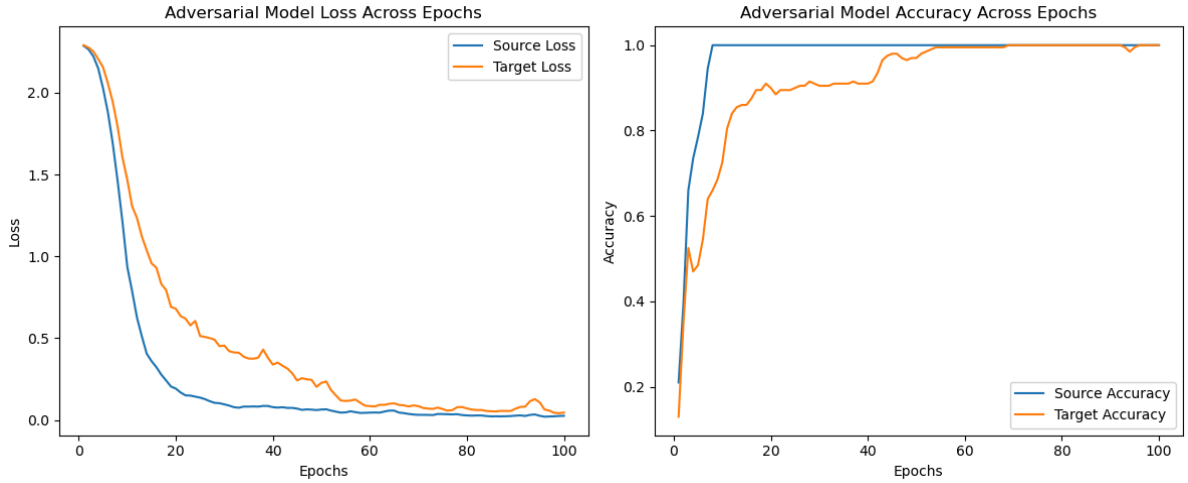
Coral Loss Model:
The following figure shows the training curves for the coral model.



We noticed that it is quite similar to our baseline model, and we actually do not gain much from changing to the Coral loss. This could be explained in several ways.

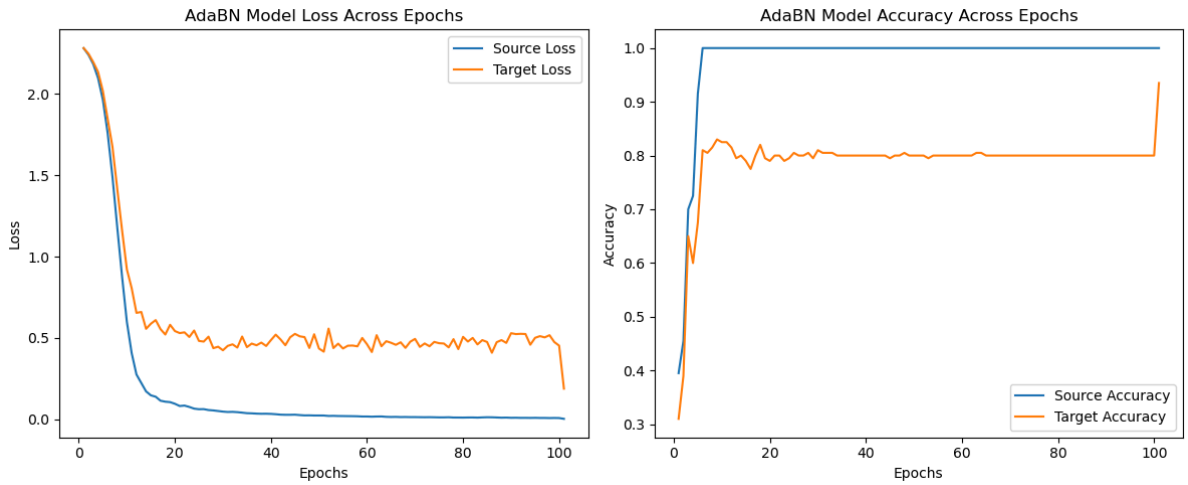
Firstly, coral loss is based around calculating the distance between the second-order statistics of the source and target domains. However, it is possible (and likely, given our results), that the actual domain shift is more complex than just the second order statistics, or is in some way not captured well by this covariance distance. Another reason could be that our model architecture is too small, which actually does not allow the model to learn more complex features which could be more representative of the target domain.

Adversarial Model:



From the training curve, we observe that the adversarial model performs very well – better than any of the other models on the target domain. We achieve near perfect accuracy around epoch 55. This suggests that the source and target domain are quite similar but with a domain shift.

Adaptive Batch Norm Model:



For the adaptive batch norm, we actually noticed that the accuracy could be much greater when NOT using batch norm in the model architecture in between the convolutional layers

(ie. just using dropout, and batch norm in the fully connected layers but not in the convolutional layers). Thus, the above plot shows the evaluation of the model using the baseline model without batchnorm in between the convolutional layers (the plot with batchnorm in the baseline model can be found in the Annex).

This is our interpretation of the above plot. During training, the model learns the source domain features very well (as seen with the very quick saturation to 100% accuracy). As mentioned in the baseline model interpretation, our model is only then able to get to around 80% accuracy on the target domain (probably due to overfitting). However, after Adaptive Batch Norm, we actually see that the model is well adapted to the target domain and we see an effective domain shift, jumping to 93.5% accuracy on the target domain.

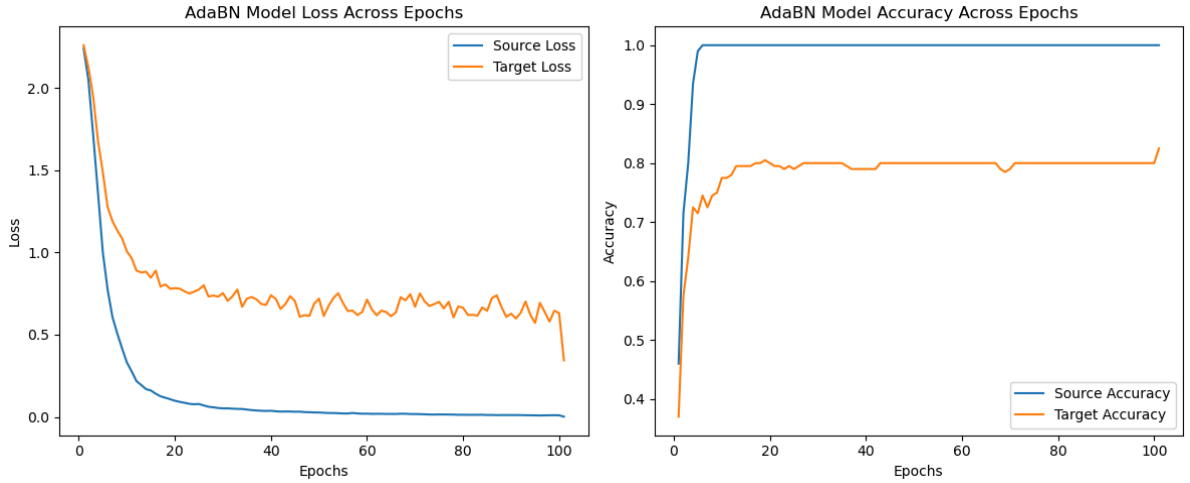
Our theory as to why the model performs better if we have a baseline model without batchnorm in the feature extraction layers is because as mentioned in the adversarial model interpretation, we believe that the source and target domains are probably similar but have a set distribution shift. Thus, by training our model on learning the features of the source domain well (willingly overfitting), but then performing the domain shift using AdaBN, we thus correct for most of the domain shift and can get a large accuracy improvement. Whereas in the case where we use batchnorm when learning the features, our learned features are probably more domain invariant, which makes adaptive batch norm less effective.

Summary results table:

Model	Source Loss	Source Accuracy	Target Accuracy
Baseline	0.009795	100%	80%
Coral	0.01	100%	80.50%
Adversarial	0.03	100%	100%
AdaBN	0.0032	100%	93.50%

Adversarial achieves the highest accuracy on the target domain, with AdaBN second. From the table, it appears that adversarial has the highest source loss, but we sum several losses so we shouldn’t directly compare the losses.

Annex:



Above is the plot of the AdaBN model with Batch Norm layers after the convolutional layers. As we can see, the addition of the AdaBN does not meaningfully change the result.