Transfer learning for embedded domains

Matt Beton

Abstract

Transfer learning is a technique that has been applied in many formats to transfer knowledge between two related, or seemingly unrelated, domains. While transfer learning has shown great strength in the computer vision, it has been less thoroughly researched in the field of time-series data. In this essay, we propose transfer learning between two *embedded tasks*; tasks where one covariate domain is a dimension-wise slice of the other. We first train a source model on the smaller domain, and then use parameter duplication and finetuning to create a target-domain model with the correct dimension. We supplement the theory with experiments on transferring single-lead to multi-lead ECG data, both with real and semi-synthetic data.

1 Introduction

Transfer learning has been a hot topic of machine learning research for some years, a technique that utilises transfer of knowledge between multiple related datasets. We hope to use skills learnt on source datasets to improve performance on a target dataset. This has parallels in human learning processes, for example a person that can already play piano may find it easier to learn guitar, due to the overlap of skills transferrable in the domain of playing music. One notable domain where transfer learning has been applied is to image recognition.

Transfer learning has been less well-explored in alternative domains; in this essay, we talk about applying transfer learning to timeseries data. Specifically, we consider the scenario where the source and target domain are timeseries with different feature spaces. In chapter 2, we introduce and formalise the concept of *embedded tasks*. This is defined as a scenario where our source covariate domain can be seen as a dimension-wise slice of the target domain. We explore using parameter duplication, fine-tuning and other architectural changes to solve the problem of embedded transfer learning, and different modelling decisions we can make based on the task we may be dealing with.

We supplement our theory with experiments in the field of ECG classification; taking an ECG reading and identifying whether the patient has a certain cardiac condition. We will provide some background on this field in chapter 3, and explore experimentally applying embedded tranfer learning to ECG in chapter 4. We disagree with some of the methodology that has been used in existing papers on ECG classification, giving justification for why we took the approach we did (and why we don't use the MIT-BIH dataset) in chapter 5. Finally, in chapter 6, we discuss some further potential applications of embedded transfer learning.

2 Transfer Learning

2.1 Domains, Tasks, and Manifolds

We begin by defining transfer learning between domains. First, we need to define the notion of a *domain* and *task* [1].

All source code for this project can be found at `https://github.com/mattyab/ecg_transfer`

Definition 2.1 (Domain)**.** A domain $\mathcal{D} = (\mathcal{X}, P(X))$ is the composition of a feature space $\mathcal{X}$ with a distribution $P(X)$, such that X is a random variable over $\mathcal{X}$. $\mathcal{X}$ gives the shape of the feature set, whilst $P(X)$ defines the distribution that samples $\mathbf{x} \in \mathcal{X}$ are drawn from.

Definition 2.2 (Task)**.** A task $\mathcal{T} = (\mathcal{Y}, f)$ is the composition of a label space $\mathcal{Y}$ and a decision function $f : \mathcal{X} \to \mathcal{Y}$, defining the true relationship between covariates and labels.

We write the joint distribution of X, Y as $P = P(X, Y)$. The goal of our model is to get the best approximation for f, given the dataset of examples $\{(\mathbf{x}_i, y_i)\}_i$. In some cases (most notably with some classification models) our model outputs the predicted conditional distributions instead of point predictions,

$$f(\mathbf{x}_j) = \{P(y|\mathbf{x}_j) : y \in \mathcal{Y}\}$$

2.2 Transfer Learning Definition

Definition 2.3 (Transfer learning)**.** Transfer learning uses knowledge learnt from a *source* domain and task $(\mathcal{D}_S, \mathcal{T}_S)$, to improve performance over a *target* domain and task $(\mathcal{D}_T, \mathcal{T}_T)$, compared with if a decision function were learnt solely on the target domain.

Between our two domains, there is assumed to be some shift in distribution $(P_S \neq P_T)$, which is why we need to 're-learn'; we cannot typically use the source model out-of-the-box on target data. Scenarios where we do reuse our model without any retraining between $\mathcal{D}_S$ and $\mathcal{D}_T$ are called *zero-shot.* Domain shift can take multiple forms:

- **Covariate Shift** is shift in feature (covariate) distribution, $P_S(X) \neq P_T(X)$ [2].
- **Concept Drift** is shift in conditional distribution, $P_S(Y|X) \neq P_T(Y|X)$ [3].
- **Heterogeneous Transfer** allows for differing feature spaces, $\mathcal{X}_S \neq \mathcal{X}_T$.

Despite domain shift between our two domains, by performing transfer learning we hope that there is information in the source distribution that can be useful in building a model for the target distribution. This could, in parallel with our possible domain shifts, take the form of a similarity relationship between $P_S(X)$ and $P_T(X)$, or between $P_S(Y|X)$ and $P_T(Y|X)$, that can be exploited by the use of transfer learning.

Transfer learning can be useful in a variety of settings. It can be beneficial when we have plentiful data from our source domain, but lack sufficient data for training a model from scratch on our target distribution; this is the setting we will explore in this essay. With transfer learning, we are able to harness the similarities between source and target distribution to build an effective model for our target task without falling victim to overfitting issues due to data unavailability. A common workflow for transfer learning is to utilise generalised predictive power of a publicly available model, beneficial when we lack the data or compute resources to train a model from scratch. Taking a large pretrained model, trained on a general dataset such as ImageNet[4], one could fine-tune it on a set of images of previously unseen objects. The original model may have never seen a picture of an insect before, but given fine-tuning examples of bees and wasps, the network's general image-recognition ability is able to refocus to the task of discerning bees from wasps.

2.3 Embedded Transfer Learning

In this section, we define transfer learning in *embedded domains*, a pair of domains with one contained in the dimensions of the other.

Definition 2.4 (Embedded Domains)**.** An *embedded domain pair* is a pair $\mathcal{D} = (\mathcal{X}, P(X)), \mathcal{D}^\star = (\mathcal{X}^\star, P^\star(X^\star))$, where elements $\mathbf{x}^\star \in \mathcal{X}^\star$ are given by $\mathbf{x}^\star = (\mathbf{x}_1^\star, \ldots, \mathbf{x}_k^\star)$, and crucially each $\mathbf{x}_j^\star \in \mathcal{X}$. Each $\mathbf{x}_j^\star$ is called a *channel*, and k is the *channel count* of the embedding. We say that $\mathcal{D}$ is *embedded* in $\mathcal{D}^\star$.

The joint distribution $P^\star(X^\star, Y^\star)$ can be flattened to a single channel. We write the i'th coordinate of $X^\star$ as $X_i^\star$, and write the joint distribution $P_i^\star = P_i^\star(X_i^\star, Y^\star)$, referred to as the *slice distribution* after Python slice notation.

Definition 2.5 (Embedded Transfer Learning)**.** Embedded transfer learning is transfer learning between source-target pairs $((\mathcal{D}_S, \mathcal{T}_S), (\mathcal{D}_T, \mathcal{T}_T))$, where $\mathcal{D}_S$ is *embedded* in $\mathcal{D}_T$.

To perform transfer learning over an embedded domain pair, we are assuming some relationship between the source distribution $P(X, Y)$ and the slice distribution $P_i^\star(X_i^\star, Y^\star)$. So crucially we not only have a related shape of the feature spaces $\mathcal{X}, \mathcal{X}^\star$, but also some relevance of the source distribution P to the marginal slice distribution $P_i^\star$.

2.4 Fine-Tuned Models

In this section, we will explore how transfer learning is done in practice. A common transfer learning setting for deep learning models is *parameter fine-tuning.*

We define a model that is completely parameterised by its parameter set $\{w_i\}$, written as a decision function $f(\mathbf{x}; w_1, \ldots, w_i)$. We have a source-domain dataset $D_S = \{(\mathbf{x}_i, y_i) : \mathbf{x}_i \in \mathcal{X}_S, y_i \in \mathcal{Y}_S, i \in \{1, \ldots, n\}\}$ of examples drawn from (X, Y). When training, we aim to minimise an objective function that incorporates the *loss function* $\mathcal{L}(f; \mathbf{x}_1, \ldots, \mathbf{x}_n, y_1, \ldots, y_n)$, a function that represents the overall error of the model on labelled source-domain instances. The objective function will often also include a *regularisation term* Ω acting on the set of model parameters w_i, introduced to improve model generalisation. Common examples of regularisation term are L1 regularisation $\sum_i \|w_i\|$, and L2 regularisation $\sum_i (w_i)^2$. Our overall objective is to find weights that minimise a linear combination of these terms,

$$\underset{\{w_i\}}{\arg\min}\{\mathcal{L}(f) + \lambda\Omega(f)\},$$

where the regularisation parameter λ is chosen to reflect how much regularisation we want in our model. The higher the parameter, the more regularisation we apply, and the more we enforce the belief that 'simpler models are better'. This objective function can be minimised with some preexisting optimisation algorithm, such as SGD[5] or Adam[6], which are cleanly implemented by computational graph based machine learning frameworks such as PyTorch [7].

We'll assume from now on that our model f uses a neural network architecture. Denote the model trained on the source dataset as f^S. We want to use the knowledge learned when training f^S to improve accuracy on the target dataset. We achieve this through parameter sharing; reusing the pre-trained weights for the source model, we can selectively choose layers of our model to freeze or allow finetuning to the target domain. Note that sometimes we may have a different output shape between source and target domain (e.g. classification with a different number of categories). The solution to this can be to 'chop off' the final fully-connected layer, replacing with a layer of the correct output shape with randomly initialised weights.

When fine-tuning our model to our target datapoints, we may want to enforce the prior that the model should stay as close to the original (source-trained) model as possible. One way to achieve this is through early stopping, halting our training process after a certain number of iterations without any improvement in our validation set. Another method is to regularise by enforcing a prior that the learned parameters should be close to the source model f^S parameters. Assume our final source model f^S has weight set $\{\bar{w}_i\} = \{\bar{w}_i : i \in F\} \cup \{\bar{w}_i : i \notin F\}$, where the set F is the set of parameters that we are fine-tuning. F^C is the set of frozen parameters. We then introduce a L1 regularisation term for the fine-tuned parameters, which, in contrast with L2 regularisation, persuades sparse deviation; few parameters are allowed to deviate from their original value $\bar{w}_i$.

$$\arg\min_{\{w_i\}}\{\mathcal{L}(f^T) + \lambda\Omega(f^T) + \mu\sum_{i\in F}\|\bar{w}_i - w_i\|\} \tag{1}$$

Here, the parameter μ allows us to adjust how much we allow the model f^T to deviate from the source model f^S. How much will we enforce what was learnt from our source domain, versus how much we'll allow for respecialisation.

2.5 Fine-Tuned Embedded Learning

We now demonstrate how we adapt the techniques in the previous section to achieve transfer learning on a set of embedded tasks, as in definition 2.5.

We firstly train a model f^S on our source dataset $(\mathcal{D}_S, \mathcal{T}_S)$ with feature space $\mathcal{X}$. Now, noting that $\mathcal{X}^\star = \mathcal{X}_1^\star \times \cdots \times \mathcal{X}_k^\star$ with each $\mathcal{X}_i^\star = \mathcal{X}$, we make k copies of our source model f^S, which we call $\{f^{(i)}\}_{i=1}^k$. These k copies of f^S have the same input shape as is given by $\mathcal{X}^\star$. We also need to reshape to get the correct output size via an architecture change. If the final layer of f^S is a fully-connected (feedforward) layer with shape (input, output) $= (L, c)$, we chop off this final layer of each $f^{(i)}$, replacing with a single fully-connected layer of shape (kL, c) to ensemble the outputs from each constituent network's deep layers. This creates our transfer model f^T, as demonstrated in figure 1. This new layer can be randomly initialised, or it could be initialised with $1/k$ of the weights in the fully-connected layer of f^S. In our testing, we use random initialisation, based on the hypothesis that the final deep layer of $f^{(i)}$ outputs the 'useful features' learnt from the source domain, but that we may want our model to use these features in a different way with our larger domain $\mathcal{X}^\star$.

If we have the saved weights from our source model f^S as $\bar{w}_j$, and each constituent $f^{(i)}$ of our target model has weights $w_j^{(i)}$, then our objective becomes,

$$\arg\min_{\{w_j^i\}}\{\mathcal{L}(f^T) + \lambda\Omega(f^T) + \mu\sum_{\substack{i\in\{1,\ldots,k\}\\ j\in F}}\|\bar{w}_j - w_j^{(i)}\|\}$$

Training on this objective function allows us to harness the learned properties from the slice domain $\mathcal{X}$, and transfer them to the larger encompassing domain $\mathcal{X}^\star$.

Let's consider what our model distribution looks like immediately after we have done the copying step (before any fine-tuning has been made). We use μ to define a probability density; $\mu_{D_i}(\mathbf{z}|\mathbf{x})$ is the density of values $\mathbf{z} = (z_1, \ldots, z_m)$ outputted from the penultimate layer of model i given inputs to the network $\mathbf{x} = (x_1, \ldots, x_n)$. We write $A = \mathbb{R}^m$ to be the output space of the penultimate layer of our source model. If we our working with a classification problem, our source model outputs the probabilites per class,

$$f^S(X) = \{P^S(y|X) : y \in \mathcal{Y}\}$$

We can then write the analogous distribution for the target model, and since the channels do not mix until the final (fully-connected) layer, we can split our conditional distribution $P(y_i|X^\star)$ into each channel's contribution:

$$\begin{aligned} f^T(X^\star) &= \left\{P^T(y|X^\star) : y \in \mathcal{Y}\right\} \\ &= \left\{\int_{(\mathbf{z}_1,\ldots,\mathbf{z}_k)\in A\times\cdots\times A} P^T_{FC}(y|\mathbf{z}_1,\ldots,\mathbf{z}_k)\mu_{D_{1,\ldots,k}}(\mathbf{z}_1,\ldots,\mathbf{z}_k|X_1^\star,\ldots,X_k^\star)d\mathbf{z} : y \in \mathcal{Y}\right\} \end{aligned} \tag{2}$$

We have separated by output $\mathbf{z}_j$ of each $f^{(i)}$ using chain rule of probabilities. However our model architecture has no connections between channels until the FC layer, so each (Z_i, X_i) is independent of each other pair (Z_j, X_j); we can split by channel:

$$= \left\{\int_{(\mathbf{z}_1,\ldots,\mathbf{z}_k)\in A\times\cdots\times A} \underbrace{P^T_{FC}(y|\mathbf{z}_1,\ldots,\mathbf{z}_k)}_{\text{FC Layer}}\underbrace{\mu_{D_1}(\mathbf{z}_1|X_1^\star)\ldots\mu_{D_k}(\mathbf{z}_k|X_k^\star)}_{k\text{ channels}}d\mathbf{z} : y \in \mathcal{Y}\right\} \tag{3}$$

This demonstrates how our model uses each channel independently, finally feeding these individual 'votes' into a fully-connected layer that amalgamates the prediction from each channel.
At the point of copying (before any fine-tuning occurs), we have each μ_{D_i} is equal. If we assume our data is a k-times copy $\mathbf{X}^\star = (X_1^\star, \ldots, X_1^\star)$, then our independence assumption for equation (3) is no longer true. Going back to equation (2), we find:

$$= \left\{\int_{\mathbf{z}_1\in A} P^T_{FC}(y|\mathbf{z}_1,\ldots,\mathbf{z}_1)\mu_{D_1}(\mathbf{z}_1|X_1^\star)d\mathbf{z} : y \in \mathcal{Y}\right\} \tag{4}$$

If we use the $1/k$ weight initialisation for P^T_{FC} described above, then $w_{ij}^{(l)} = \frac{1}{k}\bar{w}_{ij}$ in the final layer. Each channel output is equal, $(\mathbf{z}_l)_j = z_j$. Our fully-connected layer becomes,

$$\sum_{l=1}^{k}\sum_{j=1}^{m} w_{ij}^{(l)}(\mathbf{z}_l)_j = \sum_{l=1}^{k}\sum_{j=1}^{m}\frac{1}{k}\bar{w}_{ij}z_j = \sum_{j=1}^{m}\bar{w}_{ij}z_j$$

Hence $P^T_{FC}(y|\mathbf{z}_1,\ldots,\mathbf{z}_1) = P^S_{FC}(y|\mathbf{z}_1)$ (regardless of activation function used). So (4) becomes,

$$\begin{aligned} &= \left\{\int_{\mathbf{z}_1\in A} P^S_{FC}(y|\mathbf{z}_1)\mu_{D_1}(\mathbf{z}_1|X_1^\star)d\mathbf{z} : y \in \mathcal{Y}\right\} \\ &= \left\{P^S(y|X_1^\star) : y \in \mathcal{Y}\right\} \end{aligned}$$

Hence immediately after we copy parameters and under the assumption that our channels are a k-times copy of $X_i^\star$, our model f^T acts exactly the same as our source model f^S.

2.6 Cross-channel Connections

In our setting above, our only architectural change in the transfer is the final (fully-connected) layer. The separate channels are purely taking a 'vote' on their belief based on the information from their channel data $\mathcal{X}_i \subset \mathcal{X}^\star$. But what if there are more complicated interactions between the channels

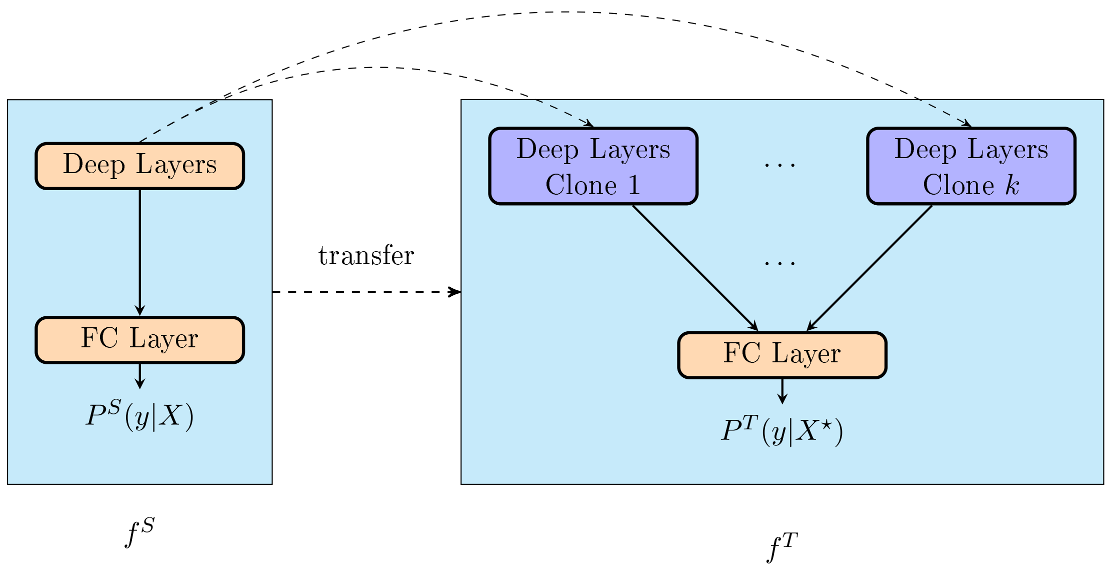

Figure 1: Transfer from embedded domain to larger domain. Orange represents randomly initialised parameters, while blue represents copied parameters

than each channel providing a vote in the feedforward layer? Our current setting is just equivalent to a logistic regression on the outputs of each channel's deep layers. To model for more complicated interactions, we need to allow interactions between channels earlier in our model.

We take inspiration from *adapters* described in [8]; we hope to take two intermediate layers of the deep network $l^{(j)}$ and $l^{(j+1)}$ and allow for cross-channel communication in the stage between these two layers.

Assume the output of layer $l^{(j)}$ has dimension m_j. Then, the total shape of the output at layer j (across channels) is $k \times m_j$ We create a new fully-connected layer with dimension $km_j \times km_j$ (square), and use this fully-connected layer immediately after the output of layer $l^{(j)}$. Crucially, we initialise the weights of this new layer with the identity matrix $\mathbb{I}_{km_j}$, so that without any training, this layer causes no change the the behaviour of the network; see figure 2 for a depiction of this.

Adding these extra connections is at the expense of increased parameter count of our model, so higher capacity for overfit. During training, we apply a L1 norm regularisation of this weight against the identity matrix $\mathbb{I}_{km_j}$, where the amount of regularisation applied controls how much we allow cross-channel connections.

3 ECG Classification

3.1 Background

We will now explore an application of the methods we have described above. The problem of electrocardiagram classiciation is a natural task to apply machine learning to; a patient is admitted to hospital, and they get connected to an ECG machine. These ECG machines produce a large amount of data, which a doctor will then manually interpret to confirm whether they have a regular heartbeat, or if they contain any abnormalities that need to be managed. This is the process we hope to automate; to give guidance to doctors we can increase efficiency of hospitals and reduce wait-times.

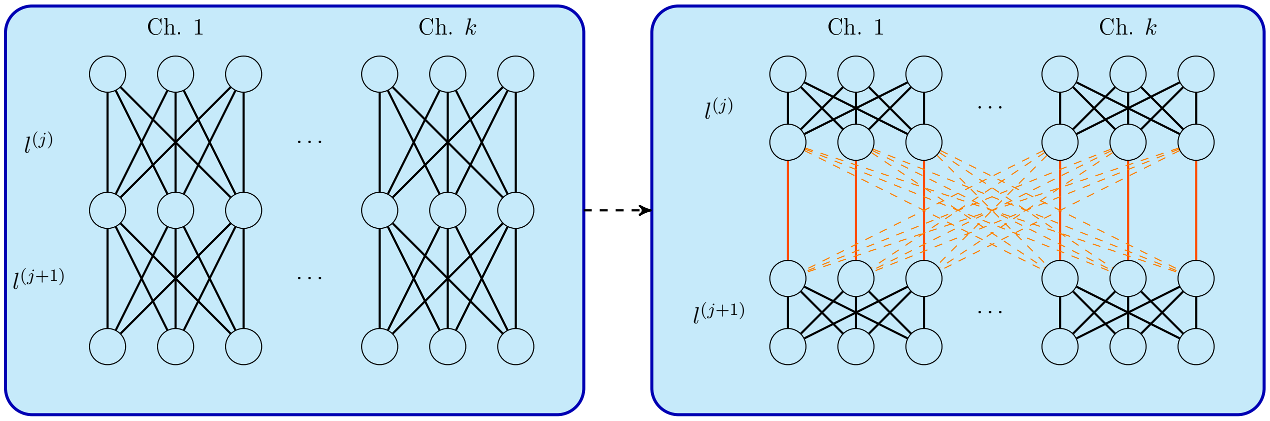

Figure 2: Left: No cross-channel connections between layers j and $j+1$. Right: Allow cross-channel connections between layers j and $j+1$ via inserted layer. Solid red connections are initialised to 1; dashed connections are initialised to 0, but are allowed to be finetuned away from 0.

One such abnormality is *cardiac arrhythmia*, a condition that is estimated to afflict 2 million people in the UK. Cardiac arrhythmia is characterised by an 'irregular' heartbeat. In this work, we train our models to identify between three categories; normal rhythm, atrial fibrillation, and 'other' irregularity. The goal is accurately identifying cardiac conditions to assist doctors, which could help to speed up diagnosis procedures.

There are two scenarios for ECG identification; a single-dimensional ECG timeseries, obtained from a single electrode pair on a patient's body, and 12-lead ECG data, the recordings of 12 electrodes across the body recording simultaneously. This leads to two related classification problems; one with 1-dimensional timeseries, and the related 12-lead problem with 12 stacked ECG traces.

There exist public databases (primarily within PhysioNet[9]) containing labelled ECG traces, which have been cited by much of the existing work on this field. The MIT-BIH Arrhythmia Database[10] contains 48 half-hour single recordings of 47 different patients, recorded between 1975 and 1979. In 2017, a challenge dataset[11] was created for atrial fibrillation classification from single-lead ECG recordings; this contains 8528 recordings of between 9 and 61 seconds each. In 2020, a 12-lead challenge dataset[12] was created drawing together 5 different data sources. This dataset has detailed labelling of the cardiac abnormality observed in each instance.

3.2 Prior Work

Deep learning as a method for ECG classification has been explored by multiple sources. Two recent papers are *Deep Bidirectional LSTM for ECG Classification*[13], and *Automated diagnosis of arrhythmia using CNN and LSTM techniques*[14], which we will build upon in this essay. Both papers use LSTM cells for classifying a short segment of single-lead ECG readings. I will discuss the approach they took, as well as some of the shortcomings I believe their methods to have, and I will discuss in later sections how I approached this differently. My source model for single-lead identification will be based on the architecture detailed in these papers.

Deep Bidirectional LSTM for ECG Classification takes each datapoint to be a single heartbeat, with one instance in our dataset having length 360, corresponding to a second of recording at 360Hz. Their architecture first uses a discrete wavelet transform[15] to separate the signal into *detail bands*, distinct timeseries separated by characteristic frequency. The constituent timeseries are then fed into two layers of LSTM; they test both unidirectional and bidirectional LSTM layers. They show that their model performs better when given the multiple detail bands, compared with using the raw ECG trace as input.

Automated diagnosis of arrhythmia using CNN and LSTM techniques splits longer ECG recordings into 5-second segments. Their architecture uses 3 layers of convolution and max-pooling, followed by a layer of unidirectional LSTM cells, and three fully-connected feedforward layers.

3.3 Dataset Differences

The papers referenced above both use only the MIT-BIH Arrhythmia Databse in their experiments. The MIT-BIH database contains half-hour-long ECG recordings from 47 distinct patients. These papers first split the 47 recordings into shorter segments (individual heartbeats or 5s intervals). They then perform their train-test split by random sampling these individual segments. Neither of the papers make reference to stratifying across patient.

We replicated the models in these papers, using randomised sampling, and were able to reach a similar accuracy of 90%. However by switching the process to stratified sampling by patient, performance dropped drastically to 60%. Our prior belief was that a person's heartbeat does not change drastically in distribution across time; but two unique patients, even with the same diagnosis, will have very varying heartbeats. In section 5, we use KL-Divergence to further support our claim that one patient's heartbeat is highly self-correlated. As a result, using the MIT-BIH dataset stratified by patient has in effect only 47 independent datapoints. Because of this, I made the choice to use the Challenge 2017 Dataset[11], as this contains 8528 short single-lead recording that have been recorded from independent patients.

Automated Arrhythmia Classification based on CNN and LSTM[14] does remedy some of the issues raised regarding dataset, by performing an independent validation stage on the AFDB and NSRDB datasets. However, I still believe the 2017 Challenge Dataset to be a better fit for my study, due to the independence of instances within the dataset. During testing, we typically had much lower accuracies than reported in the papers based on the MIT-BIH Dataset. However, by considering other studies based on the Challenge 2017 dataset such as *Limam et. al.*(2017)[16], I found reported performance in these papers comparable to what my experiments achieved.

4 Transfer Learning ECG Experiments

4.1 Baseline Model

As mentioned in section 3.1, we have two related domains; single-lead feature space $\mathcal{X}$, and 12-lead space $\mathcal{X}^\star$. Viewing a single 'image' of an ECG trace as one feature, our 12-lead domain $\mathcal{X}^\star$ is 12-dimensional. This is an example of an embedded domain by definition 2.4 with $k = 12$, where we hope to exploit a relationship between the single lead joint distribution $P(X, Y)$ and the slice distribution $P_i^\star(X_i^\star, Y)$ of the larger domain $\mathcal{X}^\star$. We will first train a baseline model on the single-lead scenario, and then transfer this to the 12-lead domain.

Our source dataset taken from the PhysioNet 2017 Challenge[11] provides us with 8528 distinct ECG recordings from different patients, with recording length between 10 and 60 seconds. We hope to classify these into three categories: normal rhythm (N), atrial fibrillation (A), and other rhythm (O). The distribution of our dataset across these categories is shown in table 1. Since our dataset is imbalanced across the three categories, we use a weighted crossentropy loss, and F1 score for measuring prediction accuracy.

ECG recordings in this dataset are sampled at 300Hz. We downsample to 150Hz by using one pass of the db6 discrete wavelet transform, denoising to remove imperfections created by the ECG

Type	# Recordings	Mean Length (s)
N	5154	31.9
A	771	31.6
O	2557	34.1

Type	# Recordings	Mean Length (s)
N	32813	11.5
A	1106	15.0
O	9182	16.9

Table 1: Left: Distribution of datapoints across categories in challenge (single-lead) 2017 dataset Right: Distribution of datapoints across categories in challenge 2020 (12-lead) dataset.

recording equipment. During each training epoch, we randomly sample a length-1000 window from each recording, equivalent to 6.7 seconds of recording. This random subsampling serves as a method of data augmentation, helping to reduce overfit in a relatively small dataset. It also ensures that all our datapoints are the same length, allowing for simple batch vectorisation. We also apply z-score normalisation - ensuring that the ECG signals are normalised to mean 0, standard deviation 1.

The architecture we use is a combination of the architectures seen in the previous work[13][17], four layers of convolution leading into two layers of bidirectional LSTM. The full layer composition used can be seen in table 2.

Layer Type	**Activation**	**Output Shape**	**Kernel**	**Stride**	**# Parameters**	**Dropout**
Conv1d	ReLU	8×981	20	1	168	N/A
MaxPool1d	None	8×489	5	2	0	0.1
Conv1d	ReLU	16×478	12	1	1552	N/A
MaxPool1d	None	16×237	5	2	0	0.1
Conv1d	ReLU	32×230	8	1	4128	N/A
MaxPool1d	None	32×113	5	2	0	0.1
Conv1d	ReLU	32×110	4	1	4128	N/A
MaxPool1d	None	32×53	5	2	0	0.1
LSTM (32)	Tanh	32×53	N/A	N/A	6272	0.2
LSTM (32)	Tanh	32×53	N/A	N/A	6272	0.2
Linear	Sigmoid	3	N/A	N/A	132	0.0

Table 2: Single Lead Model Architecture

For evaluation, we train our model with 5-fold cross-validation, but when training our baseline model that we will use in transfer, we train over the entire dataset. After 200 epochs of training, our model achieves a mean F1 score of 70.9% on the validation set. Observe our train-test plot in figure 3. The higher F1 score in the test set is attributed to the dropout we apply on the train set.

4.2 Embedded Transfer

Using our trained 'baseline' model for classifying single-lead ECG, we hope that this model has learnt useful representations of ECGs, knowledge that we can then transfer to the domain of 12-lead classification. We utilise parameter sharing from the source- to target-domain architecture by duplicating the baseline model weights 12 times. We then modify the architecture with randomly initialised weights to combine each constituent model together.

We can also view this as a form of greedy learning, as described in section 8.7.4 of *Goodfellow et. al.*[18]. We are picking a sensible initialisation for the 12-lead problem that is our 'best guess' of starting point, given the single-lead data we have available; training each channel independently first, before finetuning on the whole (12-lead) dataset.

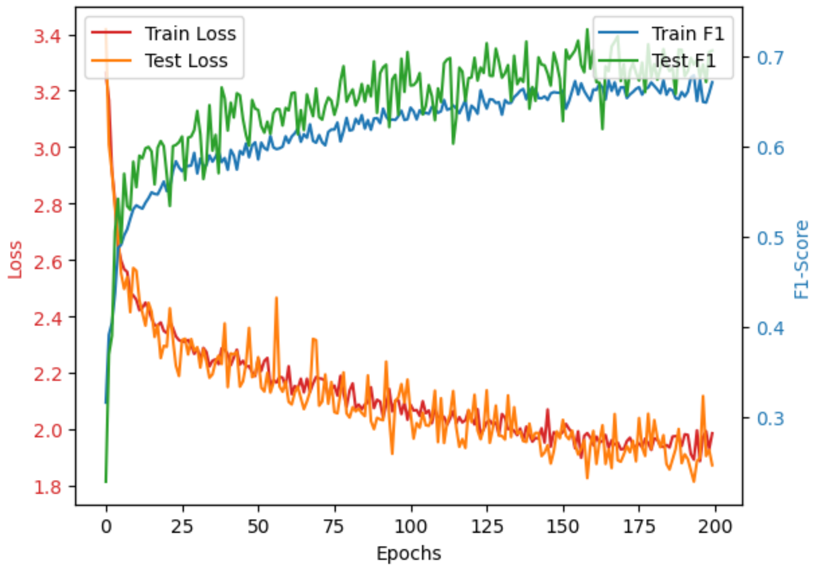

Figure 3: Train-test graph for single-lead (source domain) model

The target dataset we use is the PhysioNet Challenge 2020 [12], which contains approximately 40k 12-lead recordings. This data is highly imbalanced, similar to the Challenge 2017 dataset. Since we don't need large amounts of data for our transfer stage, we balance our dataset by trimming the oversized categories. The dataset we create has 1106 datapoints from each of the 3 categories.

Although we do have an abundance of data for the 12-lead problem, we want to test the performance when we have limited data in the target domain. Can embedded transfer learning, with limited data in the target domain, generalise better than if we were to train a model from scratch on the target domain? This is to simulate a more complex task where we are able to train a model on the source domain, but it is infeasible to train a model from scratch in the target domain. This could be for a variety of reasons; data unavailability, or computational resource constraints. For example, the setting of image-to-video transfer, as described in section 6 on further work, may be infeasible to train a model from scratch.

To simulate data scarcity, we use 10-fold cross validation with leave-9-out; we train on 10% of the data and evaluate on the remaining 90%. We provide two control experiments with randomly initialised weights using our first target-domain architecture (described below in section 4.2.1):

- Training on limited data, in the same way as our transfer training. We only use 10% of the data to train, and the remaining 90% to test, with cross-validation.
- Training on all the data; regular 10-fold cross-validation, where we train on 9 folds and test on the remaining one.

These provide lower- and upper-bounds of the performance of our transfer learning model. Our goal is a transfer learning model that gets as close accuracy to the second control experiment as possible.

4.2.1 Finetuned Fully-Connected Model

As detailed in section 2.5, we duplicate our baseline model 12 times and replace the 12 (32×3) fully-connected layers with a single (384×3) fully-connected layer, allowing for the separate channels to form a 'vote' on the classification result. We punish the L1 norm of how much each finetuned parameter deviates from the baseline model parameter set, as described in equation (1). This is to enforce the prior belief that the source- and target-domains should be highly related, a method to reduce overfit to the limited target dataset.

With this setup, we were able to achieve an F1 Score comparable to the control model trained on a large amount of data. Note in figure 4 how the finetuned model 'snaps' to high accuracy in just a single epoch.

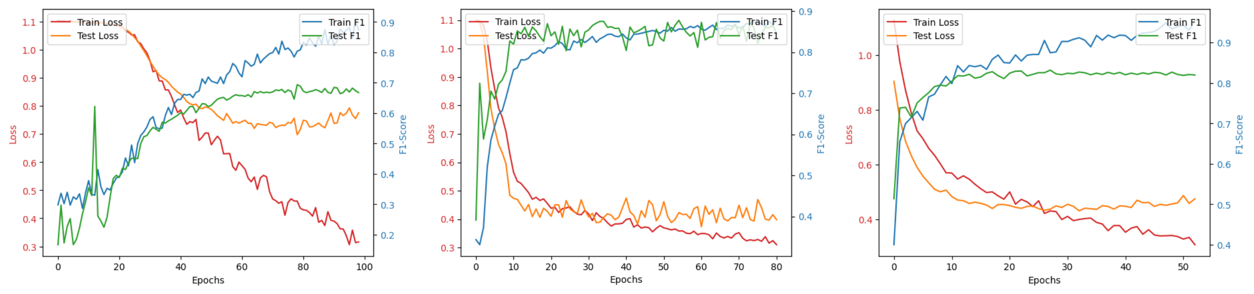

Figure 4: *(left)* control model trained on 10% of the data *(center)* control model trained on 90% of the data *(right)* finetuned transfer model trained on 10% of the data. Note that an 'epoch' is not a consistent measure of training iterations, as the training set varies in size between tests.

4.2.2 Cross-Channel Model

Now, we will explore whether there are relationships that exist within the 12-channel models, deeper than what can be modelled with an indepenedent 'voting' model.

Our models will employ cross-channel connections as described in section 2.6. By adding a stronger regularisation term to the adapters than the rest of the model, we ensure that cross-channel connections are only formed when necessary.

We then run our experiments with two different models:

- Cross-channel connections allowed between the two layers of LSTM. Final layer still randomly initialised as above.
- Cross-channel connections allowed between the second and third convolution layers. Final layers still randomly initialised as above.

We can see in table 3 that our more complex transfer models are not improving from the more complex interactions allowed; instead, they are overfitting due to overparameterisation.

	F1 Score - Train	F1 Score - Test
Control, train set 10%	90.4%	70.0%
Control, train set 90%	86.4%	83.6%
Transfer only FC layer (vote model)	90.0%	81.7%
Transfer w/ cross-channel LSTM1->LSTM2	93.3%	79.4%
Transfer w/ cross-channel CNN2->CNN3	89.7%	79.1%

Table 3: F1 score comparing control experiments with finetuned transfer model

4.3 Semi-synthetic Data

In our above tests with the 12-lead data from the 2020 Challenge dataset, our cross-channel models did not yield any significant increase in performance versus the basic finetuned model. Are there any scenarios where these cross-channel connections can improve performance? For this, we will explore semi-synthetic data using our existing 12-lead dataset.

4.3.1 Perlin Noise

Perlin Noise was initially introduced by Ken Perlin in 1985 [19] as a method for generating realistic-looking image textures. It has been since become a standard algorithm for generating 'realistic' randomness, with applications such as terrain generation and (pre-GAN) image creation. We use one-dimensional perlin noise to augment our 12-dimensional datapoints for use in our transfer model. Perlin noise is the suitable choice for adding randomness to our ECG datapoints, due to it being smooth and 'realistic'. See an example of Perlin noise generation in 1 and 2 dimensions in figure 5.

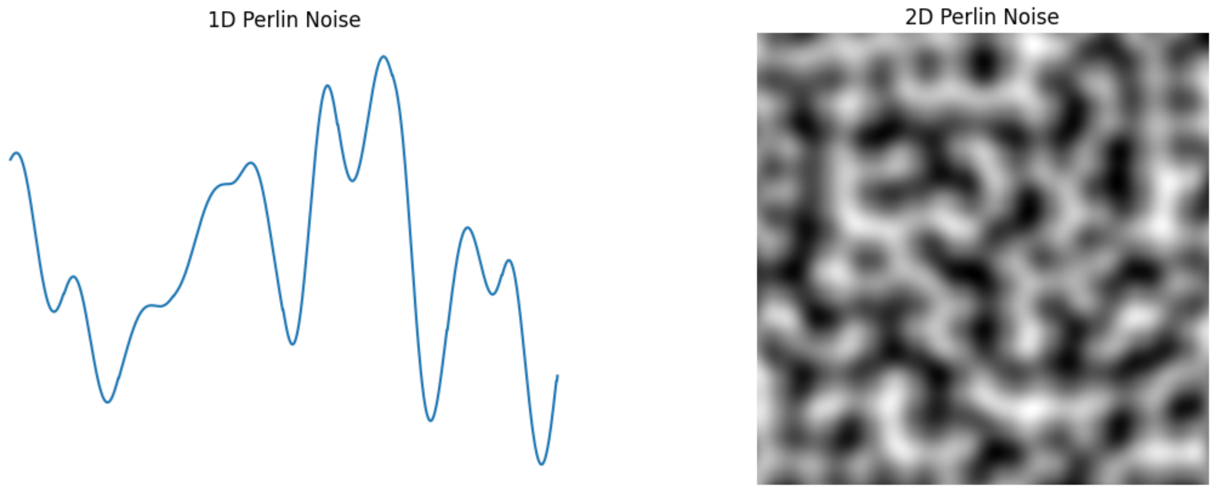

Figure 5: Perlin noise in 1 and 2 dimensions

4.3.2 Linear Dependence Augmentation

We want to augment our 12-lead dataset with a noise component on each channel. Instead of each channel being independent, we want dependencies between the channels; to synthetically create this, we will have a *constant* linear dependence between each channel. We draw a $k \times m$ matrix M_{ij}; each element drawn independently from $U(-a, a)$ distribution, where a is the scale factor for how much noise we choose to apply. This matrix is kept constant throughout training. Then, each time we use a training example, we generate $m < k$ indepenedent 1-dimensional sources of noise $v_i, i = 1 \ldots m$. The noise we add to the ith channel of our 12-lead ECG trace is $M_{ij}v_j$

The goal is that our model will be able to learn the dependencies between the noise in each channel, and learn to classify the ECG traces correctly regardless of the randomness added from noise. Observe the results of this test in table 4. Here, we use the 'vote model' transfer as our control test, and have shown that by allowing cross-channel connections performance is improved.

	F1 Score - Train	F1 Score - Test
Transfer only FC layer (vote model)	82.4%	71.0%
Transfer w/ cross-channel LSTM1->LSTM2	85.8%	**73.7%**
Transfer w/ cross-channel CNN2->CNN3	83.6%	71.4%

Table 4: F1 score comparing control experiments with finetuned transfer model

5 Distribution Difference via Autoencoders

5.1 Distribution Difference - Introduction

In this section, we try to quantify the distribution difference across the MIT-BIH dataset [10], to assess whether small sections of ECG from the same patient can be treated as independent. We want to compare patients A and B with the same diagnosis with the variation within the half-hour recording of patient A. Does a person's heartbeat now differ from what their heartbeat looks like in 10 minutes? Our claim is that there is much stronger variation between patients than exists within one patient's half-hour recording. This is heuristically supported by comparing two person's heartbeats over time visually, as can be seen in figure 6. If this is the case, it would imply that the MIT-BIH dataset is not suitable for training a deep learning model, as it has only 47 independent data sources, leading to our model being heavily overparametrised.

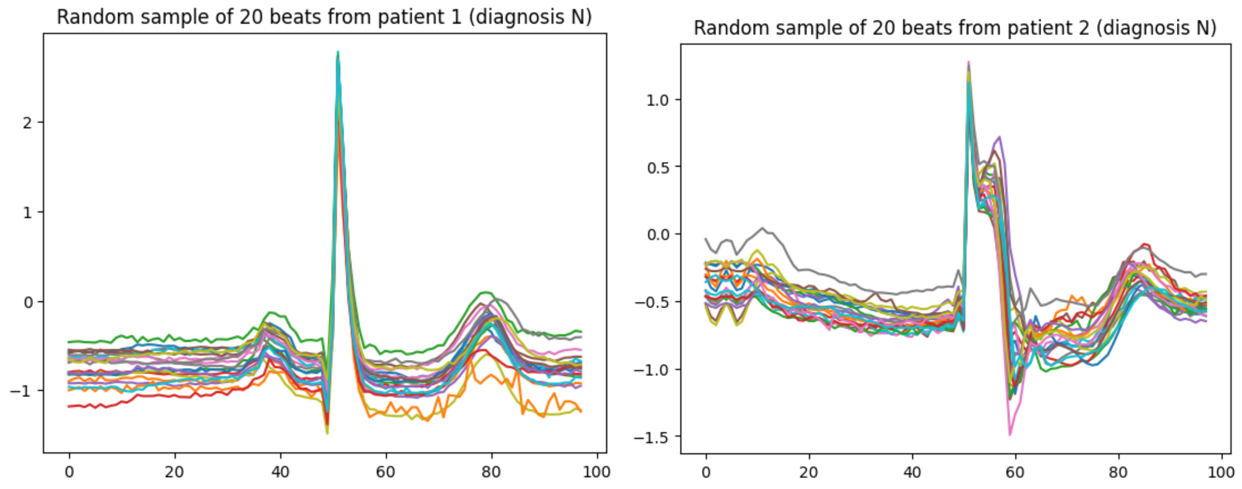

Figure 6: Random sample of 20 heartbeats from two patients with diagnosis 'normal'

We will use an information-theoretic approach to measuring the *distribution difference* of two datasets. The KL divergence of two probability distributions $P(x), Q(x)$ is defined by,

$$D(P||Q) \triangleq \int_{x\in\mathcal{X}} P(x) \log \frac{P(x)}{Q(x)} dx \tag{5}$$

This metric measures similarity between probability distributions, where $D(P||Q) \geq 0$, and $D(P||Q) = 0 \iff P = Q$. Noting that $D(P||Q) = -D(Q||P)$, we define symmetrised KL divergence,

$$KL(P,Q) \triangleq D(P||Q) + D(Q||P)$$

This is the metric we will use for quantifying how similar/different two datasets are.

However, since out datapoints are timeseries, we cannot use KL divergence directly. We have interactions and symmetries in the distribution that won't be accurately captured by information-theoretic methods. For example, in our ECG data, we have assume some smoothness of the time-series: if a heartbeat is $\mathbf{x} = (x_1, \dots, x_t) \in \mathcal{X}$, then we have strong (auto)correlation between x_i and x_{i+1}. For this reason we have a translational invariance; we want to view a translation up/down as only a small movement on the data manifold. Another consideration is the invariance in moving through time: if we have two similar datapoints $x^{(a)}, x^{(b)}$, we may have a feature that is characterised by the point $x_i^{(a)}$ in one instance, and $x_{i+1}^{(b)}$ in the other instance. We want to recognise these recordings as 'similar', but by viewing each trace as $\mathbf{x} = (x_1, \dots, x_t)$ the KL divergence will not capture these relationships.

This leads us to the conclusion that we need to apply some transformation to our timeseries; an encoding that learns useful representations of the features in heartbeats, without a time dimension. The solution that we adopt is a temporal autoencoder; a model to encode our sequence as a lower-dimensional object without a time dimension.

5.2 Autoencoders

Theorem 5.1 (Manifold Hypothesis)**.** For most 'naturally occuring' high-dimensional datasets, the set of 'likely' elements lies on a lower dimensional manifold [8, 20].

Autoencoders were initially introduced by M. Kramer as an abstraction of principal component analysis allowing nonlinear interactions[21]. Autoencoders aim to learn a latent representation $\mathbf{z} \in \mathcal{Z}$ of the lower-dimension manifold $\mathcal{M} \subset \mathcal{X}$ in which the the likely elements sit, where $\dim \mathcal{Z} < \dim \mathcal{X}$. We achieve this using a model with a *bottleneck*; a layer of the network with dimension $\dim \mathcal{Z} < \dim \mathcal{X}$. We train this model to minimise *reconstruction error* on instances of $\mathbf{x}$. By minimising the mean squared error between the original instance $\mathbf{x}$ and the reconstructed output $\mathbf{x}'$,

$$\mathcal{L} = \frac{1}{n} \sum_{i=1}^{n} \|\mathbf{x}_i - \mathbf{x}_i'\|^2,$$

We have taught our model to replicate training examples, and in the process we have produced a lower-dimensional encoding $\mathbf{z}$ for our datapoints in $\mathcal{X}$.

In the case where our encoders and decoders are single-layer networks (so each element y_i in our encoding is produced by a linear model), it can be shown that the optimal solution is for the encoding to choose the m eigenvectors of the covariance matrix with the heighest eigenvalues (the most variance). This is equivalent to principal component analysis, proven in *Baldi et. al.*(1989)[22]. This demonstrates how our autoencoder learns the task of pulling out the 'most important' factors. So a multi-layer autoencoder can be seen as a deep learning abstraction of PCA, where the model allows for a more complex, nonlinear encoding function.

We are dealing with sequential data, so instead of purely feedforward layers, we wish to use a network with a temporal architecture. The predecessor to the LSTM was the RNN cell; allowing for passing a 'remembered' value along the time dimension. However, the RNN suffers the issue of gradient vanishing or explosion after many time steps. The LSTM alleviates the issues of the RNN this because it allows the flow of information to be *gated*; the information will be only remembered, forgotten, or released, when it is given a signal to do so.

We took inspiration from the paper on ECG classification by *Hou et. al.*(2020)[23], which uses a LSTM autoencoder to encode ECG traces, which is then classified by a SVM.

In our setting, we take each heartbeat to be an individual datapoint, and each segment is the same length - a second of recording. Our encoder model has two layers of LSTM, with 16 LSTM cells per layer. We take the hidden state from our second LSTM layer (with length 32) to be our encoding, which we hope has 'remembered' useful features of the sequence through time. We can then use this hidden state as the initialisation for the decoder LSTM, and provide the decoder LSTM cells with all zeros as input.

Since this 32-dimensional encoding is the pure output from the LSTM hidden layers, we might think there may be some redundancy in these 32 dimensions. To test this, we place a three-layer feedforward autoencoder inside the LSTM autoencoder; encoding the representation **z** created by the autoencoder. This reduces our embedding dimension from 32 to 16. We also speed up our training process by using greedy learning, defined in section section 8.7.4 of *Goodfellow et. al.*[18]. We first train the autoencoder to create 32-dimensional representations $f : \mathbf{x}^{(i)} \mapsto \mathbf{z}^{(i)}$. Then, train a feedforward autoencoder to encode these representations in 16-dimensions $g : \mathbf{z}^{(i)} \mapsto \bar{\mathbf{z}}^{(i)}$. Finally, we finetune the combined model $g \circ f : \mathbf{x}^{(i)} \mapsto \bar{\mathbf{z}}^{(i)}$. We are able to achieve very similar accuracy with a 16-dimensional encoding as with the original 32 dimensions, showing that the 'most likely' datapoints lie on a manifold with dimension ≤ 16 according to the manifold hypothesis stated above. See figure 9 for an example of a heartbeat replicated by our encoding.

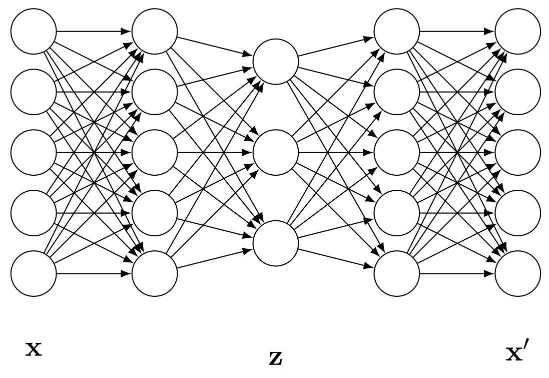

Figure 7: Fully-connected autoencoder

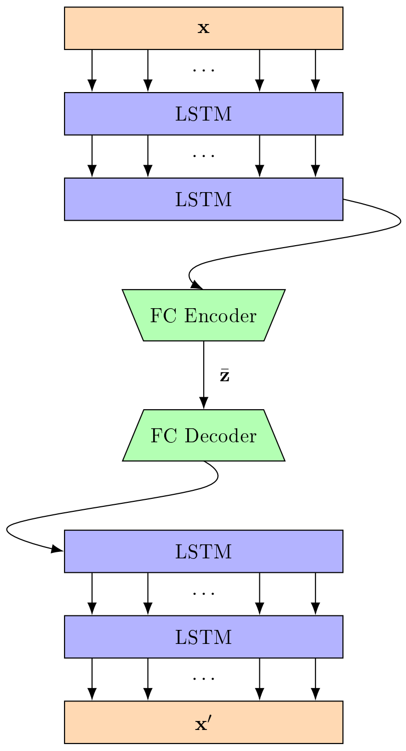

Figure 8: LSTM Autoencoder Architecture

5.3 Distribution Difference

We now have a 16-dimensional encoding for every heartbeat in the MIT-BIH dataset, each heartbeat labelled by patient and diagnosis. To quantify the distribution difference between heartbeats from the same patient, versus heartbeats from different patients, we take two patients A and B that both had the majority of their heartbeats classified as 'normal'. We can compare the two patients heuristically by plotting the distributions of individual variables, as seen in figure 10, to see that the distributions are clearly well-defined for each patient, but differ between patient.

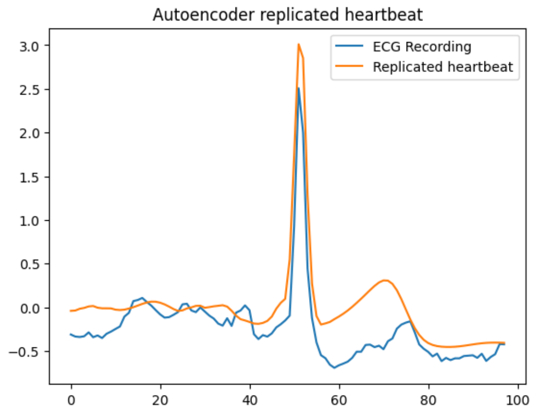

Figure 9: Heartbeat replicated by autoencoder model

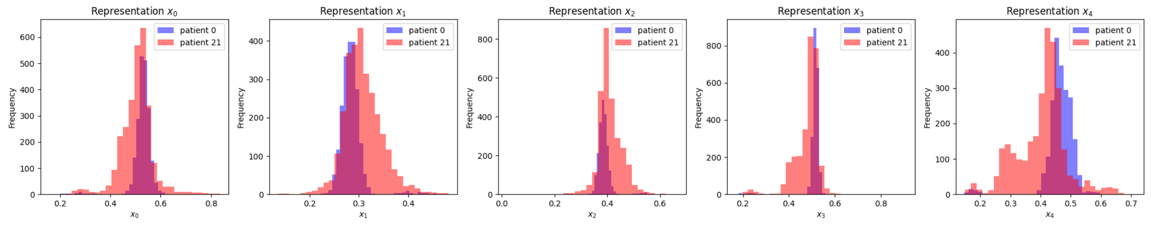

Figure 10: Distribution difference of encoded beat, between patient 0 and patient 21

To quantify this distribution difference, we now return to KL Divergence, as defined in equation (5). We cannot use KL Divergence on our empirical observations; first we need an estimate of the density function. For this, we will use Kernel Density Estimation, a method for producing an estimate $p(x|D)$ for the probability density given a set of observations D.

Definition 5.1 (Density Kernel)**.** A Density Kernel is a function $\mathcal{K} : \mathbb{R} \to \mathbb{R}_+$ such that,

- $\int \mathcal{K}(x)dx = 1$ (probability density)
- $\mathcal{K}(x) = \mathcal{K}(-x)$

Two straightforward examples of density kernels are:

- the Gaussian kernel,

$$\mathcal{K}(x) = \frac{1}{(2\pi)^{\frac{1}{2}}} e^{-\frac{x^2}{2}}$$

- the boxcar kernel,

$$\mathcal{K}(x) = \frac{1}{2} 1(|x| \leq 1)$$

Lemma 5.2. For $\mathcal{K}$ a density kernel,

$$\int_{\mathbb{R}} x\mathcal{K}(x - x_n)dx = x_n$$

Definition 5.2 (Scaled Density Kernel)**.**

$$\mathcal{K}_h(x) \triangleq \frac{1}{h}\mathcal{K}(\frac{x}{h})$$

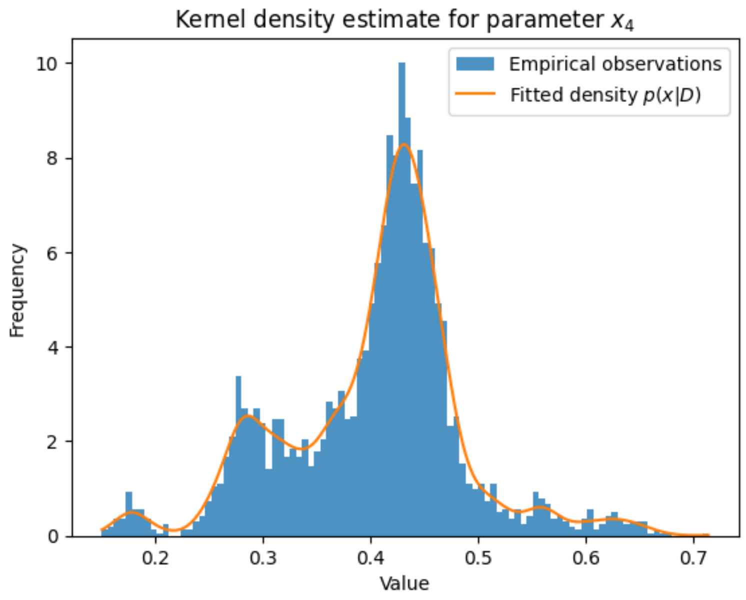

Figure 11: Kernel density estimate fitted to our observations

This allows us to control the scaling of our window, while ensuring that $\int \mathcal{K}_h(x)dx = 1$.

Then the Kernel Density Estimator for a dataset D, first defined by E. Parzen(1962)[24], is defined as,

$$p(\mathbf{x}|D) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{K}_h(\mathbf{x} - \mathbf{x}_n)$$

Observe figure 11 for how the KDE fits a distribution function to our dataset.

So, to quantify the distribution difference between patient A and patient B, we will take the symmetrised divergence of the two patients, $KL(D^{(A)}, D^{(B)})$. We need to be able to compare this with how much a single patient's ECG distribution can vary. For this, we split patient A's dataset in two. We want to be as generous as possible to our testing; to take the split that will maximise the distribution difference among A. If we split by random sampling, we could almost guarantee that the two subsets of A's empirical observations are identically distributed. We want to consider the maximum amount that the distribution of A can vary within the observations we are given. Hence, we split A in time - taking the first half of heartbeats to be our set $D_{(1)}^{(A)}$, and the second half of heartbeats to be $D_{(2)}^{(A)}$, as seen in figure 12.

5.3.1 The Curse of Dimensionality

A common problem in machine learning is tasks become exceedingly difficult when the dimension of data gets high. This problem is due to the number of datapoints in a certain neighborhood as dimension increases; if dimension increases with the same number of datapoints, we get vanishing probability of having a datapoint in a certain neighborhood; the datapoints are *lost to dimensionality*. One place this arises is in density estimation; we need a superlinear number of datapoints to be able to gain an accurate estimate for the probability density.

In 1969, Epanechnikov quantified losses to dimension in density estimation. We define the mean integrated square error as a metric for quantifying accuracy of our estimated density $p(x|D)$ based

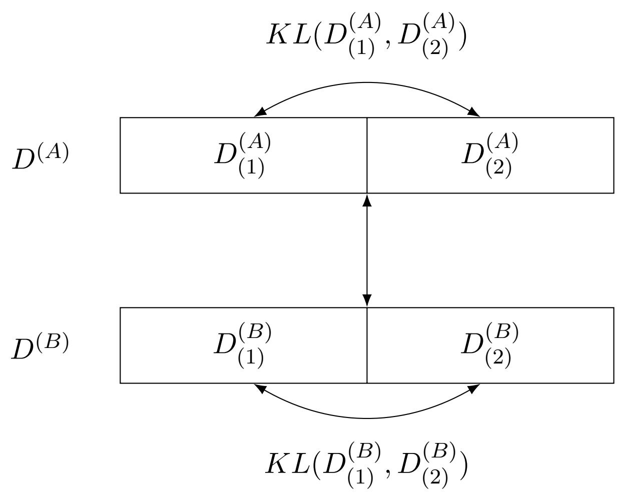

Figure 12: Comparison of distributions between, and within, patients (A) and (B).

on $|D| = n$ observations.

Definition 5.3 (Mean Integrated Square Error)**.** Given the true distribution $p(x)$ and our empirical estimate of distribution $p(x|D)$, we define:

$$\text{MISE}(x) = \mathbb{E} \int (p(x|D) - p(x))^2 dx$$

This is otherwise known as the L^2 risk function.

In the case where both our target distribution $p(x)$ and our kernel $\mathcal{K}(x)$ are multivariate Gaussian $N(0, I_d)$, Worton [25] showed that $\text{MISE}(h, n, d)$ can be calculated explicitly:

$$\hat{M} = (4\pi)^{\frac{d}{2}}\text{MISE} = \frac{1}{nh^d} - \frac{(1+h^2)^{-\frac{d}{2}}}{n} + 1 - 2\left(1 + \frac{h^2}{2}\right)^{-\frac{d}{2}} + \left(1+h^2\right)^{-\frac{d}{2}}. \quad (6)$$

The patient with the most data has approximately $n = 2000$ datapoints in $d = 16$. We seek the value of our kernel parameter h that gives the best estimate of our distribution, so we use a numerical optimiser to find $\min_h \hat{M}(h, n, d) = 0.8$. So, how many datapoints in $\mathbb{R}^1$ is this equivalent to? We can solve equation (6) numerically, finding the maximum n that could give the same value for $\hat{M}$ in $d = 1$ dimensions;

$$\max\{n : \min_h \hat{M}(h, n, 1) = 0.8\} = 0.023$$

This shows out dataset gives equivalent information to $n = 0.023$ datapoints in $\mathbb{R}^1$, completely insufficient for providing an accurate estimate of distribution.

When we were training our autoencoder models, we found $d = 16$ to be the minimum size of the representation before we start losing replication accuracy in our model. Hence we can view all 16 dimensions in our encoding as 'necessary'. Therefore, we assume that there is some notion of independence between the variables. So we instead will just try to quantify the distribution difference in each dimension separately; flattening to each dimension in turn and take average over dimensions.

5.3.2 Empirical Distribution Difference

We are in effect testing the hypothesis 'does a person's heartbeat shift over time, compared with another patient's heartbeat'. In each diagnosis category, we tested each patient against themselves, and every patient against every other patient, averaging over each test. See table 5 for a summary of the divergence results, showing how the distribution difference amongst one patient is considerably lower than the distribution between independent patients. Hence, we can conclude that the dataset of all heartbeats labelled by diagnosis has strong correlations between datapoints from the same patient. We cannot treat this dataset as IID, and since there are only 47 independent subjects, this dataset is not suitable for training large machine learning models.

	N (patients)	$KL(D^{(A)}, D^{(B)})$	$KL(D^{(A)}_{(1)}, D^{(A)}_{(2)})$
N	31	1.70	**0.14**
P	2	0.14	**0.033**
R	4	1.98	**0.20**
L	4	**0.15**	0.24

Table 5: Average divergence between- and within-patient, averaged across diagnosis group. Lower values imply closer distributions.

6 Further Work

The setting of embedded transfer learning for ECG traces is just one application of this paradigm, that has been feasible given the resources and time constraints of this project. Here, we will detail some of the other potential areas that embedded transfer learning could provide an application.

6.1 Speech Enhancement

The problem of speech enhancement relates to taking a recording of speech with imperfections and recovering high-quality audio. 'Imperfections' may refer to; background noise, low quality due to poor equipment, low quality due to compression, environmental factors such as echoes.

Current literature on speech enhancement uses deep denoising autoencoders to attempt to remove these imperfections in audio signals [26]. These models generally rely on domain-specific knowledge; being trained on the specific speaker, or specific noise type, and some works have attempted to combine domain identification into an ensemble model [27], first determining the noise type, and then selecting the model that was trained accordingly.

The hypothesis of applying embedded transfer learning to this task is that to get studio-quality audio recording, a single poor-quality audio recording fundamentally doesn't have enough information to achieve generalised denoising (and instead must have some awareness of noise type, subject's voice etc.). With multiple low-quality recordings from different locations in a room, we may have enough information to denoise speech in a general setting to a higher accuracy than from a single recording.

When we have multiple simultaneous recordings, we may also introduce geometric information into the model, encoding where the microphones and the speaker are in relation to each other. This could provide for better ability to remove echoes from the samples.

6.2 Embedded Transfer Image Classification

Embedded transfer learning can be applied to image classification in multiple ways; If we have an image recognition for single-channel (greyscale) images, we could transfer this to the three-channel scenario, where the extra dimension of colour adds some more information.

We may want to build an image recognition model that can take multiple images of the same object, and have *spatial* awareness. For this, we could use embedded transfer learning where the source domain is a single image classifier. Similarly to the speech enhancement setting explained above, we could encode geometric data (about where each image was taken from) as an additional input, to allow the model to develop an understanding of 3d shape of objects.

We may hope to build a model that takes *video* data as input, given an image classification model. For this, we could use embedded transfer learning, appending a recurrent model to the end of our image model.

7 Conclusion

In this essay, we have introduced and formalised the concept of embedded transfer learning, for suitably related tasks where one domain is a slice of another.

Our experiments were able to show that when data is scarce, applying embedded transfer learning to the ECG classification task is able to achieve considerably higher performance than training a model scratch. However, embedded transfer learning is *not* as strong as training a model from scratch on the target domain given an abundance of data. This is in contrast with some cases of transfer- and meta-learning, where by having previously trained on a related source domain, transferred models able to achieve *higher* performance in the target domain. For example in *Progressive Neural Networks* [28], the multi-layer architecture used achieves even higher performance after transfer; it is compiling knowledge from multiple domains to learn to be even stronger. This leads us to also consider embedded transfer, at least in the ECG setting, as a novel form of weight initialisation (greedy learning).

While ECG classification is not a problem where it is challenging to train a target-domain model from scratch, we hope that the concepts in this essay could transfer over to problems where it may be challenging to build a target-domain model from scratch, such as has been explained in section 6.

References

[1] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He, "A comprehensive survey on transfer learning," *CoRR*, vol. abs/1911.02685, 2019. [Online]. Available: http://arxiv.org/abs/1911.02685

[2] H. Shimodaira, "Improving predictive inference under covariate shift by weighting the log-likelihood function," *Journal of Statistical Planning and Inference*, vol. 90, no. 2, pp. 227–244, 2000. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0378375800001154

[3] K. P. Murphy, *Probabilistic machine learning: Advanced topics.* MIT press, 2023.

[4] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

[5] L. eon Bottou, "Online learning and stochastic approximations," *Online learning in neural networks*, vol. 17, no. 9, p. 142, 1998.

[6] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

[7] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32.* Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[8] K. P. Murphy, *Probabilistic Machine Learning: An introduction.* MIT Press, 2022. [Online]. Available: probml.ai

[9] A. L. Goldberger, L. A. Amaral, L. Glass, J. M. Hausdorff, P. C. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C.-K. Peng, and H. E. Stanley, "Physiobank, physiotoolkit, and physionet: components of a new research resource for complex physiologic signals," *circulation*, vol. 101, no. 23, pp. e215–e220, 2000.

[10] G. B. Moody and R. G. Mark, "The impact of the mit-bih arrhythmia database," *IEEE engineering in medicine and biology magazine*, vol. 20, no. 3, pp. 45–50, 2001.

[11] G. D. Clifford, C. Liu, B. Moody, H. L. Li-wei, I. Silva, Q. Li, A. Johnson, and R. G. Mark, "Af classification from a short single lead ecg recording: The physionet/computing in cardiology challenge 2017," in *2017 Computing in Cardiology (CinC).* IEEE, 2017, pp. 1–4.

[12] E. A. P. Alday, A. Gu, A. J. Shah, C. Robichaux, A.-K. I. Wong, C. Liu, F. Liu, A. B. Rad, A. Elola, S. Seyedi *et al.*, "Classification of 12-lead ecgs: the physionet/computing in cardiology challenge 2020," *Physiological measurement*, vol. 41, no. 12, p. 124003, 2020.

[13] Ö. Yildirim, "A novel wavelet sequence based on deep bidirectional lstm network model for ecg signal classification," *Computers in biology and medicine*, vol. 96, pp. 189–202, 2018.

[14] C. Chen, Z. Hua, R. Zhang, G. Liu, and W. Wen, "Automated arrhythmia classification based on a combination network of cnn and lstm," *Biomedical Signal Processing and Control*, vol. 57, p. 101819, 2020.

[15] S. Mallat, *A wavelet tour of signal processing.* Elsevier, 1999.

[16] M. Limam and F. Precioso, "Atrial fibrillation detection and ecg classification based on convolutional recurrent neural network," in *2017 Computing in Cardiology (CinC).* IEEE, 2017, pp. 1–4.

[17] S. L. Oh, E. Y. Ng, R. San Tan, and U. R. Acharya, "Automated diagnosis of arrhythmia using combination of cnn and lstm techniques with variable length heart beats," *Computers in biology and medicine*, vol. 102, pp. 278–287, 2018.

[18] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* MIT Press, 2016, http://www.deeplearningbook.org.

[19] K. Perlin, "An image synthesizer," *ACM Siggraph Computer Graphics*, vol. 19, no. 3, pp. 287–296, 1985.

[20] C. Fefferman, S. Mitter, and H. Narayanan, "Testing the manifold hypothesis," 2013.

[21] M. A. Kramer, "Nonlinear principal component analysis using autoassociative neural networks," *AIChE journal*, vol. 37, no. 2, pp. 233–243, 1991.

[22] P. Baldi and K. Hornik, "Neural networks and principal component analysis: Learning from examples without local minima," *Neural networks*, vol. 2, no. 1, pp. 53–58, 1989.

[23] B. Hou, J. Yang, P. Wang, and R. Yan, "Lstm-based auto-encoder model for ecg arrhythmias classification," *IEEE Transactions on Instrumentation and Measurement*, vol. 69, no. 4, pp. 1232–1240, 2020.

[24] E. Parzen, "On Estimation of a Probability Density Function and Mode," *The Annals of Mathematical Statistics*, vol. 33, no. 3, pp. 1065 – 1076, 1962. [Online]. Available: https://doi.org/10.1214/aoms/1177704472

[25] B. J. Worton, "Kernel methods for estimating the utilization distribution in home-range studies," *Ecology*, vol. 70, no. 1, pp. 164–168, 1989.

[26] P. G. Shivakumar and P. G. Georgiou, "Perception optimized deep denoising autoencoders for speech enhancement." in *Interspeech*, 2016, pp. 3743–3747.

[27] L. Sun, J. Du, L.-R. Dai, and C.-H. Lee, "Multiple-target deep learning for lstm-rnn based speech enhancement," in *2017 Hands-free Speech Communications and Microphone Arrays (HSCMA)*, 2017, pp. 136–140.

[28] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, "Progressive neural networks," 2022.