

We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

6,900

Open access books available

185,000

International authors and editors

200M

Downloads

Our authors are among the

154

Countries delivered to

TOP 1%

most cited scientists

12.2%

Contributors from top 500 universities



WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index
in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com



Chapter

Delivering Precision Medicine to Patients with Spinal Cord Disorders; Insights into Applications of Bioinformatics and Machine Learning from Studies of Degenerative Cervical Myelopathy

Kalum J. Ost, David W. Anderson and David W. Cadotte

Abstract

With the common adoption of electronic health records and new technologies capable of producing an unprecedented scale of data, a shift must occur in how we practice medicine in order to utilize these resources. We are entering an era in which the capacity of even the most clever human doctor simply is insufficient. As such, realizing “personalized” or “precision” medicine requires new methods that can leverage the massive amounts of data now available. Machine learning techniques provide one important toolkit in this venture, as they are fundamentally designed to deal with (and, in fact, benefit from) massive datasets. The clinical applications for such machine learning systems are still in their infancy, however, and the field of medicine presents a unique set of design considerations. In this chapter, we will walk through how we selected and adjusted the “Progressive Learning framework” to account for these considerations in the case of Degenerative Cervical Myelopathy. We additionally compare a model designed with these techniques to similar static models run in “perfect world” scenarios (free of the clinical issues addressed), and we use simulated clinical data acquisition scenarios to demonstrate the advantages of our machine learning approach in providing personalized diagnoses.

Keywords: Precision Medicine, Personalized Medicine, Neural Networks, Degenerative Cervical Myelopathy, Spinal Cord Injury, Continual Learning, Bioinformatics

1. Introduction

The classical practice of medical is undergoing a transition as new scales of data production become increasingly common. This transition presents the field with major analytical challenges that necessitate new and creative ways of engaging. The

use of machine learning is one of the most important developments supporting this transition, as its methods are ideally suited (and in fact benefit from) the massive scale of data collection that is increasingly becoming the norm. With the rapid growth in high-throughput technologies for genetics, proteomics, and other biological metrics, alongside the recent wide-spread adaptation of electronic health records [1, 2], more data than ever has become available to feed into a machine learning system. The classical practice of medicine is typified by giants such as Dr. William Osler [3] whose diagnostic acumen improved the lives of many. In the very near future (and, in many cases, the present), physicians and diagnosticians will work with more data than they could possibly interpret. Machine learning is one of many tools which will help alleviate this, helping to guide many diagnostic and therapeutic decisions made by the clinical team, and if implemented well, should support patients' overall health. This potential realization of "precision medicine" is based on the belief that each patient has unique characteristics which should be accounted for when treating them [4].

While precision medicine has already demonstrated major benefits in fields like pharmacology [5] and oncology [6–8], a number of potential applications remain in other medical fields. In this chapter, we will demonstrate this using spinal cord disease, specifically by examining its application to Degenerative Cervical Myelopathy (DCM). DCM is a condition when the bones and joints of the human neck (cervical spine) degenerate with age, causing a slow progressive 'squeeze' of the spinal cord. This progressive condition has a significant effect on patient quality of life. Symptoms include pain, numbness, dexterity loss, gait imbalance, and sphincter dysfunction [9, 10], with symptoms often not appearing until permanent damage has already occurred [11]. MRI scans are typically used as part of the diagnostic process, and demographic factors have also been shown to be effective in predicting DCM severity [12]. An additional challenge is that patients can exhibit the hallmarks of DCM without developing symptoms [13], suggesting that a wide range of factors may be contributing to the illness's severity. Despite all of this, research into precision medical approaches and diagnostics have been sorely lacking; to the best of our knowledge, only 4 published studies involving DCM (also referred to as Cervical Spondylotic Myelopathy) exist which utilize machine learning [7, 14–16], coming from only three different groups (including our own), and with only one utilizing MRI data [16].

In this chapter, we will discuss how we went about designing a machine learning process, focusing on considerations required for clinical data specifically. We first explore how data should be managed and stored, before moving into data preparation procedures. Finally, we move onto the design considerations for the machine learning model. We will focus on models made for diagnostic prediction, rather than outcome prediction; however, we intend this only as a first step in using machine learning to support patient care, with future work moving toward models that provide personalized therapeutic recommendations as well. Throughout this chapter we will apply the techniques being discussed to DCM to help contextualize them. Some preliminary results for the final resulting system will also be shown, as a 'proof-of-concept', using the CIFAR-10 dataset modified to replicate clinical circumstances. We hope that this will provide a road-map for future machine learning driven precision medicine projects to follow.

2. Precision medicine machine learning system design

We have previously published work using spinal cord metrics generated by the Spinal Cord Toolbox [17] alongside simple linear and logistic regression models [16].

While it found moderate success, our results suggested that complex aspects of the spinal cord morphology are likely the key to an accurate model, with simple regression analyses alone appearing to be insufficient. Said study also only used MRI-derived metrics, resulting in our model being unable to use non-imaging attributes to support its diagnostic conclusions, something which has been shown to aid model accuracy in other studies [18]. Finally, our prior models were static in nature, and thus had to be rebuilt each time new data became available. While this may be tractable for simple models (which can be rebuilt very quickly), more complex models require more computational investment, and as such would become far too difficult to manage as the dataset grows. As an additional concern, there is reason to believe that the trends in our collected metrics are likely to change over time as societal, behavioral, and environmental changes occur, influencing DCM epidemiology [19], resulting in prior trends becoming obsolete or less significant. As such, an ideal model would be able to adapt to these changes as they arise, without the need of manual correction.

2.1 Data management

As previously mentioned, a key consideration in the clinical use of machine learning is that clinical data does not remain fixed. As new patients arrive and have their data collected and current patients see their disease state change, the relevant data that can be leveraged will change and expand over time. One possible approach is to retrain our machine learning model from scratch each time we update our dataset; this would become incredibly time and resource consuming as the dataset grows, however. Thankfully, advancements in *continual learning* in the last 5 years provide an elegant solution [20] (which we discuss in Section 3). To use these techniques effectively, we will need to consider the best way of optimizing how data is collected, stored, accessed, processed, and reported. Ideally, these data management systems should be malleable, extendable, and easy to use, so they may remain useful long-term in a ever-changing clinical environment. This section will focus on detailing methodologies for achieving this, accounting for the challenges presented by ongoing clinical data collection in the process.

2.1.1 Acquisition and storage

Ideally, our clinical dataset would include any and all relevant features that can be reliably and cost-effectively obtained. In reality, the specific data elements (or “features”) will vary both across patients and over time (as new diagnostic tests come available or as ethical rules/constraints are updated). As such, an ideal data management approach should be capable of adapting to variable data feature collection over time, while still allowing new patients to be included. For ethical reasons, the storage system also needs to be set up so that data can be easily removed, should patients request their data be purged or if privacy rules require it.

In our facility, we addressed these considerations by creating a non-relational document database system using MongoDB. This allows for new features to be added and removed on-the-fly via a modular framework of ‘forms’, which specify sets of related features that should exist inside a single document ‘type’. These documents can then be stored within a larger super-document (which we will refer to as a ‘record’) for each specific patient. This results in a large dataset containing all relevant features organized in an individual-specific manner. Each form acts as a ‘schema’, specifying what features can be expected to exist within each patient’s record. With

MongoDB, this allows features to be added and removed as needed without restructuring the entire database [21], which would risk data loss. If new features are desired, one can simply write a new form containing said features and add it to the system; previous entries without these new features can then be treated as containing “null” values in their place, thereby enabling them to still be included in any analyses performed on the data. Should features need to be removed, the form containing them can either be revised or deleted entirely. This results in the features effectively being masked from analysis access without deleting them from the database itself, allowing for their recovery in the future.

Our system also has the added benefit of allowing for the creation of ‘output’ forms, which capture and store metrics generated from data analyses. This enables the same system that collects the data to also report these analytical results back to the original submitter via the same interface. These output forms can also be stored alongside forms containing the original values that were provided to the analysis, making both easily accessible when calculating error/loss.

In our DCM dataset, all features (including MRI sequences) were collected at the time of diagnosis and consent to participate in our longitudinal registry associated with the Canadian Spine Outcomes and Research Network [22]. This registry collects hundreds of metrics on each patient, including a number of common diagnostic tests, with each being stored in the database as a single form. Most notably, this includes the modified Japanese Orthopedic Association (mJOA) scale form [23]. This is important for our study as we used this diagnostic assessment of DCM severity as the target metric for model training purposes. The MRI sequence form (which contains our MRI sequences alongside metadata associated with how they were obtained) and demographic information about the patient (including metrics such as name, age, and sex, among others) are also represented by one form each within our system. A simplified visualization of this structure can be seen in **Figure 1**.

This system can also allow pre-built structures to be re-created within it. For example, our MRI data is currently stored using the Brain Imaging Data Structure (BIDS) format [24]. This standardized data structure has directory hierarchies according to the contents of the file, with metadata describing the contents of the directory “falling through” to sub-directories and documents nested within it. These nested directories can then contain new metadata which overrides some or all of the previously set values, allowing for more granular metadata specification. Such a structure is conducive to our system, with said “nested” directories acting as features within forms, or forms within records; features could even consist of sets of sub-features (such as our MRI feature, which contains the MRI image *and* its associated metadata bundled together). Such nested structure can then specify “override” values, as they become needed.

2.2 Cleaning and preparation

The raw data collected in a clinical setting is almost never “analysis ready”, as factors like human error and/or missing data fields must be contended with. Strategies for “cleaning” data can vary from dataset to dataset, but for precision medicine models there are some common standards. First, such protocols should work on a per-record basis, not a full-data basis. This is to avoid the circumstance where adding entries with extreme values would skew the dataset’s distribution, compromising the model’s prior training (as the input metrics are, in effect, re-scaled), resulting in an unrealistic drop in model accuracy. Per-record internal normalization, however,

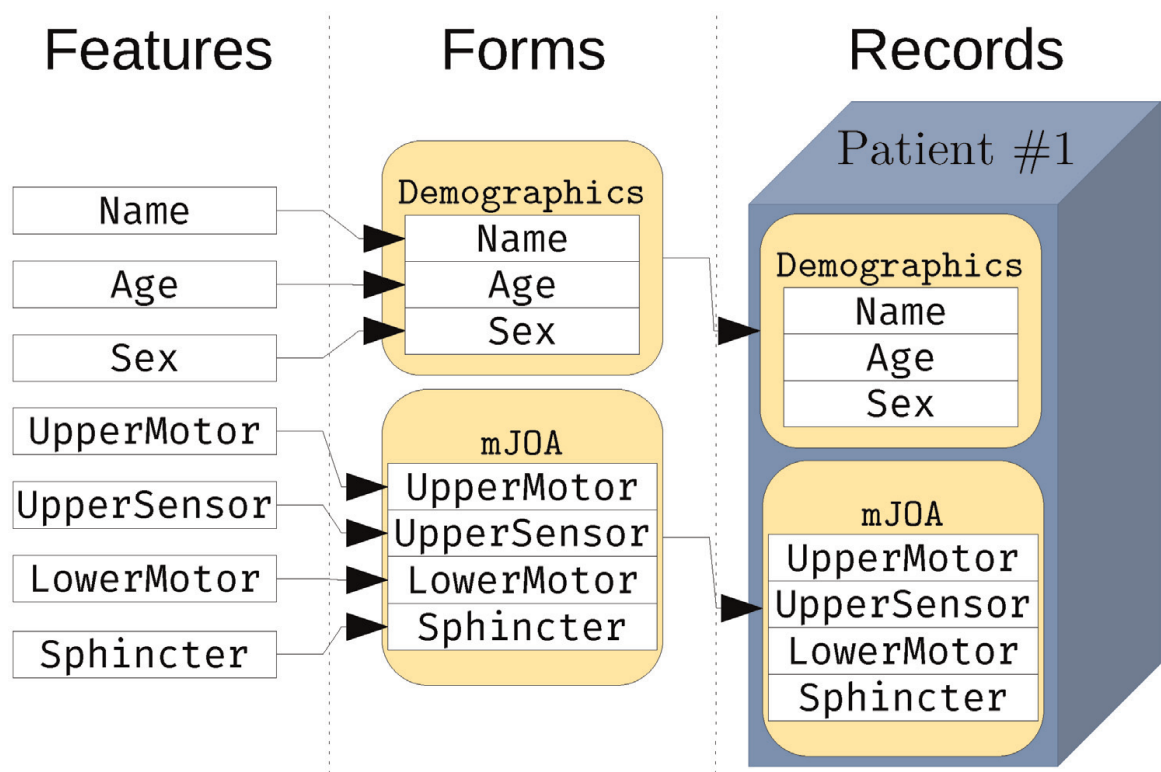


Figure 1.
A simplified example of how data is stored and managed in our theoretical system. Each feature tracked is first bundled into a ‘model’, which groups related features together alongside a descriptive label. These models act as a schema for any data analysis procedures to hook into, and can be modified, removed, and created as needed. Model instances are then stored in ‘records’, which represent one entry for any analysis system which requires it (in our case, that of one patient enrolled in our DCM study). A data structure like this can be easily achieved with any non-relational database system; in our case, we opted to use MongoDB.

typically works well, so long as it remains consistent over the period of the model’s use. Some exceptions to this exist; for example, exclusion methods may need to be aware of the entire dataset to identify erroneous records. Likewise, imputation methods will need to “tap into” other available data to fill in missing or incorrect data points within each record.

It is often the case that data is obtained from multiple different sources (e.g. different clinics, practitioners, hospitals, labs, databases, etc.), which may have varying protocols and/or environmental differences that can structurally influence the resulting measurements. If the model could be retrained from scratch every time new data was obtained, these batch effects could be easily removed [25]. In iteratively trained systems, however, this would result in the same issue as full-data normalization; new entries causing fall-through changes in the entire dataset. However, under the assumption that batch effects have less influence on the data than ‘true’ contributing effects, it has been shown that systems which learn iteratively can integrate batch effect compensation directly into their training for both numeric [26] and imaging [27] metrics, thereby resolving the issue.

Coming back to our DCM example, our data consists of demographic information (which included a mix of numerical and categorical data), diagnostic data (also numerical and categorical), and 3-dimensional MRI sequences data (which also contains meta-data describing its acquisition method). For numerical and categorical data, our processing procedures are minimal, consisting of a quick manual review to confirm that all required features were present. As our dataset was relatively large, we opted to simply

drop entries which contained malformed or missing data. New patient entries with errors were either met with a request to the supplier for corrected data, or had true values imputed for prediction purposes [28]. Categorical data is then one-hot encoded, while numerical data is scaled between 0 and 1 for values with known minimums and maximums. We had access to multiple different MRI sequencing methodologies as well, but focus on T2w sagittal oriented sequences based on our prior tests with the data [16]. MRI sequences are then resampled to a voxel size of 1mm^3 and the signal values normalized to a 0 to 1 range. Unlike our numerical results, this was done based on per-image signal minimum and maximum, in an attempt to account for variations in signal intensity variation, aiding in batch effect removal in the process.

3. Machine learning model design

Like the data management system, machine learning models designed for precision medicine need to be able to accept new data on an ongoing basis. The data contents may change over time as new discoveries about the illness are made, though it can be safely assumed that new data will be related to old data in some way. Contents of new data cannot be expected to be well distributed across target all target metrics. All of these requirements make precision medicinal systems a perfect use-case for continual learning systems.

Continual learning systems are characterized by their iterative training, as well as the ability to ‘recall’ what they learn from prior tasks to help solve new ones. Each of these tasks are assumed to be related, but contain non-trivial variation. This means the model must be flexible to change, while avoiding completely reconstructing itself after each new task, which could result in it ‘forgetting’ useful prior learning. These capabilities are referred to, respectively, as forward transfer (the ability to leverage prior learning to improve future analyses) and backward transfer (the ability leverage new knowledge to help with prior tasks).

Promising progress has been made in designing continual learning systems [20], to the point of a preliminary frameworks being devised to develop them. For this chapter, we will be using Fayek et. al’s *Progressive Learning* framework [29] as a baseline reference, though some changes were made to account for precision medicine applications.

3.1 Initial network structure

All networks need to start somewhere, which for all intents and purposes acts like a classical static machine learning system. Neural networks are the system of choice for these processes, as they allow for multiple data types to be analyzed simultaneously, being able to be constructed in a modular fashion to match up with our data storage structure detailed prior. Depending on the data, what this entails will differ. For data with implicit relations between features (such as MRI images with their spatial relations), Convolutional Neural Network (CNN) systems have been shown to be extremely effective [30]. CNNs are also among the most computationally efficient neural networks to train and run [31, 32], making them ideal for low resource systems. For other data, a Densely Connected Learning Networks (DCLNs) may be more appropriate. The complexity of these networks can be tuned to fit the problem. These models tend to be over-parameterized, however, potentially causing them to “stick” in one place or over-fit to training data; this is mediated somewhat via model pruning, discussed later in this section. The choice of available models is ever-changing, however, so one should find the model structure which best fits their specific case.

For progressively learning models, one further constraint exists; it must be able to be cleanly divide its layers into ‘blocks’. As discussed in Fayek et. al’s framework [29], this is necessary to allow for the model to progress over time. How these blocks are formed can be arbitrary, so long as each block is capable of being generalized to accept data containing the same features, but of different shape (as the size of the input data grows resulting from the concatenation operation discussed later in this section). One should also keep in mind that the block containing the output layer will be reset every progressive iteration, and should be kept as lightweight as possible.

For DCM, this would be accomplished via multiple layers running in parallel. For MRI inputs, being 3D spatial sequences, something like a 3D DenseNet similar to that employed by Ke et al. [33] could be used. The DenseNet could be run alongside DCLN blocks in parallel to read and interpret our linear data (demographics, for example), grouped with the DenseNet blocks to form the initial progressive learning model. A diagram of this structure, using the same structure mentioned prior (**Figure 1**), with a simplified “MRI” model present, is shown in **Figure 2**.

For the purposes of comparison with the original *Progressive Learning* framework, however, our testing system will instead use their initial model structure [29].

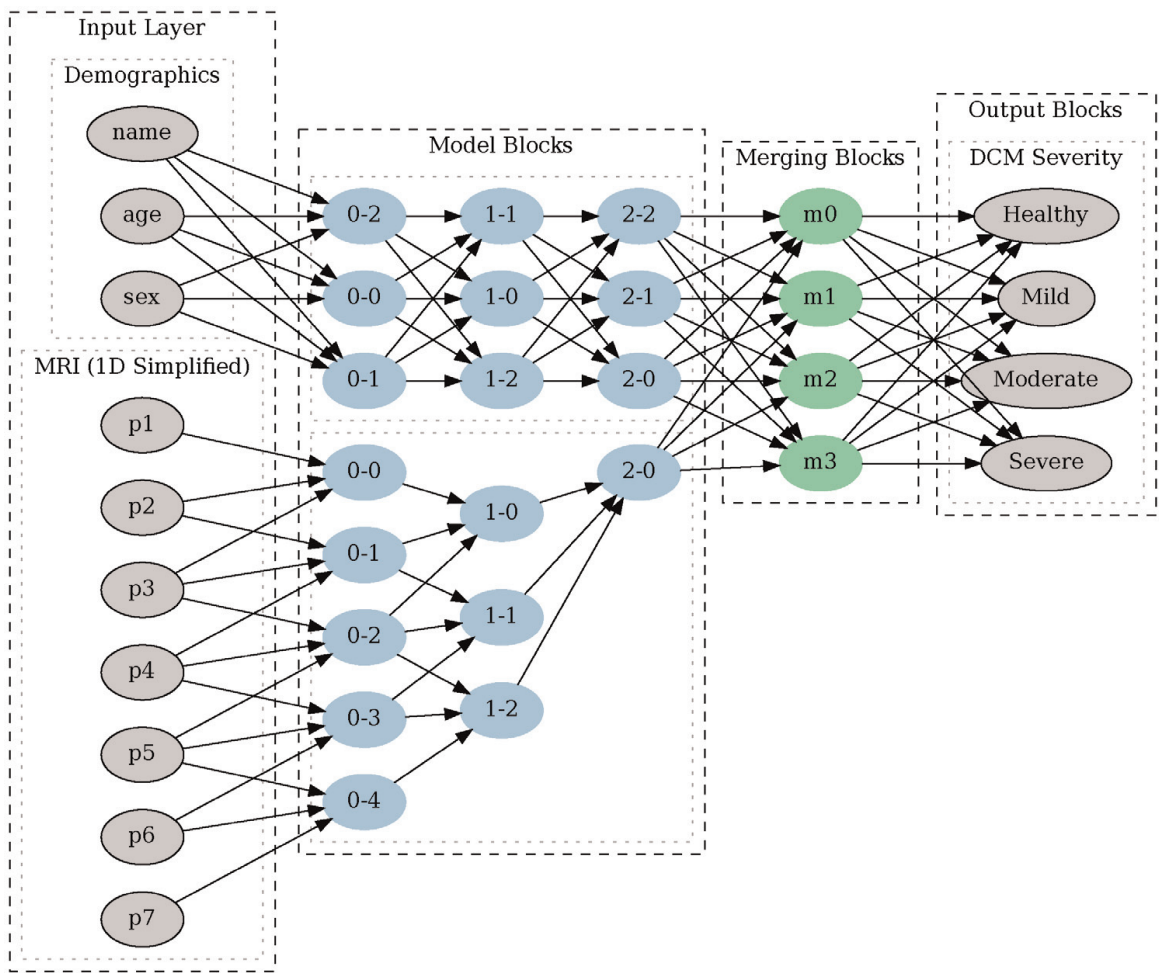


Figure 2.
An example of the initial neural network structure for use in precision medicinal systems. Note that each form receives its own “branch” block (presented within the model block column) which is used to interpret the form’s contents. As a result, each branch’s structure can be tailored to suit the form’s contents, allowing for modular addition or removal of model’s feeding into the network’s design as needed. The results of each of these branches’ interpretations are then submitted to a set of “merging” blocks, which attempts to combine these results together in a sensible manner, before a final “output” layer reports the model’s predictions for the input. The output layer is also modular, allowing for extension and/or revision as desired.

3.2 Iterative training data considerations

Once this initial framework is in place, it then needs to prove capable of accepting new patient data, updating itself as it does so. Given that the measurements of patients enrolling in clinical illness studies can be sporadic in terms of when and how often they are made, data for this system will need to be collected over time until a sufficiently large ‘batch’ of new records is acquired. Ideally large batches would be collected which are sizable enough to be split into multiple smaller batches, allowing for curriculum formation as detailed in the subsequent section. In many cases this is simply not feasible due to the time required to obtain such large batches. In this circumstance, each batch acts as a single ‘curriculum’ provided to our network in effectively random order. Thankfully, the curriculum stage appears to be the least significant stage of the Progressive Learning framework [29]. The size of these batches will depend heavily on how much data one expects to be able to collect in a given period of time and how regularly one wishes to update the model. For categorical data, each batch should include at least two of every category (one for testing, one for validation), which may influence how many samples one needs to acquire. We recommend a slightly larger batch sizes when linear data is brought into the fold, to account for the increased variety. With our DCM dataset, with a categorical output metric (the mJOA-derived DCM severity class, consisting of 4 classes), a batch of 20 patient records was selected. Data augmentation of this *new* data can also be utilized to increase the number of effective records being submitted. However, one should avoid using data from records in previous training cycles, as it can lead to a the model failing to adopt novel data trends in newer results.

3.3 Continual learning

Here, we will focus on detailing a framework based on Fayek et. al’s Progressive Learning Framework [29], which consists of three stages; *curriculum*, *progression* and *pruning*.

3.3.1 Curriculum

Given sufficiently large batches of new data can be collected in a timely manner, one can utilize the curriculum stage; at least three times the number of records per batch being collected in a 6 month period seems to be a good cutoff for this, though this can differ depending how rapidly one expects disease trends to change. This stage, as described in Fayek et. al’s framework [29], is composed of two steps; curricula creation and task ordering. In the creation step, the batch is split into sub-batches, with each being known as a ‘curriculum’. How this is done depends on the data at hand (i.e. categorical data requires that each curriculum contains data from each category), but can otherwise be performed arbitrarily. Once these curricula are formed they are sorted based on an estimate of how “difficult” they are, from least to most. Difficulty estimation can be as simple as running a regression on the data and using the resulting loss metric. The sorted set of curricula are then submitted to the network for the progression and pruning stages, one at a time. This allows for the network to learn incrementally, picking up the “easier” trends from earlier curricula before being tasked with learning more “difficult” trends in later ones.

In precision medicine, however, collecting sufficient data in a useful time span is often not possible. In this case, this stage can be safely skipped; the smaller batches will simply act as randomly sampled, unordered curricula. How “large” this is depends on the

batch size selected earlier; one should collect at least three batches worth to warrant the additional complexity of the curriculum stage. For our DCM setup, we fell below this level, as we intend on updating as often as possible, and as such intended on utilizing new batches as soon as they were collected. We believe that in most precision medicine examples this is likely to be the case, though in some situations (such as the ongoing COVID-19 pandemic), the scale of patient data collection may make the curriculum stage worth considering. Employing few-shot learning techniques may also allow for smaller subsets of data to form multiple batches as well, though the efficacy of such procedures has yet to be tested in this context.

3.3.2 Progression

In this stage, new blocks of layers are generated and concatenated to previously generated blocks in the model. The new input-accepting block is simply stacked adjacent to the prior input blocks in the model, ready to receive input from records in our dataset. Each subsequent new block, however, receives the concatenated outputs of *all* blocks from the prior layer, allowing it to include features learned in previous training cycles. The final block, which contains the output layer, is then regenerated entirely, resulting in some lost training progress that is, thankfully, usually quickly resolved as the model begins re-training.

The contents of these added blocks depends on the desired task and computational resources available. Large, more complex blocks require more computational resources and are more likely to result in over-fitting, but can enable rapid adaption of the network and better forward transfer. In the original framework [29], these blocks were simply copies of the original block's architecture, but reduced to approximately half the parameters. However, one could instead cycle through a set of varying block-types, based on how well the model performed and whether new data trends are expected to have appeared. They could also be changed as the model evolves and new effective model designs are discovered, though how effective this is in practice has yet to be seen.

Once these blocks are added, the network is then retrained on the new batch of data, generally with the same training setup used for the original set of blocks. During this retraining, prior block's parameters can be frozen, locking in what they had learned prior while still allowing them to contribute to the model's overall prediction. This prevents catastrophic forgetting of previously learned tasks, should they need to be recalled, though this usually comes at the cost of reduced overall training effectiveness. However, if one does not expect to need to re-evaluate records which have already been tested before, one can deviate from Fayek's original design and instead allow prior blocks to change along with the rest of the model. An example of progression (with two simple DCLN blocks being added) is shown in **Figure 3**.

For our DCM data, this is a pretty straightforward decision. New blocks would simply consist of new 3D DenseNet blocks run in parallel to simple DCLN layers, both containing approximately half the parameters as the original block set. The output block is then simply a linear layer which is fed into a SoftMax function for final categorical prediction. As we do not expect prior records to need to be re-tested, we also allow prior blocks to be updated during each training cycle.

3.3.3 Pruning

In this stage, a portion of parameters in the new blocks are dropped from the network. What parts of the model are allowed to be pruned depends on how the prior

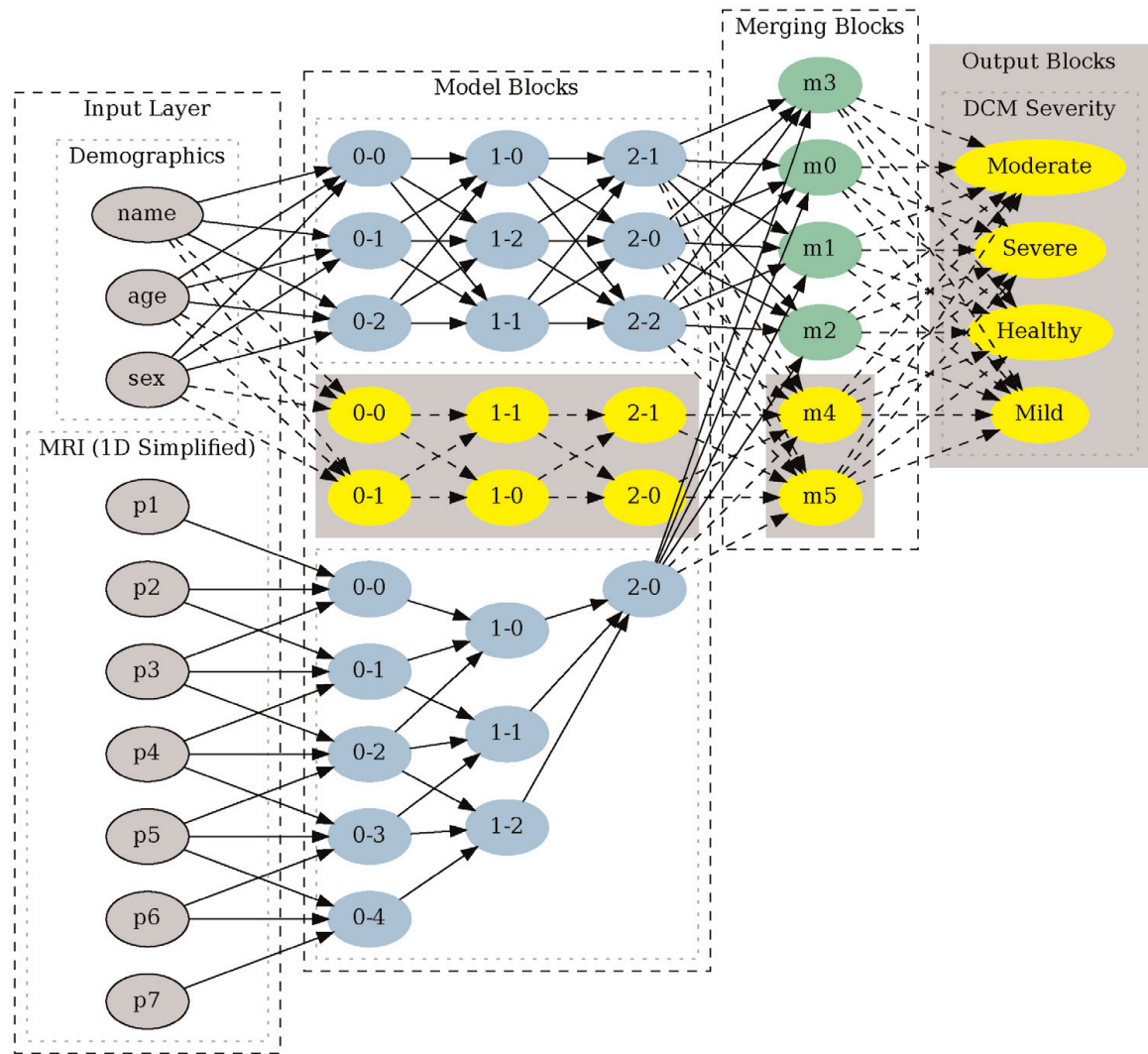


Figure 3. An example of the progression stage, building off of the initial model shown in **Figure 2**. New nodes are contained within the gray boxes, with hashed lines indicating the new connections formed as a result. Note that input connections are specific to each form, only connecting to one's inputs (in this case, only the Demographic's input), and not to those in the other branches (such as the MRI branch); this allows for shortcomings in particular model's contributions to be accounting for independently, without an extreme growth in network complexity. Note as well that the merging layer (representing all non-input receiving blocks) forms connections with all prior block outputs, however, regardless of which forms have received a new connected block. The entire output block is also regenerated at this stage, providing some learning plasticity at the expense of initial learning.

progression stage was accomplished; if previously trained blocks have been frozen, then only newly added elements should be allowed to be pruned to avoid catastrophic loss of prior training. Otherwise, the entire model can be pruned, just as it has been allowed to be trained and updated. The pruning system can also vary in how it determines which parameters are to be pruned, though dropping parameters with the lowest absolute value weights is the most straightforward. These can also be grouped as well, with Fayek et al. choosing to prune greedily layer-by-layer. However, we have found that considering all parameters at once is also effective. The proportion q dropped per cycle will depend on the computational resources and time available. Smaller increments will take longer to run, whereas larger values will tend to land further away from the “optimal” state of the network. An example of the pruning stage is shown in **Figure 4**.

The network, now lacking the pruned parameters, is then retrained for a (much shorter) duration to account for their loss. In Fayek et al's example, this process is then

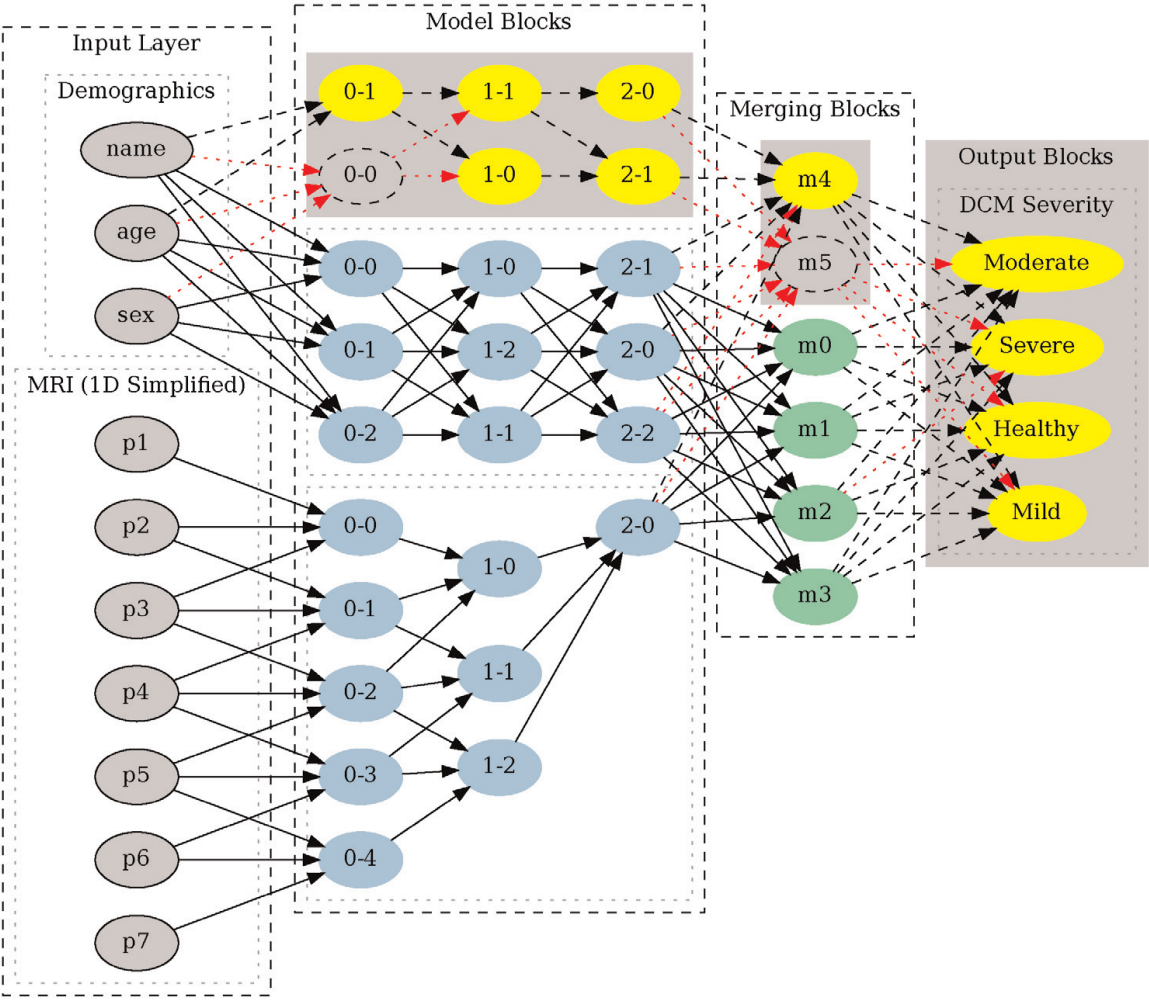


Figure 4.
An example of the pruning stage, building off of the progression network shown in **Figure 3**. Note that only newly formed connections are targeted for pruning by default, with pre-existing connections remaining safe. Parameters themselves can also be effectively lost entirely (shown as nodes with no fill and a dashed outline) should all connections leading into them get removed. This results in all connections leading out of them also getting pruned by proxy.

repeated with progressively larger q proportions until a loss in performance is observed. Alternatively, one can instead repeatedly drop the same percentile of parameters each cycle from the *previously pruned* network. This has the benefit of reducing the time taken per cycle slightly (the same weights do not need to be pruned every cycle), while also leading to the total proportion of the pruned model increasing more gradually per cycle, improving the odds that the model lands closer to the true optimal size for the system. This pruning system has the potential to be much slower, however (should the rare circumstance occur where all the new parameters are useless, requiring more iterations overall). As such, in time-limited systems, Fayek's approach remains more effective.

This stage also allows for, in theory, dynamic feature removal. Should a model (or feature within said model) cease to be available, one can simply explicitly prune the parameters associated with that feature, in effect performing a targeted pruning cycle. One would need to re-enable training of previously trained nodes to account for this, however, leading to the possibility of reduced backward transfer. Depending on how significant the to-be-removed features have become in the network, this may need to be done over multiple pruning cycles; this should allow the network to adapt to changes over time, reducing the risk of it getting 'stuck' in a sub-optimal state.

For our DCM data, the complexity of the illness and scope of the data makes it extremely unlikely for a worst-case pruning issue to occur. As such, a 10% lowest absolute weight pruning system, applied globally, is selected as our starting point, iteratively applied until a loss of mean accuracy over 10 post-prune correction epochs is observed.

4. Protocol assessment

4.1 Progressive learning ConvNet

4.1.1 Methodology

To confirm our protocol functions effectively in practice, we replicated the CIFAR-100 analysis used in the original *Progressive Learning* paper [29], with a few major changes to replicate a precision medicine environment (i.e. the kind of clinical context in which data is typically collected). First, only the CIFAR-10 dataset was used [34], rather than it being used as the model initialization dataset. This was done to better reflect the clinical environment, where we are unlikely to have a pre-established dataset which is known to be effective at preparing our model for the task. The 10 categories of the CIFAR-10 dataset also represent the average granularity usually used to assess many illnesses. Second, our datasets were randomly split into 10 subsets of 6000 random images, with 5000 used for training and the remaining 1000 used for validation. The contents of these subsets was completely random, allowing for imbalance in the number of elements from each of the 10 categories, reflecting how data collected in a clinical setting could occur. Third, we skipped the curriculum stage, again to reflect the circumstances of clinical data collection (wherein the scale of collection is insufficient). Fourth, our framework was implemented in PyTorch [35] rather than TensorFlow [36] due to its more robust network pruning support. Finally, data augmentation was performed on each image to both discourage the model from memorizing data and to simulate human error/variation in clinically acquired data. This results in a problem which is slightly more difficult than the original setup devised by Fayek et al., though for parity sake, we continued to use the same Convolution Network design.

We tested 6 procedures, representing combinations of two different variations. The first variation was whether the learning model trained as an independent learning model, a progressive learning model with prior blocks frozen, or a progressive learning model with prior blocks being freely pruned and updated. For independent learning procedures, the model was completely reset after each training cycle, whereas for progressive learning procedures the model persisted across cycles (allowing for it to “apply” prior knowledge to new data). The second was whether data was provided in batches (similar to a clinical setting), or submitted all at once (the “ideal” for machine learning analyses). In batched procedures, data was submitted one subset at a time, as described prior. A strict max wall time of 8 hours was put in place for all protocols to simulate the limited resources (in both time and hardware) that clinical settings often have. All protocols were run on a single Tesla V100-PCIE-16GB GPU with 16GB of RAM and two Intel(R) Xeon(R) Gold 6148 CPUs run at 2.40GHz (speeding up initial protocol setup).

The initial architecture for all procedures is shown in **Table 1**. For progressive learning procedures, new blocks were added which were half the size of the original

Block Number	Type	Size	Other
1	2DConvolution	32, 3x3	Stride = 1
	2DBatchNorm		
	ReLU		
	[Concatenation]		
2	2DConvolution	32, 3x3	Stride = 1
	2DBatchNorm		
	ReLU		
	2DMaxPooling	2x2	Stride = 2
	Dropout		r = 0.25
3	[Concatenation]		
	2DConvolution	64, 3x3	Stride = 1
	2DBatchNorm		
	ReLU		
4	[Concatenation]		
	2DConvolution	64, 3x3	Stride = 1
	2DBatchNorm		
	ReLU		
	2DMaxPooling	2x2	Stride = 2
5	Dropout		r = 0.25
	[Concatenation]		
	Flatten		
	Linear	512	
6	1DBatchNorm	512	
	ReLU		
	Dropout		r = 0.5
	[Concatenation]		
	Linear	20	
	Softmax		

Table 1.
The basic structure of the convolutional neural network being tested on the CIFAR-10 dataset. Based on the model used by Fayek et al. [29]. [Concatenation] indicates where the output of one set of blocks would be concatenated together before being fed into new blocks in the following layer, and can be ignored for independent learning tasks.

blocks, set to receive the the concatenated outputs of all blocks in the prior layer of each set of blocks. All parameters were initialized randomly using PyTorch version 1.8.1 default settings. We used an ADAM optimizer with a learning rate of 0.001, first moment β_1 of 0.99, second moment β_2 of 0.999, and weight decay λ of 0.001 during training. For progressive learning models, an identical optimizer with one tenth the learning rate was used for post-pruning model optimization. Each cycle consisted of 90 epochs of training. Progressive procedures were given 10 epochs per pruning cycle, with pruning being repeated until the mean accuracy of the prior set of epochs was

greater than that of the new set of epochs, with the model's state being restored to the prior before continuing. The model's training and validation accuracy was evaluated and reported once per epoch. Protocol efficacy was measured via the max validation accuracy of the model over all cycles and epochs and mean best-accuracy-per-cycle (BAPC) for all cycles.

4.1.2 Results

For our full datasets, the model achieved diagnostic classification accuracy values of 80-85% for most of the results. The simple model, without progression and with full access to the entire dataset, reached a max accuracy of 81.13%, with a mean BAPC of 80.71%. Adding progression to the process further improved this, primarily through the pruning stage, with a max accuracy of 84.80%. However, the mean BAPC dropped to 77.44%, as prior frozen parameters in the model appeared to make the model "stagnate". Allowing the model to update and prune these carried-over parameters improves things substantially, leading to a max accuracy of 90.66% and a mean BAPC of 84.83%.

When data was batched, a noticeable drop in accuracy was observed, as expected. Without progressive learning, our model's max observed accuracy was only 73.7% (a drop of 7.37%), with a mean BAPC of 71.75%. The progressive model with frozen priors initially performed better, reaching its maximum accuracy of 75.9% in its first cycle, but rapidly fell off, having a mean BAPC of 66.0%. Allowing the model to update its priors greatly improved the results, however, leading to a maximum accuracy of 82.4% and a mean BAPC of 79.02%, competing with the static model trained on all data at once.

A plot of each model's accuracy for each model setup, taken over the entire duration (all cycles and epochs) for both the training and validation assessments, is shown in **Figure 5**.

4.2 DenseNet

4.2.1 Methodology

To confirm that the success of the setup suggested by Fayek et al. was not due to random chance, we also applied the technique to another model which is effective at predicting the CIFAR-10 dataset; the DenseNet architecture [37]. DenseNets are characterized by their "blocks" of densely connected convolution layer chains, leading to a model which can utilize simpler features identified in early convolutions to inform later layers that would, in a more linear setup, not be connected together at all. These blocks are a perfect fit for our method, as they can be generated and added to our progressive learning network just like any other set of layers. DenseNets have also been shown to have better accuracy than classical convolution nets within the CIFAR-10 dataset, reaching error rates of less than 10% in many cases [37]. However, the dense connections make the networks extremely complex, and they are generally highly over-parameterized as well, making them prone to over-fitting in some cases.

Our testing methodology was largely identical to that of the Convolutional network tested in the previous section. One change was to use a Stochastic Gradient Descent (SGD) optimizer with an initial learning rate of 0.1. Training was done in batches of 64, for a total of 300 epochs per cycle. The learning rate is reduced by a factor of 10 when 50% and 75% of the epochs for each cycle has passed. The SGD

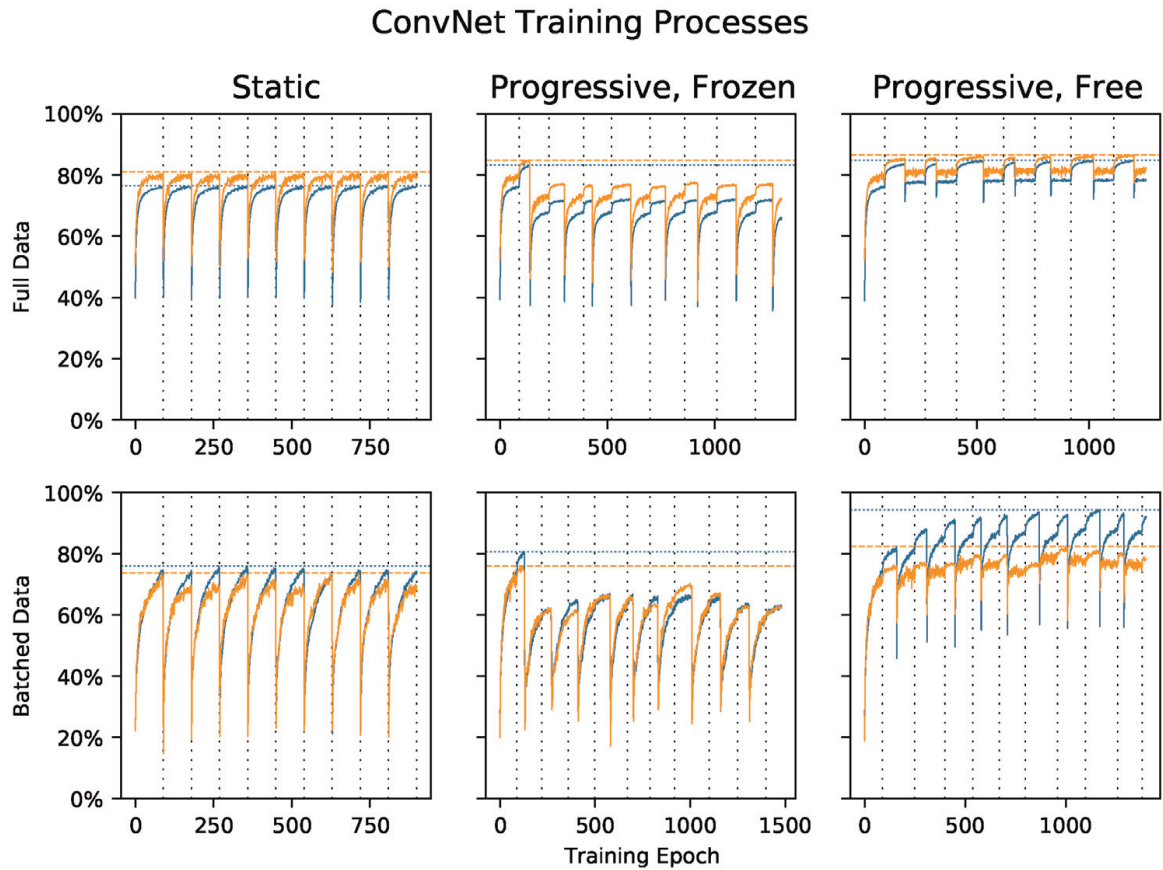


Figure 5. The training progression of the ConvNet model replicated from Fayek et. al's study [29] in various forms. From left to right, the model on its own, reset after every cycle (static), a progressively learning model, with prior traits frozen (progressive, frozen), and a progressively learning model, with all traits open to training and pruning each cycle (progressive, free). Training accuracy is shown in blue, with validation accuracy shown in orange. The maximum observed accuracy for each is indicated via a horizontal dotted (training) or dashed (validation) line. The dotted horizontal lines indicate where the training of the model for a given cycle was complete (not including the pruning of progressive models). Note that the total number of epochs taken between these cycles differs from cycle to cycle in progressive models, as a result of the pruning stage cycling until a validation accuracy loss was observed.

optimizer was set with a weight decay of 0.0001 and a momentum of 0.9. Dropout layers with a drop rate of 0.2 were added after each block as well. The initial architecture for the network was based on the 'densenet-169' architecture, and is shown in **Table 2**, having a growth rate of 32, an initial feature count of 64, and consisting of 4 blocks of densely connected convolution layers, each with 6, 12, 32, and 32 convolution layers respectively. For progressive learning systems, new blocks followed the same architecture with half the growth rate (16) and initial features (32). These changes were made to maintain parity with the original DenseNet CIFAR-10 test [37].

4.2.2 Results

For our full datasets, we saw an accuracy values of around 90%. The simple model, without progression, reached a max accuracy of exactly 90%, but was only able to run one cycle to completion before the 8 hour time limit was reached. Adding progression to the process improved this slightly, resulting in a max accuracy of of 90.66%, but only barely completed its first pruning cycle before the time limit was reached. As a result, the same accuracy was observed for both progressive models with and without

Block Number	Type	Size	Other
1	2DConvolution	64, 7x7	Stride = 2, Padding = 3
	2DBatchNorm		
	ReLU		
	2DMaxPooling		
2	[Concatenation]	Layers = 6	Bottleneck = 4, r = 0.2
	Dense Block		
	Transition Block		
	[Concatenation]		
3	Dense Block	Layers = 12	Bottleneck = 4, r = 0.2
	Transition Block		
	[Concatenation]		
	[Concatenation]		
4	Dense Block	Layers = 32	Bottleneck = 4, r = 0.2
	Transition Block		
	[Concatenation]		
	[Concatenation]		
5	Dense Block	Layers = 32	Bottleneck = 4, r = 0.2
	Transition Block		
	[Concatenation]		
	[Concatenation]		
6	2DBatchNorm	1664	
	[Concatenation]		
7	ReLU	1x1	
	2DAdaptiveAveragePool		
	Flatten		
	Linear		

Table 2.
The structure of the DenseNet mdoel being tested on the CIFAR-10 dataset. Based on the model used by Huang et al. [37]. Dense block indicates a densely connected convolution block, with transition block indicating a transition layer, both being detailed in Huang et. al’s original paper. [Concatenation] indicates where the output of one set of blocks would be concatenated together before being fed into new blocks in the following layer, and can be ignored for independent learning tasks. Where it appears, r indicates dropout rate for the associated block.

priors being trainable, as no new prior blocks were added. Slight variations were still observed, however, due to how the model’s initialization process differs.

When data was batched, a much more significant drop in accuracy occurred as compared to the Convolutional network. Without progressive learning, our model’s max observed accuracy was only 69.9% (a drop of more than 30%), with a mean BAPC of 65.87%. However, it was able to run for all 10 cycles within the allotted 8 hour time span. The progressive model with frozen priors performed even worse, reaching a maximum accuracy of 67.1% in its first cycle, and only completing 5 cycles before the time limit, only reaching a mean BAPC of 64.28%. Allowing the model to update its priors somewhat improved the results, leading to a maximum accuracy of 71.7% and a mean BAPC of 68.28%, showing some slight recovery over the static model in a batch scenario. However, it also only managed to run through 5 cycles before the time limit was reached.

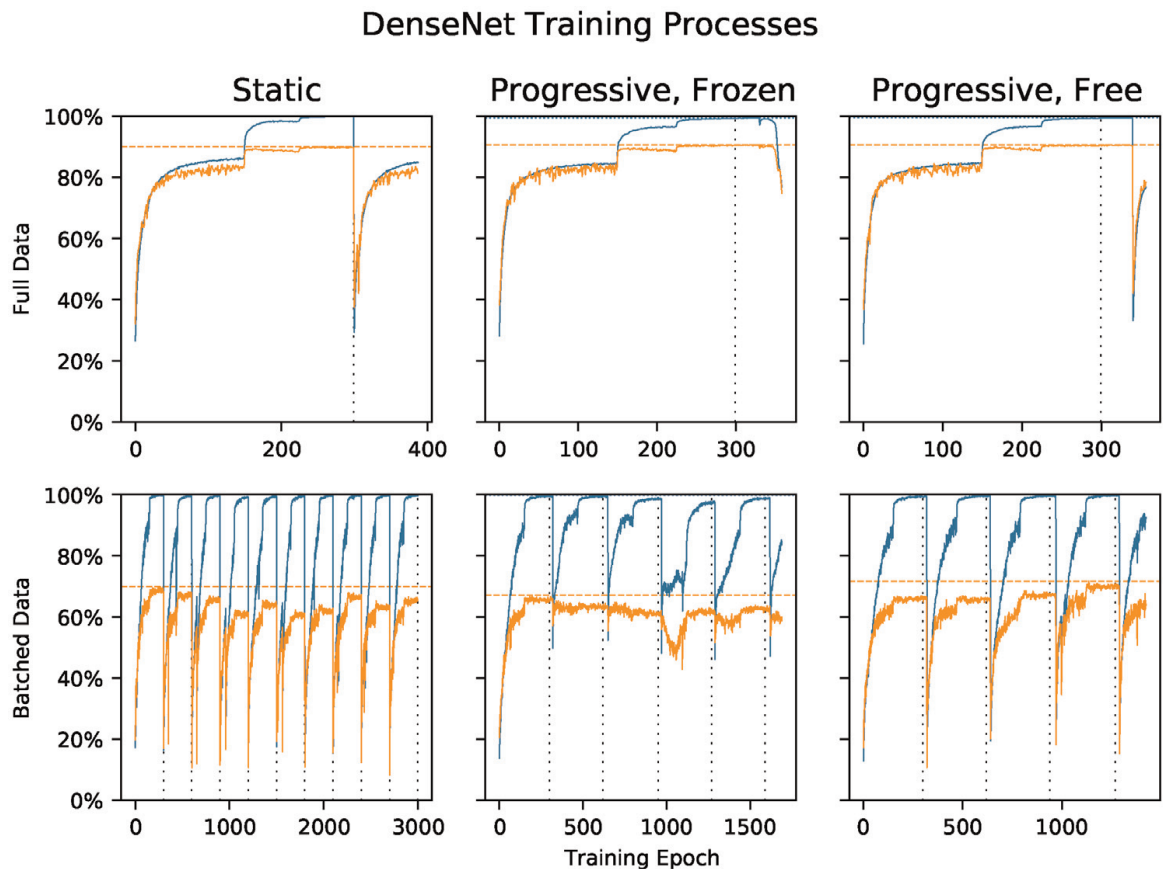


Figure 6. The training progression of the DeepNet model replicated from Huang et. al's original 'densenet-169' model [37] in various forms. From left to right, the model on its own, reset after every cycle (static), a progressively learning model, with prior traits frozen (progressive, frozen), and a progressively learning model, with all traits open to training and pruning each cycle (progressive, free). Training accuracy is shown in blue, with validation accuracy shown in orange. The maximum observed accuracy for each is indicated via a horizontal dotted (training) or dashed (validation) line. The dotted horizontal lines indicate where the training of the model for a given cycle was complete (not including the pruning of progressive models). Note that the total number of epochs taken between these cycles differs from cycle to cycle in progressive models, as a result of the pruning stage cycling until a validation accuracy loss was observed.

A plot of each model's accuracy for each model setup, taken over the entire duration (all cycles and epochs) for both the training and validation assessments, is summarized in **Figure 6**.

5. Discussion and conclusions

The results of our tests show great promise for how these approaches to machine learning use in precision medicine can be used, while nonetheless highlighting some significant shortcomings which will need to be considered should this framework become common practice. Most notably, we see that model's which over-fit the available data are extremely detrimental to the system, even if the underlying model would be better with all data immediately available to it. This is shown very clearly with the effectiveness of pruning in all our models, with clear gains in accuracy observed, likely as a result of the process helping counteract over-fitting resulting from over-parameterized models. Finding an "ideal" model for a given task is already a difficult task, and our results show that this is only exacerbated by the conditions of

a clinical environment. Nevertheless, there is clearly potential in this framework, with the Convolutional network tested on clinical-like batch data being near identical in effectiveness to its static counterpart trained on the full dataset.

We are also optimistic that many opportunities remain for improvement in progressive learning implementations. Our current implementation of the progressive learning framework is locked to a specific set of initial data inputs, being unable to add new ones should they become available. In theory, this could be as simple as adding a new set of initial blocks to the existing network, in effect acting like a “progression” stage with custom new blocks (as well as an update to existing new block generation procedures to match). However, this has a number of issues that we have not, at present, found a way to resolve. First, each branch is likely to “learn” at different rates, resulting in one set of blocks associated with a given set of input data containing more redundant features per-progression stage than the rest. This proves problematic during pruning, however; we either over-prune blocks with important features within them, or under-prune those which contain an abundance of redundant and/or noise-contributing features. We believe this can be resolved, but were simply unable to do so by the time of this publication.

Another potential improvement would be to “carry-over” output layers weights between progression stages. This would allow for the network to have better forward transfer, so long as the task’s end goal (categorical prediction, single metric estimation etc.) remains the same. In our implementation, this is currently not the case, with the output layer being regenerated every cycle, keeping it in line with the original Progressive Learning framework’s design [29]. The difficulty of implementing such a system, as well as its effectiveness in improving model outcomes, has yet to be tested.

One other major hurdle is that of long term memory cost. As currently implemented, pruning does not actually remove parameters from the model; it simply masks them out during training and evaluation, preventing them from contributing to the models predictions. While this improves the speed and accuracy of the model being generated, its memory footprint expands infinitely as more cycles are run. Resolving this issue is difficult, however, requiring the model to effectively fully re-construct itself to account for any now-removed parameters. Doing so would allow the model to come to a “static” memory cost, as the number of pruned parameters approaches the number of new ones added every cycle. In turn, this would enable applications where the model is expected to exist for very long duration in limited resource systems. Such compression techniques are an ongoing field of research at time of writing; as such, we believe such a implementation will be possible in the near future.

Finally, testing our methodology on a real-world clinical dataset is needed before we can be sure it is truly effective. While the CIFAR-10 dataset [34] has been shown to work effectively for machine learning testing purposes, our assumptions about clinical data still need to be confirmed. We intend to put our framework to the test soon, assessing its effectiveness at predicting DCM severity using the DCM data mentioned throughout this chapter; nevertheless, this framework should be considered experimental until such results (from ourselves or others) are acquired. Continual learning systems trained for clinical data also retain the limitations of continual learning, such as increased potential to over-fit and the inability to transfer new knowledge obtained to help with the understanding of prior knowledge. Modifications to the progression procedure have been proposed to amend this [29], though these have not been tested at time of writing.

Overall, however, we believe our framework for machine learning system design in precision medicine should work well as a road-map for future research, even

though refinements remain to be made. With systems such as the Progressive Learning framework available, these new systems can adapt to changes in data trends while accepting new data in effectively random batches, both important requirements for a clinical environment. Well designed data storage and management also allows such systems to easily access, update, and report important metrics to all necessary parties, while remaining open to changes as new research is completed. Through the application of these techniques, modern medicine should be able to not only adapt to the age of information, but to benefit immensely from it.

Abbreviations

DCM	Degenerative Cervical Myelopathy
CIFAR	Canadian Institute for Advanced Research
mJOA	Modified Japanese Orthopedic Association
BIDS	Brain Imaging Data Structure
CNN	Convolutional Neural Network
DCLN	Deeply Connected Learning Network
BAPC	Best Accuracy Per Cycle
SGD	Stochastic Gradient Descent

Author details

Kalum J. Ost^{1,2}, David W. Anderson^{2†} and David W. Cadotte^{1,2,3,4*†}

1 Hotchkiss Brain Institute, Calgary, Alberta, Canada

2 Cumming School of Medicine, Calgary, Alberta, Canada


3 Division of Neurosurgery, Departments of Clinical Neurosciences and Radiology, University of Calgary, Alberta, Canada

4 Combined Orthopedic and Neurosurgery Spine Program, University of Calgary, Alberta, Canada

*Address all correspondence to: david.cadotte@ucalgary.ca

† These authors contributed equally.

IntechOpen

© 2021 The Author(s). Licensee IntechOpen. This chapter is distributed under the terms of the Creative Commons Attribution License (<http://creativecommons.org/licenses/by/3.0>), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited. 

References

- [1] Julia Adler-Milstein and Ashish K Jha. Hitech act drove large gains in hospital electronic health record adoption. *Health Affairs*, 36(8): 1416–1422, 2017. doi: 10.1016/j.cell.2019.02.039.
- [2] R Scott Evans. Electronic health records: then, now, and in the future. *Yearbook of medical informatics*, (Suppl 1):S48, 2016.
- [3] Michael Bliss. *William Osler: A life in medicine*. Oxford University Press, 1999.
- [4] Zheng-Guo Wang, Liang Zhang, and Wen-Jun Zhao. Definition and application of precision medicine. *Chinese Journal of Traumatology*, 19(5): 249–250, 2016.
- [5] Christina L Aquilante, David P Kao, Katy E Trinkley, Chen-Tan Lin, Kristy R Crooks, Emily C Hearst, Steven J Hess, Elizabeth L Kudron, Yee Ming Lee, Ina Liko, et al. Clinical implementation of pharmacogenomics via a health system-wide research biobank: the university of colorado experience, 2020.
- [6] Jessica R Williams, Dalia Lorenzo, John Salerno, Vivian M Yeh, Victoria B Mitrani, and Sunil Kripalani. Current applications of precision medicine: a bibliometric analysis. *Personalized medicine*, 16(4):351–359, 2019. doi: 10.2217/pme-2018-0089.
- [7] Omar Khan, Jetan H Badhiwala, Giovanni Grasso, and Michael G Fehlings. Use of machine learning and artificial intelligence to drive personalized medicine approaches for spine care. *World Neurosurgery*, 140: 512–518, 2020. doi: 10.1016/j.wneu.2020.04.022.
- [8] Renato Cuocolo, Martina Caruso, Teresa Perillo, Lorenzo Ugga, and Mario Petretta. Machine learning in oncology: A clinical appraisal. *Cancer letters*, 481: 55–62, 2020. doi: 10.1016/j.canlet.2020.03.032.
- [9] Aria Nouri, Lindsay Tetreault, Anoushka Singh, Spyridon K Karadimas, and Michael G Fehlings. Degenerative cervical myelopathy: epidemiology, genetics, and pathogenesis. *Spine*, 40(12):E675–E693, 2015. doi: 10.1097/BRS.0000000000000913.
- [10] Benjamin M Davies, Oliver D Mowforth, Emma K Smith, and Mark RN Kotter. Degenerative cervical myelopathy. *Bmj*, 360, 2018. doi: 10.1136/bmj.k186.
- [11] Ivana Kovalova, Milos Kerkovsky, Zdenek Kadanka, Zdenek Kadanka Jr, Martin Nemec, Barbora Jurova, Ladislav Dusek, Jiri Jarkovsky, and Josef Bednarik. Prevalence and imaging characteristics of nonmyelopathic and myelopathic spondylotic cervical cord compression. *Spine*, 41(24):1908–1916, 2016. doi: 10.1097/BRS.0000000000001842.
- [12] Lindsay A Tetreault, Branko Kopjar, Alexander Vaccaro, Sangwook Tim Yoon, Paul M Arnold, Eric M Massicotte, and Michael G Fehlings. A clinical prediction model to determine outcomes in patients with cervical spondylotic myelopathy undergoing surgical treatment: data from the prospective, multi-center aospine north america study. *JBJS*, 95 (18):1659–1666, 2013. doi: 10.2106/JBJS.L.01323.
- [13] Josef Bednarik, Zdenek Kadanka, Ladislav Dusek, Milos Kerkovsky, Stanislav Vohanka, Oldrich Novotny, Igor Urbanek, and Dagmar Kratochvilova. Presymptomatic spondylotic cervical myelopathy: an

updated predictive model. *European Spine Journal*, 17(3):421–431, 2008. doi: 10.1007/s00586-008-0585-1.

[14] Benjamin S Hopkins, Kenneth A Weber II, Kartik Kesavabhotla, Monica Paliwal, Donald R Cantrell, and Zachary A Smith. Machine learning for the prediction of cervical spondylotic myelopathy: a post hoc pilot study of 28 participants. *World neurosurgery*, 127: e436–e442, 2019. doi: 10.1016/j.wneu.2019.03.165.

[15] Omar Khan, Jetan H Badhiwala, Muhammad A Akbar, and Michael G Fehlings. Prediction of worse functional status after surgery for degenerative cervical myelopathy: A machine learning approach. *Neurosurgery*, 2020. doi: 10.1093/neuros/nyaa477.

[16] Kalum Ost, W Bradley Jacobs, Nathan Evaniew, Julien Cohen-Adad, David Anderson, and David W Cadotte. Spinal cord morphology in degenerative cervical myelopathy patients; assessing key morphological characteristics using machine vision tools. *Journal of Clinical Medicine*, 10(4): 892, 2021. doi: 10.3390/jcm10040892.

[17] Benjamin De Leener, Simon Lévy, Sara M Dupont, Vladimir S Fonov, Nikola Stikov, D Louis Collins, Virginie Callot, and Julien Cohen-Adad. Sct: Spinal cord toolbox, an open-source software for processing spinal cord mri data. *Neuroimage*, 145:24–43, 2017. doi: 10.1016/j.neuroimage.2016.10.009.

[18] Takashi Kameyama, Yoshio Hashizume, Tetsuo Ando, and Akira Takahashi. Morphometry of the normal cadaveric cervical spinal cord. *Spine*, 19(18):2077–2081, 1994. doi: 10.1097/00007632-199409150-00013.

[19] Nitin B Jain, Gregory D Ayers, Emily N Peterson, Mitchel B Harris, Leslie

Morse, Kevin C O’Connor, and Eric Garshick. Traumatic spinal cord injury in the united states, 1993–2012. *Jama*, 313(22):2236–2243, 2015.

[20] Raia Hadsell, Dushyant Rao, Andrei A Rusu, and Razvan Pascanu. Embracing change: Continual learning in deep neural networks. *Trends in Cognitive Sciences*, 2020. doi: 10.1016/j.tics.2020.09.004.

[21] Mongoddb. <https://github.com/mongodb/mongo>, 2013.

[22] Nathan Evaniew, David W Cadotte, Nicolas Dea, Christopher S Bailey, Sean D Christie, Charles G Fisher, Jerome Paquet, Alex Soroceanu, Kenneth C Thomas, Y Raja Rampersaud, et al. Clinical predictors of achieving the minimal clinically important difference after surgery for cervical spondylotic myelopathy: an external validation study from the canadian spine outcomes and research network. *Journal of Neurosurgery: Spine*, 33(2):129–137, 2020. doi: 10.3171/2020.2.spine191495.

[23] Lindsay Tetreault, Branko Kopjar, Aria Nouri, Paul Arnold, Giuseppe Barbagallo, Ronald Bartels, Zhou Qiang, Anoushka Singh, Mehmet Zileli, Alexander Vaccaro, et al. The modified japanese orthopaedic association scale: establishing criteria for mild, moderate and severe impairment in patients with degenerative cervical myelopathy. *European Spine Journal*, 26(1):78–84, 2017. doi: 10.1007/s00586-016-4660-8.

[24] Krzysztof J Gorgolewski, Tibor Auer, Vince D Calhoun, R Cameron Craddock, Samir Das, Eugene P Duff, Guillaume Flandin, Satrajit S Ghosh, Tristan Glatard, Yaroslav O Halchenko, et al. The brain imaging data structure, a format for organizing and describing outputs of neuroimaging experiments. *Scientific data*, 3(1):1–9, 2016. doi: 10.1038/sdata.2016.44.

- [25] A. Chen, J. Beer, N. Tustison, P. Cook, R. Shinohara, and H. Shou. Removal of scanner effects in covariance improves multivariate pattern analysis in neuroimaging data. *bioRxiv* p, 2019.
- [26] Xiangjie Li, Kui Wang, Yafei Lyu, Huize Pan, Jingxiao Zhang, Dwight Stambolian, Katalin Susztak, Muredach P Reilly, Gang Hu, and Mingyao Li. Deep learning enables accurate clustering with batch effect removal in single-cell rna-seq analysis. *Nature communications*, 11(1):1–14, 2020. doi: 10.1038/s41467-020-15851-3.
- [27] Samuel J Yang, Scott L Lipnick, Nina R Makhortova, Subhashini Venugopalan, Minjie Fan, Zan Armstrong, Thorsten M Schlaeger, Liyong Deng, Wendy K Chung, Liadan O’Callaghan, et al. Applying deep neural network analysis to high-content image-based assays. *SLAS DISCOVERY: Advancing Life Sciences R&D*, 24(8):829–841, 2019. doi: 10.1177/2472555219857715.
- [28] Jake Turicchi, Ruairi O’Driscoll, Graham Finlayson, Cristiana Duarte, Antonio L Palmeira, Sofus C Larsen, Berit L Heitmann, and R James Stubbs. Data imputation and body weight variability calculation using linear and nonlinear methods in data collected from digital smart scales: simulation and validation study. *JMIR mHealth and uHealth*, 8(9):e17977, 2020.
- [29] Haytham M Fayek, Lawrence Cavedon, and Hong Ren Wu. Progressive learning: A deep learning framework for continual learning. *Neural Networks*, 128:345–357, 2020. doi: 10.1016/j.neunet.2020.05.011.
- [30] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.
- [31] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25: 1097–1105, 2012.
- [32] Ting Zhang, Guo-Jun Qi, Bin Xiao, and Jingdong Wang. Interleaved group convolutions. In *Proceedings of the IEEE international conference on computer vision*, pages 4373–4382, 2017.
- [33] Liangru Ke, Yishu Deng, Weixiong Xia, Mengyun Qiang, Xi Chen, Kuiyuan Liu, Bingzhong Jing, Caisheng He, Chuanmiao Xie, Xiang Guo, et al. Development of a self-constrained 3d densenet model in automatic detection and segmentation of nasopharyngeal carcinoma using magnetic resonance images. *Oral Oncology*, 110:104862, 2020. doi: 10.1016/j.oraloncology.2020.104862.
- [34] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.
- [35] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.
- [36] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Man’e, Rajat Monga, Sherry Moore, Derek Murray,

Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL <https://www.tensorflow.org/>. Software available from [tensorflow.org](https://www.tensorflow.org/).

[37] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.