Capturing label characteristics in VAEs: CCVAE

Albert Gimó[1], Lucas Murray[1], and Pau Redon[1]
[1]Ecole Normale Supérieure, Paris-Saclay

1. Introduction

The main focus of this project is the study of the work developed in Joy et al., 2022, where they propose a methodology to generate latent spaces through the well-known VAE models harnessing the availability of labels that express particular properties of the original data. More specifically, in Joy et al., 2022 they use categorical labels by splitting latent vectors into subsets of entries, where each subset deals with the encoding of a particular characteristic via an auxiliary classification task. This encoding is not done through hard coding of the labels in the latent vectors; instead, it is done through the introduction of characteristic latent variables associated with the label, generating a disentangled latent space. This work is particularly relevant in the context of image encoding and generation, allowing not only the prediction of data points but also their manipulation through the characteristic latent variables.

In this project, we extend the formulation of the original paper to continuous labels by replacing the classification task with an auxiliary regression task over the labels. Furthermore, we provide an extensive analysis of the properties of the generated latent space and contrast it with one generated by a regular VAE. For this purpose, we applied this technique to a data set composed of images that correspond to solutions of the firebreak allocation problem in a particular instance, with their associated performance.

Finally, we explore a possible application of this methodology, where the properties of the latent space are used to find relevant data points in this new structured space. This is done by coupling the exploration process of genetic algorithms with the CCVAE space.

The work was divided into three parts: Lucas dealt with the implementation of the VAE, CCVAE, and genetic algorithms, Pau with the study of the latent space and proposal of mutation techniques, and Albert with the extensive study of the original paper, latent space study, and mutation techniques proposal. All of the members of the group have actively engaged with the project throughout its development (idea generation) and have contributed to the write-up. Implementations of VAE, CCVAE, and genetic algorithms are available and open source[1].

2. Paper Introduction

In semi-supervised VAEs data is generally assumed to be split into those with and without labels: given a dataset D, we can split it into $D = U \cup S$, where every $x_S \in S$ comes with its corresponding label y, and $x_U \in U$ doesn't have assigned labels. Most approaches treat y as a latent variable, which may or may not be available. Hence, the latent space is divided into $z = \{z_y, z_{\setminus y}\}$, where $z_y = y$ when y is available, and it is marginalized over when not available. Inputting labels in this way generates several problems: 1) the properties of a data point associated with the label are generally richer than the label itself, encoding the label explicitly into the latent space misses the possibility of capturing such characteristics in a human-interpretable way 2) progressive change of the properties related to the label is not possible since we can only generate modifications by changing the label, 3) inputting labels directly can generate a mismatch between the prior $p(z)$ and $q(z)$.

To address this issue, in Joy et al., 2022 the authors introduce the *Characteristic Capturing Variational Autoencoder* (CCVAE). In this work, the authors side-step the limitations of explicitly encoding labels by, instead, conditioning a subset of latent variables to capture label characteristics. Considering this, they split the latent space in a similar way as previous approaches: $z = \{z_c, z_{\setminus c}\}$, but now z_c is intended to encode information associated with the label, not the label itself. To avoid the existence of coupling between latent variables relevant to different labels, z_c is divided into several disjoint parts z_c^i, named *characteristic latent variables for label i* each containing information corresponding to one of the labels. In contrast, $z_{\setminus c}$ now should encode information relevant to the reconstruction of the image but not necessarily related to the labels. To force the latent variables z_c^i to store characteristics relevant to each label, besides the usual VAE architecture, a set of classifiers are learned to predict y^i from z_c^i:

$$L = \sum_{x \in U} L_{VAE}(x) + \sum_{(x,y) \in S} \Big(L_{VAE}(x) + \alpha E_{q_\phi(z|x)} \big[\sum_i log\ q_{\varphi^i}(y^i | z_c^i) \big] \Big) \tag{1}$$

[1]https://github.com/lucasmurray97/fireEncoder.

2.1 Classification, Conditional Generation and Intervention

The authors argue that a VAE model should be able to perform successfully three main tasks: classification, conditional generation, and intervention. In their work, *Joy et al.* show that the CCVAE framework matches previous approaches for classification tasks (which seems reasonable since the classifiers are trained jointly with the VAE) and outperforms alternative approaches for intervention. This is because now the model is trained to learn characteristics relevant to each label in z_c^i, which can be modified without changing the label itself. This provides the potential for producing multiple samples consistent with the desired modifications and even adapting the magnitude of said changes (e.g. making a face more or less smiling), ultimately resulting in richer intervention possibilities.

Conditional generation turns out to be a harder challenge. Given the loss in Equation 2, it's unclear how to generate new samples from a given label. One approach presented in the paper consists of sampling from the prior for z and only accepting the sampling if the prediction for the labels of interest fulfills the desired characteristics. Although effective, this method can prove inefficient for certain use cases. To solve this problem *Joy et al.* present a modification to the expression of the training loss in 2 by introducing a *conditional generative model* $p_\psi(z_c|y)$ learned along the other networks. Since these modifications are less relevant to our work exposed in the remainder of the report, we transfer the details and discussion of these changes to Appendix A.

2.2 Extension to continuous labels

Although the work focuses mainly on discrete labels, the authors themselves point out that this approach could also be used for continuous labels. Certainly, the classifier on labels z_c^i can be replaced by a regression function $f_{\varphi^i}(z_c^i)$ that predicts continuous labels. As a result, the right-most term in Equation 2 must be replaced by a loss measuring the distance between a predicted label and its ground truth value y^i. As a result, the model should be trained with the following loss:

$$L = \sum_{x \in U} L_{VAE}(x) + \sum_{(x,y) \in S^D} \Big(L_{VAE}(x) + \alpha^D E_{q_\phi(z|x)}\big[\sum_i log\ q_{\varphi^{D,i}}(y^{D,i}|z_c^{D,i})\big] + \alpha^C E_{q_\phi(z|x)}\big[\sum_i (y^{C,i} - f_{\varphi^{C,i}}(z_c^{D,i})^2]\Big) \quad (2)$$

Where the superscripts ^{C} and ^{D} indicate whether a variable is continuous or discrete, respectively. In the following sections, we explore an application of this framework to a problem with continuous labels.

3. The Firebreak Problem

The Firebreak Allocation problem is a classical forestry problem that has received particular interest in recent years due to the progressive increase in wildfires both in terms of frequency and magnitude. This problem is formulated as follows: considering a forest as a grid of cells, each with specific characteristics, one must decide where to allocate firebreaks in this landscape in order to minimize the expected number of cells that are burned for an arbitrary wildfire, subject to a maximum number of firebreaks. The effect of firebreaks is to deviate or stop the advance of an ongoing fire. To generate the data, we used the Cell2Fire Wildfire Simulator (Pais et al., 2019) developed by Fire Management & Advanced Analytics group (where Lucas used to work). The main motivation when solving this problem is to generate firebreaks that achieve a high fitness level according to the Cell2Fire simulator.

3.1 Data Description

The dataset is composed of 50.000 randomly generated solutions of the firebreak allocation problem for a specific forest and their corresponding percentage of non-burned cells, which was computed using the Cell2Fire simulator. The solutions correspond to binary matrices, where 1's correspond to cells being treated as firebreaks, and 0's otherwise. An example is presented in Figure 8(a). Concerning the non-burned percentages, they are continuous numbers in $[0, 1]$. Their distribution over the dataset can be observed in Figure 8(b).

4. The Trained Model

With the data and motivation for the Firebreak Allocation problem established, we now apply the CCVAE framework to address this challenge.

Two models were fitted to the dataset, a regular VAE and a CCVAE, with the intention of comparing their representations. Both models were implemented in almost identical architectures, with the CCVAE adding a regression head to predict the performance of a solution (more in Table 1). Training and validation losses are presented for both models in Figures 9and 10. With respect to model performance, we computed reconstruction metrics for both VAE and CCVAE for both training and validations sets (see Tables 2, 3 respectively). For CCVAE, regression metrics were computed for the performance prediction as well. Both losses and metrics presented in appendix C indicate both models achieve reasonable performance and are well adjusted to the data.

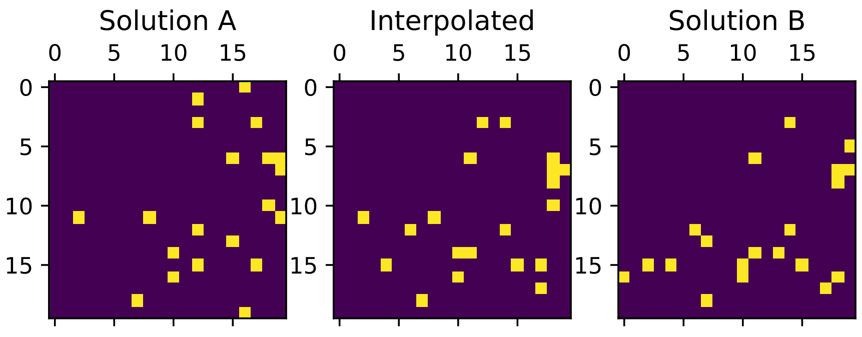
4.1 Latent Space Analysis

We have performed an analysis of the latent space in order to understand the properties of the CCVAE model for this particular problem, ensure good implementation, and provide inspiration for techniques to obtain high-fitness solutions in the Firebreak Allocation problem.

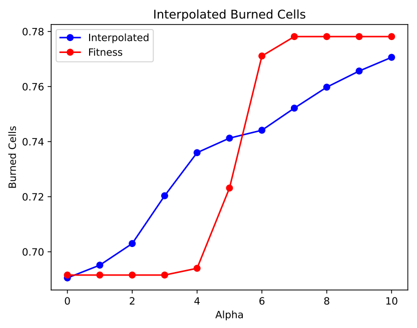
The first experiment done in the analysis of the latent space was to select the two solutions x_A and x_B of the dataset with the lowest and highest fitness respectively, encode them with our CCVAE, and perform interpolation in the latent representations. In Figure 1(a), we can see the result of decoding the interpolation between the two latent space points z_A and z_B (in this case, the midpoint between them, $\frac{1}{2}(z_A + z_B)$). We can see that, while Solution A and Solution B in the figure share very few similarities, the reconstruction of the interpolation shares similarities with both solutions.

A similar experiment can be seen in Figure 1(b), where, we have computed the simulated and predicted fitness of the interpolations $\alpha z_A + (1-\alpha)z_B$ for 10 equidistant values of α between 0 and 1. In the endpoints of the plot the accuracy is very high, while in the interpolated points, there is more error. We give two reasons for this phenomenon. First of all, the simulator is very non-smooth, since a change in one pixel can make the fitness of a solution vary greatly (shown in section B.1 of the appendix). Therefore, the predictor neural net has trouble predicting the fitness close to its abrupt change, since it is very hard for the neural net to translate one-pixel changes to a big change in accuracy. The second reason is very related to the first one, and it is the following one: very close points in the latent space can correspond to the same embedding of a single image. For example, in Figure 1(b) the last 3 points of the latent space correspond to the same simulated accuracy and same figure (more details in B.1 of the appendix). This explains why the predicted accuracy curve is smooth: a single point in the latent space may be obtained by sampling from the normal distribution corresponding to inputs with different fitness values.

In figure 2, we can see how in the subset $z_{\setminus c}$ of the latent space of the CCVAE, the simulated fitness is a main component in the dimension reduction, while in the other cases, this phenomenon isn't observed. This shows that the most important features of the embeddings of the CCVAE have a high correlation with the fitness, as expected, and, more importantly, they are embedded in $z_{\setminus c}$. In addition, in figure 7 in the appendix, we see that fitness is also a main component of the whole latent space z.



(a) Reconstructing interpolated points



(b) Performance of interpolated points

Figure 1: Interpolating in the latent space.

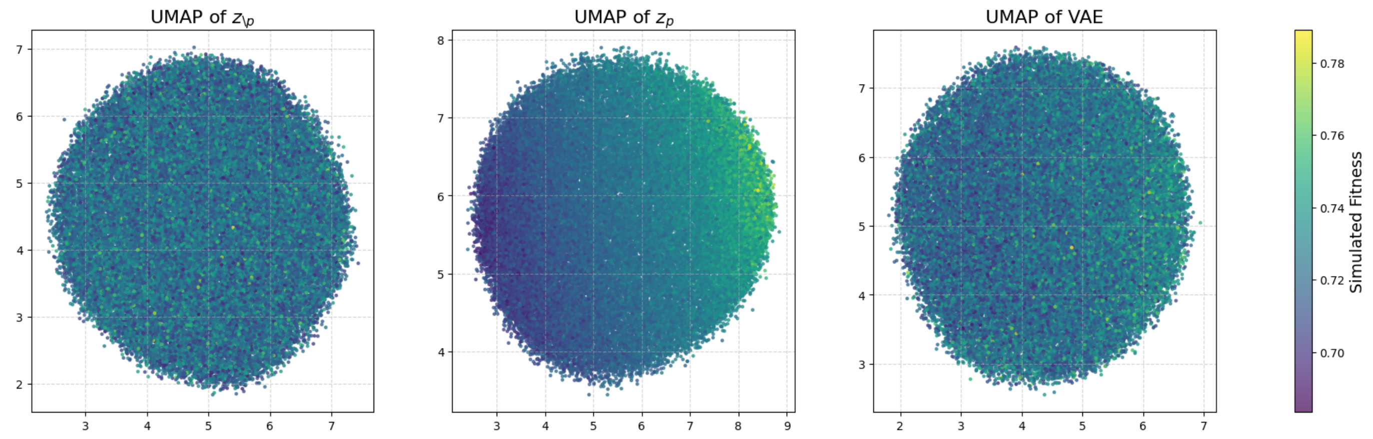


Figure 2: Dimensionality reduction (UMAP) of the embeddings of all the points in the dataset, for both parts of the latent space of the CCVAE (left) and for VAE (right), with points colored by their simulated fitness.

5. Experiments

5.1 Conditional Generation

An approach to conditional generation presented in the paper consists on using rejection sampling, where samples are drawn from the prior $p(z)$ and only accepted if they meet the criteria: $q_\varphi(y|z_c) > \lambda$. In other words, conditional generation is just an inference problem where we seek to draw samples from:

$$p(z|\{q_\varphi(y|z_c) > \lambda\}) \propto p(z)I(q_\varphi(y|z_c) > \lambda) \tag{3}$$

Through this process, samples from the posterior can be conditioned on exhibiting certain properties to different levels. We test out this conditional generation method 2 by sampling from the prior $p(z)$ and evaluating the generated embeddings on the regression head. We tried different value thresholds for the rejection sampling and evaluated the corresponding reconstructions on Cell2Fire (Pais et al., 2019), results are shown in Figure 3.

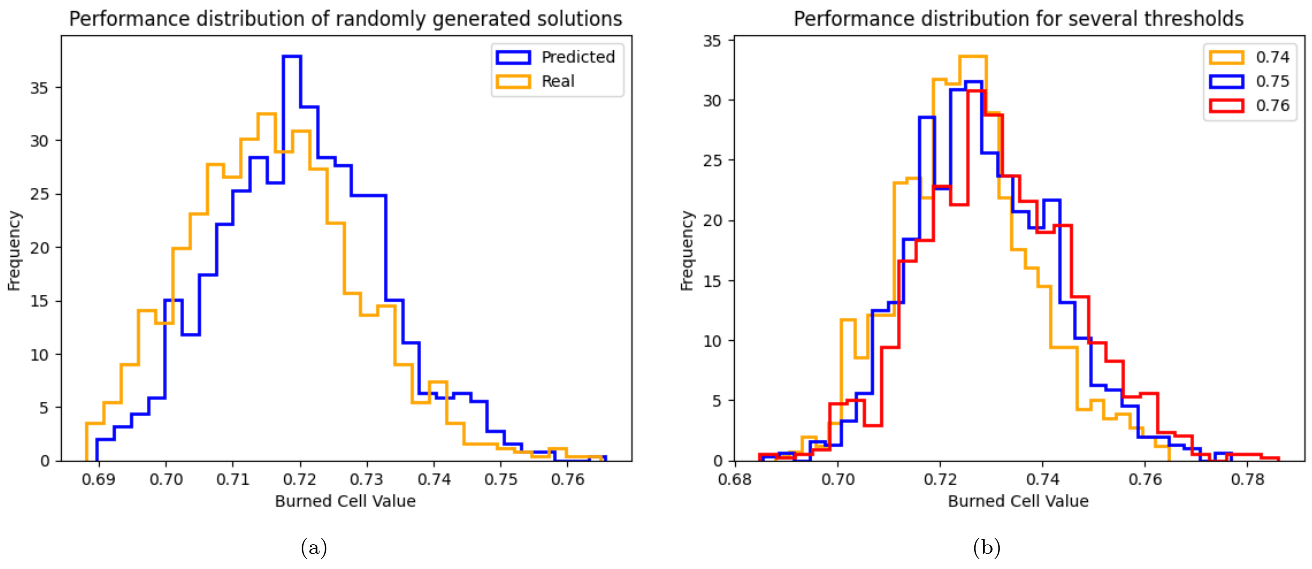


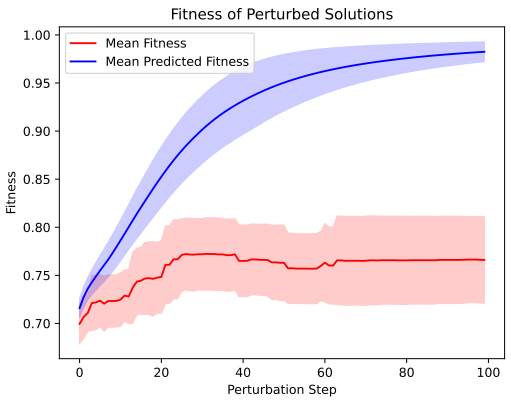
Figure 3: On the left, distribution of real and predicted performances of 1000 samples without conditional generation. On the right, the distribution of real performances of 1000 samples conditionally generated for various thresholds.

As can be seen in Figure 3(a), sampling from the prior approximates the original data distribution, both when analyzing the predicted and simulated fitness, where this last shows a small shift toward smaller values. By means of applying rejection sampling with the rule: $I(q_\varphi(y|z_c) > \lambda)$, the distribution of simulated fitness can be effectively shifted by modifying the values of λ. This is shown in Figure 3(b), where higher values of λ yield a shift of the distribution toward higher values.

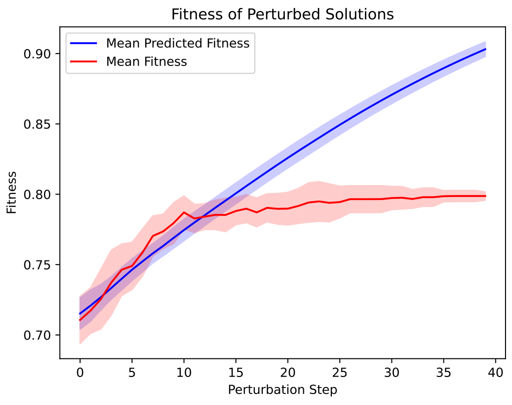
5.2 Interventions

A main feature of the CCVAE model is the ability to perform interventions. With a vanilla VAE, directed interventions are not possible, since we have no clear way to relate the latent space with the variables we want to intervene on. Nonetheless, in our CCVAE the features of half of the latent space are related to the fitness, and thus, given an embedding of an observation, we are able to improve the fitness of a data point.

One way to intervene on an embedding is to treat the latent variables $z_{\setminus c}$ as parameters, freeze the predictor network, and apply a gradient descent step to maximize the predicted fitness. Then, for a given embedding, we are intervening by moving half of the embedding space in the direction of maximum fitness and leaving the rest unchanged. The result of applying this procedure repeatedly is shown in figure 4(a). In this figure, we can observe how the mean simulated fitness improves until around 20 iterations, where the fitness reaches a maximum. We argue that the improvement stops because of the new embeddings being out of distribution (in the training dataset there are only solutions with fitness below 0.77). In a similar way, we computed the direction between the solutions that maximize and minimize the fitness in the dataset and moved solutions towards this direction (for the fitness half). Results are shown in Figure 4(b) and are similar to those using gradient descent, with improvement plateauing for the same alleged reason.



(a)



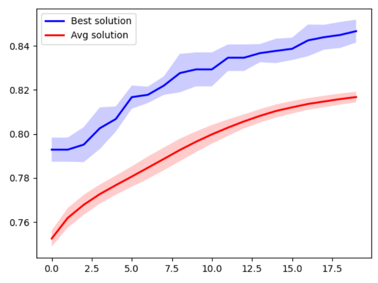
(b)

Figure 4: On the left, results of applying the gradient descent procedure on the latent variables z_c in the direction of maximum Fitness. On the right, the results of moving solutions toward the direction between maximum and minimum solutions in the dataset. The results are averaged over 100 initial embeddings of dataset solutions.

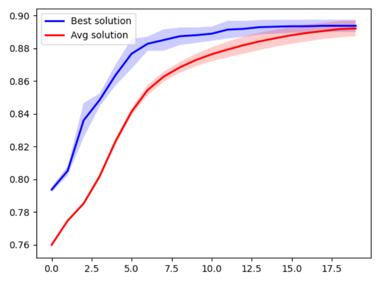
5.3 Fitness maximization

Considering the nature of the data we worked with, a natural application of this work was to consider the possibility of devising an algorithm that made use of the properties of the latent space that were discussed in the previous section. A natural choice for this family of algorithms was the use of genetic algorithms, replacing classic mutation and cross-over rules with ones that leverage latent space properties. For this purpose, we implemented a vanilla, VAE-based and CCVAE-based versions of genetic algorithms that sought to improve the performances exhibited in the dataset. Cross-over and mutation rules for the vanilla algorithm consist of mixing two solutions equitably and modifying randomly a fixed % of the cells, replacing their values with their opposite. With respect to VAE and CCVAE, we used as a cross-over operation the interpolation between solutions, while for mutations we considered sampling from the approximate posterior. Additionally, we implemented a mutation version based on the conditional generation process shown in 3.4 (CCVAE-CG) and one that applies the gradient descent intervention described in 3.5 (CCVAE-GD).

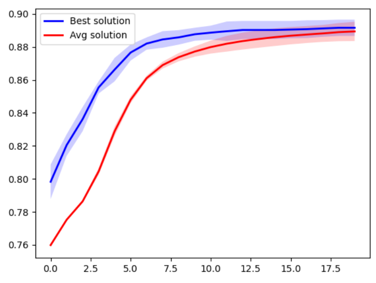
As can be seen in Figure 5, there is a clear improvement when comparing the Vanilla Genetic Algorithm with the VAE-based one. The algorithms have a much steeper improvement, achieving better results after fewer iterations. The contrast between VAE and CCVAE-based algorithms is far less apparent. General trends are maintained, both in terms of the maximum fitness obtained and improvement rate. Nevertheless, taking a closer look at CCVAE-GD and CCVAE-CG, both achieve marginally better finesses than VAE and CCVAE. This suggests there might be some advantage to custom-made mutation rules. But again, improvements are modest, and further study is required to assert this with confidence.



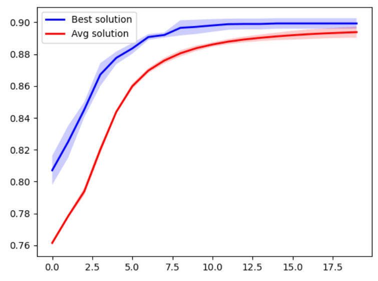
(a) Vanilla



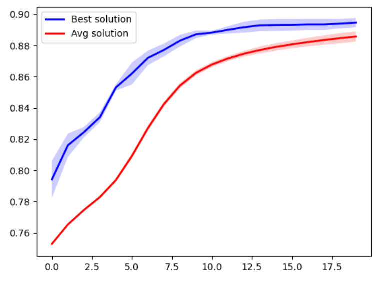
(b) VAE



(c) CCVAE



(d) CCVAE-GD



(e) CCVAE-CG

Figure 5: Results for 20 iterations of the genetic algorithms, each plot shows the average of 8 runs with it's corresponding standard deviation.

6. Discussion

We have presented an application of the CCVAE model to the firebreak allocation problem. In this application, we have extended the scope of the original paper to admit continuous variables and implemented the three main motivations of the semi-supervised VAE (prediction, intervention, and conditional generation), as well as the main motivation of the CCVAE (directed interventions). We have successfully implemented prediction, as can be seen in table 3 in the Annex. For the intervention task, we have proposed a directed intervention based on gradient descent that locally provides good results. Finally, we have implemented conditional generation and found good results for in-distribution fitness values.

The main challenge we have faced in tackling this problem has been out-of-distribution (OOD) data. The main objective of the Firebreak Allocation problem is to generate solutions that have a high simulated fitness - in other words - generate out-of-distribution samples. As we have said, this issue can be spotted in figure 4(a), where the latent space point exits the training distribution after a few gradient steps, so the gradients stop being informative about the real fitness. Moreover, the conditional generation algorithm we have used is not suited to generate OOD solutions, since it samples from the model's learned distribution. We conclude that the "vanilla" CCVAE is not suited to tackle the generation of out-of-distribution samples. One possible solution to this problem would have been data augmentation. Given our ability to produce samples with higher simulated fitness (see figure 5 for example), we could add these high-fitness samples to our initial dataset, in order to extend the fitness interval in which the model has been trained. To overcome OOD, we have coupled our intervention and conditional generation results with genetic algorithms. Given that at each iteration the samples with high simulated fitness are conserved for the following iteration, we overcome the problem of low fitness results due to OOD data.

Some of the methods developed in this project could be applied to real-world firebreak allocation problems. For instance, in cases where an actual firebreak exists, interventions in the direction of gradient descent could be useful to change the structure of the firebreak minimally while greatly improving its functionality.

Overall, we have proven that the CCVAE framework can successfully be extended to continuous variable problems. However, the nature of our problem (i.e. the search for OOD solutions and the existence of only 1 label) makes it harder to exploit the advantages of the CCVAE framework. With perspective, although the results for this specific problem are satisfactory, we could have chosen better applications to showcase the CCVAE's strengths had we understood the original work a few weeks ago as well as we do now.

References

Joy, T., Schmon, S. M., Torr, P. H. S., Siddharth, N., & Rainforth, T. (2022). Capturing label characteristics in vaes. https://arxiv.org/abs/2006.10102

Pais, C., Carrasco, J., Martell, D. L., Weintraub, A., & Woodruff, D. L. (2019). Cell2fire: A cell based forest fire growth model. https://arxiv.org/abs/1905.09317

Appendix

The source repository for all the implementations and notebooks is https://github.com/lucasmurray97/fireEncoder

A. The conditional generative model

In their work, Joy et al., 2022 present the *conditional generative model* $p_\psi(z_c|y)$. This model, given a value for the discrete label y^i models a distribution over the possible latent variables z_c^i values that might correspond to inputs with these labels. In this section, we explain this model and its uses. Finally, we explain why it is not very useful in our specific use case (and therefore we haven't implemented it).

Performing conditional generation using the method in subsection 5.1 can prove to be inefficient in certain cases. For example, if the desired values for a label don't appear often (say in a 0.1% of the input images, then many values of z will have to be sampled and reconstructed in order to obtain an image corresponding to the desired label value. This is even more true for continuous labels or discrete labels that can take multiple values. Having access to a NN that models the probability distribution from y^i to $z^i_{\setminus c}$ can greatly simplify and accelerate this process. To generate the desired characteristics in an input, one can just sample from the distribution $p_\psi(z_c|y^i = d)$ where d is the desired value of the label in the reconstruction.

This also allows diversity and graduality in generations. *Diversity* because several values for the characteristics can be replaced into an embedding, producing different outputs. *Graduality* because this allows for interpolation between characteristics (or selecting characteristics that give different probabilities between the classes.

In our specific use case, the Firebreak Allocation problem, fitting such a model would not have been very useful. There is no interest in finding solutions with a level of fitness that can be found in the data. It would instead be useful to sample from unseen (high) values of the label. However, it would be unreasonable to expect such an approach to work, since it would only be operating out of distribution (and we have already discussed the limitations of operating OOD in previous sections). We instead are more interested in performing small modifications that can improve the performance. In our work we show that a conditional generative model is not required to make such interventions successfully.

B. Misc.

B.1 The simulator is not smooth

In figure 6, we can see how changing a single pixel can produce an increment of 4% in the simulated fitness. Moreover, at the beginning of the interpolation, the generated solutions from different latent space points are identical.

B.2 CCVAE complete latent space

In figure 7 we can see the plot of the complete latent space.

C. Figures and Tables

This section contains some additional plots not included in the main body of the work.

Data Description

The inputs to the models are 20x20 binary tables with exactly 20 1s. The simulated performance of the inputs is distributed following (roughly) a normal distribution.

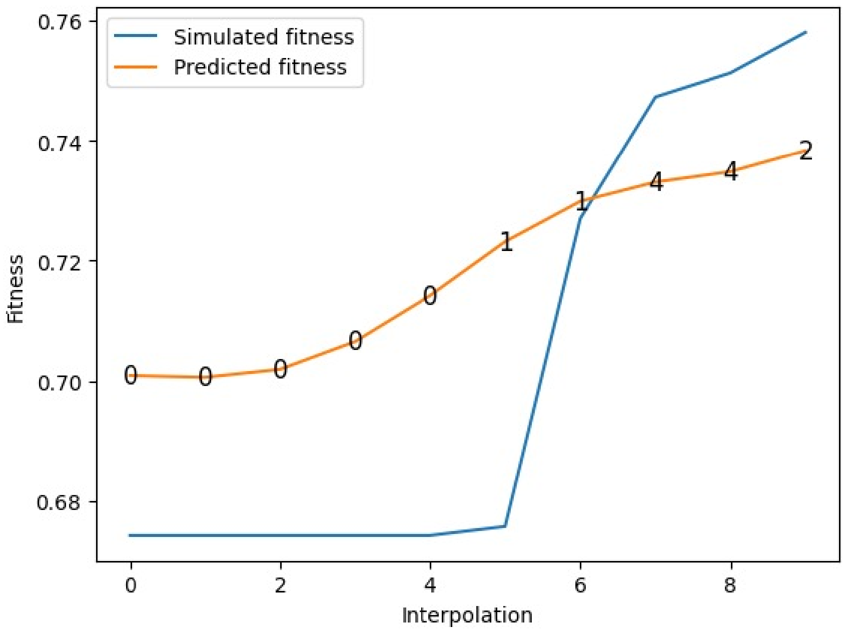
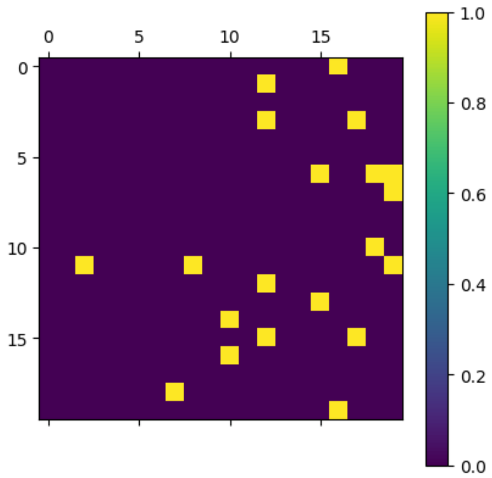
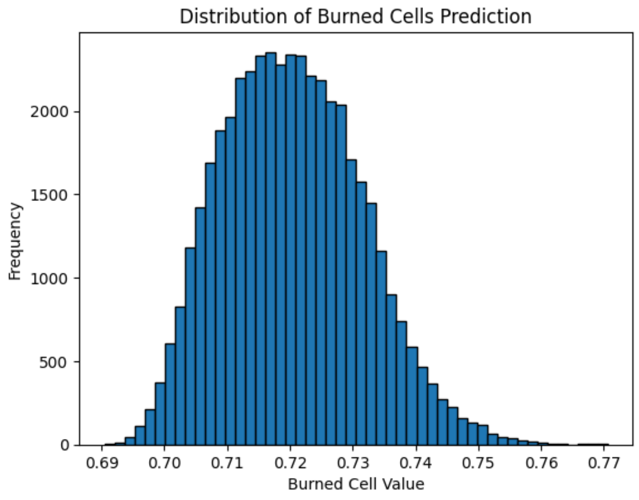


Figure 6: Similarly to figure 1(b), we have plotted the simulated and predicted fitnesses for interpolation between two points of the latent space. Additionally, in this figure, the number shown on top of the predicted fitness line represents the number of pixel changes in the reconstruction of the latent spaces.



(a) Example Solution



(b) Distribution of performances in Dataset

Figure 8: Data description

Model Architectures

The following table illustrates the similarities and differences in the two models' architectures.

	VAE	CCVAE

Encoder	conv(4, 128)
	conv(128, 256)
	Linear(6400, 256)
		conv(4, 128)
		conv(128, 256)
		Linear(6400, 256)

Decoder	Linear(256, 6400)
	t_conv(256, 128)
	t_conv(128, 1)
		Linear(256, 6400)
		t_conv(256, 128)
		t_conv(128, 1)

Head	-	Linear(128, 128)
		Linear(128, 64)
		Linear(64, 1)

Table 1: VAE and CCVAE network architectures

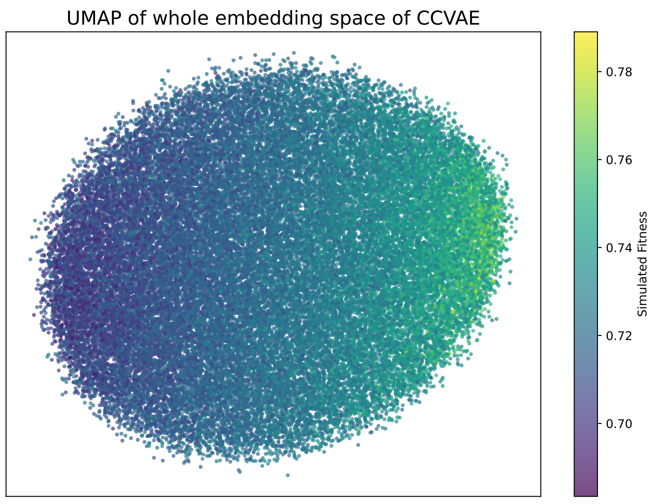
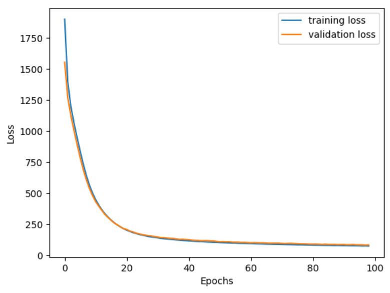


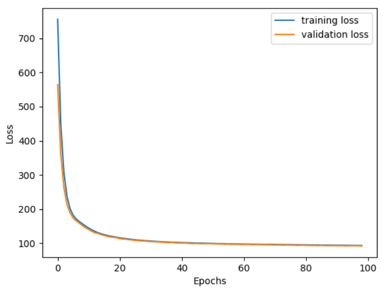
Figure 7: UMAP dimensionality reduction of full latent space, for all points in the training dataset.

Model Fitting

In what follows, plots and tables showing the losses for the trained model in shown.



(a) Reconstruction Loss



(b) KL-divergence Loss

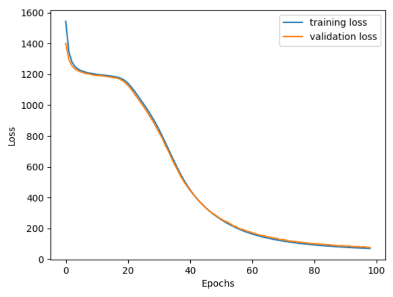
Figure 9: VAE Losses

	accuracy	precision	recall
train	0.99	0.99	0.99
validation	0.99	0.99	0.99

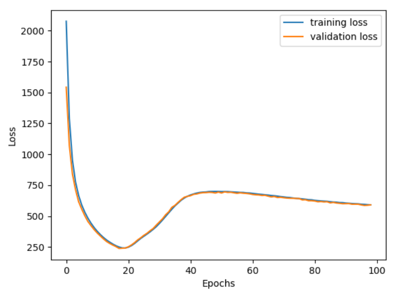
Table 2: VAE

	accuracy	precision	recall	MAE	MAPE
train	0.99	0.99	0.99	0.004	0.005
validation	0.99	0.99	0.99	0.004	0.005

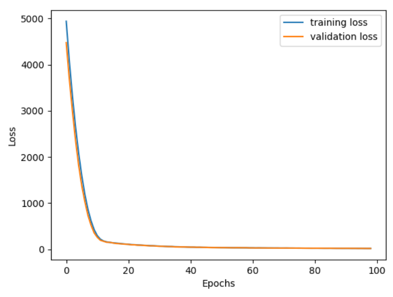
Table 3: CCVAE



(a) Reconstruction Loss



(b) KL-divergence Loss



(c) Regression Loss

Figure 10: CCVAE Losses