Architexa

High-level description:

Context:

This project is an example of an image generating AI, which means it is designed to enable people to generate images, or artworks, in a small amount of time based on textual prompts.

These kinds of systems are useful both to facilitate personal expression, and for use as prototyping and design aide tools (ours being the latter kind).

When this system is finished; users will be able to access our website, and input textual prompts corresponding to one of our supported classes, and then get back a randomly seeded new and unique image within that class.

Also, as a side effect of our efforts we have created modular case tools for database population and hyperparameter optimisation which can be substituted into other systems for use by more technical users.

Architecture:

The solution currently has four major components, each of which has been assigned a development team. First, there is the CGAN model itself, which consists of two adversarial neural networks and is responsible for taking in data (conditioned with prompts) and outputting generated images.

Secondly, we have the hypertuning, which performs the computationally complex task of tuning the model's hyperparameters efficiently by employing multi-threading (and distributed computing) to run multiple training branches simultaneously (as well as allowing partially trained models to be saved).

Thirdly, a proprietary dataset consisting of images of buildings labelled with various features (including the style and purpose of the building), as well as the CASE tools and methodologies developed to construct said dataset.

Finally, a website written directly in HTML5, CSS3 and JavaScript, hosted on the Firebase web hosting services. It will have a sign up, login, home, terms, and about pages. The login pages are set up with Firebase tools using Javascript. Furthermore, JavaScript is used for the dynamic interaction aspect of the website, which includes internal navigation and the calling of the model API from a remote server location.

(make this more descriptive of the problem we are trying to solve and maybe throw in a diagram of our primary use cases, e.i calling the model, or using one of our case tools)

Divergence from planned design:

We had originally anticipated that the bulk of the effort for this module would lie in the construction of an effective model. We would be able to use industry standards for our hyperparameters and rely on an existing dataset for training.

However, in reality multi labelled datasets are unusual (and are in fact, much more a limiting factor in the construction of a multi-labelling CGAN than upgrades to the architecture itself), and it became necessary to produce our own proprietary dataset (which, due to the lack of appropriate images on line consists largely of images generated by the mid-journey application)

Conversely, it became apparent that it was too easy to identify effective hyperparameters using existing research (as all image generators use very similar ones). Therefore, we created CASE tools to run multiple training loops with different hyperparameters (as an exercise in optimising hyperparameters), which is optimised using distributed computing to allow multiple versions of the algorithm to run simultaneously.

Due to the extra difficulty in completing these elements and the small size of the proprietary dataset concerning professional offerings, the system's function is slightly restricted to only simple (though still multi-class) prompts.

Data

Data Collection:

For this project, ***BulkAI*** will be used with ***Midjourney*** to retrieve image training and testing datasets. BulkAI is an automation tool designed to interact with AI image generators like Midjourney, which can create bulk submissions of prompts. This capability allows for the generation of architectural images on a large scale, which will be key to our dataset development.

BulkAI's primary function in the data collection workflow is to automate the submission of numerous prompts to Midjourney. This process can be executed hundreds of times without manual intervention, significantly enhancing the efficiency of data collection. Following the generation of images, we can efficiently download the outputs in bulk directly from the Midjourney gallery. The images generated through this process are considered our own creations, ensuring compliance with copyright policies and avoiding any legal complications related to image rights.

Midjourney, on the other hand, is an advanced AI image generator that interprets textual prompts to produce high-quality images. This tool's capability to generate diverse and high-resolution architectural images based on textual descriptions is significant in creating a varied and comprehensive dataset tailored to our project's needs.

Integrating BulkAI with Midjourney presents a significant advancement over traditional data collection methods, often involving manual scraping from the internet. Such traditional methods are time-consuming and prone to inconsistencies and the need for extensive post-collection processing. The chosen approach furthers the data collection process. It ensures high consistency and relevance in the dataset, which are critical factors for effectively training our Conditional Generative Adversarial Network (CGAN).

The initial method was to scrape images on Pinterest or on Google search but noticed that the image variation and quality of image was too extreme so it was decided to scale the project back onto simulated images as this will be much more accurate to the conditions we want to test it on.

As the project progresses, we plan to further enhance our data collection strategy. By expanding the variety of prompts and potentially integrating additional image sources, we aim to improve the dataset's diversity. These improvements are expected to refine the realism and accuracy of the architectural visualisations produced by our CGAN, enhancing the overall quality and applicability of our project outputs.

Harvard Reference for BulkAI: Igolaizola, I. (n.d.) BulkAI. Available at: https://github.com/igolaizola/bulkai (Accessed: 12 March 2024).

Midjourney. (n.d.) Documentation. Available at: https://docs.midjourney.com/docs/midjourney-discord (Accessed: 12 March 2024).

Data Preprocessing:

For this section, A Python Script will be used to perform the following techniques on the images:

Resizing: Ensure all images are to a consistent dimension to maintain uniformity during model training.

Augmentation: Applying transformations like grayscale conversion, rotation, and possibly flipping or slight distortion. This will allow the model to become more robust to such changes, improving its ability to generalise from the training data to prompts.

Normalisation: The pixel values of the images will be normalised. This means adjusting the pixel values to have a similar range, typically between 0 and 1 or -1 and 1, which can help stabilise the training process and improve the model's fit.

Noise Addition: Introducing slight random noise to the images will make the model more robust and less likely to overfit the training data. This simulates real-world imperfections in images, preparing the CGAN for a broader range of inputs.

Splitting Dataset: The dataset will be split into training and test sets. This split is crucial for accurately evaluating the model's performance and generalisation capabilities.

Architecture:

Like all neural networks, the generator and discriminator components learn by calculating a loss function (some metric of the gap between expected and actual output) and using a procedure called backpropagation to update node weights accordingly. What is unusual about these networks is that the loss function for the generator is, in fact, a heuristic for the inverse of the discriminators performance (which makes to combination of the two networks an adversarial network since improvements to the accuracy of one network reduce the accuracy of the other)

Generator:

```
Input1          input2
|               |
V               V
Concatenation
```

```
|
V
Transpositional Convolution←——--|
|                                       |
V                                       xn
Batch Normalisation                     |
|                                       |
V                                       |
Elu——------------------------------------|
|
V
Convolution
|
V
output
```

Justification for network shape:

layer 1 - the first input is a batch of white noise in the shape of an array of feature maps (ready to be upsampled by the Transconv layers, and the second is a prompt processed into the same dimensions as the noise.

layer 2 - these inputs are concatenated so that the prompt is treated as an additional colour stream (e.i R,G,B, prompt)

Layer 3 - the input is then passed through a transpositional convolution layer, which means they go through the mathematical process of transposition convolution. This consists of the input being broken down into subsections of size n, then those subsections being multiplied by the contents of each n-sized subsection of a > n-sized filter, producing intermediary regions of size > n, the overlapping areas of the intermediary regions are summed leaving a final result of size greater than the input conditioned on the learned filter (the effect of this is to combine the conditioned random feature maps (in a manner learned from feedback from the discriminator to compile noise as if it correspond to logical features) into high order features until they correspond to a set of images)

Layer 4 - whilst mathematically complicated to implement, the purpose of a batch norm layer is simply to clip data
so that it has a mean of 0 and a standard deviation of 1. This reduces the extent to which
changes to the weights before the previous layer affect the distribution of data input to the next layer.

This, in turn, reduces the need for layers to update their weights to compensate for changes made to the weights of previous layers, thus speeding up training (sometimes at the cost of accuracy).
Layer 5 - This is a dense layer (where the sum of outputs from nodes in the previous layer is passed to each node in the dense layer) with an Elu activation function; this means that the input is raised as an exponentiate of e in some function with exponential degree. The result of this is that values subtending left of 0 approach -1 and values subtending right of 0 approach infinity (which means that data entering this layer can be split into categories based on its sign (allowing for nonlinear functions, where different ranges of values are subject to different processing) to be learnt. It is distinct from a Relu function (which sets all values less than 0 to 0) in that values on either side of the boundary are not all treated the same,

and thus Elu can be used where true function is extremely non-linear and inexact splitting can be used a heuristic to account for splits at a lower granularity than is used).

layer 6 - the data is then passed through a convolutional layer, meaning that it goes through the mathematical process
known as convolutional; whereby features are extracted from a multi-dimensional data structure by splitting it into
a complete set of subsets (sometimes with subsets overlapping) and summing the values of each subset, with the value in each subset conditioned on the learned values of a filter. (the general purpose of such a filter is to extract feature maps into smaller ones whilst removing logical noise, and this typically results in a larger number of smaller feature maps; however, in this instance, the first two dimensions of the output must be the same as the input, and the input is a number of (5.2) potential generated images since the only dimension available to trim is the number of images (as opposed to their dimensions) this layer, therefore, serves to select the learnable best image to generate from the options.

Preprocessing of data:

Before reaching the generator, the image-shaped noise goes through a further batch normalization and Elu layer, which normalises the data and arbitrarily splits it (leading to a lower chance of dis-convergence but greater generalisation error)

Before reaching the generator, the label is embedded as a vector of arbitrary length and then goes through a dense layer with an output shape equal to that of the image (this allows multiple class labels to be encoded as part of the embedded vector and transforms the shape of the input to match that of the other stream (allowing concatenation))

Discriminator:

```
input1    input2
|         |
V         V
concatenation
|
V
convolution<----|
|               |
V               |
batch norm      |xn
|               |
V               |
ReLu------------|
|
V
flatten
|
V
dropout
|
V
output
```

Justification for network shape:

layer 1 - the first input is a batch of 2d images and their colour streams, the second is a prompt processed into the same dimensions as the image.

layer 2 - these inputs are concatenated so that the prompt is treated as an additional colour stream (e.i R,G,B,prompt)

layer 3 - the data is then passed through a convolutional layer, meaning that it goes through the mathematical process
known as convolutional; whereby features are extracted from a multi-dimensional data structure by splitting it into
a complete set of subsets (sometimes with subsets overlapping) and summing the values of each subset, with the value in each subset conditioned on the learned values of a filter.. (each of these subsets is known as a feature map and is learned as a logical component of the image being discriminated, this kind of convolution works to split them into increasingly fine and precise features (thereby eliminating noise) and making the resulting core data easier to classify))

layer 4 - the data is then passed through a Relu function, meaning values greater than 0 remain the same, and values less than 0 become 0. This is important as it causes certain ranges of values to have different
gradients, allowing the network to learn to fit the data to a nonlinear function by weighting certain inputs to land in one or the other range at this level.

layer 5 - whilst mathematically complicated to implement, the purpose of a batch norm layer is simply to clip data
so that it has a mean of 0 and a standard deviation of 1. This reduces the extent to which
changes to the weights prior to the previous layer affect the distribution of data input to the next layer.

this in turn, reduces the need for layers to update their weights to compensate for changes made to the weights of previous layers, thus speeding up training (sometimes at the cost of accuracy).

layer 6 - flatten layers simply take n > 1-dimensional data structures and replace the data in a linear list, So, at least a 2-dimensional update of a convolutional layer can be fed to a linear layer.

layer 7 - a dropout layer is used to prevent overfitting by randomly setting some of the data it is input to 0, and thus adding random noise to the output. this forces the discriminator to get used to classifying arbitrarily
divergent data at every epoch, which forces it to ignore distributions that may appear only in training data (and be damaged
by dropout) and focus on whatever distributions all valid data must have in common.

Layer 8 - This layer is a not actually a layer, but the point at which the information from a given forward propagation of the network is passed back to whatever function called it, note that this output will take the form of a feature map expressing the discriminators best guess at mean average values for all features within the distribution of images passed to it.

Notes on optimisation:

Whilst the implemented architecture will be very similar to the above networks, each data distribution requires a network to be carefully fine tuned to it before it can run properly. Therefore elements such as the networks loss function, filter size, batch size, latent dimension, learning rate, optimizer, and number of residual layers, are not mentioned here as they will be subject to change right up until the demonstration, as we attempt to optimise the models performance.

Preprocessing of data:

The image passed from the generator is not pre-processed before entering the discriminator network.

Before reaching the generator, the label is embedded as a vector of arbitrary length. It then passes through a dense layer with an output shape equal to that of the image (this allows multiple class labels to be encoded as part of the embedded vector and transforms the shape of the input to match that of the other stream (allowing concatenation)).

Support:

Initial Training loop:

The initial training loop based on the original GAN paper. This training process pits the generator and discriminator against each other with the generator trying to fool the discriminator and the discriminator trying to classify images as real or fake. This was one of the first GAN training processes to be developed and uses binary cross entropy (BCE) to determine the losses for both models. The discriminator's loss is correlated to how many fake and real images it classifies incorrectly while the generator's loss is correlated to how many fake images the discriminator can correctly identify. In practice, this means that the losses of both models are opposed to one another, when one is high the other is low. This is important as the higher the loss the larger the updates to that model will be each batch.

Hyperparameter optimiser:

As well as the training loop our project will also use a simple script to run multiple versions of the training loop at once, often on different computers, with different values for key hyperparameters (such as the learning rate for both networks, the batch size, and the latent input dimension) and display the results of this training to users upon completion, so that they can assess whether the current set of values are more or less optimal than the previous ones. This can be easily extended by adding in more hyper parameters into the search space.

The Training Process:

The training process initially started with hypertuning the models that had been created with a classic GAN type of training. This type of training suffers from a few flaws however and this became apparent within 3-4 days. Firstly, if one model is much more complex than the other it will overpower and dominate the other model. This leads to complete breakdown in training where no noticeable increase in image quality happens. This was the first thing we experienced and many small changes were made such as increasing the amount of “blocks” in each model to help balance both of them correctly. Each run took 2 days when run to 200 epochs and it took around a week of painfully slow progress before we got a run that generated something resembling a building. However, it was very blurry and dark artefacts all over the image. This was our fatal flaw in our training methodology as we believed this was purely due to having imperfect hyper parameters like learning rate and batch size. It took a further week of tuning before we realised this was actually a flaw in the GAN training process. A process called mode collapse can cause the generator to find and exploit a vulnerability in the discriminator so it can get a certain type of image past. If this happens the discriminator usually cannot recover and the image quality will not improve and in fact get worse.

After noticing this we researched a more stable version of this training process known as WGAN or the even better WGAN-GP. This training process improves on the older GAN process in a number of key ways.

Instead of being a classifier (by mapping the output to 0-1 to be a probability) the discriminator becomes a classifier simply giving an image a score based on how “real” it looks. In the WGAN framework the discriminator's job changes from being a classifier to being a “critic” giving each image an objective rating. This opens up a world of mathematical formulas to define loss rather than just tying it to how many the discriminator gets right. WGAN specifically uses a formula called wasserstein’s distance also called earth mover distance in some cases.

This means we can edit what the losses are tied to. The critics job now becomes to maximise the difference in score between real and fake images. This can be thought practically as being the larger the difference the more confident the critic is that real images are real and fake are fake. The generator's loss on the other hand is trying to minimise the difference in scores. As it cannot influence the real scores its loss is tied to the negative of the average score of fake images. This means that because the critic is giving an objective score for how real a fake image looks, the critic will have a lower loss (and therefore update less) only when it's producing more real images. Mode collapse can still happen in this process but it does dramatically reduce the chances of this happening early in training when compared to GAN. The other change in WGAN is that the critic needs to be much more powerful than the generator to give it effective feedback. This is achieved by training the critic n more times for every generator update. For us we set n = 5.

WGAN also has a flaw however due to how it enforces something called the Lipschitz continuity condition or constraint. This constraint is essentially the thing that enforces that the scores be “objective”. This is done in the original WGAN paper by “clipping” weights. This puts an upper bound on how large any-one weight can be and if it goes above it is simply redefined to be this upper bound. This is a poor way to enforce the constraint however as a weight 5% above and 500% above will be clipped to the same value.

This is where the WGAN-GP framework comes in as it replaces clipping with a gradient penalty. This is a term added on to the loss calculation of the critic which penalises large gradients. We calculate this by combining a real image and a fake image and getting the score. The score should be proportional to the proportions of real vs fake in that image. This discourages large drop offs where images could be getting more realistic but not changing their score much until hitting the dropoff and having the change be reflected in the score.

Overall we saw huge improvements during these changes in training process, and should we have had the time WGAN-GP would have been sufficient to provide clearer and higher resolution images but due to the large training time we were only able to run a handful of tests of this new variant before having to commit to the final model training.

System interface:

The user interface is a web application and is built using the languages of the web HTML, CSS, and JavaScript. Vanilla JavaScript is used because of its familiarity, which allows for quicker development cycles and its compatibility on all modern browsers without needing transpilation, and faster loading times. The IDE used is Visual Studio Code for its great editing and debugging capabilities. The entire codebase of the web application has been developed on Visual Studio Code. After finishing the first version of the web application we deployed it on Firebase web hosting services. The choice of hosting our website on Firebase is due to its developer friendly features such as, user authentication, analytics, database, and server side functions. Architexa incorporates the user authentication aspect to validate users, for managing and tracking usage accurately. Furthermore, to understand how our services are being used to carry updates and maintenance.

The layout of the information is organised in two main components; the main content of the page is displayed in the middle of the screen, and to the left is a sidebar that contains the links to the other pages. When designing the branding and colours, we were aiming for a minimalistic, sleek look that encapsulates the futuristic aspect of using AI technology. For a good user experience we used CSS flexbox to provide a responsive design that functions on various screen sizes.

When a user enters a prompt in the input bar the web application uses an API call to fetch the generated designs from our AI model hosted on a Render server. (add a diagram showing how this networking is done)

For the API (Application Programming Interface) we adapted a Flask-based REST API. Flask is a web framework that is designed to build APIs, and it's conducted using python language. A REST API, sometimes referred to as a RESTful API, is an application programming interface (API) that adheres to the principles and limitations of the REST architectural style. It enables communication and interaction with RESTful web services. REST is an acronym that stands for representational state transfer. This is necessary to accommodate the AI model within the hosting provider's environment ensuring stability, compatibility and enhancing the performance of the overall model.

In our python code, To ensure sending the image over HTTP as a string we had to transform binary data into characters, so we used the base 64 encoding. We also imported the Flask library.

The function is to retrieve the photo generated from the image generator to the website. When it receives a GET request from our'/hi' endpoint, the home() function is called. The AI model is called within the home() function to produce an image. Subsequently, the AI model constructs a neural network then loads the weights in from generator.h5 to produce an image. The model utilises the noise and label input to produce an image, which is further processed and stored as a JPEG file. The disc attempts to read the generated image. Then, the image data is transformed into base 64 string. This transformation is important because the format used for the response (JSON) is unable to process data which is not preprocessed, which means it can only process binary data if it goes through a string data transformation. This is possible by using base 64 encoding. If the location of the image file can't be found, the server will respond with an error message and a 500 status code, which signifies an internal server fault.

Evaluation:

For our evaluation we have to rely heavily on black box testing since neural networks (especially highly complex ones) are inherently black box systems, meaning that it is impossible to understand how their set of learned weights corresponds to their output. Equally, it is challenging to judge the quality of our output mathematically, and so we have set a benchmark of 'human recognisability' for output quality.

With these provisions, our testing scheme consists of unit testing for the individual components, integration testing of their combined operation, and a little bit of performance testing (made highly necessary by the resource constraints of AI systems).

This means that we will begin by testing that all the functions of the individual components operate correctly in a vacuum (being fed dummy) data, before we test them to see if they work in tandem. This is important because it allows us to isolate the location of errors (e.i if the neural networks and the training loop both work, but will not run together, then the problem lies in their cohesion).

Beyond this, we will run performance checks after the basic functionality of the code is confirmed, so that we can ensure that our use of our limited processing resources is optimised, and that as many users as possible can access the system as quickly as possible.

Unit testing:

Neural network:

Test:	Outcome:
Neural network can successfully learn some features	
Neural network can learn to differentiate distinct classes	
Neural network can produce a human-recognisable image in response to a stripped version of the rock-paper-scissors dataset containing only rocks	
Neural network can produce a human-recognisable image in response to all 3 prompt combinations of the rock-paper-scissors dataset listed on github	

Neural network can generate an image in less than 60 seconds	
Neural network can be trained in less than 2 days	
Neural network can be altered to accept images of size 64*64, or 256*256 and still function	
Neural network can be trained for 1 epoch without crashing an average computer	
Loss values for the discriminator and generator converge when trained using both the rock-paper-scissors data set and the multi-category dataset on GitHub.	

Dataset:

Test:	Outcome:
All images are used legally and without civil liability.	
Each category combination has at least 100 images in it.	
A random sample of AI-generated images (of size 100) contains only images that humans can correctly categorise.	
Dataset can be transmitted in less than 10 minutes over university wi-fi.	
A random sample of 100 images from the dataset are all the same size.	

Optimisation tool:

Test:	Outcome:
Optimisation tool can save an example neural network in a partially trained state.	
Optimisation tool can run simultaneously on two physically distinct computers, two distinct virtual machines, and two distinct processing threads.	
Optimisation tool can optimise learning rate for an example neural network in less than 4 days.	
Optimisation tool can use feedback from previous tests to automatically generate new test values for hyperparameters.	

Website:

Test:	Outcome:
Website can be accessed via Google Chrome, Microsoft edge, and Mozilla firefox.	
Websites can load in less than 20 seconds on wi-fi throttled down to 1Mbs, 10 for 5Mbs, and 1 for 10Mbs.	
All pages of the website can be accessed via the taskbar in all other pages of the website.	
Website prompt field can accept textual input.	
Website API calls can be received by dummy script.	
Websites can receive (and then display in browser) images from a dummy script.	

Integration testing:

Test:	Outcome:
Neural Network can be trained via training apparatus in less than 1 day.	
Neural Network can be saved in a partially trained state via a training apparatus.	
A Neural Network can be trained simultaneously on multiple computers, virtual machines, and processing threads to optimise.	
Learning rate of the Neural Network can be optimised by training apparatus in less than 4 days.	
Neural Network can be trained on a dataset.	
Neural Network can produce human-recognisable images in response to all combination categories in the dataset.	
Website can call the Neural Network without triggering the training apparatus.	
Neural Network can receive prompts from Website.	

Website can receive (and then display) images generated by Neural Network..	
Neural networks can produce human recognisable images in response to a stripped version of the true dataset containing only brick houses.	