Deep Learning Driven Architectural Style Classifier

Project Proposal

Daniel Lennox
12/07/2019

Domain Background

Problem Domain:

Architecture is all around us. The buildings we have lived, played and worked in over the millennia are living bookmarks in the story of human history, an aesthetic and functional record in stone, timber or brick of our values and dreams. Knowledge and appreciation of architecture is important for the continued evolution of building construction and design as well as the preservation of important buildings and our history.

Technology/Solution Domain:

Computer vision is concerned with computers being able to understand images in a similar way to humans, extracting complex descriptive information from visual scenes [1]. The application of computer vision in real world projects has been accelerated by recent technological (computer/graphical processing power) improvements, as well as advancements to the way artificial neural networks are used to understand video and imagery, in particular, the improvements to deep learning using convolutional neural networks (CNN's) [2]. The ability for a computer to classify images is one popular application of these new technologies that can be applied to a variety of domains. Image classification tasks have recently benefited from the availability of huge labelled image datasets such as ImageNet and the development and availability of CNN models pre-trained on these datasets, such as ResNet50 and Inception. Using these pre-trained models as a base, has enabled often faster and more effective training of new models, a process called Transfer Learning [3].

Motivation:

I'm motivated to solve a problem in the domain of building architecture primarily due to a growing fascination with the various architectural wonders I've encountered while travelling these past few years, as well as with the architecture in my home city of Melbourne Australia. While my interest in architecture is strong, my concrete knowledge of the subject is very weak. Computer vision is the field within machine learning that interests me the most. I suspect there are ample interesting opportunities to combine these two interests by applying this visually focussed technology to the visually focussed domain of Architecture and potentially increasing my knowledge and experience with both by building a tool or application that would be useful or educational to others.

Problem Statement

When a person walks by a building, the vague idea that the building is perhaps Gothic or maybe Victorian style might come to mind. If the person was lucky enough to have an architect for a friend that was walking beside them, they might be able to ask, "Hey, what style is that building?" and receive a much more informed opinion. However for most people and in most situations, such a friend isn't available at the required moment. The goal of this project is to find a solution that can simulate the more informed opinion of an "architect friend". A person could, for instance, snap a quick photo of the building in question, provide it to the solution and receive an approximation of the various architectural styles that the building might be influenced by. For instance, they might receive a response of "80% Gothic, 15% Victorian and 5% Modern".

To be clear, this problem does not require highly accurate architectural classification, the standard of which would be acceptable to a professional in that field. Rather the problem is that of the amatuar enthusiast who wishes to know roughly what they are looking at. If the solution can provide classifications that are **mostly correct** but are **significantly more accurate than the user's vague impressions**, then it can be considered a good solution. The classifications generated by the solution should be as reliable as the potentially accurate but ultimately fallible opinions of the aforementioned single architect friend!

Datasets and Inputs

Dataset Source

A solution that can solve the previously described problem of basic architectural classification will likely require data which includes images of architecture, labelled with the architectural style or styles that influenced the architecture.

A novel approach to obtaining such a dataset is to utilise the popular Instagram social media application. Everyday, thousands of people around the world take photos of buildings and *tag* (label) them with the architectural styles that they believe the building represents. A quick search for the tag *#brutalistarchitecture* for instance, returns a result set of 130,000 images. This represents an enormous wealth of labelled image data.

The obvious “elephant in the room” here is whether these images are reliable or appropriate, given the fact that there is no guarantee that their architectural style labels are always correct or credible. The images were tagged by a multitude of users, with varying architectural knowledge and the correctness of the tags would likely vary to the same degree. However, it can be argued that this kind of dataset **is appropriate** for use given the problem context, audience and the definition of *correctness* within the domain. Firstly, as mentioned in the problem description, the accuracy of the classifications do not need to be up to academic research or professional standards. The goal, as per the problem description, is to provide useful classifications that “are **mostly correct** but are **significantly more accurate”** than those of the amateur architecture enthusiast. Additionally, the definition of correctness is somewhat loose in this context, as architecture and architectural styles reside in the domain of art and therefore it is often to some extent subjective and debatable whether a piece of architecture falls into certain categories. Incorrect classifications might rather be re-framed as contentious or unusual classifications, further adding to the breadth and variety of the dataset.

It is also hoped that the varied and imperfect photography in the Instagram based dataset will be a natural source of image augmentation, ensuring the solution can generalise well to odd angles and perspectives, rather than perfect, professional portrait shots of buildings with perfect lighting and high end equipment.

Dataset acquisition

Another consideration when using Instagram as a datasource is the question of how to obtain or access the images for processing. An open source image scraping tool called Instaloader (https://pypi.org/project/instaloader/) has the ability to download a subset of instagram images by tag. Running this tool for each desired architectural style being considered for classification has resulted in an appropriately large repository of labelled images for processing. If during implementation it’s discovered that the solution requires

more data for reasonable results, then it should be straightforward to simply download even more images using this tool.

As an encouraging precedent, this was also the approach used by Leonard Bogdonoff when collating his New York City street art dataset and using it to build a street art detection model (https://blog.floydhub.com/instagram-street-art/). His article describing the process was the inspiration for this project.

An example of a command to retrieve and save images of a particular architectural style to disk using the Instaloader tool is below:

```
$ instaloader --fast-update --no-videos --no-metadata-json
--no-captions "#brutalistarchitecture"
```

Dataset details

Initially, the solution should be able to classify images in 28 different architectural styles. These styles were chosen as a starting point (other styles could be progressively added later) and are generally typical of architectural styles that might be encountered while walking through a western city, rather than at a museum or archeological site. The frequency of instagram tag usage for each of these architectural styles was also used as a measure of the frequency for which these styles are encountered by everyday people and hopefully their relevance to the end users of the application.

Between 350 and 400 single layer images in **JPEG format** were downloaded for each of the architectural styles below.

Total dataset size: 10431 JPEG images.

Dataset size per architectural style:

Brutalist *384 images*	**Gothic** *361 images*	**Art Deco** 351 images	**Art Nouveau** *395 images*	**Victorian** *359 images*
Baroque *360 images*	**Colonial** *362 images*	**Romanesque** *351 images*	**Tudor** *379 images*	**Deconstructivism** *372 images*
Second Empire *389 images*	**Beaux-arts** *368 images*	**Queen Anne** *368 images*	**Renaissance** *386 images*	**Postmodern** *400 images*
Gothic Revival *355 images*	**Neo Renaissance** *397 images*	**Tudor Revival** *378 images*	**Neo Classical** *383 images*	**Edwardian** *352 images*
Italianate *375 images*	**Shingle Style** 381 images	**Mid Century Modern** *352 images*	**Modernist** *391 images*	**Indo-Islamic** *357 images*
Industrial *380 images*	**Romanesque Revival** *364 images*	**Neo Baroque** *380 images*		

The dataset sizes above represent the final datasets after scrubbing of inappropriate images (such as people tagging their lunch with #secondempirearchitecture while possibly eating inside a second empire architecture building) and removal of duplicate images. Additionally, images of well known buildings (such as the Notre Dame cathedral in Paris) tended to feature prominently in the dataset, so for each category, only a few images of differing angles and features were included for each building.

Additional dataset information

Average image size: 200kB
Largest image: 2800kB
Smallest image: 20kB
Largest image width: 1080 pixels
Largest image height: 1363 pixels
Smallest image width: 301 pixels
Smallest image height: 218 pixels

Solution Statement

The proposed solution to the problem described above, that of classifying images of buildings by architectural style, will primarily be the implementation, training and deployment of a Convolutional Neural Network (CNN) model. CNN's are a cutting edge and industry standard technology well suited to image classification tasks. Architecture styles in particular, are largely defined by their various shapes, curves and patterns. Since CNN's excel at extracting patterns from images and then using these to accurately classify images, it's hoped that a well implemented CNN can be trained to identify the various patterns and attributes of each architectural style and classify images of buildings accordingly.

CNN's are also able to perform stochastic classifications, so it should not only be possible to classify a given image of a building by a single architectural style, but to identify if multiple styles have influenced the building and a rough measure as to what extent.

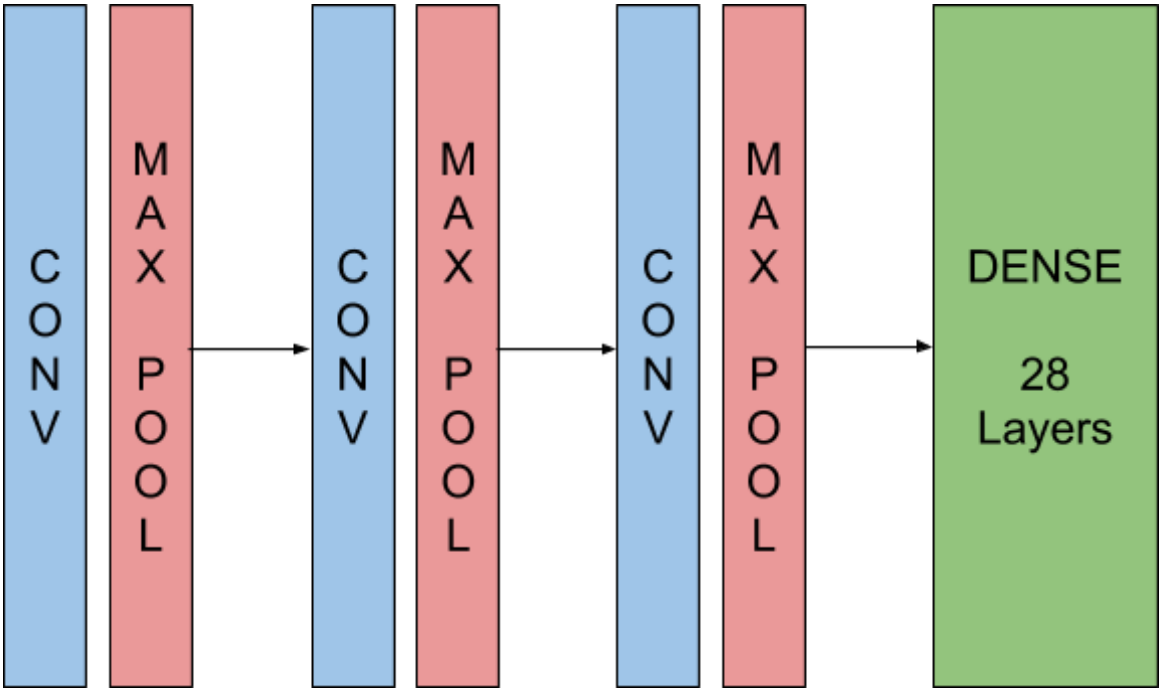
By training a CNN model on the dataset described above and splitting the dataset into training, testing and validation subsets, it should be possible to accurately measure the performance of the model using the known labels (architectural styles) of the building in each image. The performance can be measured using an accuracy metric that simply calculates the percentage of images that the model was able to correctly categorise from the set of testing data. Once a reasonable level of performance (see following sections) has been achieved, it should then be possible to classify new, unlabelled images of buildings by their architectural styles.

Benchmark Model

Primary Benchmarks

1. **Random chance model**
   As an initial starting point. The model should be able to perform better than a simple model that simply classifies each image using random chance.

2. **A vanilla CNN model**
   The solution model should, using the same dataset, also be able to outperform a simple baseline CNN model using the evaluation metric discussed in the next section. This model will be well structured, but use the simple CNN model architecture below.



With the following initial, untuned hyperparameters.

```
model.add(Conv2D(filters=16, kernel_size=3, padding='same'
activation='relu', input_shape=(TBA, TBA, TBA)))
model.add(MaxPooling2D(pool_size=2))
model.add(Conv2D(filters=32, kernel_size=3, padding='same',
activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Flatten())
model.add(Dense(28, activation='softmax'))
```

The solution model will attempt to improve results obtained by the above model, by experimenting with additional model architectures and hyperparameter tuning.

A secondary benchmark (just for fun!):

As described in the problem statement, the solution is designed to fill the role of an “Architect friend” that can tell the user about the architectural styles of buildings they might encounter. Therefore, while not particularly robust benchmark, comparing the performance of the solution to an actual architects performance will be an interesting, if not definitive, way to test if the solution is solving the original problem.

Evaluation Metrics

For the solution and for each of the benchmarks described above, the evaluation metric used to assess performance can simply be the **Accuracy** of the model where:

```
Accuracy (Percentage) = Number of Correct Classifications / Total Test Images * 100
```

Accuracy is an appropriate evaluation metric as it measures how frequently the solution or benchmark is able to correctly classify images of architectural buildings, which equates to measuring how well the solution or benchmark solves the problem outlined in the problem statement.

Since the categories are well balanced (similar number of images in the dataset for each architectural style) and neither precision or recall is more important for this problem domain, accuracy should be a solid measure of the performance of the model and benchmarks.

Project Design

Below is an initial outline of the approach that will be taken towards developing the proposed solution.

Data Preparation

Data preparation will be a major component of this project, primarily due to the nature of the datasource. It's expected that the images scraped from instagram will, in addition to the scrubbing of unsuitable images, require some amount of pre-processing work on suitable images.

Probable steps:

1. *Download Images*
   Obtain 350 - 400 raw images using the instaloader tool for each of the 28 hashtags specified earlier. Download these to a local directory structure.

2. *Remove outliers*
   View each local image tag directory using the gallery view in the Finder application on Mac OSX. Manually delete any **obvious** outliers, such as images that do not contain buildings.

3. *Training, Validation and Testing Split*
   Split total dataset into Training, Validation and Testing datasets. The initial split that will be used is defined below.

Training	**Validation**	**Testing**
70%	15%	15%

4. *Import dataset.*
   Import the initial datasets in a Jupyter notebook, loading their various categories (labels) from folder names that match the architectural style names.

5. *Resize images.*
   Programmatically convert all images to a uniform square size.

Possible steps

- *Convert to Gray Scale*
  It might be useful to programmatically convert each image to gray scale to increase training time. The assumption is that shapes are more integral to the uniqueness of each architecture style and therefore gray scale might be acceptable. However this

should only be done if problems are encountered with either the accuracy of the model not reaching the benchmarks or if training times are too high, since the model might dispel this human assumption and decide colour is very useful after all!

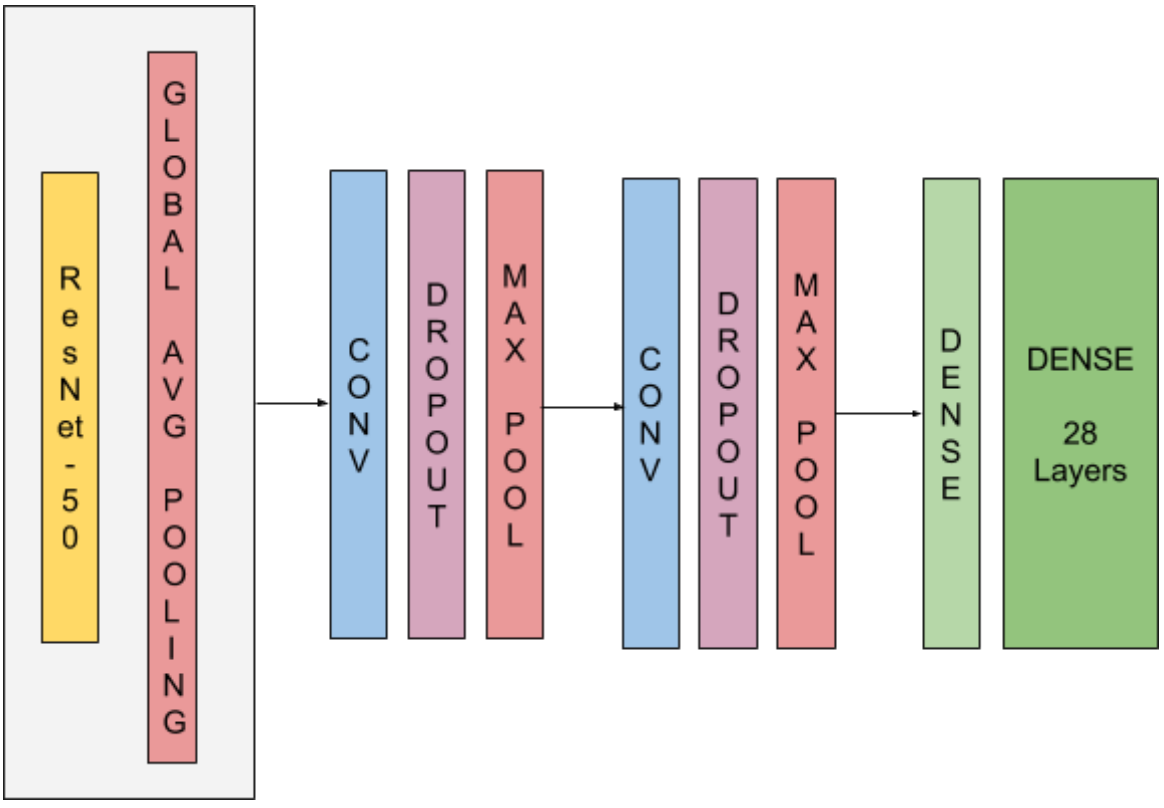
- *Data Augmentation*
  The variety of images provided by the dataset will likely include subject matter in various locations within the image. This will hopefully ensure that there's no need to tackle **translation** invariance within the dataset. However **rotation** invariance will likely be a useful image augmentation step. It's likely that most of the images in the dataset will at least show the subject matter in the right orientation. However the final use of the solution will likely result in images of buildings that are at varying angles. When a user takes a photo for instagram, some level of care is usually taken to ensure the photo has a decent composition. However a user of the "Architectural Classifier" solution might only care about identifying the building, rather than taking a good photo and therefore might take hasty snaps at no particular angle which are then fed into the solution for classification. For this reason, the model should be taught to classify images of buildings at various orientations.

Model Design

Once the data has been prepared, an initial CNN model will be designed and implemented.

Initial Model Architecture

A rough model architecture is provided below, which represents a starting point for investigating the best CNN architecture for this particular problem.



Architecture Attributes

- *Transfer Learning*
  It makes sense to leverage transfer learning, rather than training the Model from scratch to recognise basic shapes and patterns. Leveraging a pre-trained network such as ResNet50, will drastically save on training time, since the pre-trained CNN model will already know how to detect the underlying basic shapes and lines. This will also leave the new Model more time and resources to detect the specific patterns unique to each architecture.

- *Convolutional Layers*
  The Model will then potentially have additional convolutional layers added after the pre-trained network, in order to hopefully learn the high level architectural patterns and styles present within the images.

- *Dropout Layers.*
  If test dataset performance is poor, then dropout layers will be added after the convolutional layers to hopefully reduce overfitting.

- *Pooling Layers*
  Max pooling layers will be added after the convolutional layers to reduce the dimensionality of the data.

- *Layer Activation Functions.*
  ReLu activation functions will be used on the convolutional layers to avoid the vanishing gradient problem.

- *Output layer (Final dense layer).*
  The final dense layer will have 28 nodes to match the number of architectural styles that the model will attempt to categorise. It will have a softmax activation function in order to output probability percentages for each architectural style for a given image. It's also likely that there will be some post processing to truncate the architectural styles with the lowest percentages into an "Other" category, to keep the results focussed on the primary architectural styles of the building.

Model Training

The Model training process will involve:

1. *Loss function selection*
   The Model will likely use categorical cross entropy to measure the performance of the model when comparing it's probability outputs for each architectural style to the actual image labels.

2. *Optimiser selection and experimentation*
   Experiments will be conducted with various gradient descent **optimisers** to reduce error loss and increase accuracy while training the model.

3. *Configure training reporting.*
   Various logging and graphs will need to be set up in the Jupyter project file to gain visibility on the training process and measure error loss and accuracy.

4. *Implement saving of best weights*
   The Model weights with the best performing training results should be saved in order to utilise them for future predictions on unseen data and eventual deployment to a user facing application.

Evaluation

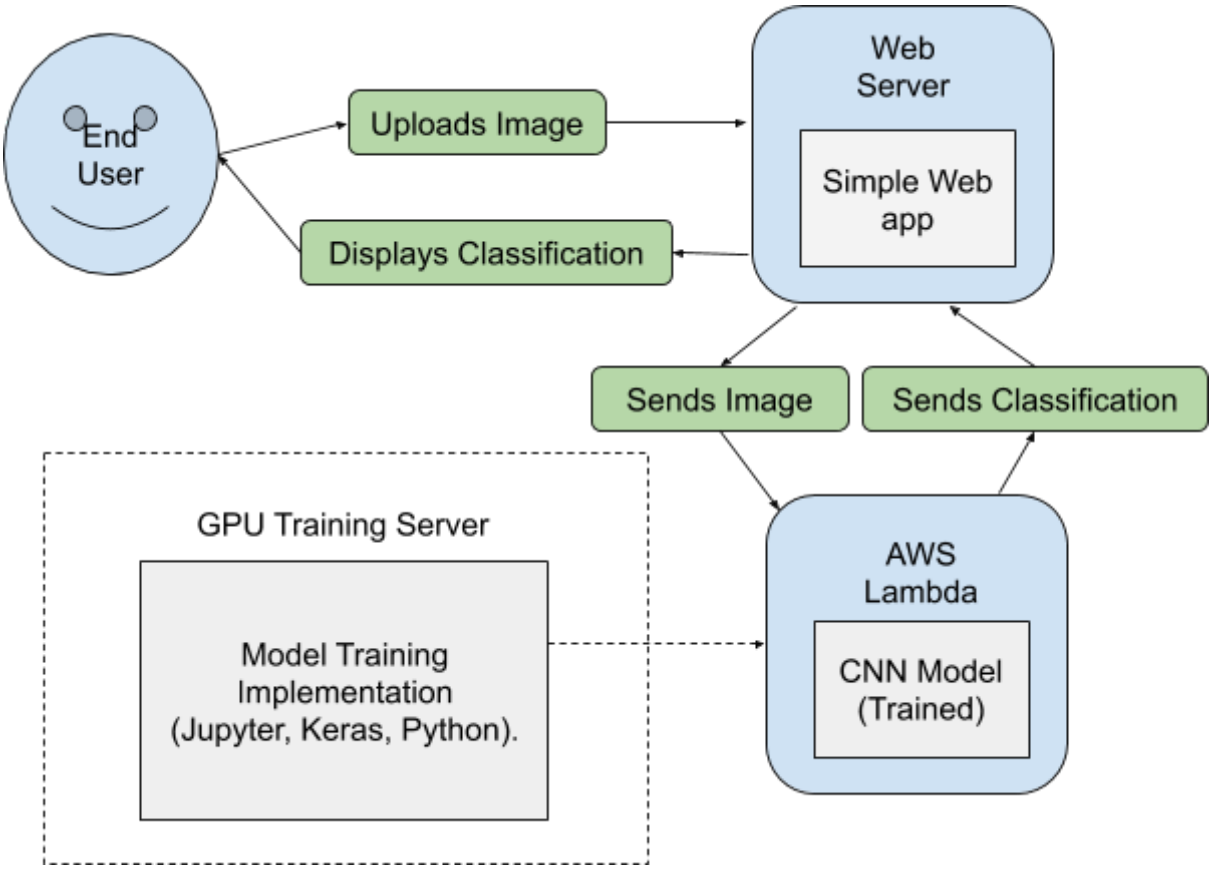
The evaluation process will primarily consist of:

1. *Benchmark tests*
   Any trained model should be evaluated against the benchmarks outlined earlier in this document.

2. *Iteration and experimentation*
   The Model training process will primarily be focussed on tweaking CNN architecture and algorithm hyperparameters until the solution yields results that can perform to the standards set by the desired benchmarks.

Deployment

The model, once performing to the standards set by the aforementioned benchmarks, will then be deployed into a production environment, with a similar architecture to the diagram below.



The final model, could be added to a serverless function, such as AWS lambda. From there a simple web application could leverage the model to classify images of architecture that were uploaded by an end user.

References

1. Dana H. Ballard; Christopher M. Brown (1982). *Computer Vision*. Prentice Hall. ISBN 978-0-13-165316-0.
2. Krizhevsky, Alex; Sutskever, Ilya; Hinton, Geoffry (2012). "ImageNet Classification with Deep Convolutional Neural Networks". *NIPS 2012: Neural Information Processing Systems, Lake Tahoe, Nevada*.
3. C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu, (2018) “A survey on deep transfer learning,” https://arxiv.org/abs/1808.01974