ISYE/CSE/MGT 6748 Practicum: Simply Business Property Insurance Risk Assessment
Team 5

Cody D. Westgard, cwestgard3@gatech.edu;

Christian D. Blandford, cblandford7@gatech.edu;

Daniel L. Broyles, dbroyles6@gatech.edu;

Introduction — In support of property insurance risk assessments: using public data to estimate building features that affect insurance value, an analytics pipeline is implemented to support the use of transfer learning for property type prediction. We hypothesize that a two-pronged approach, utilizing both a binary class and a multi-class prediction with nerual networks, will yield the best results

1 PROJECT OVERVIEW

The project goal is to develop models, including neural networks, that as input, use known building characteristics combined with visual features extracted from the forefront image to predict the property attributes to automate the process of claims risk evaluation. Specifically, the project seeks to use publicly available property forefront images from Google Street View (GSV) API to estimate the type of property (e.g., flat, terraced house, semi-detached, detached house).

1.1 Data Overview

Leveraging extract, transform, load scripts (e.g., 'load_google_streetview_SHARE_v2.ipynb') and data files (e.g., 'outcodes.csv') provided by the Simply Business team, images are extracted from the GSV API that represent the forefront or "street view" of a property associated with a latitude-longitude coordinate derived from properties in outcodes (e.g., zip codes) from the rightmove.co.uk database. The retrieved images utilize the default parameters of the GSV API, with the 'heading':'0' parameter specified which corresponds to a north facing direction of the Google Street View camera. It was

assumed that given this parameter was specified in the provided scripts and used to create the providing training data files (e.g., uni_project_data.zip), the default parameter, which derives "a value ... that directs the camera towards the specified location, from the point at which the closest photograph was taken" should not be used [1].

1.2 Data Challenges & Hypothesis

Given the default southward facing of all provided images due to a northward facing google street view camera, it was hypothesized that the default facing image from the provided ETL would not be the optimal image for property type prediction. We hypothesized that a two-pronged approach, utilizing multiple API calls to get a "best image", by iterating through the cardinal directions integrated with a binary property image classifier at each direction would improve the overall model accuracy. Finding a method that is efficient (in terms of computation and limiting API calls) was the largest data challenge.

A smaller secondary challenge was the elimination of duplicate entries, which could reach up to 20% of the data for some boroughs and falsely inflated our model's performance. Upon investigation, most duplicates were multiple flats located within the same building, which results in multiple images from the street outside that building. Ultimately, we allowed each latitude longitude pair to occur only once in the dataset, resolving this issue.

1.3 Implementation

As Shown in Figure 1 below, the "default" image for a property provided in the training data and generated by the provided scripts does not directly face any building, but rather a street. As such, we implement a binary property classifier utilizing transfer learning, discussed in detail in later sections, to predict a probability of the "default" image being a good image to train the multi-class model on. The implemented solution predicts the probability of the image showing a physical property, as opposed to a street, brush, woodlands, ocean etc; should the predicted probability be below a selected threshold of .5, the image is re-queried utilizing the panorama identifier (e.g., "pano_id") found in the metadata to retrieve the same image from a different heading. This process is repeated for each 90 degrees until a 90 degree field of view ("fov":90) section of the panorama is found to be above the .5 threshold for predicted probabilities.

Image gsv_1b842cd2-15f4-463b-a736-2a2984c35a68 in Heading:0 (North) & Heading 90 (East)

GSV Image at Heading 0 & 90

As Shown in Figure 2 below, the ETL predicts images where there is no property in the image; in particular, images were identified as being inside the buildings and are saved in a dataframe of "bad images" tagged to their property_id .

"Bad Image" gsv_354eb88b-6c9a-431b-a922-5dcf245efcd6

GSV Image at Heading 270

Ultimately, the pipeline outlined above creates a folder of images to be utilized in both training and prediction with transfer learning models outlined in section three below. Further, technical risks and improvements related to the ETL, are discussed in the final section.

1.4 Pipeline Details

The first step of this approach will be to build a neural network. It will flag certain images as being potentially improved by rotating the Google Streetview camera. A list of those addresses is passed back to the API, where we collect 3 more images of that location, with the headings 90, 180, 270 degrees. The same neural network is then applied to these images, and the best option selected. That image is then passed to the primary model, another neural network to determine the class of the property. If all are below a threshold probability (we used 0.5), that property is discarded. We found that typically 1-2% of locations are discarded.

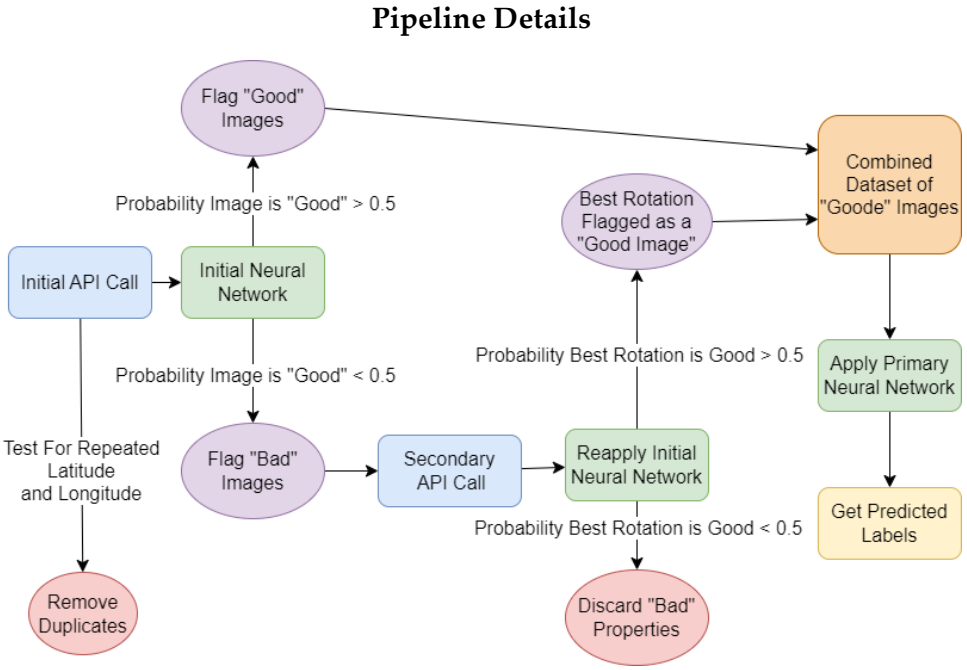

Figure 3: GSV Image at Heading 0 & 90

1.5 Training Initial Neural Network

First, the building of the initial neural network. In building this step, the main challenge here was a lack of data. Our team was unable to find an pre-existing data set containing labels for such a question, and a brief search through literature did not find any examples of papers directly emulating our question. Thus, we opted to hand label data. Samples images were collected from three boroughs, chosen to represent rural, urban and suburban based on their population density [2]. The chosen boroughs are Swindon (SN), East London (E) and Newcastle-Upon-Tyne (NE).

After downloading a random selection from these boroughs and removing duplicate entries, we are left with 538 images. These were given binary labels (0 representing that the image does not directly capture a property, 1 representing a good capture). There are 325 0's and 211 1's in this hand labeled set.

After splitting these images into a train and test set (using a randomized 80-20 split), we sought to build a neural network. The VGG16, Resnet50, and EfficientNet architectures were tested, with VGG16 performing the best (see Figure 4). The details for transfer learning and described in part 2 of this report.

Model	ResNet50	VGG16	EfficientNetB0
Validation Accuracy	0.6111	0.6565	0.6111

Figure 4: Accuracy for Various Pretrained Models

These networks were trained for 20 epochs and demonstrate a clear overfitting problem for anything higher than that. Further attempts to improve the network by allowing unfreezing also resulted in more overfitting (see Figure 5). Overall, the accuracy here is rather low for a binary classification algorithm. These performance issues will be addressed as they are a major area for possible improvement, and this piece is the weakest link of our pipeline.

6

Accuracy Over Unfrozen Layers

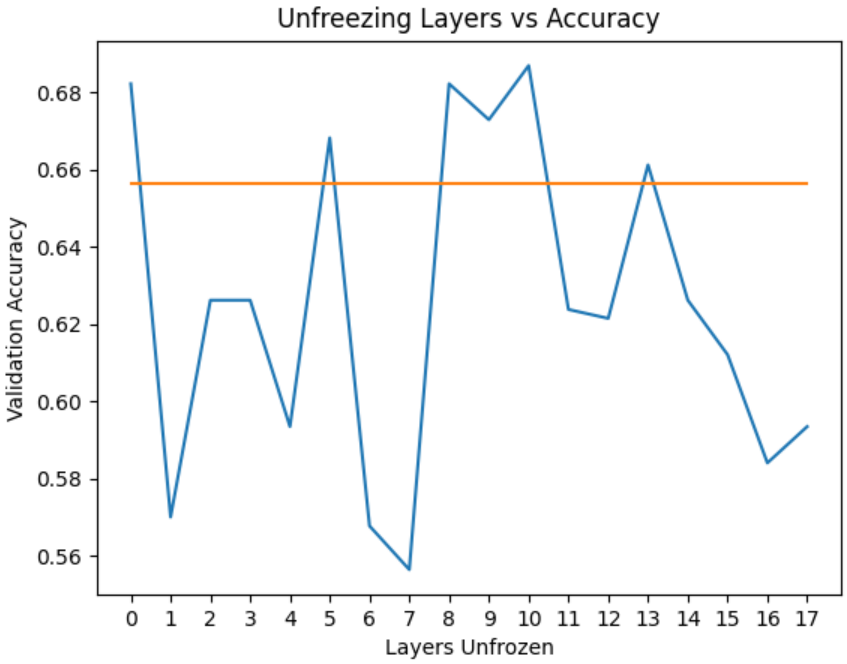

Figure 5: Unfrozen Layers vs. Accuracy

The most prominent issue is likely the training data size. We could not find an existing dataset of images with the desired labels so could only hand label, which is a time-consuming process. If seeking further improvements, expanding the size of the training dataset is a strong candidate for where to begin.

2 TRANSFER LEARNING

Transfer learning is the process of using an existing training set of features, labels and the predictive function that has already been trained to predict

labels from those features, a then modifying those pieces, especially the predictive function, to be applied to a new problem space consisting a new set of features and labels. The goal is to generate an updated predictive function. By leveraging the existing material, the goal is to apply previous material related to one problem domain and apply it to a similar, or sometimes the same, problem domain.

2.1 Transfer Learning: Homogeneous vs. Heterogeneous

As described in *A survey of transfer learning (Weiss)*, there are two broad categories of transfer learning [3]. These are Homogenous and Heterogenous respectively. Homogenous Transfer learning is when the features, labels, and predictive function of one domain is applied to a new dataset from the same, or similar domain, but usually with the goal of predicting a different label or metric. Heterogenous transfer learning is when the source labels and features from the source domain do not overlap with the target domain. The graphic below from an article titled, "An Introduction to Transfer Learning" [4] demonstrates the visual difference in the feature and label feature space between Homogenous and Heterogenous transfer learning. This effort could be considered a Homogenous transfer learning process as we utilized an existing models trained partially on features that shared the same feature and domain space.

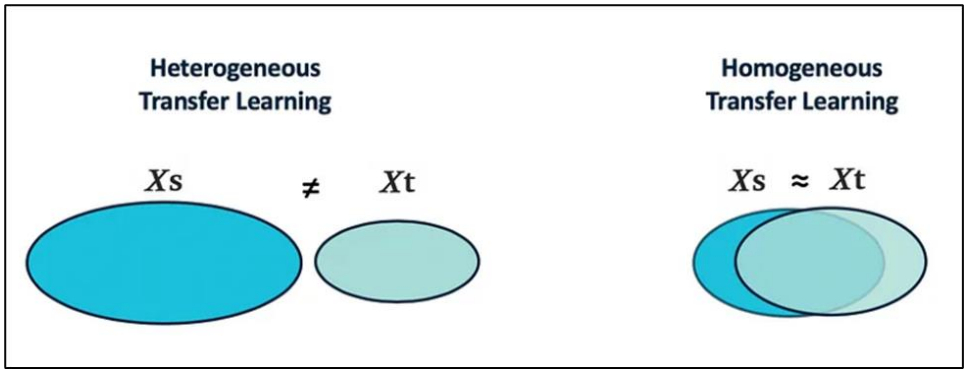

Figure 6: The difference in feature space between Homogenous and Heterogenous Transfer Learning

8

a. Transfer Learning: Parameter Based Approach

Beyond Heterogenous and Homogenous Transfer learning there are at least 5 difference broad classes of approaches used for transfer learning. These approaches are Instance, Feature, Parameter, Hybrid and Relational based (Asgarian). For this effort, a parameter-based approach was used so that will be discussed more below. More information on the other approaches can be referenced in [3][4].

Parameter based transfer learning consists of leveraging the existing parameters from a predictive function and then reweighting the components (also referenced as "learners") and/or applying additional convolutional layers on top of the existing model to adapt to the problem domain. The main steps for the parameter-based transfer learning approach are as follows [4][5]:

1. Select an existing model that optimally has a source domain that overlaps the target domain as much as possible: this will be referred to as the "base model"
2. Decide whether to freeze or unfreeze the base model as necessary (discussed below)
3. Add additional classification layers on top of the existing model
4. Determine how the new classification layers and (if using an unfrozen base model) how the layers should update their weights based on the selected loss function (discussed below)
5. Train the model
6. Evaluate the Model

The parameter transfer learning process is visualized below in Figure 7 [4].

Parameter Transfer Learning Process

9

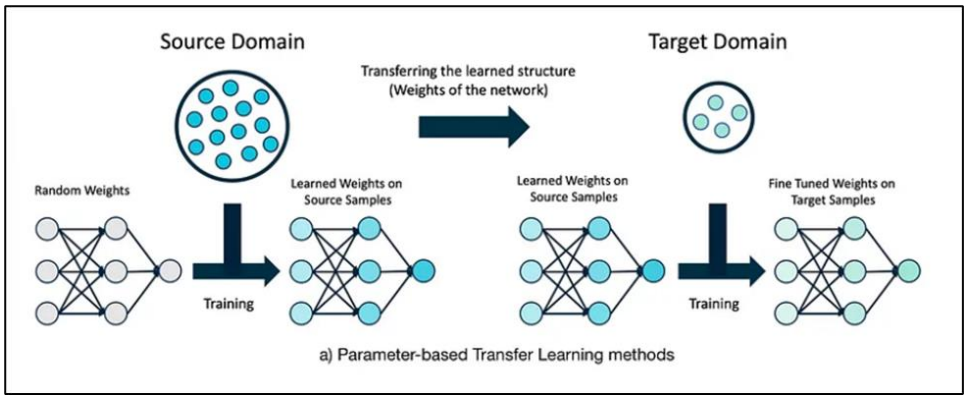

Figure 7: Parameter Transfer Learning Process

The specific decisions and configuration used for every step of the process highlighted above for this project will be discussed in more detail in the individual sections below as the decision and tests made for each are critical to understanding our approach and the results obtained. The goal of these following sections is to elucidate our team's approach and rationale for the steps taken.

4. Transfer Learning: Base Models

All models were obtained via the TensorFlow Hub repository. The following models were tested using a variety of configurations that will be detailed in subsequent sections below:

Model Name	Source
mobilenet_v2_100_224	**https://tfhub.dev/google/imagenet/mobilenet_v2_100_224/feature_vector/5**
mobilenet_v3_large_075_224	**https://tfhub.dev/google/imagenet/mobilenet_v3_large_075_224/feature_vector/5**
efficientnet_v2_imagenet21k_ft1k_b0	**https://tfhub.dev/google/imagenet/efficientnet_v2_imagenet21k_ft1k_b0/classification/2**

10

convnext_large_21k_1k_384_fe	**https://tfhub.dev/sayakpaul/convnext_large_21k_1k_384_fe/1**
resnet_v1_200	**https://tfhub.dev/google/supcon/resnet_v1_200/imagenet/classification/1**

a. mobilenet_v2_100_224 Details

Mobilenet V2 is trained using Imagenet-1k, which is also referred to as ILSRV 2012. This image dataset consists of 1.2 million images spit into 1000 categories (image-net). These categories include mobile homes, prison houses, movie theatres, restaurants, and greenhouses. While these categories are not houses or house types, they are building related. Thus, these can be considered for use in a homogenous transfer learning process, which is our team's goal.

Mobilenet V2's structure is detailed below in Figure 8 [6]:

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

Figure 8: Mobilenet V2 Structure

b. mobilenet_v3_large_075_224 Details

MobileNet V3 Large is, as the name suggests, and evolution of MobileNet V2. MobileNet V3 Large was again, trained on ImageNet data. The model consists of many changes, but of primary note is that in Mobile Net V3 a 7x7 pooling layer was added to before the final convolution layers (see third to last row in Figure 9 below). In addition, MobileNet V3 authors added h-swish to the model, which is a more computationally efficient method for calculating the sigmoid function for the classification nodes. The bottom line is that MobileNet V3 Large in testing is faster and more accurate than MobileNet V2.

Mobilenet V3 Large's structure is detailed below in Figure 9 [6]:

12

Input	Operator	exp size	#out	SE	NL	s
$224^2 \times 3$	conv2d	-	16	-	HS	2
$112^2 \times 16$	bneck, 3x3	16	16	-	RE	1
$112^2 \times 16$	bneck, 3x3	64	24	-	RE	2
$56^2 \times 24$	bneck, 3x3	72	24	-	RE	1
$56^2 \times 24$	bneck, 5x5	72	40	✓	RE	2
$28^2 \times 40$	bneck, 5x5	120	40	✓	RE	1
$28^2 \times 40$	bneck, 5x5	120	40	✓	RE	1
$28^2 \times 40$	bneck, 3x3	240	80	-	HS	2
$14^2 \times 80$	bneck, 3x3	200	80	-	HS	1
$14^2 \times 80$	bneck, 3x3	184	80	-	HS	1
$14^2 \times 80$	bneck, 3x3	184	80	-	HS	1
$14^2 \times 80$	bneck, 3x3	480	112	✓	HS	1
$14^2 \times 112$	bneck, 3x3	672	112	✓	HS	1
$14^2 \times 112$	bneck, 5x5	672	160	✓	HS	2
$7^2 \times 160$	bneck, 5x5	960	160	✓	HS	1
$7^2 \times 160$	bneck, 5x5	960	160	✓	HS	1
$7^2 \times 160$	conv2d, 1x1	-	960	-	HS	1
$7^2 \times 960$	pool, 7x7	-	-	-	-	1
$1^2 \times 960$	conv2d 1x1, NBN	-	1280	-	HS	1
$1^2 \times 1280$	conv2d 1x1, NBN	-	k	-	-	1

Figure 9: Mobilenet V3 Structure

c. efficientnet_v2_imagenet21k_ft1k_b0

EfficentNet V2 is a convolutional neural network that, at the highest level, uses a logic in which a "controller" which will evaluate the training of the network using a reward parameter, uniquely for EfficentNet, that reward parameter is the floating-point operations per second (FLOPS) [8]. For full details on the architecture please refer to *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks* (Tan et al.)

The main takeaway is that EfficentNet V2 has less training parameters versus comparable models (24 million parameter in EfficentNet V2 vs Vision Transformers 86 million)[9]. Thus, EfficentNet is faster to train, especially when unfreezing the base layers. At the same time, EfficentNet V2 has maintained or exceeded accuracies from other models when tested against ImageNet datasets.

The version of EfficentNet used for this project was trained on ImageNet-21k which consisted of 14 million images separated into 21 thousand classes, which

includes mobile homes, prison houses, movie theatres, restaurants, and greenhouses as previously mentioned in addition to other features classes.

Efficientnet V2 structure is detailed below in Figure 10 [10]:

Stage	Operator	Stride	#Channels	#Layers
0	Conv3x3	2	24	1
1	Fused-MBConv1, k3x3	1	24	2
2	Fused-MBConv4, k3x3	2	48	4
3	Fused-MBConv4, k3x3	2	64	4
4	MBConv4, k3x3, SE0.25	2	128	6
5	MBConv6, k3x3, SE0.25	1	160	9
6	MBConv6, k3x3, SE0.25	2	256	15
7	Conv1x1 & Pooling & FC	-	1280	1

Figure 10: EfficientNet V2 Structure

d. convnext_large_21k_1k_384_fe

ConvNeXT was released in 2020 and is an evolution of previously released Convolutional Network Models, specifically ResNeT50. The architecture of ConvNeXT has been updated to include Swin Transformers (SWIN-T) which is a method of performing classification on merged patches of images. Figure 11 below is a good illustration of how SWIN-T works versus a Vision Transformer (ViT) method [12]. For full details on ConvNeXT's architecture please refer to *A ConvNet for the 2020s* (Liu et al.)

14

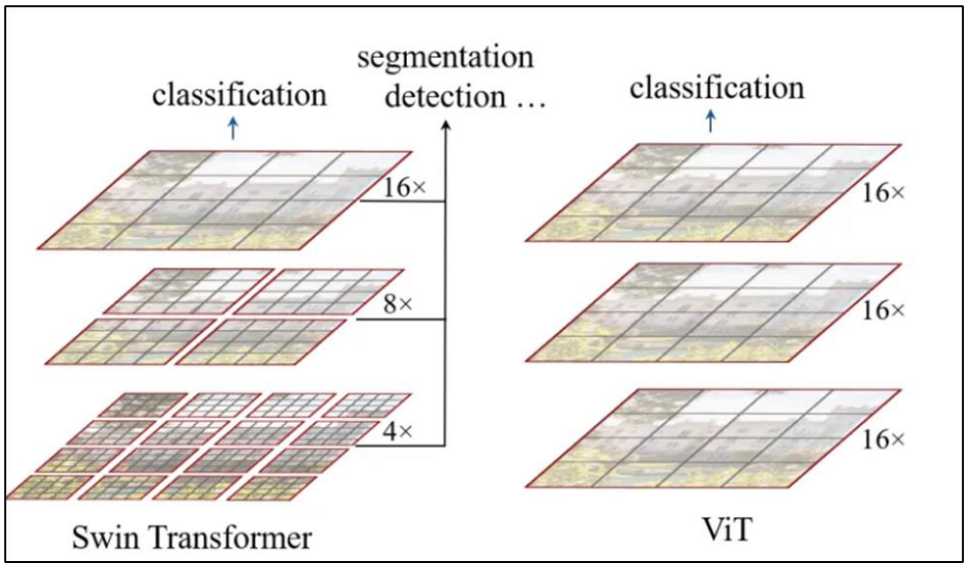

Figure 11: SWIN-T Overvie

ConvNeXT Large was originally trained using the 2017 COCO training set, but the version used for this project was retrained using ImageNet 21K. This is interesting because the original COCO 2017 classes do not include any building types. COCO is focused on household goods, as can be seen on their website (https://cocodataset.org/#explore). However, since this model was retrained via transfer learning on ImageNet 21K classes which includes mobile homes, prison houses, movie theatres, restaurants, and greenhouses as previously mentioned, it was determined that this model was appropriate to test for our application.

e. resnet_v1_200

ResNet V1 is trained using Imagenet-1k, which is also referred to as ILSRV 2012. This image dataset consists of 1.2 million images split into 1000 categories (image-net). These categories include mobile homes, prison houses, movie theatres, restaurants, and greenhouses. While these categories are not houses or house

15

types, they are building related. Thus, these can be considered for use in a homogenous transfer learning process, which is our team's goal.

ResNet consists of "residual blocks" which is a mechanism in which a layer not only provides training inputs to subsequent layer, but also to layers below the next immediate layer. Thus, instead of a purely sequential training model on the network you can think of the training process occurring in chunks. This offers the advantage of the initial training vectors not only being used to training the first layer, but the first few (depending on the configuration in question) which can help to avoid problems when the first layer does a poor job learning form the initial training vectors. In addition, the blocks can be trained in batches which makes training a "deep" model more efficient [14]. See Figure 12 below for a simple representation of Residual Block.

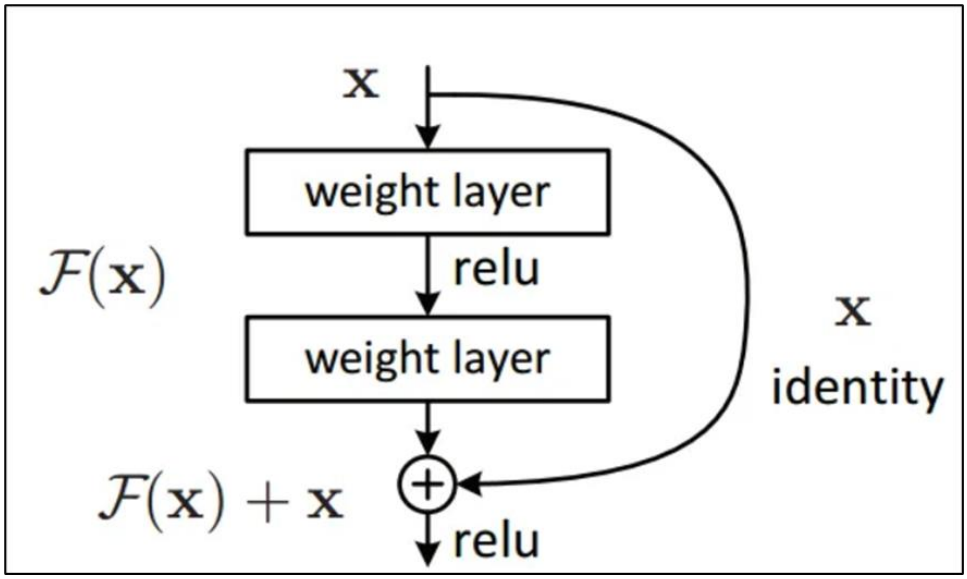

Figure 12: Residual Block Sample

5. Transfer Learning: Freeze or Unfreeze Base Model

The next decision in the parameter-based transfer learning process after you pick the base model is to decide on whether to "freeze" or "unfreeze" the base model.

16

Freezing refers to not adjusting the weights of the base models in the Transfer learning process and simply training the classification layers you add to the base model. This is much faster and can reduce overfitting issues if you allow the base model to update its weights.

Unfreezing a model is when you allow the layers from the base model to be retrained in the training process. This offers the benefit of allowing you to optimize models in which the source targets and labels may have less overlap with your target features and labels, and thus more changes to the base model may be warranted. Referring to Figure 6, this is when those two overlapping domain ovals share a smaller common area.

In conjunction with deciding to freeze or unfreeze the base model, an appropriate Learning Rate (LR) needs to be determined. The LR is the parameter which controls "how much to change the model in response to the estimated error each time the model weights are updated" [16]. If you choose too high/large of a LR then the model will make large adjustments each run based on the error, and this can lead to an erratic fitting process which will skip over the optimal fit. However, if you select too low of a LR then the model may get stuck and not be able to progress as it cannot make big enough changes to the node weights to resolve a misclassification issue. A low learning rate can also increase the number of epochs needed to fit the model and lead to overfitting. In general, it is best practice to use lower learning rates when perform transfer learning with unfrozen layers.

For this project, all models except ResNet and Mobilenet V3 Large were tested using frozen and unfrozen base layers. To do this, different Learning Rates were applied. If the layers were frozen a learning rate of 1e-2 of 1e-3 was used. If the layers were unfrozen, a much lower learning rate of 1e-4 or 1e-5 was used to slow the rate of change on the base layer weights during the transfer learning process. The training time for training an unfrozen model increased by several orders of magnitude, for example with EfficentNet V2 the training time increases from ~12 seconds per epoch to ~200 seconds (about 3 and a half minutes) per epoch.

6. Add additional classification layers on top of the existing model

Once the base model is picked, and frozen decision is made, it is time add at least one additional layer to your model. For this activity, the team opted to add a

single dense layer consisting of about 4000 parameters outputting to 4 classes. This was because the team used an approach where the model would predict a multiclass vector in which the vector index position corresponds to a given building type. Please see figure 13 below of a sample model summary from the ResNet model that was tested.

This sample shows that in this sample as can be seen in the final layer(dense_1), there are 800 samples, each with 4 class options. An interesting thing to note is in the second layer (keras_layer_1) you can see the model outputs a vector containing 1000 class scores for each sample. These samples are based on ResNet model trained on the Imagenet-1k dataset, hence the 1000 response classes.

```
: model_bce.summary()
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 random_flip_1 (RandomFlip)  (800, 224, 224, 3)        0

 keras_layer_1 (KerasLayer)  (800, 1000)               65111976

 dense_1 (Dense)             (800, 4)                  4004

=================================================================
Total params: 65,115,980
Trainable params: 4,004
Non-trainable params: 65,111,976
_________________________________________________________________
```

Figure 13: ResNet Model Sample with Frozen Base Model

7. Select Loss Function

After selecting a base model, freezing or not, adding your layer(s), it is time to select a loss function. The loss function is the mechanism in which a neural network determines how well it is doing, and thus is critical to the training process as the loss function will directly influence the node weight update process. There is a plethora of loss functions available, and it is possible to create custom loss functions. For this project, the team tested the models using Binary Cross-Entropy/Log Loss and Categorical Cross Entropy Loss.

Binary Cross-Entropy/Log Loss is when a model tags an input and classifies it into one of two categories and assigns the log probabilities of that point to the first or second class. Typically, Binary Cross-Entropy is used for binary

classification problems, but it can be used in a one vs all manner to perform multiclass classification, which is how it is deployed in our project. For Binary Cross-Entropy to work, the model needs to provide a sigmoid curve representing the probability of an observation belonging to a given class, hence the activation function for a Neural Network fitted using Binary Cross-Entropy will be a sigmoid. Please refer to Figure 14 below for more details on the Binary Cross-Entropy/Log Loss Function [18].

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

Figure 14: Binary Cross-Entropy/Log Loss Function

Categorical Cross Entropy Loss is like Binary Cross-Entropy/Log Loss but does differ in some key respects. First, Categorical Cross Entropy will output a series of probabilities for each class, referred to as the Softmax probability. The class with the highest probability will be predicted to be the response class. This loss function can be used to multiclass classification natively and can handle one hot encoded label vectors with ease, this no one versus all fit is needed. Please refer to Figure 15 below for details on the Categorical Cross-Entropy formula sample for a 3-class sample.

19

$$CE = -\sum_{i=1}^{i=N} y\_true_i \cdot log(y\_pred_i)$$

$$CE = -\sum_{i=1}^{i=N} y_i \cdot log(\widehat{y}_i)$$

$$\implies CE = -[y_1 \cdot log(\widehat{y_1}) + y_2 \cdot log(\widehat{y_2}) + y_3 \cdot log(\widehat{y_3})]$$

Figure 15 Categorical Cross-Entropy Function

20

8. **Train the Model**

Figure 16 below overviews the training process that was implemented.

21

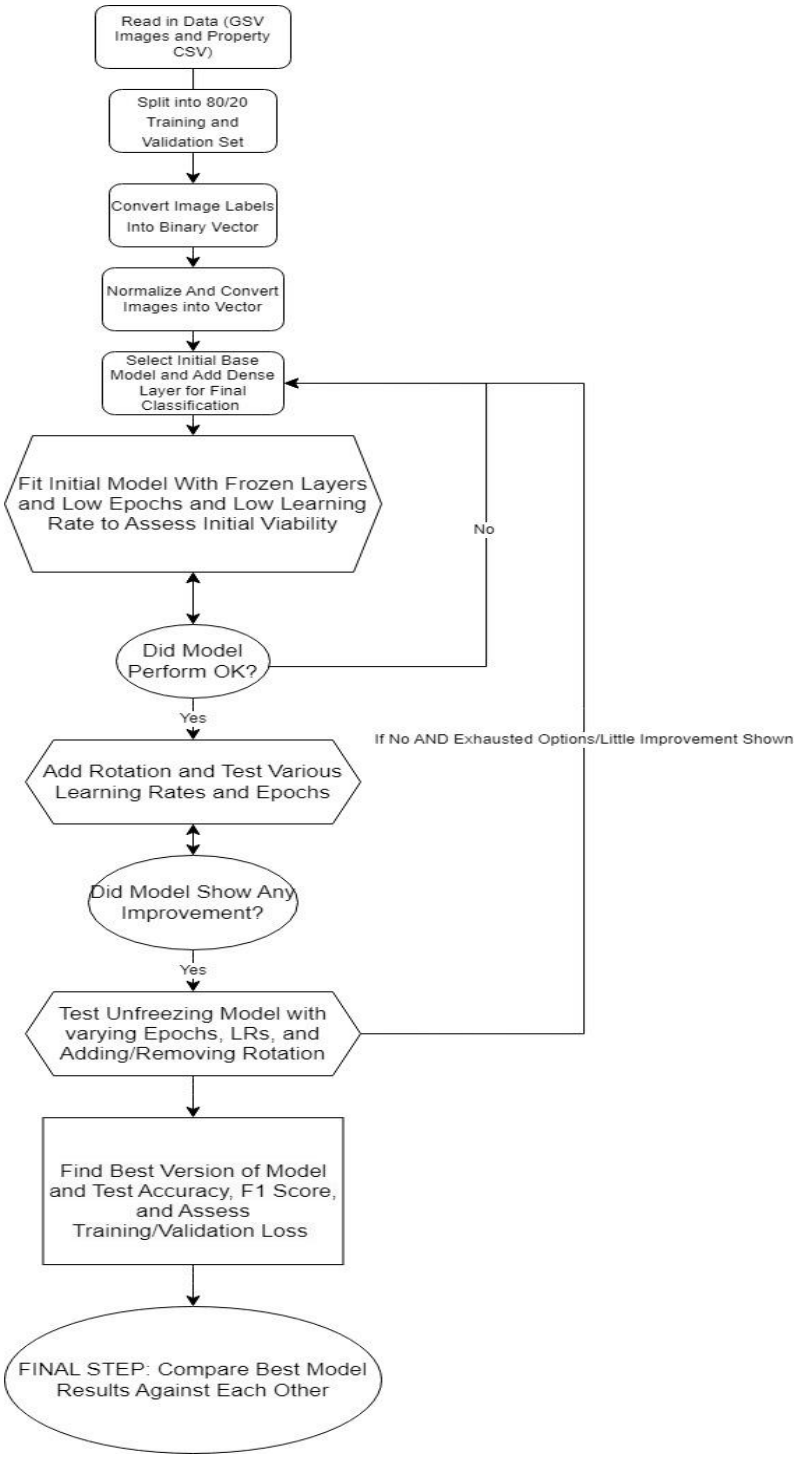

Figure 16: Model Training Process

22

This training process was utilized against all 5 models tested (detailed in Section 6.2: Training the Models). The following models were tested through the entire process detailed in Figure 16 above: mobilenet_v2, efficentnet_v2, and convnext_large. Resnet and mobilenet_v3 large were not tested through the entire process. In the case of Resnet, this model was not tested through the entire process because the initial model had extremely poor initial performance (less than 25% accuracy). In the case of mobilenet_v3_large, it was not tested through the entire process as its initial performance was worse than mobilenet_v2, thus a decision was made to save training time and focus on other models.

For those models that were tested through the entire process the following configuration values show in Figure 17 below were tested in either the unfrozen model, frozen model, or both:

Learning Rates	Epochs	Rotation Along Horizontal Axis
1e-2	**10**	**Yes**
1e-3	**15**	**No**
1e-4	**20**	**N/A**
1e-5	**40**	**N/A**
N/A	**50**	**N/A**
N/A	**200***	**N/A**

Figure 17: Configurations of Parameters Tested

For the learning rates, all the learning rates were applied to the to the frozen models, but only the low learning rates (1e-4 and 1e-5) were applied to the unfrozen models.

23

The Epoch values were applied to all model configuration permutations for both the frozen and unfrozen models. Of special note, the 200-epoch value was only tested against efficientnet_V2 for reasons discussed below.

The rotation was applied to all permutations of the models for both frozen and unfrozen versions.

Around 80 permutations for the 3 models that were completely tested for this effort using the configurations above and following the process covered in Figure 16.

As illuded to above, the major constraint was training time. The time it took to train a single epoch using a batch process for a frozen model could exceed 3.5 minutes per batch. This, in addition to the number of permutations tested, limited the team to only fully evaluating 3 different models, with partial evaluations of a further 2

9. Evaluate the Model

Each model was evaluated for its accuracy, and the most accurate model was further evaluated on its F1 Score. The most accurate results for each model, along with its configuration that had those results, will be covered below in the 'Results' section.

RESULTS

The following table details the best model found.

Model Name:	efficientnet_v2_imagenet21k_ft1k_b0 Details
Model Source:	https://tfhub.dev/google/imagenet/efficientnet_v2_imagenet21k_ft1k_b0/classification/2

Frozen/Un-frozen	Unfrozen
# of Epochs	15
Learning Rate	1e-4
Horizontal Rotation	Yes
Accuracy	~48%
F1 Scores by Class	Detached: 0.435; Flat: .182; Semi-Detached: .552; Terraced: .489

Figure 18: Characteristics of Best Model Found

Figure 19 Below shows the training and validation accuracy and loss for the best model at every fitted epoch.

25

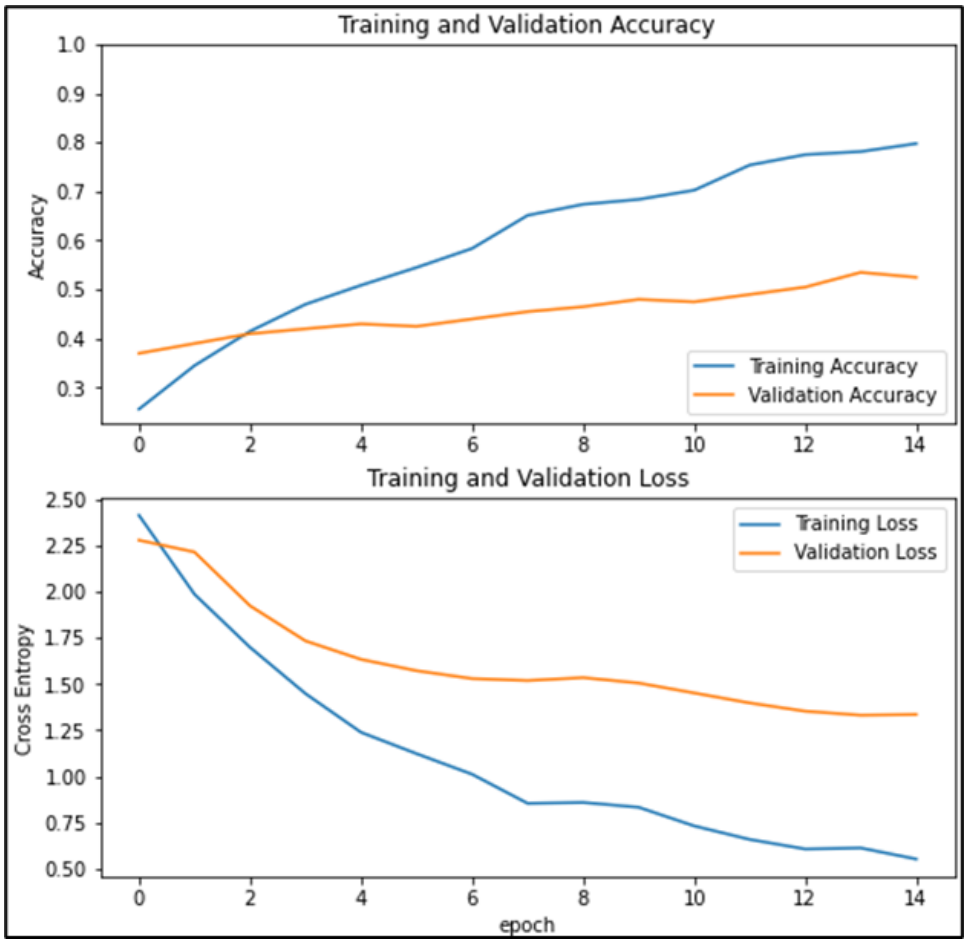

Figure 19: Best Model Training and Validation Accuracy and Loss

The following table details the best accuracy for the other 4 models (please note that only ConvNeXT and Mobilnet V2 were fully tested).

Model Name	Best Accuracy
mobilenet_v2_100_224	~47.7% (Overfitted on 100 Epochs)
convnext_large_21k_1k_384_fe	~39.7%
mobilenet_v3_large_075_224	~38.3%
resnet_v1_200	~19.5%

Figure 20: Validation Accuracy for Other Models

26

3 CONCLUSIONS AND RECOMMENDATIONS

Our team was able to assemble a model following our hypothesizes pipeline that yield results for this classification task. While the ultimate accuracy only reached about 48%, we are optimistic that this is a functional proof of concept and if further refined and developed, in the methods described below, this pipeline has potential to succeed as a broad structure

3.1 ETL Improvements

Based on knowledge gained from the implementation it is hypothesized that the ETL could potentially be improved by adjusting both the default parameters and iterative image generation parameters. Firstly, utilizing the default GSV setting as opposed to North (heading:0) as the default image and then rotating the field of view on the panorama image to get the best image with a similar approach as implemented could be explored. Secondly, if it is determined that the GSV default direction is relatively accurate, a similar approach utilizing a dynamically determined but narrower heading range, (e.g., GSV default heading +/- [x, y ,z]), as opposed to testing 360 degrees of coverage could also yield improvements. Thirdly, utilizing a larger field of view (e.g., FOV > 90) and training on "wider" images around the GSV default or otherwise queried image may also yield improvements. Lastly, a more robust training dataset set for training the binary property prediction could likely improve the ETL generation of a ultimate "Best Image" for multi-class training and prediction.

Additionally, as mentioned in section 1, the first neural network in the pipeline likely suffers from its relatively small training sample and thus this area is a place for possible improvement.

3.2 Prediction Improvements

We hypothesize that an improvement can be achieved by partitioning the data. Notably the "flat" category gave a lot of issues to our models, so it is possible that predictions would be improved by leaving out that category, or possibly building a separate model to detect only that category.

4 REFERENCES

1. Google Street View Documentation: https://developers.google.com/maps/documentation/streetview/request-streetview
2. Office for National Statistics Population density - census maps, ons. Home - Office for National Statistics. (n.d.). https://www.ons.gov.uk/census/maps/choropleth/population/population-density/population-density/persons-per-square-kilometre
3. Weiss, K., Khoshgoftaar, T. M., & Wang, D. (2016, May 28). A survey of Transfer Learning - Journal of Big Data. SpringerOpen. https://journalofbigdata.springeropen.com/articles/10.1186/s40537-016-0043-6
4. Asgarian, A. (2019, August 13). Transfer learning - part 1. Medium. https://medium.com/georgian-impact-blog/transfer-learning-part-1-edoc174ad6e7
5. Transfer learning and fine-tuning : Tensorflow Core. TensorFlow. (n.d.). https://www.tensorflow.org/tutorials/images/transfer_learning
6. MobileNet V2. (n.d.). https://huggingface.co/docs/transformers/model_doc/mobilenet_v2
7. Imagenet Large Scale Visual Recognition Challenge 2012 (ILSVRC2012). ImageNet. (n.d.). https://www.image-net.org/challenges/LSVRC/2012/browse-synsets.php
8. Jain, V. (2019, November 22). Everything you need to know about MobileNetV3 and its comparison with previous versions. Medium. https://towardsdatascience.com/everything-you-need-to-know-about-mobilenetv3-and-its-comparison-with-previous-versions-a5d5e5a6eeaa
9. Mobilenet V3. Papers With Code. (n.d.). https://paperswithcode.com/lib/torchvision/mobilenet-v3
10. Sarkar, A. (2022, October 8). EFFICIENTNETV2 - faster, smaller, and higher accuracy than Vision Transformers. Medium. https://towardsdatascience.com/efficientnetv2-faster-smaller-and-higher- accuracy-than-vision-transformers-98e23587bf04
11. Tan, M., & Le, Q. V. (2020, September 11). EfficientNet: Rethinking model scaling for Convolutional Neural Networks. arXiv.org. https://arxiv.org/abs/1905.11946

12. Singh, A. (2022, March 30). ConvNext: The return of convolution networks. Medium. https://medium.com/augmented-startups/convnext-the-return-of-convolution-networks-e70cbe8dabcc
13. Liu, Z., Mao, H., Wu, C.-Y., Feichtenhofer, C., Darrell, T., & Xie, S. (2022, March 2). A convnet for the 2020s. arXiv.org. https://arxiv.org/abs/2201.03545
14. Axon/resnet50-v1 · hugging face. Axon/resnet50-v1 · Hugging Face. (n.d.). https://huggingface.co/Axon/resnet50-v1
15. Sahoo, S. (2022, September 26). Residual blocks - building blocks of Resnet. Medium. https://towardsdatascience.com/residual-blocks-building-blocks-of-resnet-fd90ca15d6ec
16. Brownlee, J. (2020, September 11). Understand the impact of learning rate on neural network performance. MachineLearningMastery.com. https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/
17. Godoy, D. (2022, July 10). Understanding binary cross-entropy / log loss: A visual explanation. Medium. https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a
18. Neuralthreads. (2021, December 26). Categorical cross-entropy loss - the most important loss function. Medium. https://neuralthreads.medium.com/categorical-cross-entropy-loss-the-most-important-loss-function-d3792151d05b

29