An X-ceptional Solar Panel Classifier

Ana Belén Bárcenas
Masters in Interdisciplinary Data Science
Duke University
anabelen.barcenas.jimenez@duke.edu

Jacqueline Wang
Masters in Interdisciplinary Data Science
Duke University
jingwen.wang@duke.edu

Zisheng Jason Chang
Masters in Interdisciplinary Data Science
Duke University
zisheng.chang@duke.edu

Prajawal Vijendra
Masters in Interdisciplinary Data Science
Duke University
prajwal.vijendra@duke.edu

Abstract

For this Kaggle competition, we developed a machine learning technique to classify the existence of solar panels from aerial imagery. For the model, we used the Xception model and applied transfer learning to train the model on this aerial imagery dataset. We supplemented the training data with a dataset from Stanford's Deep Solar project and finished 2nd in the Kaggle Competition with a private leaderboard accuracy of 99.993%, and a public leaderboard of 98.83%

Introduction

Solar panel adoption is rapidly growing due to the benefits of reducing both costs and environmental impact. However, tracking the adoption rates of solar panels can be a difficult proposition. Often solar panel data is tracked by private parties, or are collected via self-reported surveys. These sources can be unreliable. Our group trained a convolutional neural network capable of classifying solar panels from aerial imagery. With this model, our group developed a model capable of tracking solar panel utilization across the world.

Background

The most prominent study relevant to this challenge was the Deep Solar Project by Stanford. For this study Yu, Wang, Majumdar, and Rajagopal trained a convolutional neural network to both classify and segment solar panels in images. Their model was developed from Google Inception V3. In this project, they utilized transfer learning to train the Inception neural network model on solar panel arrays. Thus our decision to choose a deep convolutional neural network to classify these solar panel images.

Neural networks were first introduced in 1983 by Kunihiko Fukushima[1]. In his paper, Fukushima developed a recognition that recognized handwritten characters and patterns. These neurons, also referred to as nodes, are the basis of how neural networks transform raw inputs into desired outputs. Neural networks are made up of many of these artificial neurons. The artificial neuron works by calculating the sum of its weighted inputs, adding a bias, and outputting a value that is a function of a summed input. In Google's Inception V3, this neural network consists of thousands of densely connected nodes.

Through training, neural networks can learn to classify data. Each node's incoming connection is assigned a number known as a weight. These weights are initialized with random numbers. Each neuron's weight dictates what it will output. The process of training involves reducing the error between the networks output and the desired output. This training process is done through backpropagation. During the feed-forward pass, the network is given input and calculates an output determined by the weights mentioned previously. As the network trains, it aims to reduce the error between the output of the network and the actual output by changing the weights of its nodes.

This training process is repeated over and over until the network reaches an error minimum. However, it is important to note that this method is vulnerable to local minima. A local minimum may not be the best solution for the problem. This issue is known as overfitting. Overfitting is a result of the bias-variance trade-off.

[1] https://www.rctn.org/bruno/public/papers/Fukushima1980.pdf

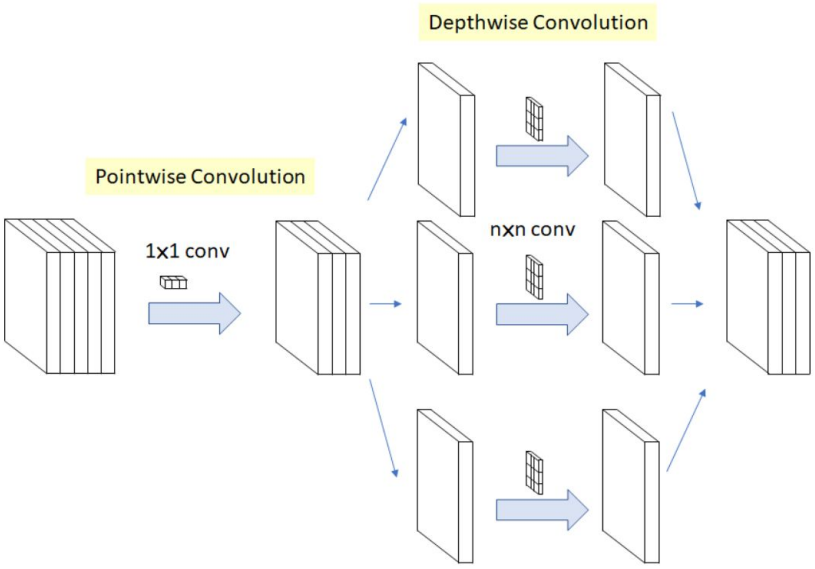
In this challenge, this was an important consideration as the dataset's training set contains only 1500 images, of size 101x101x3. In total, this translates to over 45 million training data points, and the Google Inception model has 6.797 million trainable parameters[2].

The convolutional neural network we are using is made up of these learnable node weights. Convolutional layers generate feature maps by iterating filters over an image, detecting features such as straight edges, simple colors, and curves. By stacking filters, the convolutional neural network is able to learn complex features and shapes. What separates convolutional neural networks from neural networks is their ability to analyze a volume of data using location invariant feature detectors.

The potential of convolutional neural networks applied to image classification tasks was first demonstrated in the ImageNet Project. The ImageNet Project[3] has hosted annual competitions challenging teams to classify a large visual database with over 14 million annotated images. In 2012 the AlexNet reduced the top 5 error from 26% - 15.3%. In 2014 Google Developed a neural network model codenamed Inception for the ImageNet project[4], achieving first place in 2014. Since then an iterated version of Google's Inception, Xception has been developed.

Xception stands for 'Extreme' version of Inception. In a traditional convolution neural network, the convolution layer seeks out correlation across both space and depth. But the inception architecture separated out the image region and channels. The inception architecture had 1x1 convolutions to project the original input into several separate, smaller input space, and used a different type of filter to transform the blocks into data. Xception is a modification on the Inception architecture, instead of partitioning input data into several compressed chunks, it maps the spatial correlations for each output channel separately, and then performs a 1×1 depthwise convolution to capture cross-channel correlation. This is equivalent to depthwise separable convolution, which consists of depthwise convolution and a pointwise convolution. For this project, we used transfer learning to train the Xception Model for solar panel classification in 124x124x3 images.

[2] https://arxiv.org/pdf/1409.4842v1.pdf
[3] https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf
[4] https://www.cs.unc.edu/~wliu/papers/GoogLeNet.pdf



The Modified Depthwise Separable Convolution used as an Inception Module in Xception, so called "extreme" version of Inception module (n=3 here)

Data

The data consists of images of size 101x101x3 RGB color. Each pixel in the image stores a tuple corresponding to each color channel red, green, and blue. For each image, this means that there are a total of 101x101x3 = 30,603 training points. Examples of images we classified are shown below. The first row represents images with solar panels, and the second row represents images without.

The top row shows examples of images with solar panels,
And the bottom row shows images without solar panels.

The largest challenge we faced are the different sizes, shapes, rotations, and colors of these solar panels. For example, the two images below show the following difficulties from left to right: low resolution, rotated panels, similar roof and panel colors, and non-rectangular layout. The takeaway from recognizing these difficulties is to have the model generalize on a training set that includes examples of rotated solar panels, or non-rectangular solar panel layouts.

The image on the left shows an example of a rotated solar panel, and on the right an irregularly shaped solar panel

The dataset from Kaggle is organized in a CSV file, with each image's id and output stored in each row and the training and testing images stored in a separate directory. An example of the dataset is shown below.

id	label
0	0
1	0
2	1
3	1
4	1
5	0
6	1
7	0
8	1
9	0

Methods

Our solution involved transfer learning on Google's Xception Deep Learning model. We used Google's Xception model as it is an iteration of the Inception model used in the Stanford project. The only preprocessing we applied to the images were transforming the 101x101 images to 128x128 shape using anti-aliasing scaling as per the model's recommendations. The model was built using Keras. The only modification made to the model was adding a Dense layer with two

outputs for a one hot encoding representation of a panels existence in an aerial image: [0,1] or [1,0].

The code for building the model is shown below.

```
base_model = Xception(input_shape=(img_width, img_height, 3),
weights='imagenet', include_top=False)
x = base_model.output
x = GlobalAveragePooling2D()(x)
predictions = Dense(nb_classes, activation='softmax')(x)
model = Model(base_model.input, predictions)
```

The first step of the model was to build the data pipeline. The CSV file containing the labels is loaded using NumPy. The CSV contains two columns: image ids and the label of the image (Solar PV array or no Solar PV array). The CSV is imported into a dictionary. The images are loaded using Python imaging library and saved as a NumPy array.

The dataset is shuffled, and the last 10% of the dataset is used for validation. The code below shows the shuffling process for the dataset.

```
from random import shuffle
idxs=list(range(0,imgs.shape[0]))
shuffle(idxs)

trainImgs = imgs[idxs]
trainLabels = labels[idxs]
```

The model is trained for 30 epochs. After each epoch, the model is tested on the validation set to ensure no overfitting.

```
tm = model.fit(trainImgs, trainLabels,batch_size = 20, epochs = 30,
               verbose = 1, validation_split=0.1)
```

After training, the model runs predictions on the test dataset, and the predictions are exported to a results.csv file.

```
testImgs,testLabels = load_testing_data()
raw_results = model.predict(testImgs, verbose = 0)
# Output preprocessing code concatenated
…
results = results.sort_values(by=['id'])
results.to_csv('results3.csv')
```

Results

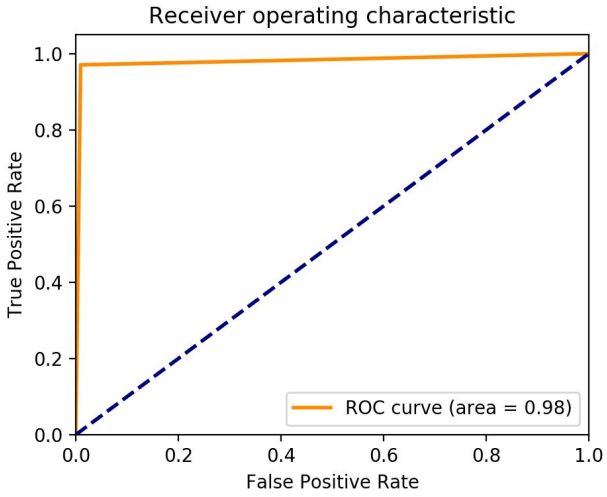
The results below are collected from the validation set: representing 10% of the randomly shuffled training set from Kaggle. The validation set contains a total of 173 samples. There are 105 positive samples and 68 negative samples.

The confusion matrix below for the validation set shows the number of true and false positives and negatives.

	Positive Classification	Negative Classification
Positive Condition	104	1
Negative Condition	2	66

Confusion Matrix on the validation set.

The dashed line represents a model with random guessing. The model performs significantly better than a model with random chance guessing.



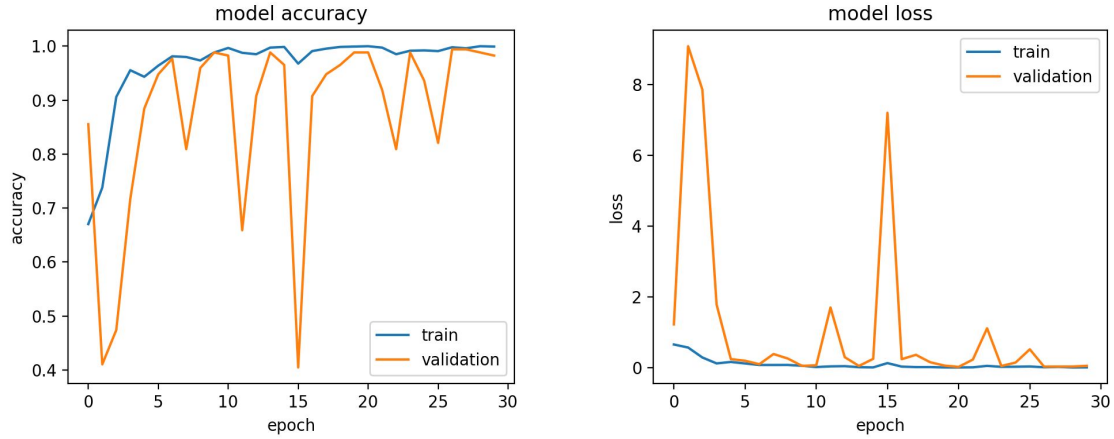
Receiver Operating Characteristic
Curve on the validation set

To compare the performance of the Xception model, we developed a baseline model with two layers: Convolutional layer, and a dense model. The convolutional layer is flattened before the output for the model to output a single layer.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_49 (Conv2D)           (None, 125, 125, 64)      3136
_________________________________________________________________
flatten_11 (Flatten)         (None, 1000000)           0
_________________________________________________________________
dense_81 (Dense)             (None, 2)                 2000002
=================================================================
Total params: 2,003,138
Trainable params: 2,003,138
Non-trainable params: 0
_________________________________________________________________
```

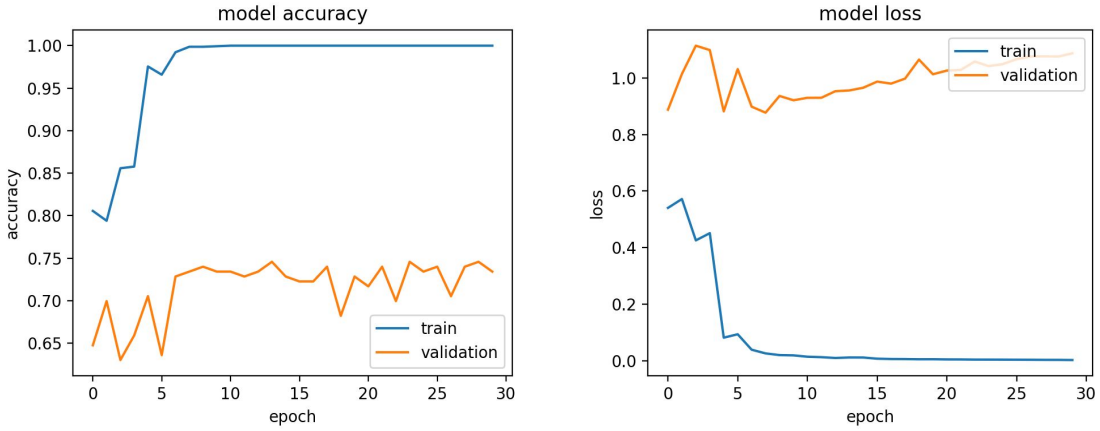
Model Accuracy and Loss vs. Epochs

The image on the left represents the model accuracy on the training and validation set for the baseline model, and the image on the right represents the baseline model loss on the training and validation set.

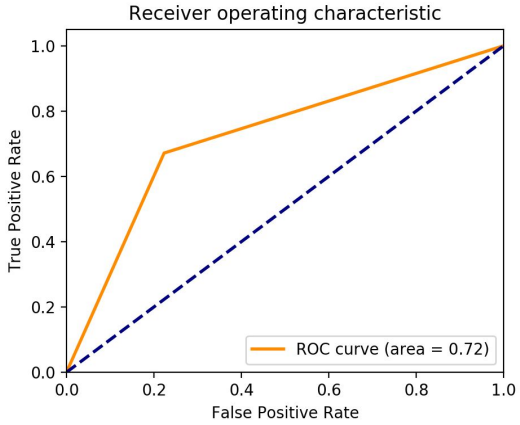


Left: Model Accuracy Accuracy vs Epochs, Right: Model Loss vs Epochs

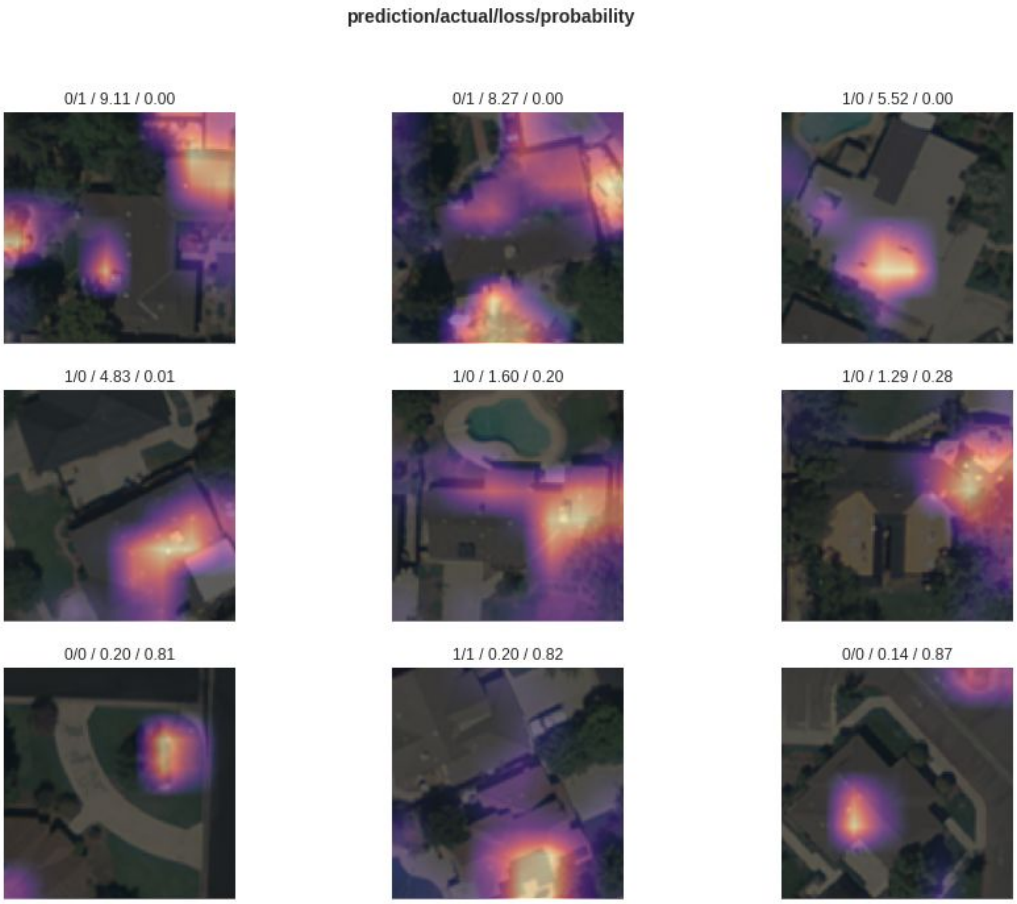
The figures below show the baseline model scores. We observed that during training the baseline model suffered from overfitting: unable to generalize on the validation set. The AUC is significantly less than the Xception Model: 0.72 vs 0.98 AUC.



Left: Base Model Accuracy Accuracy vs Epochs, Right: Base Model Loss vs Epochs



The above performance assessments are the results of the model we trained and submitted to the Kaggle Dataset. The image below shows samples during the training process that had the largest losses. We can see from the plot that the major losses in the pictures are highlighted.



The images above show examples of 'hotspots', where the neural network
Has predicted the location of a solar panel is. The numbers above
show the prediction/actual/loss/probability of a solar panel

Conclusions

For this Kaggle competition, we were challenged to create a model to classify images with and without solar panels. With Google's Xception model we demonstrated that with transfer learning the model was able to achieve 99.93% accuracy on the Kaggle private leaderboard dataset.

The largest challenge we faced during this exercise was the issue of overfitting on high dimensional data. The Kaggle dataset we were given contained 1500 101x101x3 color RGB images, and 500 testing images for model evaluation. This translates to 45 million training points for the model to train on. The Google Xception Model has 20,811,050 trainable parameters. As seen from our model's accuracy vs. epochs graphs in the results section, the model did not consistently generalize on the kaggle dataset: with spikes up and down in accuracy even as the number of epochs increased. The transformation we applied to the images was scaling the size of the images to 128x128px to match the Xception's model's input size detailed in the literature. We are curious to explore if changing parameters, such as color space, could increase the model's ability to generalize.

The key takeaway we observed was the feasibility of taking Google's Xception model and applying it to this classification challenge utilizing pre trained models and transfer learning. Worth exploring in future work involves improving the generalization performance of the model.

Our project has demonstrated the potential of using a deep learning model to discriminate images containing solar panels. Future work could include further comprehensive comparisons and metrics for comparing the different deep learning models. Another model to explore utilizing are the Deep Residual Network Model, which won the Imagenet 2015 classification competition. Of interest to pursue next also includes image semantic segmentation of the solar panels: drawing a polygon around each of the solar panels.

Roles

Prajwal Vijendra
Model Development and Training

Z. Jason Chang

Model Development and Data Cleaning

Ana B. Barcenas
Dataset Transformations Research, Versions Maintainer, Model Research

Jacqueline Wang
Dataset Transformations Research, Model Researcher

References

1. An Introduction to Statistical Learning (ISL) by Gareth James, Daniela Witten, Trevor Hastie and Robert Tibshirani, 2013. (http://www-bcf.usc.edu/~gareth/ISL/)
2. Deep Learning by Ian Goodfellow, Yoshua Bengio, and Aaron Courville, 2016.
3. Deep Residual Learning for Image recognition (https://arxiv.org/abs/1512.03385)
4. FastAI Resnet (https://www.fast.ai)
5. Google Xception (https://arxiv.org/abs/1610.02357)
6. Keras Xception Implementation (https://keras.io)
7. Lecon convolutional neural networks on text and images (http://yann.lecun.com/exdb/publis/pdf/lecun-bengio-95a.pdf)
8. Neocognitron (https://www.rctn.org/bruno/public/papers/Fukushima1980.pdf)
9. Reinforcement Learning: An Introduction, by Richard Sutton and Andrew Barto, 2018.
10. Stanford Deepsolar (http://web.stanford.edu/group/deepsolar/home)