Detecting Palm Oil Plantations Using Neural Networks

Newton Tran

INTRODUCTION

Palm oil is a type of oil that comes from the fruit of palm trees, namely from *Elaeis guineensis* or the African oil palm. Hence its name, the tree is native to Africa but was brought over to Southeast Asia over a century ago as an ornamental (decorative) tree crop. It is highly versatile because:

- It is semi-solid at room temperature, which helps keep spreads spreadable
- It is resistant to oxidation, which extends the product's shelf-life
- It has a relatively high smoke point of 235 °C (or 455 °F)
- It is odorless and colorless
- It is a highly-efficient, high-yielding crop that does not require muc
  h land

About 50% of packaged products found in a typical supermarket contain palm oil. Currently, over 85% of the global supply is produced in Indonesia and Malaysia, but there are several other countries such as Thailand, Columbia, and Nigeria that produce it.

Despite its versatility, palm oil has significantly driven the rate of deforestation in highly biodiverse forests. Additionally, this coupled with carbon-rich peat soils emit millions of tons of greenhouse gases into the atmosphere, thereby also contributing to climate change.

PROBLEM STATEMENT

The goal of this project is to build a model, utilizing neural networks, to classify whether or not a given satellite image contains the presence of a palm oil plantation. This can help government and environmental agencies monitor and mitigate further deforestation, as well as help introduce stricter regulations against it.

THE DATA

The dataset is originally from the Women in Data Science Datathon 2019 competition. It contains three compressed .zip folders of satellite images and their corresponding annotations, which are saved as .csv files. Explicitly:

- `train_images.zip`, the images used for training
- `leaderboard_holdout_data.zip`, the images used for the competition's submission / ranking
- `leaderboard_test_data.zip`, the images used for the competition's submission / ranking, similar to above
- `traininglabels.csv`, the annotations that correspond to training images
- `holdout.csv`, the annotations that correspond to the holdout images
- `testlabels.csv`, the annotations that correspond to the test images

For each of the annotations files, there are three features:

- `image_id`, the filename of the image
- `has_oilpalm`, where 0 indicates no presence of a palm oil plantation, 1 indicates otherwise (i.e. the image's label)
- `score`, which indicates the likelihood of the image's label holding true, where values closer to 1 indicate high likelihood

One critical consideration is that after the end of the competition, more images were added but were not officially documented. More notably, the test data was unusable since there were significantly more images than those listed in the corresponding annotations. Therefore, this project utilizes the training and holdout data.

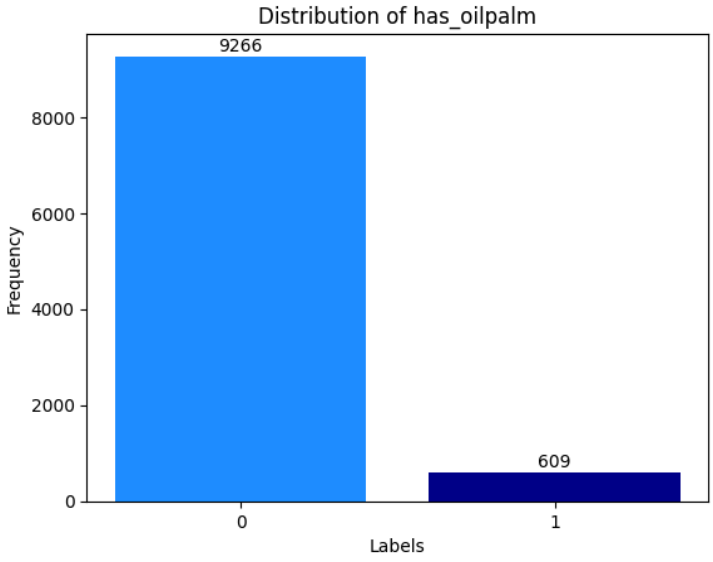
A preview of the annotations is shown below:

	image_id	has_oilpalm	score
0	img_000002017.jpg	0	0.7895
1	img_000012017.jpg	0	1.0000
2	img_000022017.jpg	0	1.0000
3	img_000072017.jpg	0	1.0000
4	img_000082017.jpg	0	1.0000

EXPLORATORY DATA ANALYSIS

The first goal is to explore the training data. When looking at the training annotations, there were no nulls. However, as aforementioned, since there was a mismatch between the filenames listed in the annotations versus the original ones, we needed to promptly address this. Particularly, the filenames listed in `trainlabels.csv` had 2017 and 2018 appended to the end, so those were removed, and because this subsequently created duplicate filenames, duplicate rows were then removed. Furthermore, because there were additional, undocumented images added, the annotations were further filtered to only contain the filenames that matched the ones in the folder of training images.

After preliminary cleaning, we are then able to observe the distribution of the image labels `has_oilpalm`:

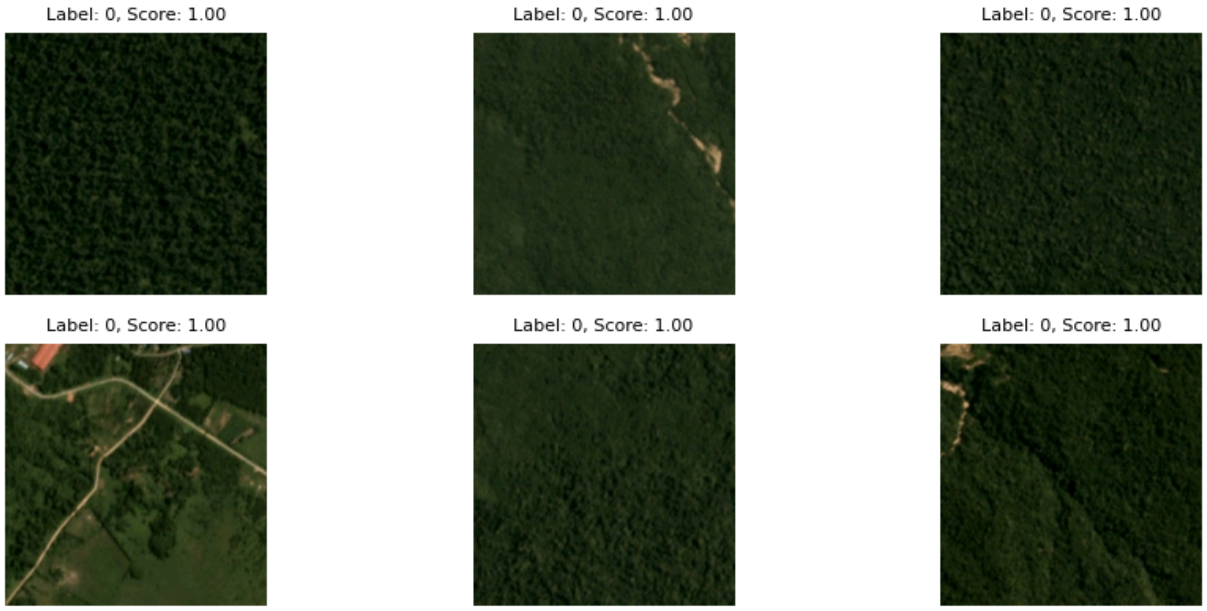


We observe that the image labels are highly imbalanced, where the negative class accounts for about 94 percent of the images, whereas the remaining 6 percent are of the positive class.

We also want to investigate the distribution of `score` among the negative class images. In this case, about 83 percent of those images contain a score of 1, where the satellite image has a high likelihood of not indicating any presence of a palm oil plantation. To further understand the distributions of `score`, we further split it into:

- `score` equals 1
- `score` is between 0.7 inclusive and 1 non-inclusive
- `score` is between 0.5 inclusive and 0.7 non-inclusive
- `score` is below 0.5

We then randomly sampled the negative class images that met these `score` values and plotted them:

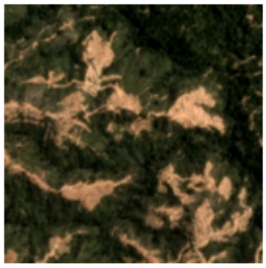


Label: 0, Score: 0.81

Label: 0, Score: 0.80

Label: 0, Score: 0.80

Label: 0, Score: 0.80



Label: 0, Score: 0.80

Label: 0, Score: 0.80

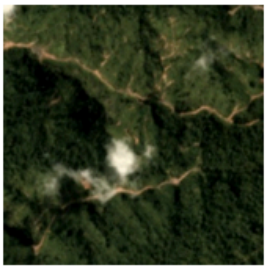
Label: 0, Score: 0.61

Label: 0, Score: 0.60

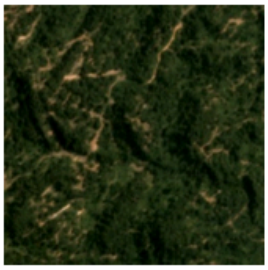
Label: 0, Score: 0.60

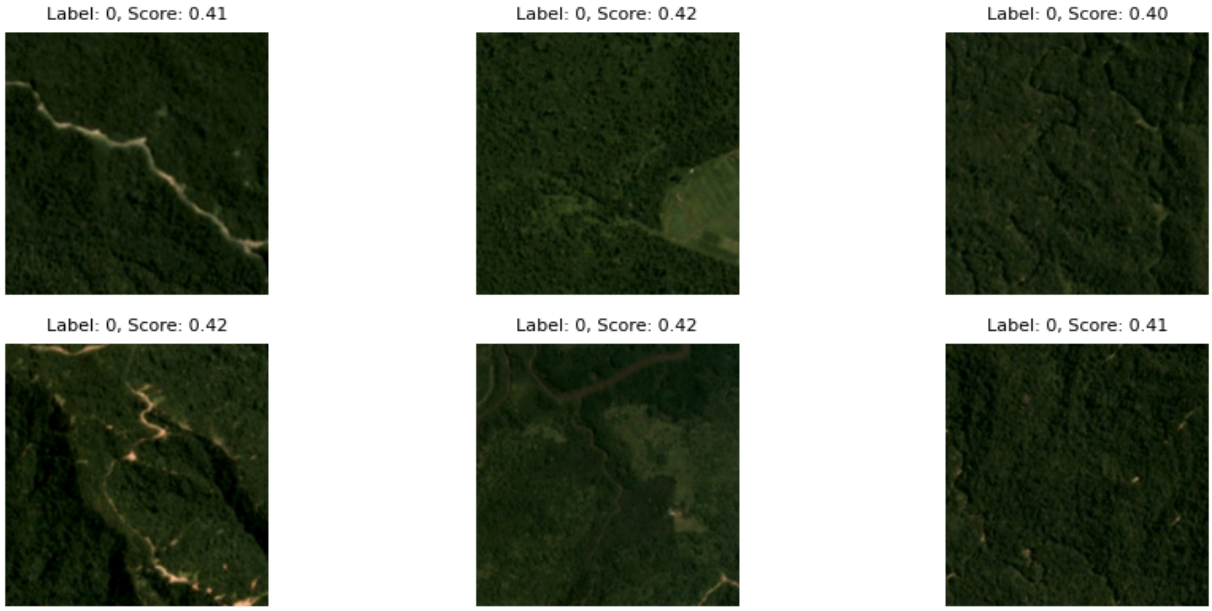
Label: 0, Score: 0.60

Label: 0, Score: 0.61



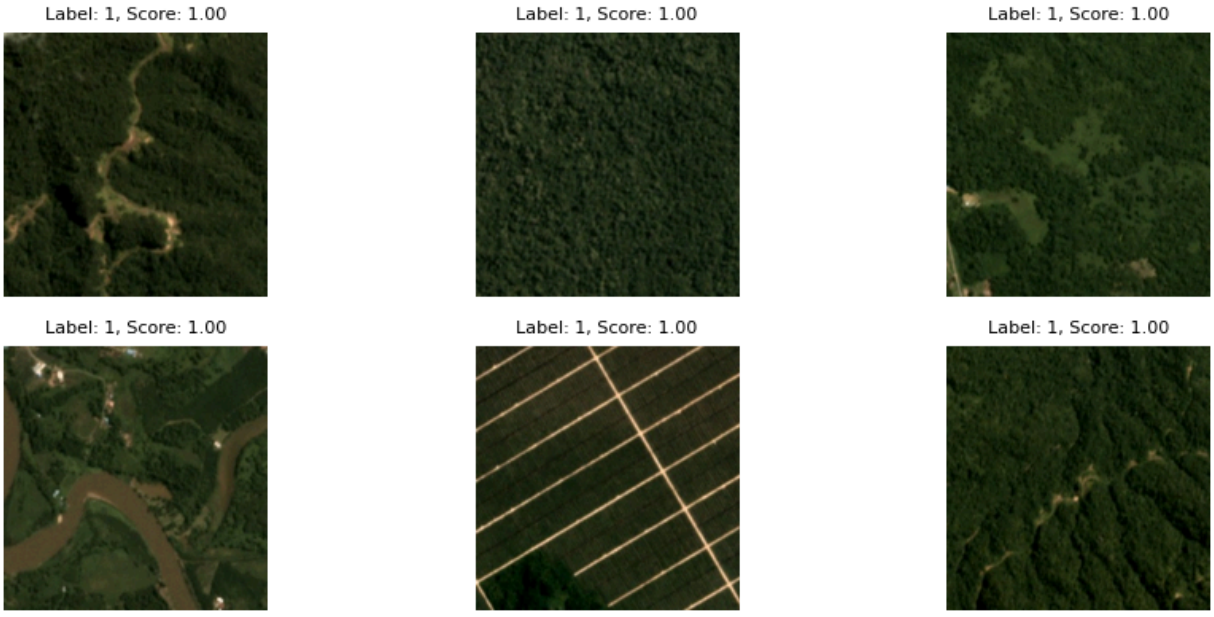
Label: 0, Score: 0.59





Throughout these images, we see that there is lush vegetation and mountainous terrain, as well as signs of human settlements. However, as the `score` decreases, it becomes less clear about any indication of a palm oil plantation.

For the positive class images, about 82 percent of them have a `score` of 1. We then repeated this process as done previously:

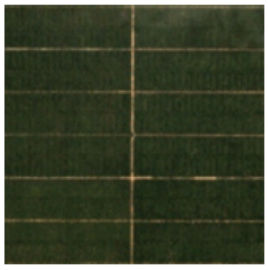


Label: 1, Score: 0.79

Label: 1, Score: 0.79

Label: 1, Score: 0.80

Label: 1, Score: 0.80



Label: 1, Score: 0.81

Label: 1, Score: 0.81

Label: 1, Score: 0.60

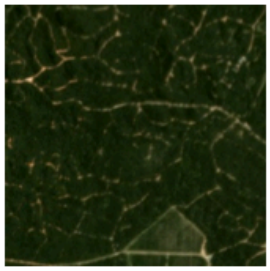
Label: 1, Score: 0.60

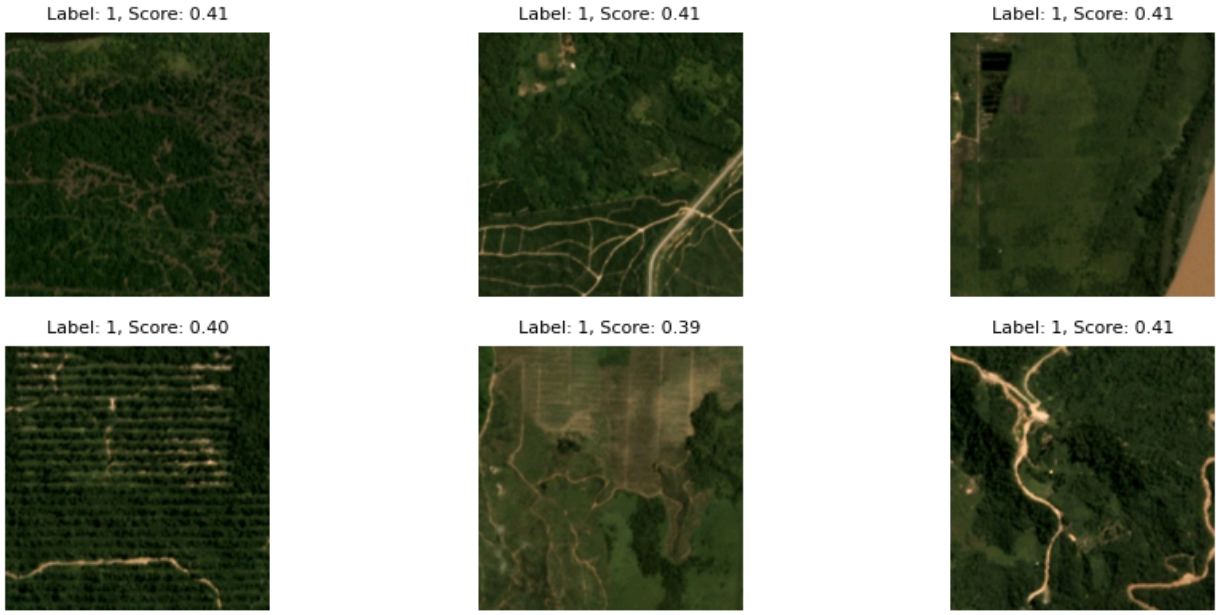
Label: 1, Score: 0.62

Label: 1, Score: 0.60

Label: 1, Score: 0.61

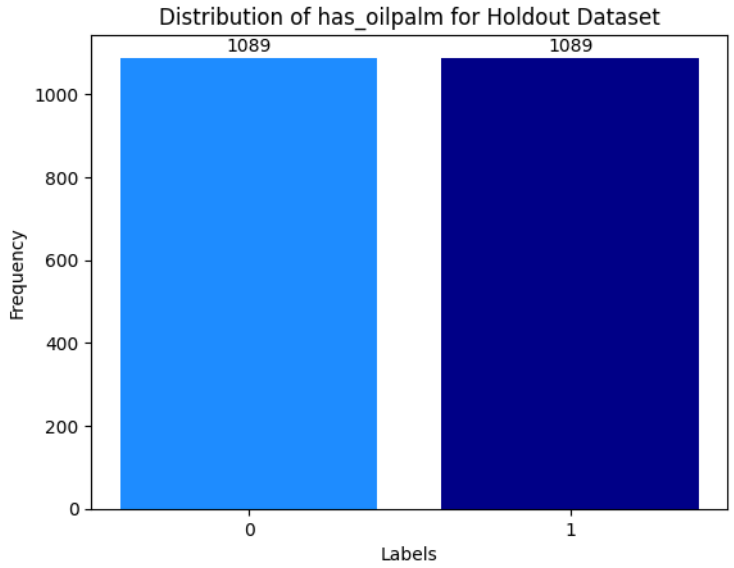
Label: 1, Score: 0.63



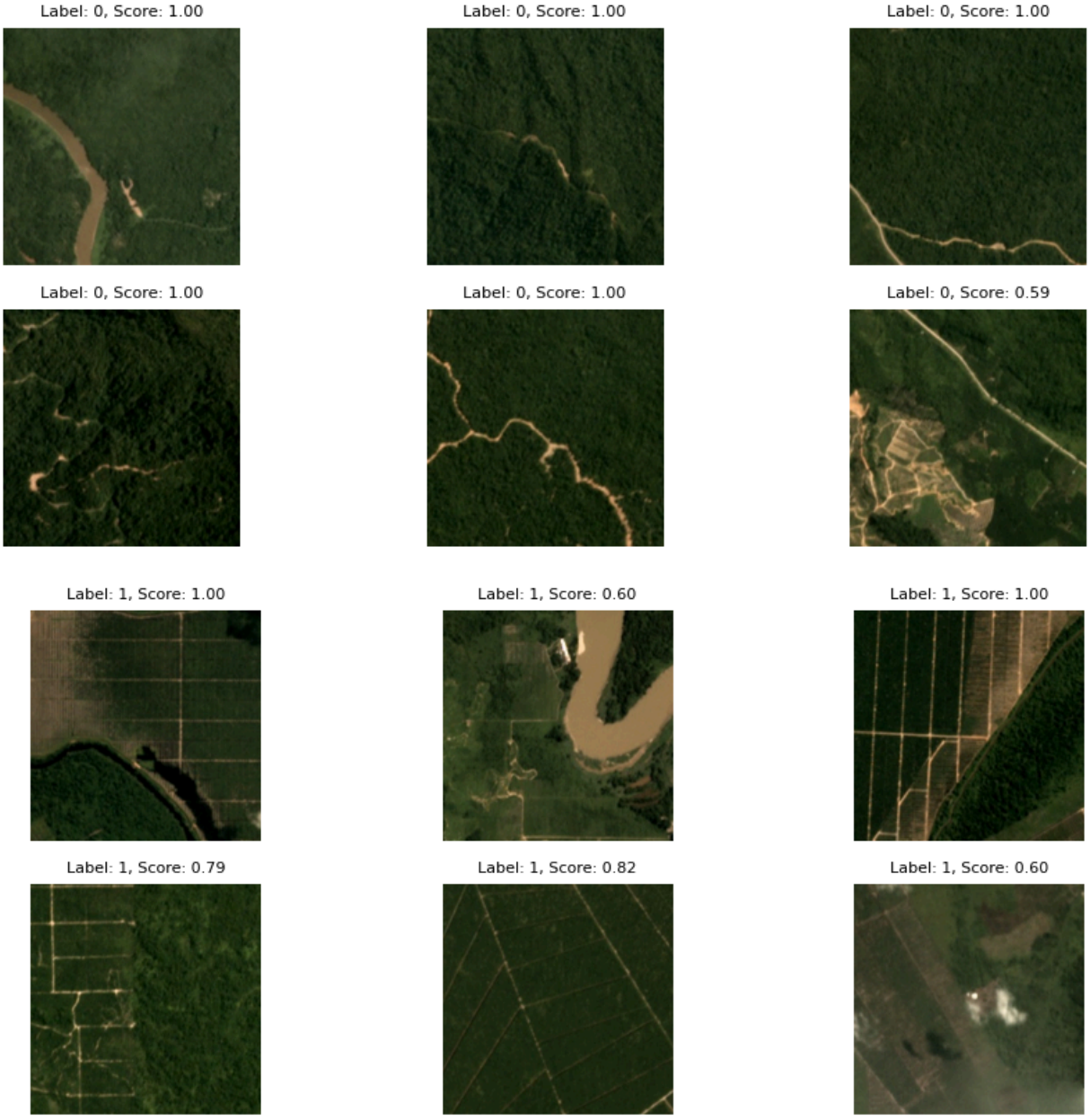


These images generally show clear indications of a palm oil plantation, where we see numerous palm plants and paved paths. However, similar to the negative class images, as the `score` decreases, the images become less clear-cut (e.g. it is possible that a given lower-scored positive class image is probably an inactive palm oil plantation or even another type of plantation).

For the holdout dataset, the image classes are perfectly balanced.



A sample of the negative class and positive class images is shown below:



DATA WRANGLING

Since the dataset was highly imbalanced, both undersampling of the negative class and oversampling of the positive class were utilized. However, it was also important to establish the criteria for `score` for both classes, namely:

- For the negative class images, set `score` equal to 1

- For the positive class images, set `score` greater than or equal to 0.5

Choosing the numerical threshold for the positive class images was to ensure robustness to our dataset. After filtering the training annotations based on those two criteria, 7671 images were of negative class and 593 images were of positive class. In this case, the negative class images were downsampled to 5930 through random sampling, and the positive class images were oversampled to 5930 via augmentation.

For the augmentation, we utilized six types of transformations:

- Random brightness, applied 2 times separately
- Random contrast, applied 2 times separately
- Random blur, applied 2 times separately
- Rotate by 90 degrees
- Rotate by 180 degrees
- Rotate by 270 degrees

The image transformations of a positive class image are shown below:

Besides the three types of rotations, for the random brightness, random contrast, and random blur transformations, we wanted to retain as much of the original image as possible while introducing robustness. In particular, this takes into account variations in color, lighting, and overall image quality. Thus, the resulting images are rather subtle.

Since the model utilizes VGG-19, the images needed to be preprocessed in a specific manner for it to handle properly, namely via the `preprocess_input` function from `tensorflow.keras.applications.vgg19`; the resulting images were resized to (224, 224) with 3 color channels. After the training images were preprocessed, their numerical arrays and their corresponding labels were serialized and zipped using the `gzip` and `pickle` libraries. The same procedure was done for the holdout images. Because of the resulting file sizes (around 8 GB), the files were uploaded to Google Drive to be later utilized in modeling.

MODELING

The serialized and zipped images were unloaded for modeling. The training data was then train-test split using an 80/20 stratified split. This was further split into training and validation using another 80/20 stratified split.

As previously mentioned, the model utilizes VGG-19, which is a pre-trained convolutional network that consists of 19 layers (16 convolutional and 3 fully

connected layers, to be precise) and is trained on over 1 million images from the ImageNet collection. More precisely, we are utilizing transfer learning since we are adapting a pre-trained neural network for this specific task. Furthermore, because VGG-19 is pre-trained, we freeze its initial layers to leverage its feature extraction capabilities to help extract certain patterns found in these satellite images. However, since VGG-19 is not trained on these specific satellite images, we also introduce our own custom layers, namely five dense layers each using the rectified linear unit (ReLu) activation function and a final dense layer using the sigmoid activation function. Respectively, the number of nodes for each dense layer were 300, 200, 100, 50, 25, and 1; the number of nodes decreased gradually. After each layer, dropout, where we randomly deactivate a proportion of nodes in a layer, was introduced at rates of 0.4, 0.3, 0.25, 0.25, and 0.2; similar to the number of nodes in each dense layer, the dropout rate starts high and then gradually lowers. More importantly, dropouts are a simple yet effective method for preventing the model from overfitting.

One of the biggest challenges early-on was that even though the training and validation losses generally kept decreasing together as the number of epochs increased, the model would overfit and produce highly-skewed predictions for one image class over the other. In this case, this behavior started happening after around six epochs.

Hyperparameter tuning was also utilized to address overfitting, and one effective way was by using Kaiming normal initialization on the first five dense layers that used the ReLu activation function. This method, introduced by Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun in their paper *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*, is used to address the vanishing and exploding gradient problems. In the derivation, He and his colleagues established the relationship:

$$W \to \mathrm{N}(0, \tfrac{2}{n})$$

More precisely, the weights are random numbers that follow a Gaussian/normal distribution of mean 0 and variance of $\frac{2}{n}$. The factor of 2 in the variance is specific to the ReLu activation function. Traditionally, in deep neural networks that utilize

ReLu activation functions, the weights that are initialized through random normal or Xavier initialization can potentially lead to gradients that vanish or explode as they are propagated through the layers during backpropagation, which hinders the model from training effectively.

Another hyperparameter that was utilized was L2 regularization (i.e. Ridge regularization or weight decay) on the final dense layer that uses the sigmoid activation function. Unlike L1 regularization that pushes the weights to zero, which in turn promotes sparsity, L2 regularization pushes the weights to be small but not equal to zero. In this case, L2 regularization adds a term to the loss function that is proportional to the sum of the squares of the weights. More precisely:

$$L_{\text{total}} = L_{\text{original}} + \lambda \sum_{i=1}^{n} w_i^2$$

L_{total} represents the total loss with regularization, $L_{original}$ is the original loss using mean-squared error, λ is the regularization parameter that controls the strength of the penalty, and each w_i is the model's weights. L2 regularization is beneficial since:

- the smaller weights reduce model complexity and prevent overfitting
- the weights are more evenly distributed (i.e. it avoids the case where some weights are much larger than others)
- there is more numerical stability, especially when there is multicollinearity or when features are highly correlated with each other
- improves interpretability by reducing the influence of less important features

L2 regularization was applied only on the final dense layer to prevent over-regularization, which can affect the model's performance.

Given the fact that the model began overfitting around after six epochs, a grid search was performed of two hyperparameters: the number of epochs and a set of L2 regularization values. Here, the list that contains the number of epochs was

defined as `epoch_values = [3, 4]`, and the list that contains the L2 regularization values was defined as `l2_values = [0.03851, 0.03852, 0.03853, 0.03854, 0.03855]`. The values in `l2_values` were chosen since in earlier iterations, values between 0.038 and 0.039 yielded the best results. This also considers the computational costs. Altogether, the best-yielding hyperparameters were `epoch = 3` and `l2_value = 0.03851`.

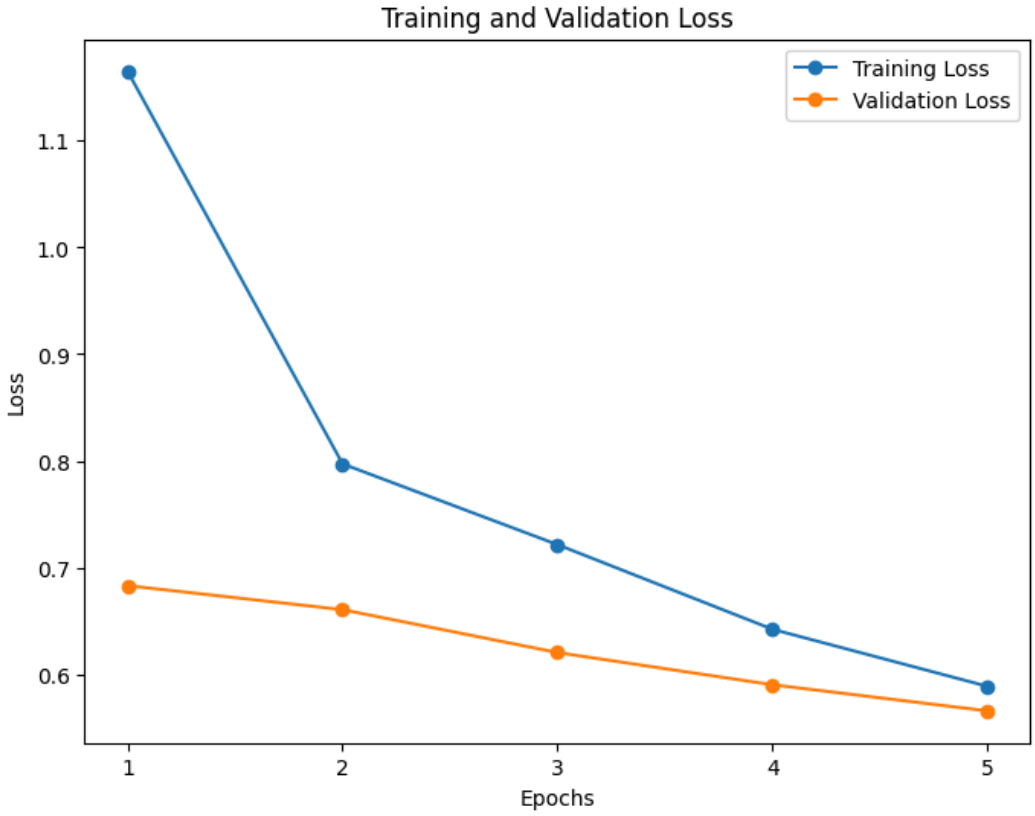
For the optimizer, AdamW was used instead of Adam, with `learning_rate = 1e-4` and `weight_decay = 5e-5`. Both optimizers handle weight regularization differently. In Adam, the weight decay is applied indirectly when updating gradients, which can unintentionally modify the model's adaptive learning capabilities and interfere with the optimization process. On the other hand, AdamW separates the weight decay from the gradient step, which ensures regularization impacts the parameters without altering the model's adaptive learning capabilities. As a result, this helps regularize models more precisely and help models generalize better.

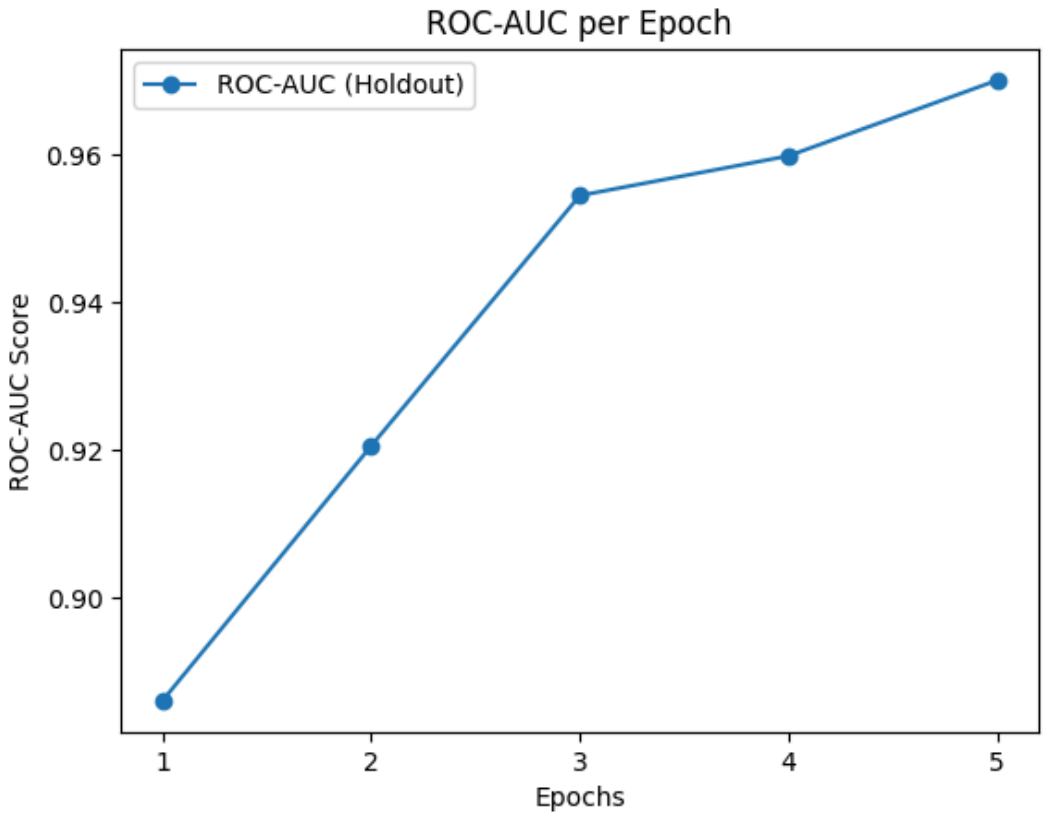
The model also utilizes a custom callback to keep record of several metrics evaluated on the holdout dataset, namely:

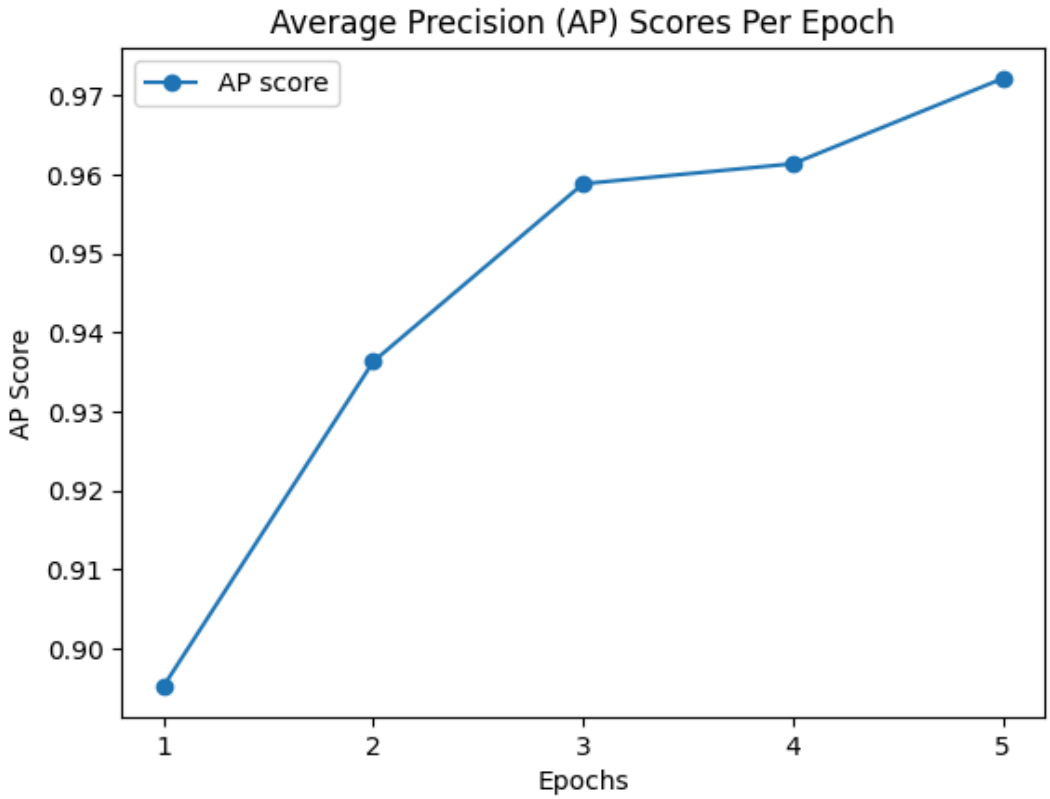
- Precision scores of both classes
- Recall scores of both classes
- F1 scores of both classes
- ROC-AUC score across epochs
- Average precision (AP) score

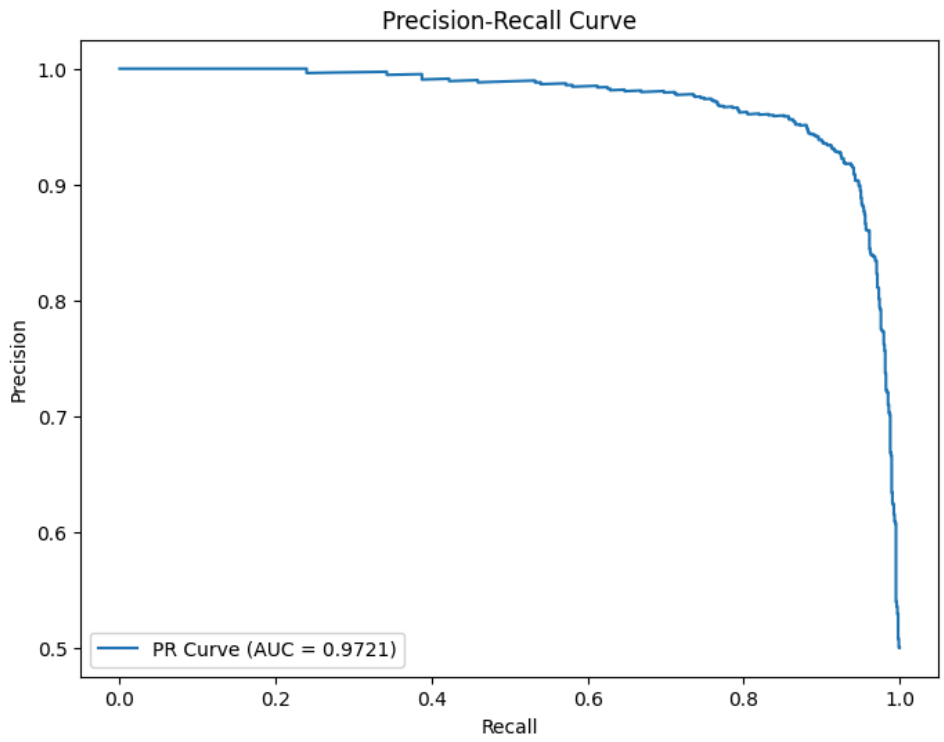
While taking into account of trying to optimize for the best receiver operator characteristic - area under ROC curve (ROC-AUC) score and average precision (AP) score – due to data imbalance – of the holdout set, the condition that was used to determine the best model in this grid search was to consider the average F1 scores of both image classes (i.e. `avg_f1_score = (f1_class_0 + f1_class_1) / 2`). This was chosen because it was important to be able to minimize both type I (predicting the presence of a palm oil plantation when there is none) and type II (predicting no presence of a palm oil plantation when in reality there is) errors. The best model yielded from the grid search produced a holdout ROC-AUC score of 0.9545 and a holdout AP score of 0.9588. This model was subsequently saved,

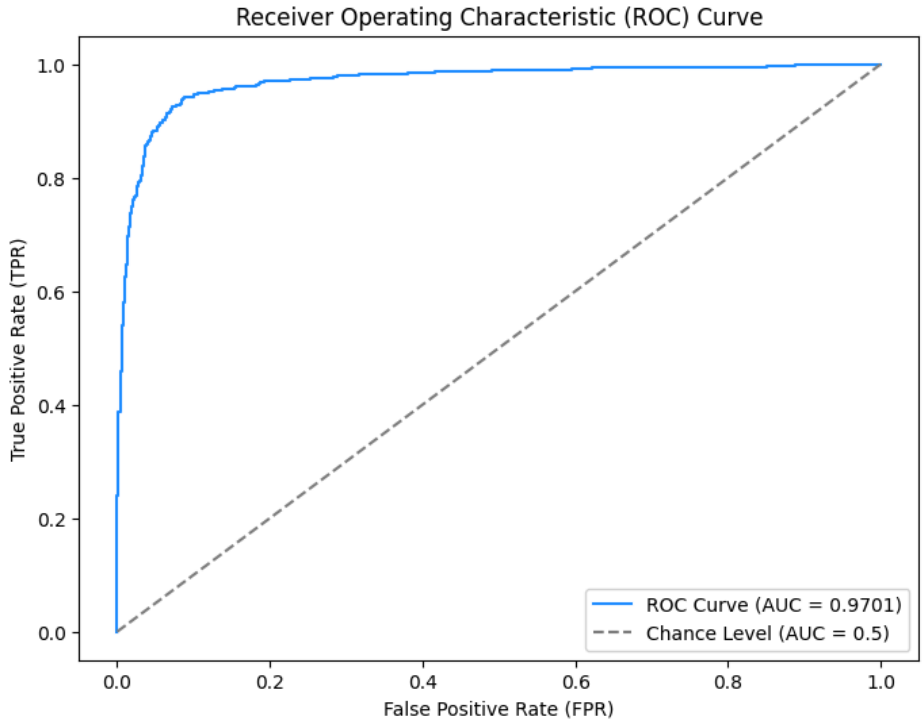
loaded, and then trained for two additional epochs with the layers unfreezed for fine-tuning. The learning rate for the AdamW optimizer was also lowered, where `learning_rate = 1e-5`. The final model yielded a holdout ROC-AUC score of 0.9701 and a holdout AP score of 0.9721. The training and validation losses, as well as the ROC-AUC and AP scores across epochs, and the confusion matrix are shown below:



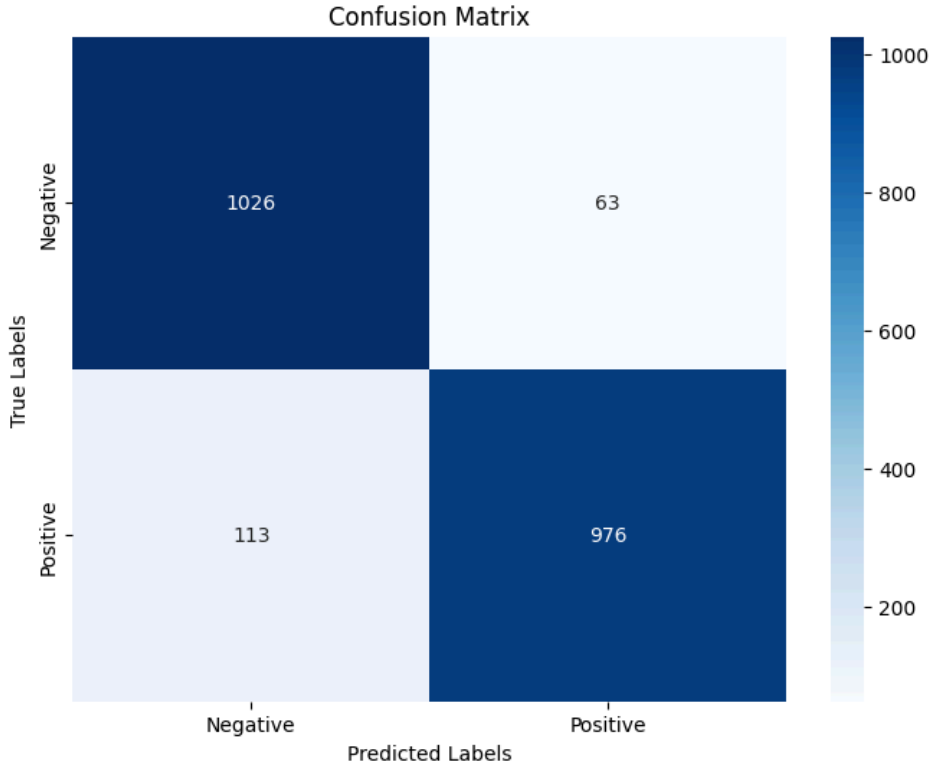

ROC-AUC per Epoch
ROC-AUC (Holdout)
ROC-AUC Score
0.96
0.94
0.92
0.90
1
2
3
4
5
Epochs


Average Precision (AP) Scores Per Epoch
AP score
AP Score
0.97
0.96
0.95
0.94
0.93
0.92
0.91
0.90
1
2
3
4
5
Epochs


Precision-Recall Curve
Precision
Recall
PR Curve (AUC = 0.9721)
1.0
0.9
0.8
0.7
0.6
0.5
0.0
0.2
0.4
0.6
0.8
1.0


Receiver Operating Characteristic (ROC) Curve
True Positive Rate (TPR)
False Positive Rate (FPR)
ROC Curve (AUC = 0.9701)
Chance Level (AUC = 0.5)
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

	precision	recall	f1-score	support
0	0.90	0.94	0.92	1089
1	0.94	0.90	0.92	1089
accuracy			0.92	2178
macro avg	0.92	0.92	0.92	2178
weighted avg	0.92	0.92	0.92	2178



In general, we see that the training and validation losses decrease together and end at around the same level after the fifth epoch. In addition, the ROC-AUC and AP scores for the holdout dataset increase after each epoch, which indicates that the model had progressively improved; both metrics were considerably high at the end since both were close to 1. For the confusion matrix, we see that the model has predicted 94% of the negative class and 90% of the positive class correctly, also achieving an F1-score of 0.92 for both classes.

CONCLUSION

Overall, the final model yielded remarkable results, particularly due to data augmentation, hyperparameter tuning, and fine-tuning. Addressing the initial data imbalance was critical, hence the need for data augmentation. In addition, hyperparameter tuning addressed overfitting and helped optimize our model, and fine-tuning allowed us to adapt the pre-trained VGG19 model for our specific problem.

In the future, one thing we can do differently is try a different pre-trained model like ResNet-50. Additionally, we can readjust the data augmentation for the positive class images, namely by applying random brightness, random contrast, and random blur a single time instead of twice. This would also require us to slightly further undersample the negative class images. That being said, another thing we could try differently is setting a different threshold for filtering the positive class images by setting the `score` greater than or equal to 0.6. Finally, one thing we could also try differently is using a random grid search to save on time and computational costs.

REFERENCES

WWF. “8 Things to Know about Palm Oil.” WWF, WWF, 12 Nov. 2018, www.wwf.org.uk/updates/8-things-know-about-palm-oil.

“Palm Oil | USDA Foreign Agricultural Service.” Usda.gov, 2024, www.fas.usda.gov/data/production/commodity/4243000.

Pykes, Kurtis. “AdamW Optimizer in PyTorch Tutorial.” Datacamp.com, DataCamp, 21 Oct. 2024, www.datacamp.com/tutorial/adamw-optimizer-in-pytorch.

MathWorks. “Mel Frequency Cepstral Coefficients.” Mathworks.com, 2020, www.mathworks.com/help/audio/ref/cepstralcoefficients.html, https://doi.org/1061245463.woff2.

Vaj, Tiya. “Why We Freeze Some Layers for Transfer Learning - Tiya Vaj - Medium.” Medium, 24 Jan. 2024, vtiya.medium.com/why-we-freeze-some-layers-for-transfer-learning-f35d9f67f99c.

“VGG-Net Architecture Explained.” GeeksforGeeks, 7 June 2024, www.geeksforgeeks.org/vgg-net-architecture-explained/.

“Kaiming Initialization in Deep Learning.” GeeksforGeeks, 27 Dec. 2023, www.geeksforgeeks.org/kaiming-initialization-in-deep-learning/.

GeeksforGeeks. “L1/L2 Regularization in PyTorch.” GeeksforGeeks, 31 July 2024, www.geeksforgeeks.org/l1l2-regularization-in-pytorch/. Accessed 7 Feb. 2025.

Brownlee, Jason. “A Gentle Introduction to Dropout for Regularizing Deep Neural Networks.” Machine Learning Mastery, 3 Dec. 2018, machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/.