FRESNO STATE

Tree Classification using Digital Game Models

Kyle Andrews and Robert Mawhinney

CSCI 166

Dr. Athanasios Panagopoulos

December 6, 2022

Code Repository: https://github.com/principles-of-ai/supervised-learning

I. Motivation

The primary motivation for this project started with image classifiers. Images classifiers are already utilized in many areas of machine learning. The goal of image classifiers, specifically convolutional neural network models, is to train labeled images and then reliably classify unlabeled images.

Image classifiers are useful because they can solve a problems that color clicking cheat clients struggle to solve. A color clicking cheat client identifies objects in a video game scene using pixel color(s) of the objects. This is a problem because game brightness and the rotation of the game scene creates instability in the pixel colors. Current solutions to this is to use advanced cheat clients which inject into the game code to find the exact models or object id. However this type of client is detectable and your account will be banned if you use these methods.

Image classification can solve some of these problems and potentially be fully undetectable. Image classification is accurate with enough training data. What if we took a lot of screenshots of the different trees in the game and tried to classify them?

II. Problem Statement

Without using a cheat client, finding objects in a game scene can only be done by using pixel color matching or injecting into the game code. Can we solve this problem using a convolutional neural network? Can we accurately identify trees in a game scene and remain fully undetectable?

III. Background & Related Work

Supervised learning is a fast solution to classification problems. Convolutional Neural Networks are especially great at image classification [1] [2].

A Convolutional Neural Network was used instead of a normal neural network to overcome the limitations. Pixel adjacency is needed to increase accuracy in identifying features. Training a fully connected neural network produces a specific result. If the pixels are shuffled in the training images, completely distorting the original image, the newly trained network produces the same result [3]. Thus pixel adjacency is not preserved. The input vector consists of all the pixels of the input image. Meaning, a 128x128 RGB image creates 2 billion trainable weights in the fully connected layer. The model takes too long to train for large data sets.

Convolutional neural networks simplify these problems. Instead of feeding all the pixels as the input vector we instead feed parts of the image into the network. These parts are scanned for features by applying a convolution on a kernel matrix. This preserves pixel adjacency. In addition, each convolutional layer added to the network increases feature detection and creates a feature hierarchy in the network.

Convolutional neural networks have been proven to be effective at image classification producing metrics with high accuracy and precision if provided with enough training data [4].

IV. Approach

To train the model we need a lot of images. We can fabricate images using image augmentation which will slightly alter the training images. This can also prevent overfitting [5].

Will 84 images be enough? Probably not. Since the model is a game model it has no variance. Can

we just duplicate those images to generate more images? More epochs or rounds of training? What about adding more convolution layers or removing them to reduce feature extraction? These are the questions that we must explore.

Classes
We have 4 classes: *Tree, Oak, Willow, and Yew*. Each class contains 21 images producing 84 total images.

Convolutional Neural Network

1. Create model
2. Evaluate results
3. Tune the model
4. Repeat step 1

A convolutional network model was constructed using TensorFlow [6]. Each iteration requires evaluating the results to see how the model is performing. We can do this using *metrics* and visualizing the performance using a *confusion matrix*. With this information we can evaluate our model and repeat the experiment. [7]

Starting Parameters

- Filters
- Kernel size
- Learning rates

The starting parameters were initialized by referring to tutorials, and also reading the TensorFlow documentation [6]. There was a unique convention used on *filters* that suggest each additional convolutional layer added to the network–double the filters into the next layer.

Training

- Training
- Validation
- Image Augmentation
- Duplicated images?

To train the network we also need a validation training set. Some of the training images will be allocated to validate the model. Image augmentation can assist with a lack of images and prevent overfitting. This inspired a question: Game models do not change–is there harm in duplicating the training set to get more training images? Will image augmentation assist in preventing overfitting?

V. Experimental Setup

Using a Jupyter Notebook the entire experiment procedure was documented block by block. Training of the convolutional neural network was done using a 1080ti GPU.

Training considerations:

- How many images to feed into the network (batch)?
- What are the image dimensions of the data?
- How many training rounds to run (epochs)?
- How many layers/filters?

Starting with an image dimension of $140x140x3$ (RGB), a batch size of 32, and 15 epochs. We tested augmented images and standard training images. The first test is to discover how many epochs lead to overfitting. From there we can reduce and find the optimal epoch.

Epoch Test

- Ran a series of training iterations on $15, 50,$ and 100 epochs
- If image augmentation trains better keep augmenting images

Duplicate Training Set

- Duplicate the training data until we have $1,200$ images of each class
- Game models have no variance so overfitting is expected

Learning Rate

- Change the learning rate to see if we need to slow down or speed up

Convolutional Filters/Layers

- Each filter added to the network can be thought of as feature extracting
- Discover how many filters we need to optimally extract features

VI. Results and Discussion

Evaluating the experiment setup can be easily observed. Some of the changes improved the model, some did nothing at all, and some adjustments completely broke the model rendering it useless.

Epoch Test

- Ran a series of training iterations on $15, 50,$ and 100 epochs.

  This test demonstrated the power of image augmentation and how it prevents overfitting during training. With more epochs we simulated more training data because all of the training images were slightly augmented. Non-augmented images overfit the training data and the validation set failed to improve. This test concluded and we decided to keep image augmentation into the final model.

Duplicate Training Set

- Duplicate the training data until we have $1,200$ images of each class

  This test had extreme results because every test with duplicated data performed the same: very poor accuracy on unseen data, and immediate overfitting with just 5 epochs. This test performed well in classifying training data but had poor accuracy in identifying unseen data. This test concluded that duplicating the training set does not improve model accuracy. It overfits the model and cannot classify unseen images accurately.

Learning Rate

- Change the learning rate to see if we need to slow down or speed up

  The learning rate did not have an impact until the rate was set to 0.025. With a high learning rate the model converged to one class and the output always produced Oak labels. Since this damaged the model's accuracy we decided to revert the learning rate back to the default 0.001.

Convolutional Filters/Layers

- Each filter added to the network can be thought of as feature extracting.

- How many filters do we need to optimally extract features?

  Adding more layers to the network did not change the model in observable ways. In the first layer we tested $32, 64, 128,$ and 256 filters. The model performed the same in all cases except with more filters reducing the amount of trainable weights. We believe that additional filters for this limited data set did not produce observable results. Testing just one filter yielded similar results to the 0.025 learning rate test. With 1 filter the model converges to one class for all predictions.

  Since the results were not observable to higher filters we set the first layer to 64 filters to maximize trainable parameters in the beginning layers.

VII. Contributions and Conclusions

The amount of images needed to accurately predict labels for unseen images was not met in this project. The adjustments made to the model attempted to train and predict using only 84 images. The lack of images created issues that no amount of parameter tuning could yield accurate results [7]. Not all changes created immediate observable results but some changes, such as the learning rate, produced immediate results. In the future if this project were to be repeated the results could potentially be improved by providing the model with a larger training set. Each image added to the network improves accuracy to the model [7].

References

[1] Abadi, M. *et al.* TensorFlow: Large-scale machine learning on heterogeneous systems (2015). URL https://www.tensorflow.org/. Software available from tensorflow.org.

[2] Education, I. C. What are convolutional neural networks? *IBM Cloud Learn Hub* (2020). URL https://www.ibm.com/cloud/learn/convolutional-neural-networks.

[3] Russell, S. J. & Norvig, P. *Artificial Intelligence: A Modern Approach, 4th edition*. Pearson series in artificial intelligence (Pearson, 2020). URL https://www.pearson.com/en-us/subject-catalog/p/artificial-intelligence-a-modern-approach/P200000003500/9780137505135.

[4] Fricker, G. A. *et al.* A convolutional neural network classifier identifies tree species in mixed-conifer forest from hyperspectral imagery. *Remote Sensing* **11** (2019). URL https://www.mdpi.com/2072-4292/11/19/2326.

[5] Jain, T. Basics of image classification techniques in machine learning. *OpenGenus IQ: Computing Expertise & Legacy* (2019). URL https://iq.opengenus.org/basics-of-machine-learning-image-classification-techniques/.

[6] TensorFlow. Convolutional neural network (cnn) | tensorflow core. *TensorFlow* (2022). URL https://www.tensorflow.org/tutorials/images/cnn.

[7] Brownlee, J. How to improve deep learning performance. *Deep Learning Performance* (2016). URL https://machinelearningmastery.com/improve-deep-learning-performance/.