Leveraging a Convolutional Neural Network (CNN) to Identify Ships

LCDR John Ross

Naval Postgraduate School

Dr. Armon Barton

4 December 2020

Implement a CNN to identify ships using a TensorFlow backend. Describe the details of this implementation and the results. How could this approach be applied in an operational environment?

**Abstract**

In this report, I address the problem of ship classification in a dataset of images. I use a deep learning method known as a CNN to distinguish between images that contain ships from images of the surrounding area that do not contain ships. Using a ResNet-34 CNN, I achieved a classification accuracy of 99.36%.

1 Problem Statement

The contagion-like growth of imagery data is rapidly overtaking the capacity of imagery analysts to perform meaningful analysis and subsequently enhance the situational awareness of theatre commanders. Commercial imagery providers are using constellations of satellites to collect images consisting of the entirety of the Earth each day. This imagery data is being produced so quickly that presently utilized methods of analysis are unfeasible. Government organizations, in addition to commercial providers, significantly contribute to this growth. In fact, government organizations have even become dependent upon the collective contribution of commercial providers to help them leverage their slice of imagery data. The advent of the connectivity enabled by globalization has facilitated this symbiotic relationship. “This flood of new imagery is outgrowing the ability for organizations to manually look at each image that gets captured, and there is a need for machine learning and computer vision algorithms to help automate the analysis process” (R Hammel, 2017).

In this report, I will demonstrate the ability of a CNN to classify images as a *ship* (label of 1) or *not a ship* (label of 0). I used a dataset from Kaggle wherein the images are provided with appropriate labels. Once the images have been preprocessed and reshaped, the convolution layers of the CNN will identify, extract, and learn useful features form the images. These features will consist of patterns within the image such as curves, contrast, lines, and placement of them within the image.

Since CNNs can be computationally expensive, I set up a TensorFlow backend library that would take advantage of CUDA cores on a Graphics Processing Unit (GPU). This allows for multiple computations to be run in parallel on each core of the GPU which reduces the total time required for a model to complete its task. To establish a baseline for comparison, I experimented with the models of other Kaggle users written in the Keras application programming interface with a TensorFlow backend library (Burachonok and Rasymas, 2017). The results on the test sets using their models were between 96 and 97%. In an attempt to improve upon these results, I applied a Residual Network with 34 layers (ResNet-34) to this dataset. A ResNet-34 is considered a deep network and uses *skip connections* to train the network. These *skip connections* connect inputs and outputs in such a way that it enables a network to learn parameters considerably faster. I will report the findings of the experimental results at the end.

2 Data Description

This dataset is presented in formats such that it allows for ease of use and may be used for classification tasks without delay (R Hammel, 2017).

The dataset consists of images (examples shown in figure 1) extracted from commercially obtained satellite imagery collected over the San Francisco Bay and San Pedro Bay areas of California. It includes 4000 80x80 RGB images labeled with either a "ship" or "no-ship" classification. Images were derived from a commercial provider of visual scene products, which are orthorectified to a 3-meter pixel size. The orthorectification of the images is similar to the imagery that an imagery analyst would assess on an operational watch floor of an intelligence center. The pixel value data for each 80x80 RGB image is stored as a list of 19200 integers within the data list. The first 6400 entries contain the red channel values, the next 6400 the green, and the final 6400 the blue. The image is stored in row-major order, so that the first 80 entries of the array are the red channel values of the first row of the image. The "ship" class includes 1000 images. Images in this class contain ships whose center of mass is located the center of the image. Ships of different sizes, orientations, and atmospheric collection conditions are included. The "no-ship" class includes 3000 images. A third of these are a random sampling of different landcover features - water, vegetation, bare earth, buildings, etc. - that do not include any portion of a ship. The next third are "partial ships" that contain only a portion of a ship, but not enough to meet the definition of the "ship" class. The last third are images that have been a challenge to machine learning models, due to bright pixels or strong linear features.

Figure 1. Sample images of the classes of *ship* and *not a ship*. From Ships in Satellite Imagery by R Hammel, 2017. No known Copyright

3 Methodology and Implementation

3.1 Pre-processing

Images collected from any imagery data repository must be properly formatted prior to being fed into a classifier. This is due to the computational nature of machine learning algorithms. The images must be converted to a purely numerical matrix format called a vector, then the numerical values in that vector must be normalized. Prior to this, though, the images must be viewed to gather a sense of the images that may reveal previously unconsidered details about the images. Then the images must be shuffled so that the algorithm does not inadvertently learn patterns related to the order of the images, which isn’t important.

The algorithms within classifiers learn patterns in normalized data more easily than data in its raw form. Once the images have been viewed and preprocessed then the algorithm can learn patterns in the preprocessed image data that we see as curves, contrast, lines, and placement of them within the image. The aforementioned preprocessing steps are not as challenging as they may seem. The preprocessing steps are as follows.

* Reshape the images to be physically reviewed.
* Shuffle the images.
* Normalize the image data so that all values are between 0 and 1.
* Restructure the image vectors to be fed into the input layer of the model.

3.1.1 Viewing Images

Viewing the images requires that the second column of the two-column vector be reshaped to be 3-dimensions. The three dimensions are the color channels of red, green, and blue in addition to the width and height of the image in pixels. In this case the pixel width and height are 80 x 80 pixels.

3.1.2 Shuffling Images

The images are shuffled by creating a variable for the index with a length that is equal to the number of images. That index variable is then shuffled and associated with every image in the dataset.

3.1.3 Restructuring Vectors

Once each image has been associated with a value from the shuffled index variable, the columns of the vector are rearranged so that they can be fed into the input layer of the CNN. If the vector columns are not rearranged, then the code with throw an error if an attempt is made to train the model with the image data.

3.1.4 Normalizing Vector Cell Values

Most machine learning algorithms are designed such that they require values to be scaled or normalized. The features that contribute to learning for image classification are each pixel and varying groupings of these pixels. As previously mentioned, and image’s pixel height and width are 80 x 80 with 3 color channels. For each image, this equates to a total of 19200 pixels with values ranging from 0 to 255. So, these values must be divided by 255 to normalize them to values between 0 and 1.

3.2 Feature Extraction and Representation

3.2.1 Convolution

The convolution layer is the building block of a CNN. The following is a comprehensive description of the purpose and conceptual design of convolution layers as envisioned by Yann LeCun in 1998 and represented in figure 2 (Geron, 2017).

The learnable parameters in a CNN are the weighted connections between neurons. Neurons in the first convolutional layer are not connected to every single pixel in the input image, but only to pixels in their receptive fields which is similar to the visual cortex of some mammals. In turn, each neuron in the second convolutional layer is connected only to neurons located within a small rectangle in the first layer. This architecture allows the network to concentrate on small low-level features in the first hidden layer, then assemble them into larger higher-level features in the next hidden layer, and so on. A neuron’s weights can be represented as a small image the size of the receptive field called a *filter*. A layer full of neurons using the same filter outputs a *feature map*, which highlights the areas in an image that activate the filter the most. During training the convolutional layer will automatically learn the most useful filters for its task, and the layers above will learn to combine them into more complex patterns. A convolutional layer has multiple filters and outputs one feature map per filter. It has one neuron per pixel in each feature map, and all neurons within a given feature map share the same parameters. Neurons in different feature maps use different parameters. A convolutional layer simultaneously applies multiple trainable filters to its inputs, making it capable of detecting multiple features anywhere in its inputs (Geron, 2017).

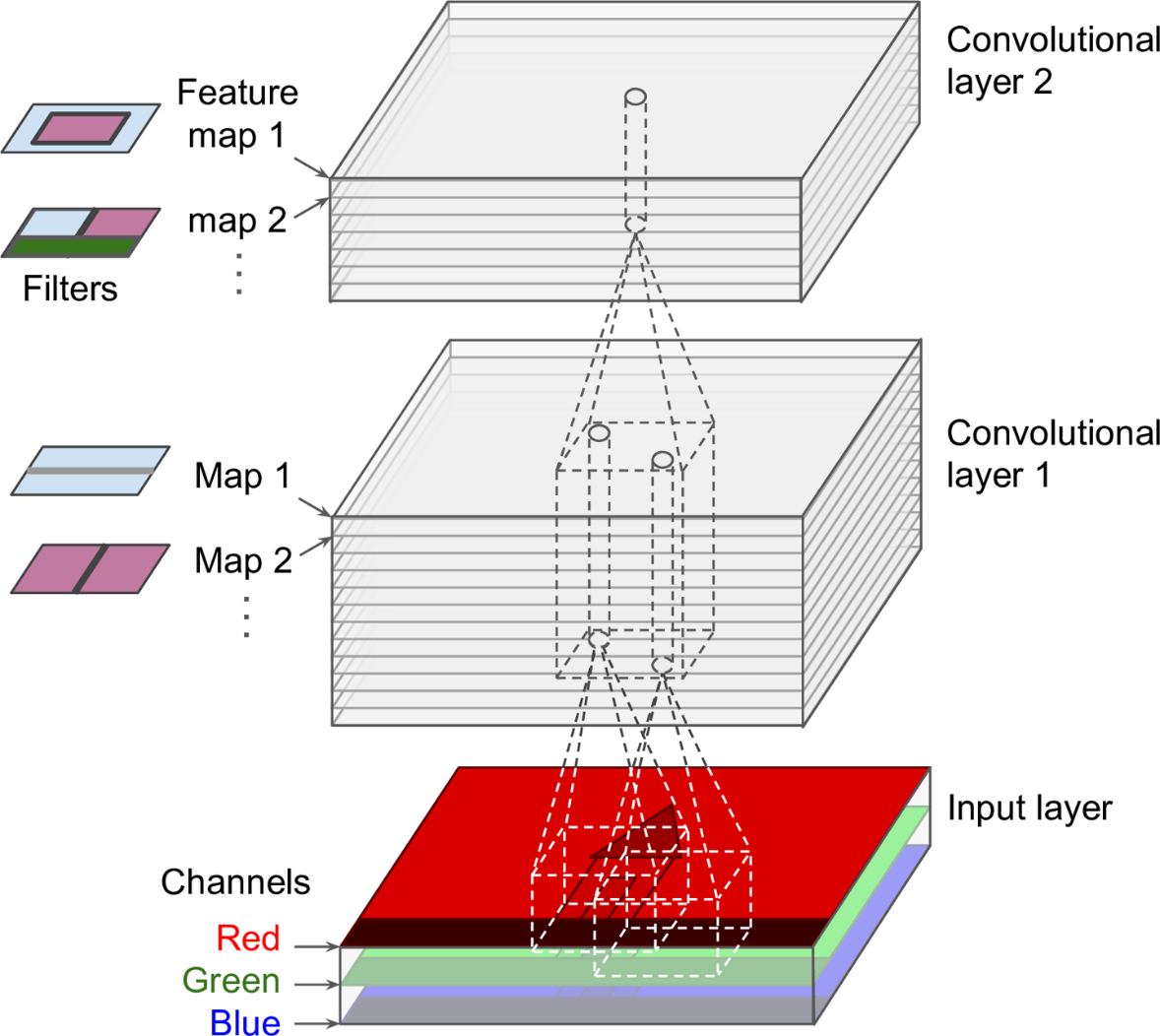
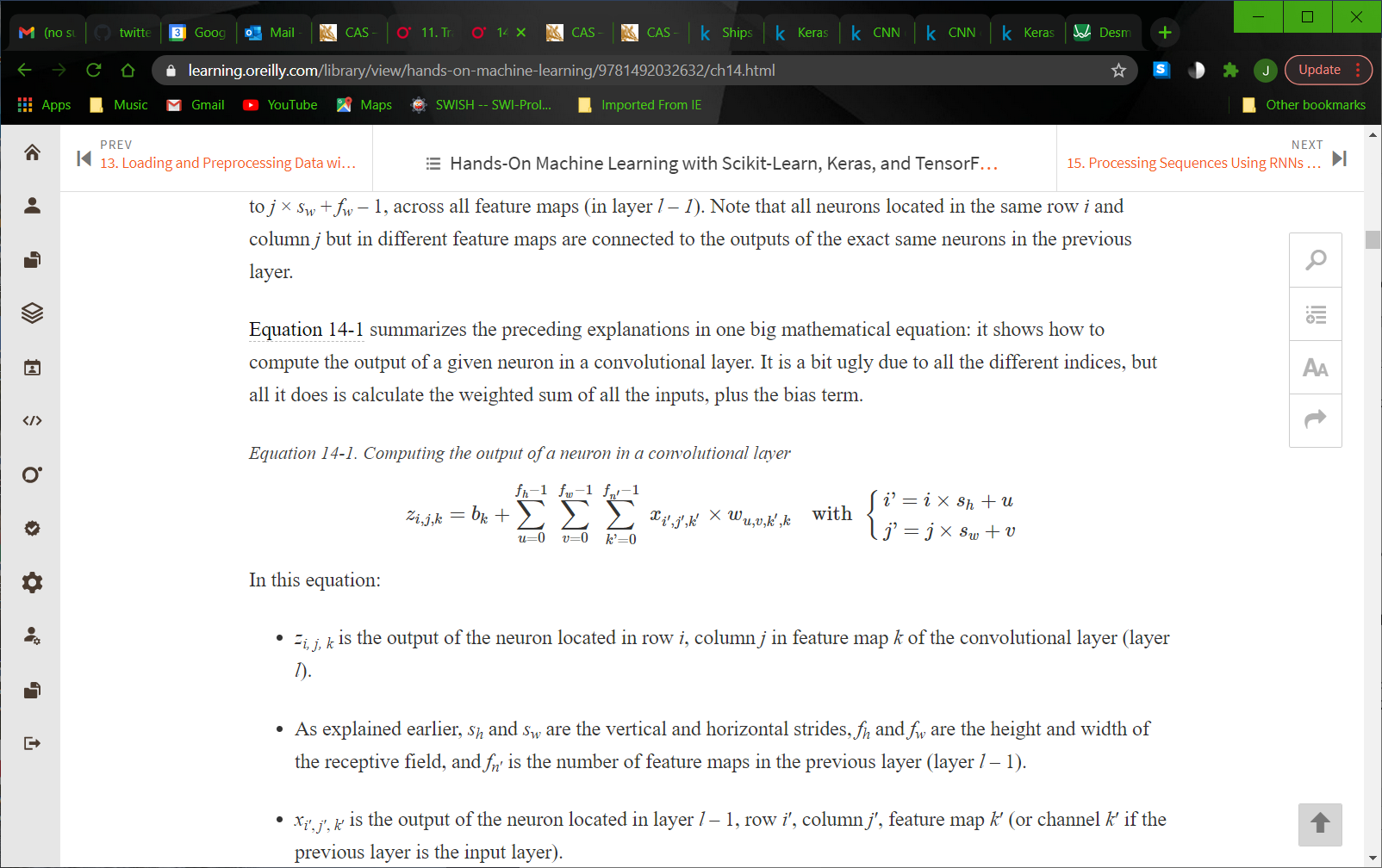


Figure 2. Convolutional layers with multiple feature maps, and images with three color channels. From Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems by A. Geron, 2017. Copyright 2020 by O’Reilly Media, Inc.

3.2.2 Pooling

The goal of pooling is to subsample the input image in order to reduce the computational load, the memory usage, and the number of parameters The first convolution layer of my model uses a 7 × 7 filter with stride 2 and "same" padding, outputting 64 feature maps of size 25 × 25. Each of the 64 feature maps contains 25 × 25 neurons, and each of these neurons needs to compute a weighted sum of its 7 × 7 × 3 = 75 inputs for a total of 5,880,000 million float multiplications (Equation 1).



Equation 1. Computing the output of a neuron in a convolution layer. From Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems by A. Geron, 2017. Copyright 2020 by O’Reilly Media, Inc.

Since these feature maps are represented using 32-bit floats, then the convolutional layer’s output will occupy 64 × 25 × 25 × 32 = 1,280,000 million bits (0.16 MB) of RAM. Each training batch contains 32 instances, so this layer would use up 5.12 MB of RAM. That would not be overburdensome for most modern household laptops and computers, but if the feature maps were large then pooling would be a useful method to reduce computational load, memory usage, and the number of learnable parameters. Similar to convolutional layers, each neuron in a pooling layer is connected to the outputs of a limited number of neurons in the previous layer, located within a small rectangular receptive field as represented in figure 3 (Geron, 2017).

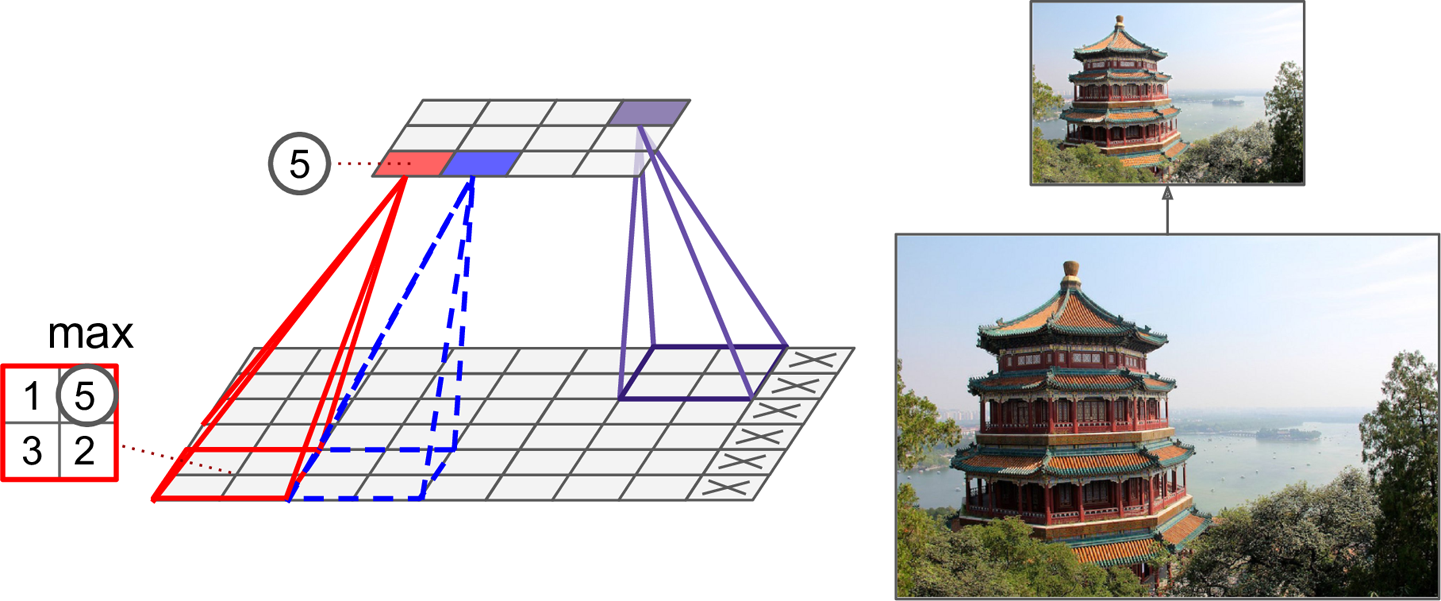


Figure 3. Max pooling layer (2 x 2 pooling kernel, stride 2, no padding. From Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems by A. Geron, 2017. Copyright 2020 by O’Reilly Media, Inc.

3.3 Classifiers

3.3.1 CNN

The CNN used as a baseline for comparison had the typical structure as described in the text. Which was to “stack a few convolutional layers with ReLU activation, then a pooling layer, then another few convolutional layers which also use ReLU activation, then another pooling layer” (Geron, 2017). Each architecture was nearly identical achieved an accuracy of approximately 96%.

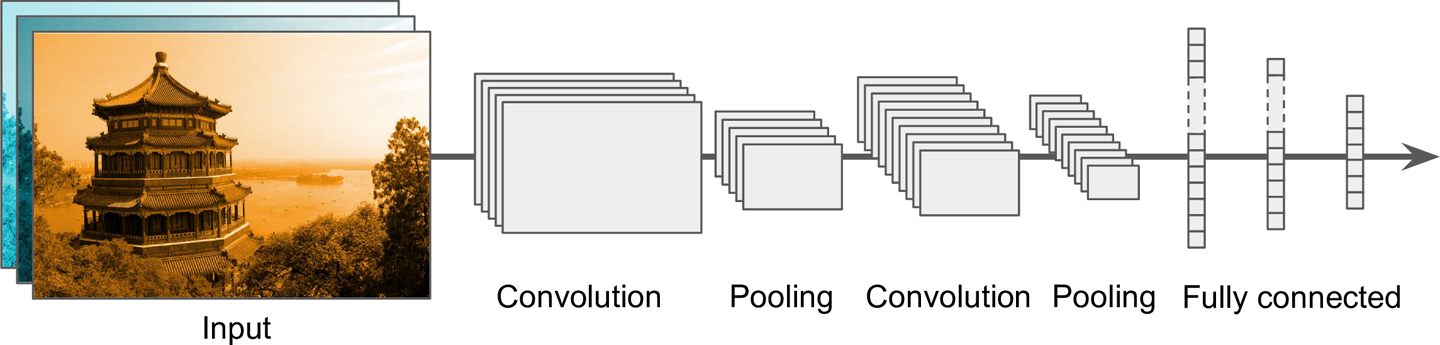


Figure 4. Typical CNN architecture. From Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems by A. Geron, 2017. Copyright 2020 by O’Reilly Media, Inc.

3.3.2 ResNet-34 CNN

The ResNet architecture expands upon the typical CNN architecture by stacking *residual units* after the pooling layer. “Each residual unit is composed of two convolutional layers, with Batch Normalization (BN) and ReLU activation, using 3 × 3 kernels and preserving spatial dimensions with a stride of 1 with "same" padding. ResNet-34, shown in figure 5, is the ResNet with 34 layers 17 containing 3 residual units that output 64 feature maps, 4 RUs with 128 maps, 6 RUs with 256 maps, and 3 RUs with 512 maps” (Geron, 2017).

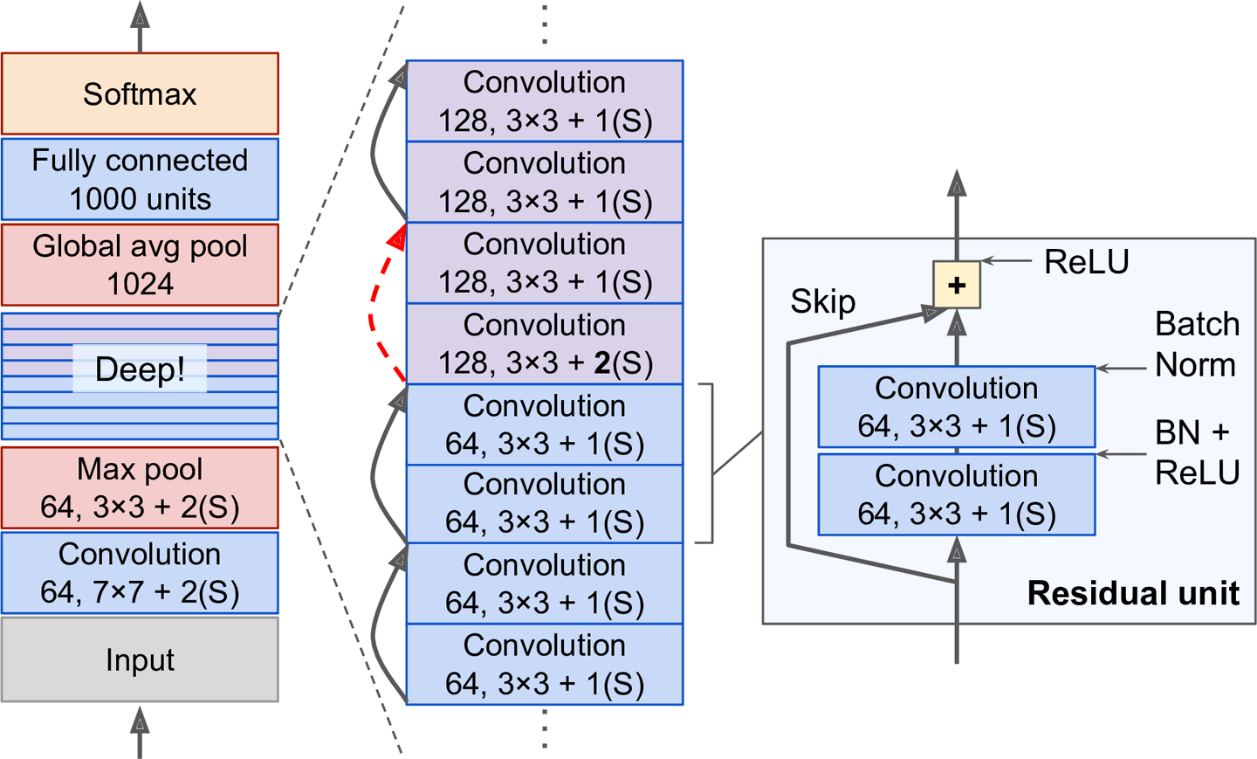


Figure 5. ResNet architecture. From Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems by A. Geron, 2017. Copyright 2020 by O’Reilly Media, Inc.

4 Experiments

The experiments that I performed involved on the baseline CNN and the ResNet-34 CNN. I made use of a validation set in both experiments to check against overfitting. For the baseline CNN, 20% of the training set was used for validation and 20% of the total dataset was used for testing. For the ResNet-34, 30% of the training set was used for validation and 20% was used for testing. The increase in the validation was performed in an attempt to maximize performance wherever possible.

4.1 Baseline CNN

The layers of the baseline models found on Kaggle consisted of Stacked Convolution 🡪 Max Pooling 🡪 Dropout (0.2) 🡪 Stacked Convolution 🡪 Max Pooling 🡪 Dropout (0.2) 🡪 Flatten (to properly vectorize the inputs to the fully connected neural network) 🡪 Dense (512 neurons) with ReLU activation 🡪 Dropout (0.2) 🡪 Dense (2 neurons) with softmax activation. They were trained using categorical cross entropy which was surprising to me as it is not recommended in the text to use categorical cross entropy for binary classification problems. Although, the softmax activation in the output layer is properly paired with the categorical cross entropy. The learning was optimized with Stochastic Gradient Descent (SGD) over 20 epochs and batches of 32. These previous implementations seemed to work well as I did not observe overfitting. Furthermore, the total training time was less than 20 minutes.

4.2 ResNet-34 CNN

The layers of the ResNet-34 models were structured according to figure 5 as shown above, with the exception of the dense output layer which consisted of a single neuron with sigmoid activation. They were trained using binary cross entropy which is appropriate since this is a binary classification problem. The learning was optimized with Nadam over 30 epochs and batches of 32 (Ross, 2020). For the first 20 epochs there were fluctuations in performance where loss and accuracy were moderately erratic as shown in figure 6. Once training reached the 25th epoch the model demonstrated steady improvements and ultimately reached convergence. However, the total training time was approximately 3 hours.

**Chart, histogram

Description automatically generatedChart, line chart

Description automatically generated**

Figure 6. ResNet-34 Loss and Accuracy plots. From the training results of a ResNet-34 model on the Ship in Satellite Imagery Dataset on Kaggle by J. Ross, 2020. No Copyright

5 Conclusion

5.1 Summary of achievements

While the ResNet-34 model was much slower to train, it achieved accuracy values on the test set ranging between 99.36 and 100% as shown in figure 7. If this model were being reviewed by theatre commanders or decision makers for implementation in an operational environment, then they would likely sacrifice faster training times for better overall performance. Analysts must be able to rely on classifiers to not be the cause of disruptions in the operational environment.

Graphical user interface, text

Description automatically generatedGraphical user interface, text

Description automatically generated

Figure 7. ResNet-34 results on test set. From the evaluation of a ResNet-34 model on the test set of the Ship in Satellite Imagery Dataset on Kaggle by J. Ross, 2020. No Copyright

5.2 Future Directions

While multi-class classifiers are an impressive feat, binary classifiers seem to achieve greater accuracy indicating that more simple models may be better suited for deployment in support of imagery analysis in theatre. Operational environments need reliable and simple methods in lieu of more complex classification models. The same multi-class classification functionality could be achieved by having a hierarchical structure to classification datasets. The first system would begin with classifying ships in the broadest of categories according to whether they are naval combatant, logistic, or civilian vessel. The next layer down would classify according to the class of naval combatant (DDG, CG, etc.). The subsequent layer would classify according to the variant of that class of naval combatant (Flight I DDG, Flight II DDG, etc.).

# References

Vitali Burachonok, Tomas Rasymas. (2017). *Ships in Satellite Imagery*. Retrieved from Kaggle: https://www.kaggle.com/rhammell/ships-in-satellite-imagery/notebooks?datasetId=2869&searchQuery=Keras

Geron A. (2017). *Hands-on machine learning with Scikit-Learn and TensorFlow*: concepts, tools, and techniques to build intelligent systems. Sebastopol, CA: O'Reilly Media. ISBN: 978-1491962299

R Hammel. (2017). *Ships in Satellite Imagery*. Retrieved from Kaggle: https://www.kaggle.com/rhammell/ships-in-satellite-imagery

John Ross. (2020). *ResNet-34 Model on the Ships in Satellite Imagery Dataset*. Retrieved from Kaggle: https://github.com/jtrii0211/Class-Projects/blob/main/ML\_Final\_Project.py