Political Meme Classification Phase 2

Kate Arendes
Introduction to Deep Learning

March 1, 2022

Contents

Overview

This phase focused on experimentation with various convolutional neural network architectures to see how small a model could be while managing to overfit the data. By altering the model's structure and monitoring relevant performance metrics, the composition of the model with the fewest parameters while achieving a close to 1.00 accuracy was determined.

The smallest architecture able to overfit the input with its targets added as another channel was found by similar means and is discussed later in this report.

1 Building an Overfitting Model

In order to build the smallest possible model that overfits the data, it was necessary to examine how adjusting the number of layers, filters, and parameters affected the model's ability to properly classify the samples.

1.1 Layers

Each architecture that was tested was based on alternating Conv2D layers with MaxPooling2D layers to create a convolutional neural network with a single dense layer at the end. While the dense layer was always necessary, as it possessed the sigmoid activation needed for binary classification, the number of Conv2D and MaxPooling2D layers was variable.

The model ultimately performed well with three pairs of Conv2D and MaxPooling2D layers, which allowed it to have enough memory to learn the data while not being too large to immediately overfit. The precision and accuracy using this structure was able to reach the 0.7-0.8 range quickly, then more slowly approach near 1.00 for both metrics. Other structures were either unable to learn the data or overfit rapidly.

1.2 Filters

While varying layer structures could yield similar parameter counts, the number of filters in each layer was critical to maintaining a reasonable number of parameters.

By increasing the number of filters in each layer, the number of trainable parameters generally increased as well. For this data set, a model in which each layer had 8 parameters was barely able to overfit with 1,401 parameters. Reducing the filter counts too much prevented the network from being able to make improvements in accuracy, precision, and loss during training, and increasing the number of filters would make overfitting occur more quickly.

Whether the number of filters in subsequent layers increased, decreased, or stayed constant also impacted the model. By having the number of filters in subsequent layers increase, the model had a more difficult time initially increasing accuracy and precision, and reducing loss.

1.3 Parameters

Because the size of a given model refers to the number of its parameters, and the number of layers and filters impacted this count, the most important element of the model's structure was its trainable parameters.

The ideal number of parameters for any model to overfit this data set was between 1,200 and 1,500, which was achieved through altering the number of layers and filters.

1.4 Selecting the Smallest Overfitting Model

The smallest model that was able to overfit the data was composed of three Conv2D layers and three MaxPooling2D layers alternating between each other. The first two Conv2D layers had 8 filters, while the third had 6. All Conv2D layers had a kernel size of 3, and each MaxPooling2D layer had a pool size of 4, resulting in a total of 1,253 parameters (Figure 1).

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[(None, 150, 150, 3)]	0
rescaling_1 (Rescaling)	(None, 150, 150, 3)	0
conv2d_3 (Conv2D)	(None, 148, 148, 8)	224
max_pooling2d_3 (MaxPooling 2D)	(None, 37, 37, 8)	0
conv2d_4 (Conv2D)	(None, 35, 35, 8)	584
max_pooling2d_4 (MaxPooling 2D)	(None, 8, 8, 8)	0
conv2d_5 (Conv2D)	(None, 6, 6, 6)	438
max_pooling2d_5 (MaxPooling 2D)	(None, 1, 1, 6)	0
flatten_1 (Flatten)	(None, 6)	0
dense_1 (Dense)	(None, 1)	7

Total params: 1,253
Trainable params: 1,253
Non-trainable params: 0

Figure 1: Architecture of the model.

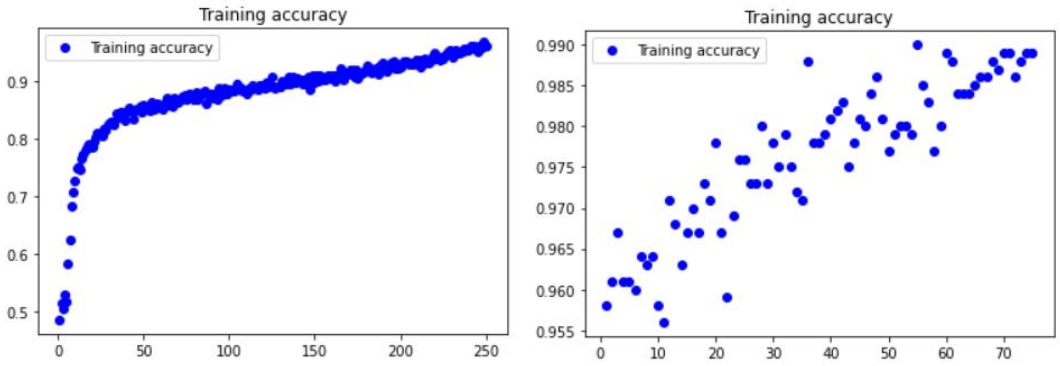

Figure 2: Accuracy of the model for the first 250 and final 75 epochs.

2 Model Performance

Over 325 epochs, the smallest overfitting model was able to achieve the following performance metrics:

2.1 Accuracy

During training, the maximum accuracy reached by the model was 0.9900, which remained above 0.97 for the final 50 epochs (Figure 2). Other models with more parameters, such as the model with 1,401, were able to reach a slightly higher maximum accuracy of 0.9980, and did so in far less than 325 epochs. However, those models were larger, and thus their learning ability was greater than that of the smaller model, which never completely memorized the data.

2.2 Loss

The minimum training loss of the model was 0.0617 (Figure 3). Although other larger models were able to achieve near 0.01 loss by the end of training, the smaller model's loss was continuing to decline when the program ended. If the network had trained for any more epochs, the loss would likely have fallen even further. However, this metric did not make the same progress that accuracy and precision did versus other models after fewer epochs.

2.3 Precision

For all models tested, the rate of improvement, stagnation, or degradation of precision remained similar to accuracy, often trailing shifts in accuracy by a few epochs. The maximum precision for the smallest overfitting model was 0.9822, and remained over 0.95 for the final 50 epochs of training (Figure 4).

2.4 Callbacks

Because the model was not being used with validation and test data, the only callback needed was ModelCheckpoint to ensure that if a model did not overfit after a given number of epochs, the progress of previous training would not be lost. Using this callback allowed the model to be run with for 325 epochs without the values of its weights being lost.

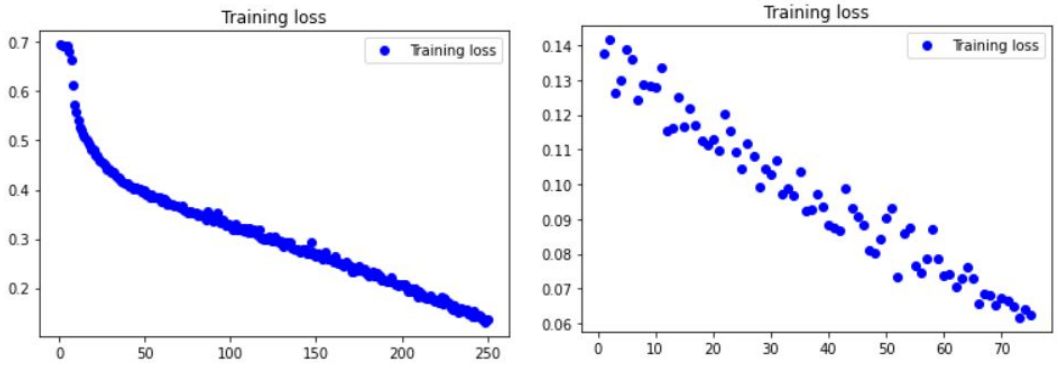

Figure 3: Training loss of the model for the first 250 and final 75 epochs.

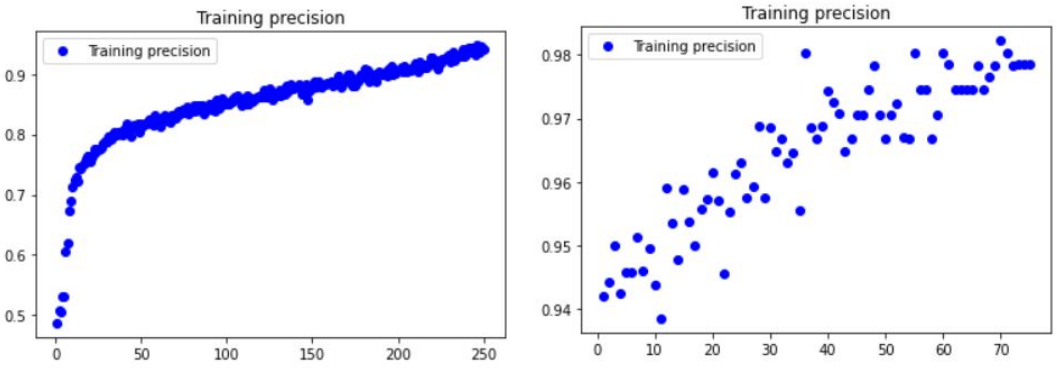

Figure 4: Precision of the model for the first 250 and final 75 epochs.

Layer (type)	Output Shape	Param #
input_50 (InputLayer)	[(None, 150, 150, 4)]	0
rescaling_49 (Rescaling)	(None, 150, 150, 4)	0
max_pooling2d_94 (MaxPooling2D)	(None, 37, 37, 4)	0
max_pooling2d_95 (MaxPooling2D)	(None, 9, 9, 4)	0
conv2d_16 (Conv2D)	(None, 7, 7, 1)	37
conv2d_17 (Conv2D)	(None, 5, 5, 1)	10
max_pooling2d_96 (MaxPooling2D)	(None, 1, 1, 1)	0
flatten_37 (Flatten)	(None, 1)	0
dense_48 (Dense)	(None, 1)	2

Total params: 49
Trainable params: 49
Non-trainable params: 0

Figure 5: Architecture of the smallest convolutional model.

3 Overfitting Model with Output as Input

While a single neuron would likely successfully classify the data based on the encoded layer, I opted instead to experiment with Conv2D layers and MaxPooling2D layers to construct the smallest possible convolutional neural network to overfit the data.

To test these architectures, each image was given an additional channel consisting entirely of 0s or 1s depending on their respective labels. The samples, along with their targets, were then fed into each of the following networks.

3.1 Smallest Model with Conv2D Layers

Using a two Conv2D layers, each with one neuron, and three MaxPooling2D layers to downsample the data, I was able to construct a convolutional network with 49 parameters (Figure 5).

After only 6 epochs, the model was able to achieve an accuracy and precision of 1.00. The training loss after 30 epochs reached 0.0101 (Figure 6).

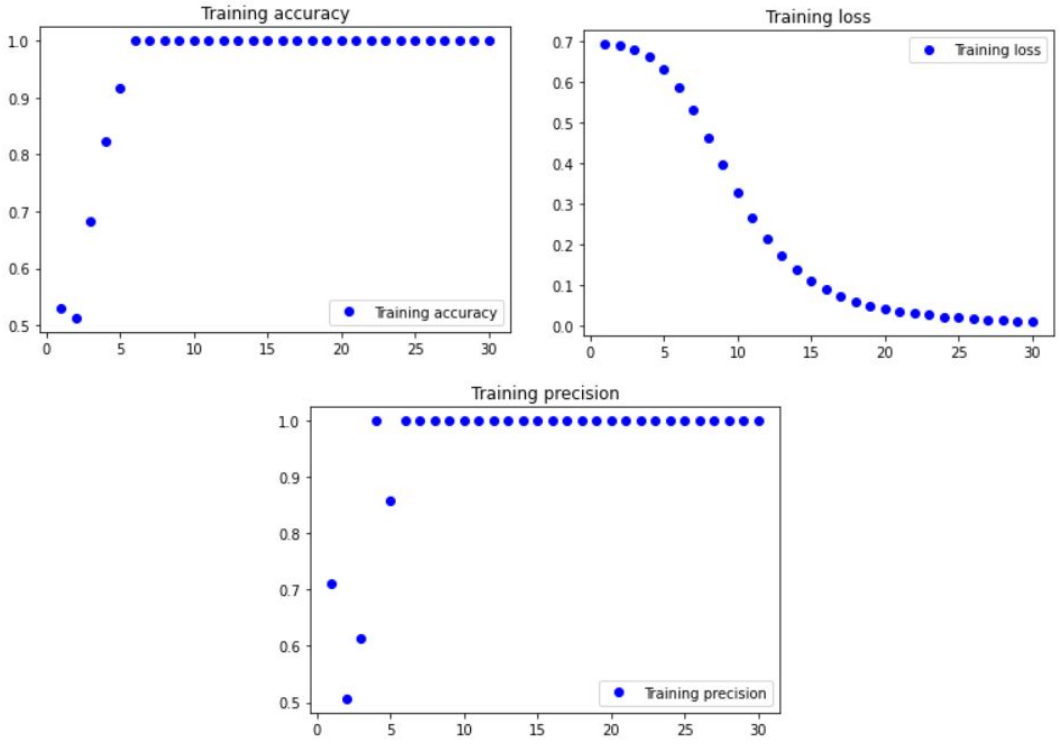

Figure 6: Performance metrics of the convolutional model.

3.2 Smallest Model Using Single MaxPooling2D Layer

Along with constructing a true convolutional network, I wanted to utilize downsampling to see how few parameters I could achieve with a model using a single MaxPooling2D layer. With a pool size of 150, I was able to reduce the number of parameters to 5, and test the architecture to check whether it was still capable of overfitting the data (Figure 7).

As with the model using Conv2D layers, this model quickly overfit the data as well, though improvements in accuracy and precision were widely varied before reaching 1.00. However, the loss of the model remained extremely high compared to the model using Conv2D layers, and never reached below 0.68 (Figure 8).

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_51 (InputLayer)       [(None, 150, 150, 4)]     0

 rescaling_50 (Rescaling)    (None, 150, 150, 4)       0

 max_pooling2d_97 (MaxPoolin  (None, 1, 1, 4)          0
 g2D)

 flatten_38 (Flatten)        (None, 4)                 0

 dense_49 (Dense)            (None, 1)                 5

=================================================================
Total params: 5
Trainable params: 5
Non-trainable params: 0
_________________________________________________________________
```

Figure 7: Architecture of the max pooling model.

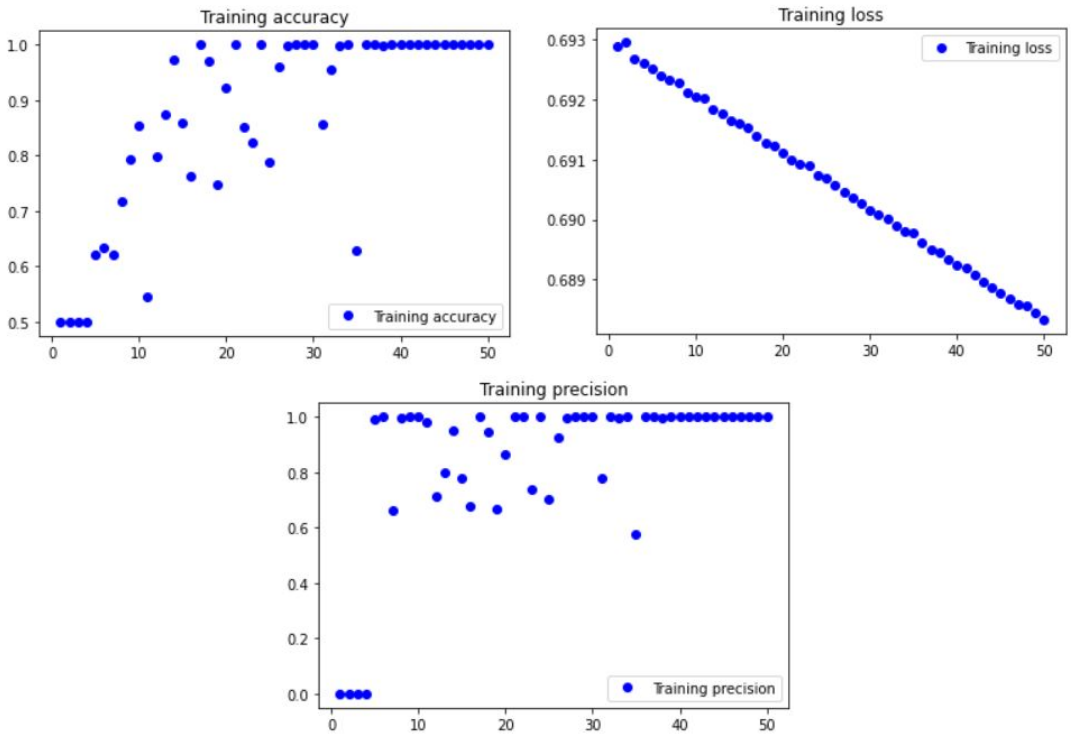

Figure 8: Performance metrics of the max pooling model.