Political Meme Classification Phase 3

Kate Arendes
Introduction to Deep Learning

March 7, 2022

Contents

Overview

For this phase of constructing the political meme classifier, the data was split into three sets to distinguish the model's performance during training, validation, and testing. Utilizing various validation metrics, the best performing model in regards to both validation and testing was selected and further analyzed. This report describes the process of data preparation, model experimentation, and performance analysis necessary to make possible the progress seen during this phase.

1 Data Preparation for Evaluation

In order to measure the model's generalization power, there had to exist a division between the process of teaching the model to make classifications and that of examining its interactions with data it had not yet encountered. While the amount of data collected for this specific task was relatively small, a method of simple holdout validation was still sufficient to see improvements in test performance substantially above the baseline accuracy.

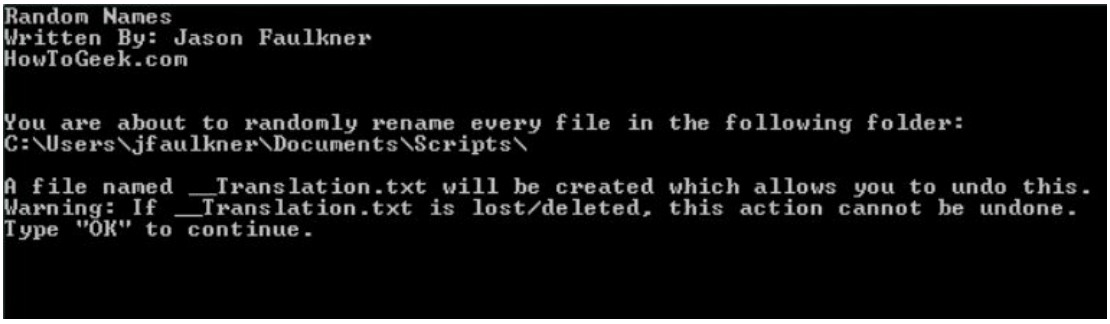

Figure 1: Interface of the name randomization script used in each directory.

1.1 Randomization

Before splitting the data into training, validation, and test sets, all images belonging to each class were placed in folders representing the two labels "conservative" and "liberal". To shuffle the images, a script that randomly named each sample was run in both folders, and the results were organized alphabetically (Figure 1).[1] Because the political memes in the data alluded to current events, this shuffling ensured that images featuring certain topics were well distributed throughout the data.

1.2 Division

Once the images were randomized, 600 images belonging to both classes were placed in a training folder, 200 images were placed in a validation folder, and the remaining 200 images were placed in a test folder.

To prevent target leaking, images used for testing were never included in the training or validation generators. Once the data set was divided, images in various folders were kept separate from one another for every experiment conducted during this phase.

1.3 Image Generators

Samples were fed into the network by using the Keras ImageDataGenerator object, three of which were instantiated for the training, validation, and test sets. With each model that was created, the first step in experimentation was to load the data and display images to ensure that the generators were functioning correctly (Figure 2).

2 Experimentation

After testing the image generators to check that samples had the proper resolution and labeling, model construction and training could commence in order to find the network with the best possible performance in validation and testing.

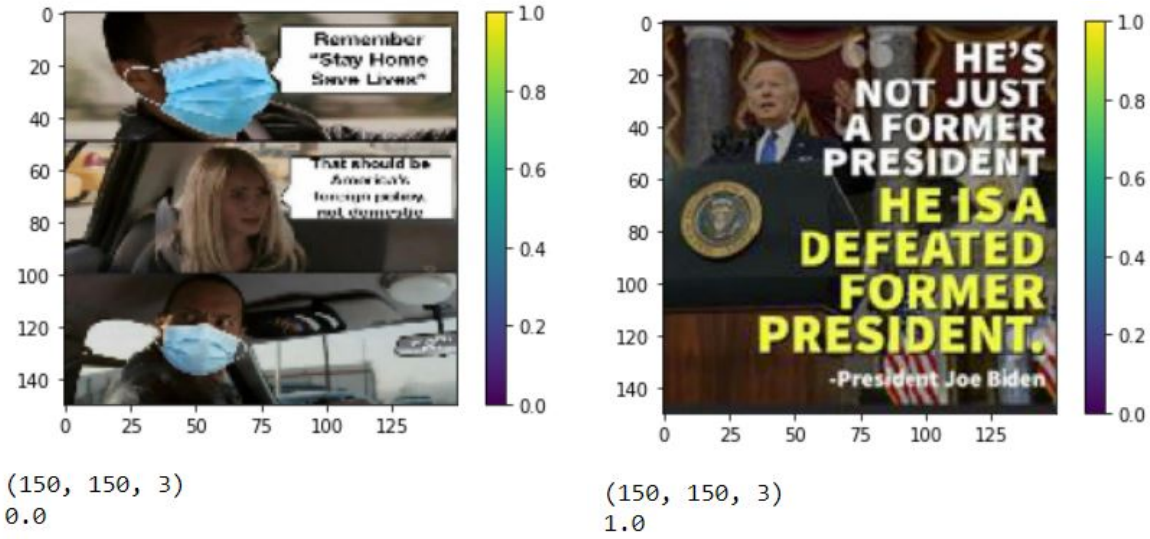

Figure 2: Images displayed from testing each generator.

```
cb_check = tensorflow.keras.callbacks.ModelCheckpoint(
    filepath="checkpoint_filepath",
    save_best_only=True,
    monitor="val_loss")

cb_early = tensorflow.keras.callbacks.EarlyStopping(
    monitor="val_loss",
    patience=30
)
```

Figure 3: Callbacks used with model.fit().

2.1 Callbacks

The callbacks used during experimentation were EarlyStopping and ModelCheckpoint. Since ModelCheckpoint was used to save the weights from the epoch that had the best validation loss score, the EarlyStopping callback was given a very liberal patience value to ensure that the model had performed as well as possible before overfitting occurred (Figure 3). In earlier experiments where the patience was set too low, it was unclear whether the lowest validation loss had been attained before training was terminated.

2.2 Model Architecture

Because the goal of these experiments was to focus on validation and evaluation metrics, and the model had less data on which to train due to the division of the samples into multiple sets, the architecture for the model had to be altered so that test performance was prioritized, rather than its ability to overfit.

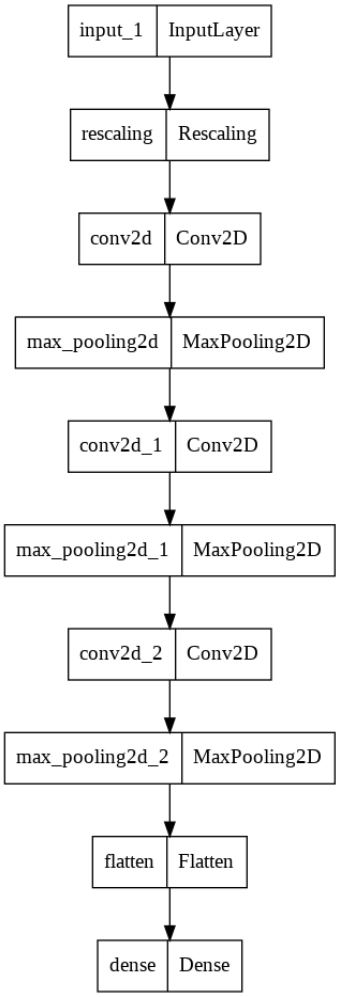

Figure 4: Visualization of the most effective layer composition, used by most of the experimental models.

As in the case of constructing a model to overfit the data, the best layer composition proved to be three pairs of Convolutional2D and MaxPooling2D layers before a final dense layer with sigmoid activation (Figure 4). Regardless of the number of parameters of each experimental model, performance was always strongest when there was a similar number of filters in each layer.

Despite having less data on which to train than the models built for the purpose of overfitting, the ideal parameter count for models in this phase remained in the 1,100 to 1,450 range (Figures 5 and 6). Though not the best-performing model overall, a network with 1,401 parameters nearly matched a 1,191-parameter model in validation statistics, likely due to a similar filter count and distribution throughout the layers.

2.3 Batch Size

Altering the batch size used during training had a small impact on the test loss, precision, and accuracy, and allowed for much longer training sessions before the EarlyStop callback occurred.

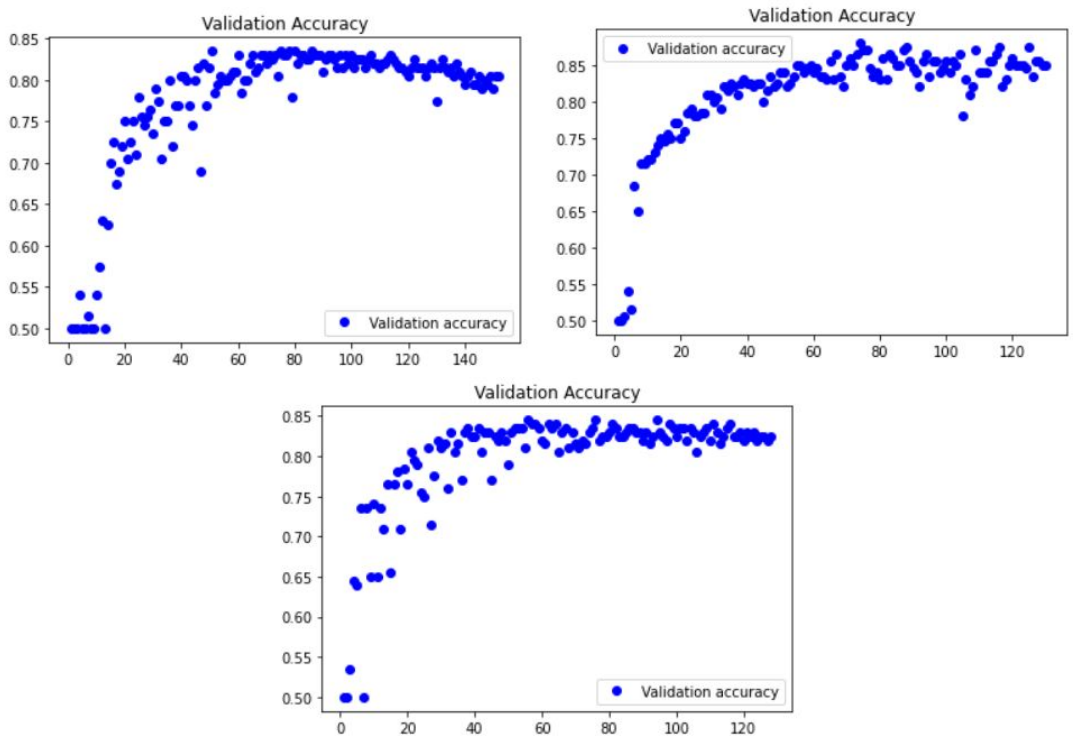

Figure 5: Clockwise starting with top left image: Validation accuracy of 1,035-parameter, 1,401-parameter, and 1453-parameter models, respectively.

In the model with 1,191 parameters, reducing the batch size of training from 10 to 8 samples allowed training to continue an additional 28 epochs before the model achieved its lowest validation loss, and the validation loss for the reduced batch size was 0.0132 points lower than that of the model's previous iteration. The curves of the model before and after this change reflect how a smaller batch size maintained consistent performance metrics as the training loss outpaced the validation loss, which subsequently deteriorated (Figure 7).

However, this alteration did not result in any notable improvements during evaluation, and reducing the batch size further negatively impacted the model's performance.

3 Best Performing Model

After experimenting with layer composition, parameter count, and batch size, the model that performed best in validation and evaluation was determined to be a model with 1,191 parameters, consisting of one Convolutional2D layer of 8 filters and two Convolutional2D layers of 7 filters, all with a kernel size of 3, alternating with MaxPooling2D layers of pool size 4 (Figure 8).

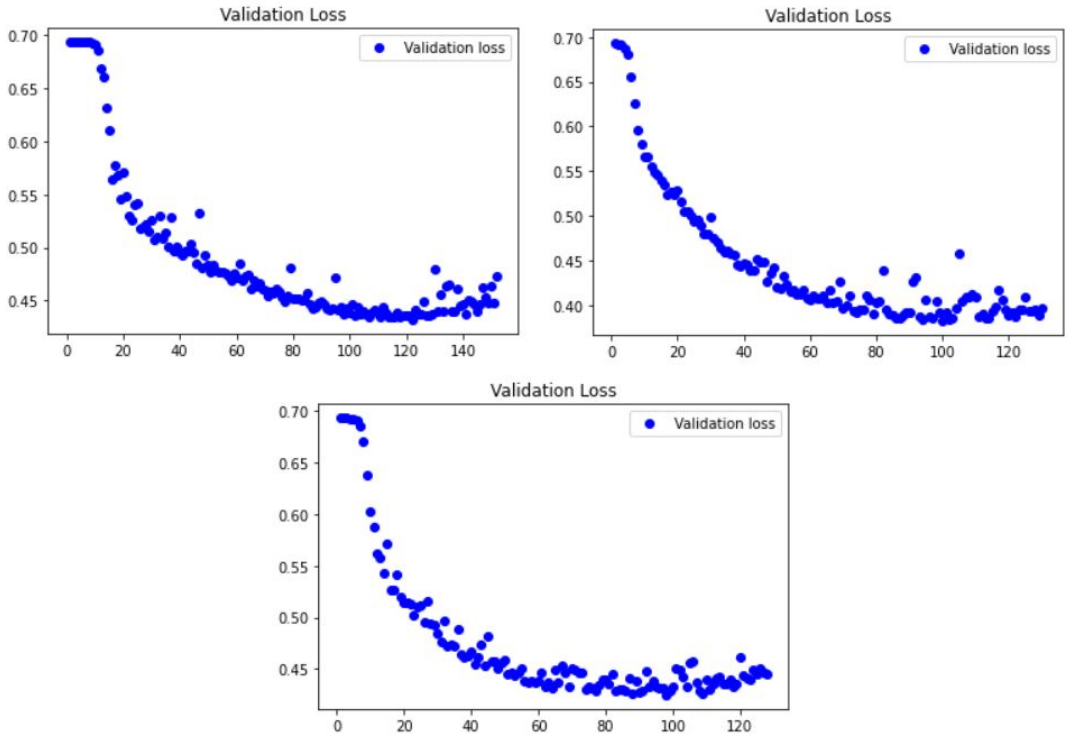

Figure 6: Clockwise starting with top left image: Validation loss of 1,035-parameter, 1,401-parameter, and 1453-parameter models, respectively.

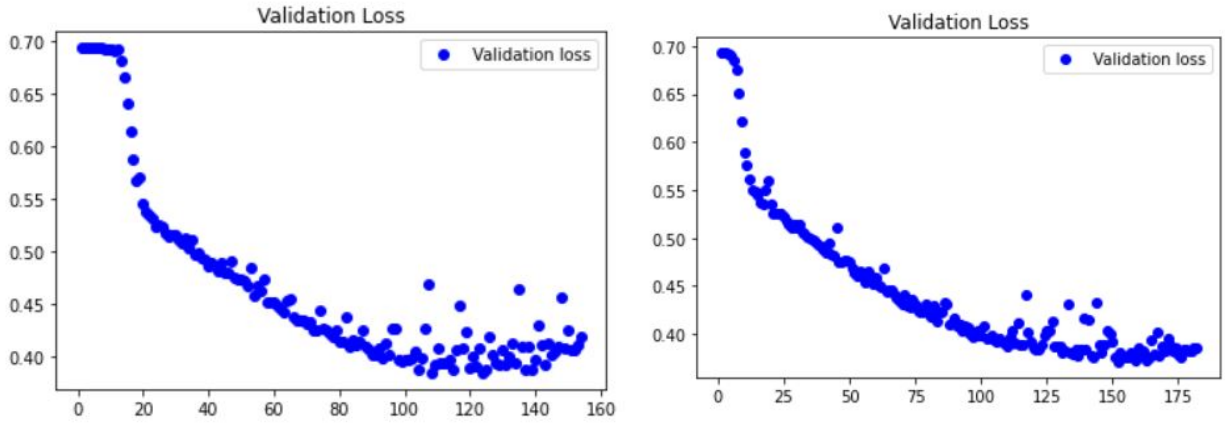

Figure 7: Before and after curves demonstrating how a change in the batch size of the same model resulted in a more gradual and controlled deterioration of the validation loss over more epochs.

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, 7)	511
max_pooling2d_4 (MaxPooling 2D)	(None, 8, 8, 7)	0
conv2d_5 (Conv2D)	(None, 6, 6, 7)	448
max_pooling2d_5 (MaxPooling 2D)	(None, 1, 1, 7)	0
flatten_1 (Flatten)	(None, 7)	0
dense_1 (Dense)	(None, 1)	8

Total params: 1,191
Trainable params: 1,191
Non-trainable params: 0

Figure 8: Architecture of best-performing model found from experimentation.

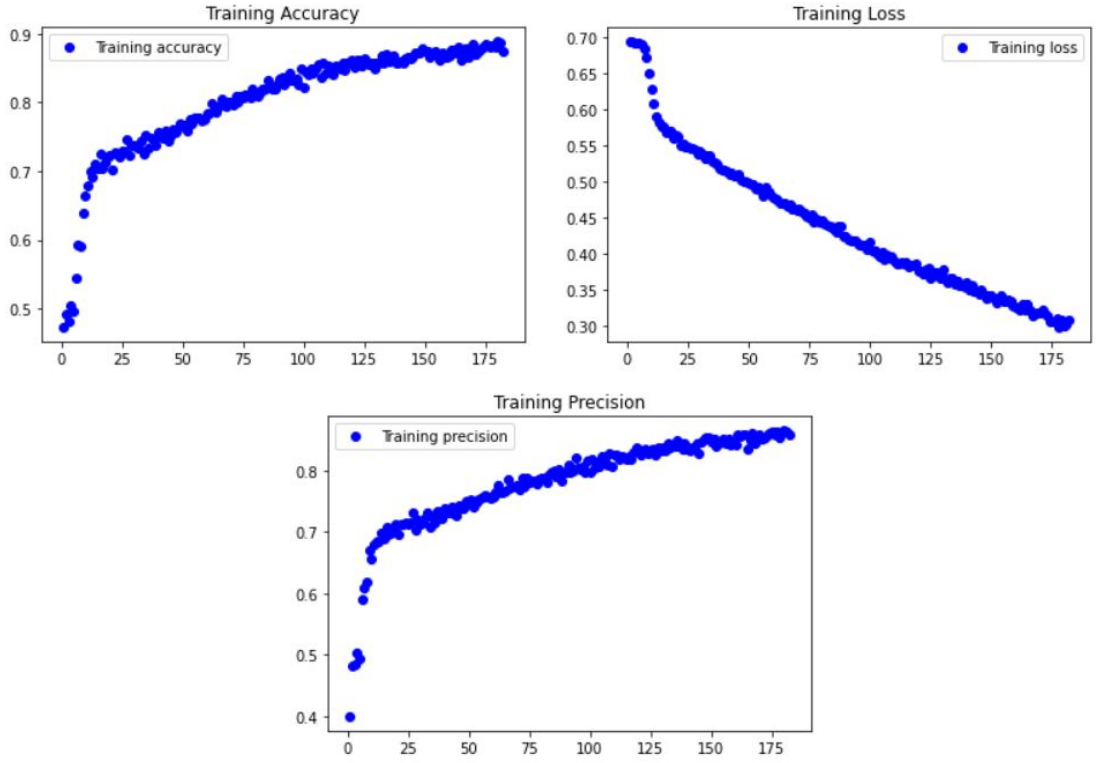

Figure 9: Training accuracy, loss, and precision of the best performing model showing a slow climb towards potentially overfitting the data.

3.1 Training and Validation

The training batch size of the best performing model was 8 samples, and the network was able to train for 182 epochs before the EarlyStopping callback ended the session. The lowest training loss was 0.2988, the highest training accuracy was 0.8883, and the highest training precision was 0.8652. Although these metrics were continuing to improve when training completed, their advancement had substantially stalled and it was unclear whether the model would have been able to eventually reach a near perfect training accuracy (Figure 9).

The lowest validation loss was 0.3709, achieved after epoch 152 of training. The highest validation accuracy, 0.8650, was reached at epoch 110, but an accuracy of 0.8600 was attained several times after that point. At epoch 157, the highest validation precision of 0.8600 was seen, and marked a point at which the model was performing at its best with all of its metrics taken into account. Precision reached its peak range more rapidly compared to other metrics, which showed more gradual curves. (Figure 10).

3.2 Evaluation

Evaluation was performed with the model saved by the ModelCheckpoint callback. During evaluation on the test set, the final model achieved a loss score of 0.4675, an accuracy of 0.8300, and a precision of 0.7895 (Figure 11). While the accuracy and precision of the model strongly demonstrate its ability to beat the baseline expectations for a binary classifier, the loss score was

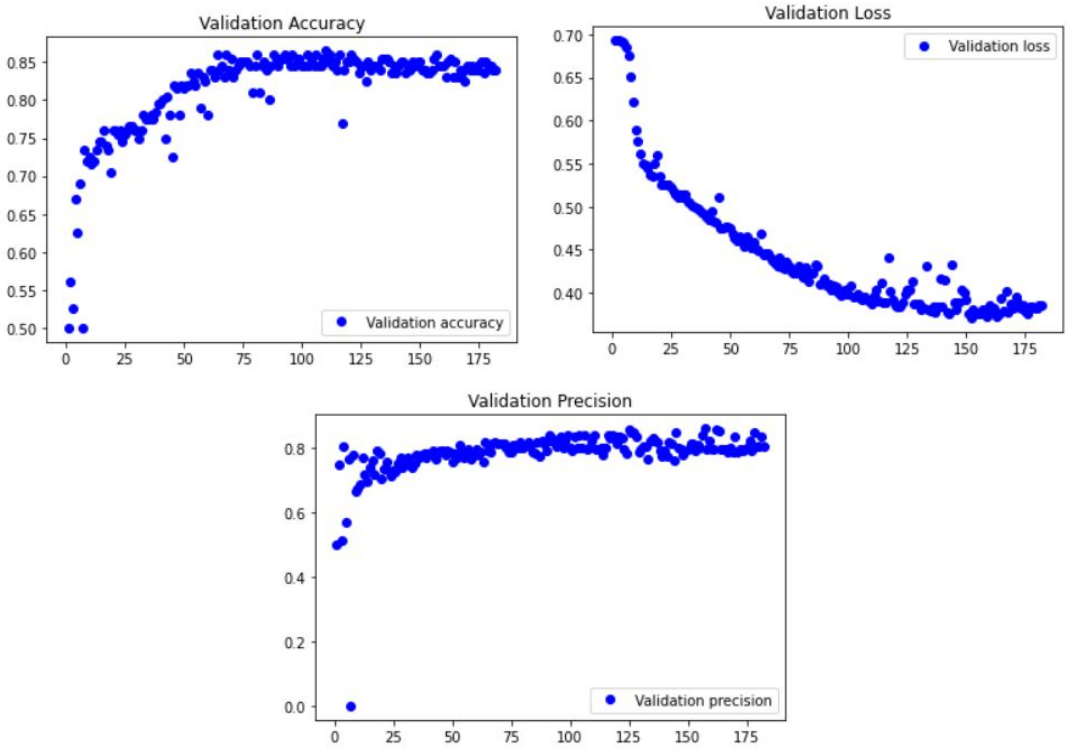

Figure 10: Validation accuracy, loss, and precision of the best performing model.

```
loss: 0.4675 - accuracy: 0.8300 - precision_1: 0.7895
```

Figure 11: Results of evaluating the model using the test data.

much higher than that seen during validation, and there will be ample room for improvement with these metrics with the introduction of data augmentation and normalization in future experiments.

References

[1] Jason Faulkner. "Stupid Geek Tricks: Randomly Rename Every File in a Directory". `https://www.howtogeek.com/57661/stupid-geek-tricks-randomly-rename-every-file-in-a-directory/`, 2016. [Online; accessed 3-March-2022].