Political Meme Classification

Kate Arendes
Introduction to Deep Learning

May 4, 2022

Abstract

A ”meme” is defined by Merriam-Webster Dictionary as ”an amusing or interesting picture, video, etc., that is spread widely through the Internet.”[1] Many such memes feature political figures and themes, and are capable of synthesizing opinions on current events and policy into easily shareable cultural artifacts. The aim of this project is to test a neural network’s ability to learn to recognize memes belonging to the two most prominent viewpoints in United States politics: conservative and liberal. Because it is sometimes difficult for humans to understand the meaning of a meme and classify its political affiliation, it will be fascinating to consider the patterns on which the model relies to make these decisions and see how successful it becomes at executing this task.

Applications of this model can help to better understand how political polarization manifests on the internet, such as through being able to immediately recognize how images re-posted by users of social media sites are skewed towards one of the two ideologies represented in the data. If the model is able to make clear distinctions between the classes, such an outcome may also reflect a wide chasm between the opinions of the two political groups, especially if a very low loss and high precision are attained. To build a model capable of making such classifications, the data was first processed and an initial model was built to overfit, or memorize, all of the images and their classes. This model’s performance in terms of accuracy, loss, and precision was then improved using data augmentation and normalization techniques, and finally entirely rebuilt with different architectures and convolutional bases. The results of these phases of experimentation are described in this report.

1 Data Collection and Normalization

To collect the images for this project, I searched through conservative and liberal meme sites on Reddit, Facebook, and Pinterest. I assembled a collection of 1000 images, evenly distributed between the two classes, shown in Figure 1. These images were then placed in folders so that they could later be loaded into generators. Figure 2 displays one image for each label.
The normalization of the images involved rescaling all images so that their resolutions were equivalent by multiplying the input data by 1/255.

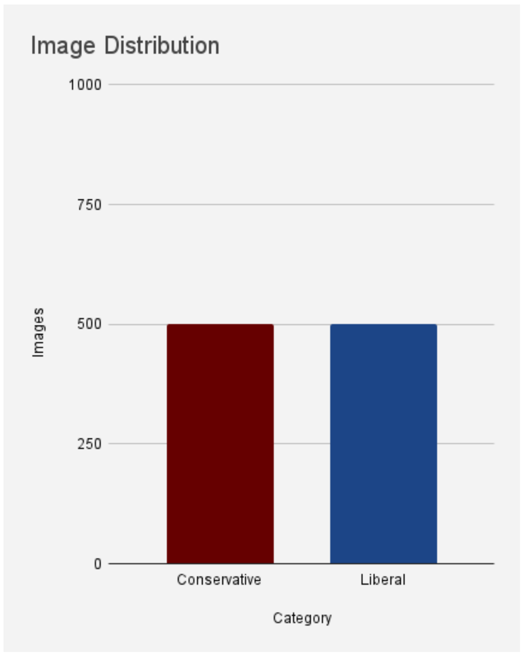


Figure 1: Even distribution of images in the ”Conservative” and ”Liberal” classes.

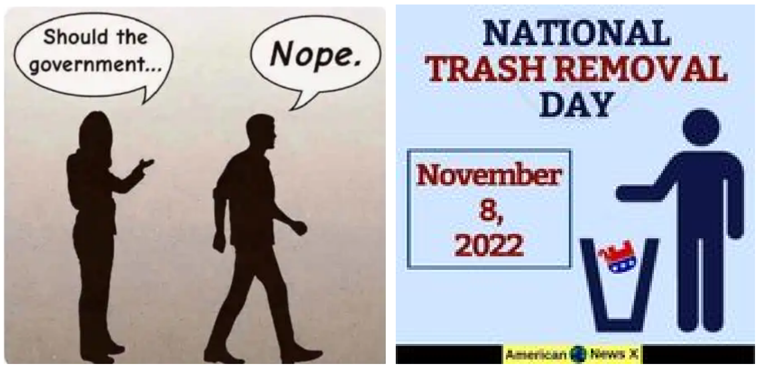
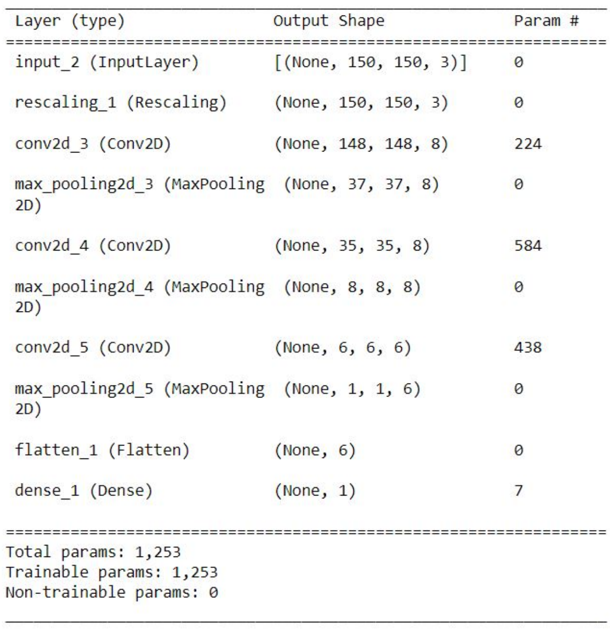


Figure 2: Sample of a conservative meme and liberal meme, respectively.



```
_________________________________________________________________
 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  (None, 37, 37, 8)        0
 2D)

 conv2d_4 (Conv2D)           (None, 35, 35, 8)         584

 max_pooling2d_4 (MaxPooling  (None, 8, 8, 8)          0
 2D)

 conv2d_5 (Conv2D)           (None, 6, 6, 6)           438

 max_pooling2d_5 (MaxPooling  (None, 1, 1, 6)          0
 2D)

 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 3: Architecture of the smallest overfitting model.

2 Overfitting the Data

After testing several candidates, the smallest model that was able to train 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 3). During training, the model reached a maximum accuracy of 0.9900, which remained above 0.97 for the final 50 epochs (Figure 4). The minimum training loss of the model was 0.0617 (Figure 5), and the maximum precision was 0.9822, which remained over 0.95 for the final 50 epochs of training.

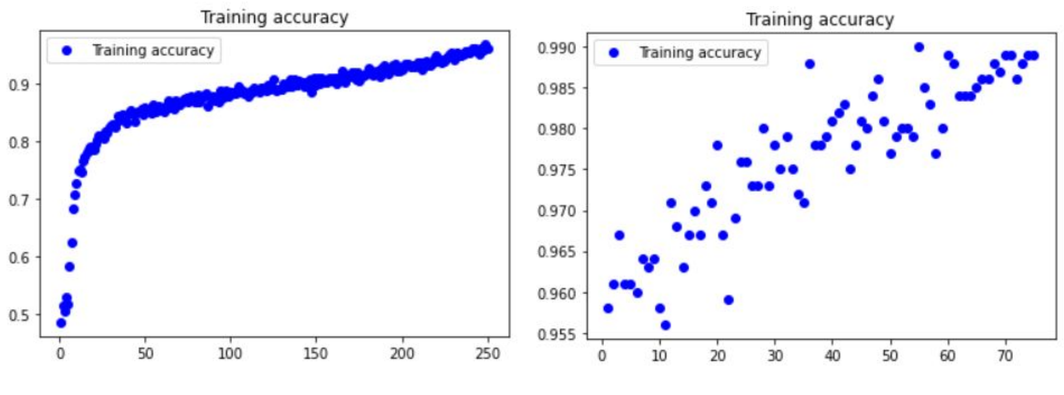


Figure 4: Accuracy of the overfitting model for the first 250 and final 75 epochs.

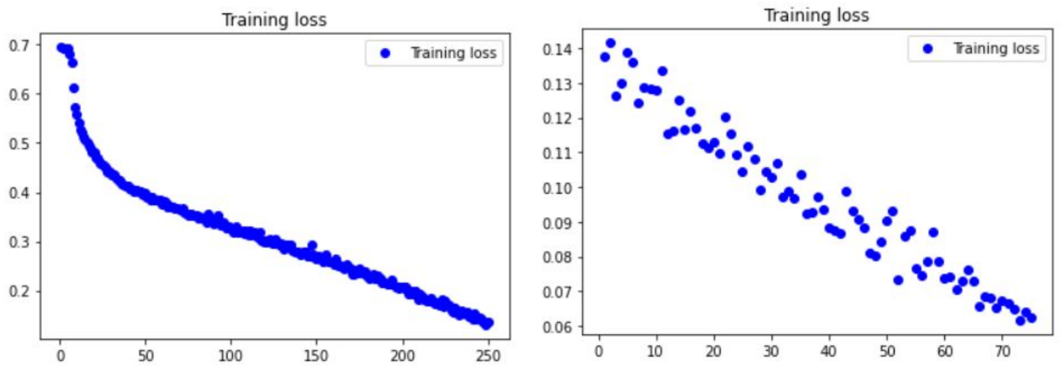


Figure 5: Training loss of the overfitting model for the first 250 and final 75 epochs.

3 Dividing the Data for Training, Validation, and Testing

For the next 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. By making this division, the effectiveness of the model could be more accurately evaluated.

3.1 Randomization and Division

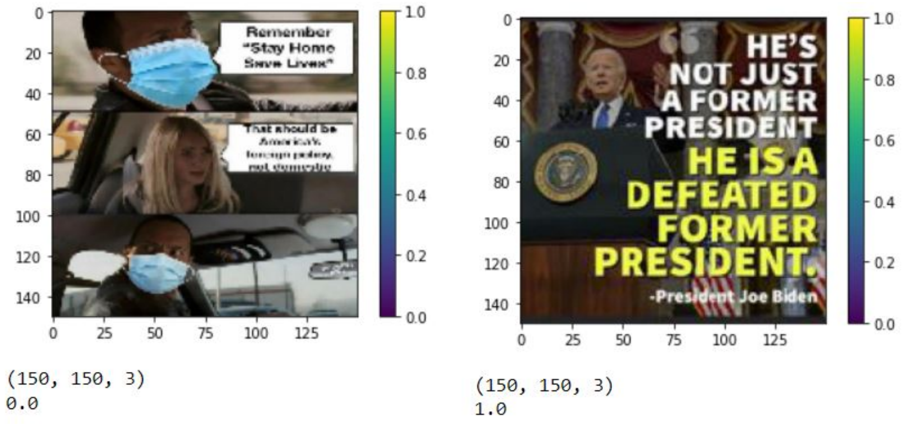
All images belonging to each class were placed in folders representing the two labels "conservative" and "liberal". To shuffle the images, a script that randomly named each sample was run in both folders, and the results were organized alphabetically.[2] Once the images were randomized, 600 images belonging 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.

3.2 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. The first step in experimentation was to load the data and display images to ensure that the generators were functioning correctly (Figure 6).

3.3 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.


Remember "Stay Home Save Lives"
That should be America's foreign policy, not domestic
HE'S NOT JUST A FORMER PRESIDENT HE IS A DEFEATED FORMER PRESIDENT.
-President Joe Biden

```
(150, 150, 3)
0.0
```

```
(150, 150, 3)
1.0
```

Figure 6: Images displayed from testing each generator.

3.4 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. In a 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.

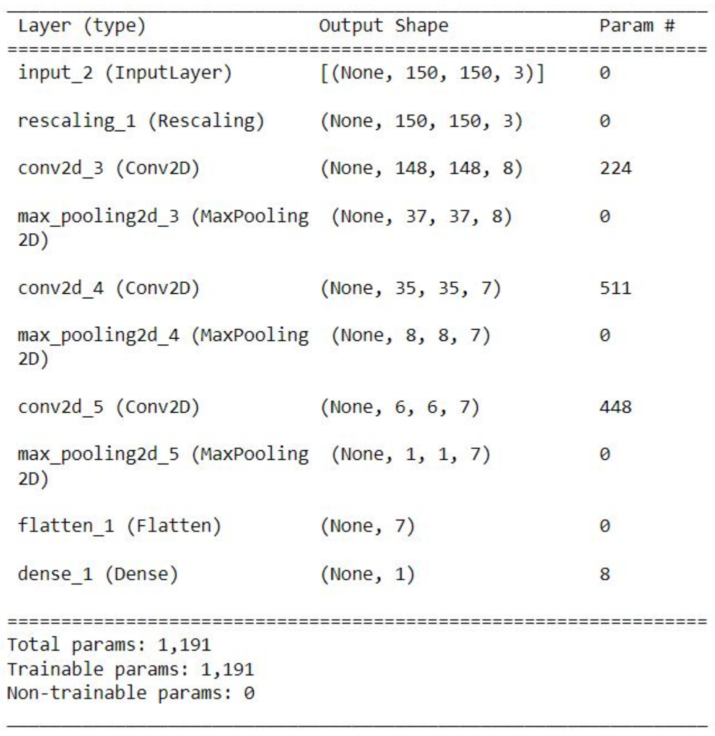
3.5 Best-Performing Evaluation Model

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 7).

The model's lowest training loss was 0.2988, the highest training accuracy was 0.8883, and the highest training precision was 0.8652. The lowest validation loss was 0.3709, highest validation accuracy was 0.8650, and highest validation precision was 0.8600. Precision reached its peak range more rapidly compared to other metrics, which showed more gradual curves. (Figure 8). 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 9). While the accuracy and precision of the model strongly demonstrated its ability to beat the baseline expectations for a binary classifier, the loss score was much higher than that seen during validation.

4 Adding Data Augmentation

To augment the data, input images were altered via three forms of random transformation: rotation, zooming, and flipping (Figure 10). While these alterations theoretically should have



```
_________________________________________________________________
 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  (None, 37, 37, 8)        0
 2D)

 conv2d_4 (Conv2D)           (None, 35, 35, 7)         511

 max_pooling2d_4 (MaxPooling  (None, 8, 8, 7)          0
 2D)

 conv2d_5 (Conv2D)           (None, 6, 6, 7)           448

 max_pooling2d_5 (MaxPooling  (None, 1, 1, 7)          0
 2D)

 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 7: Architecture of best-performing model found from experimentation after splitting the data.

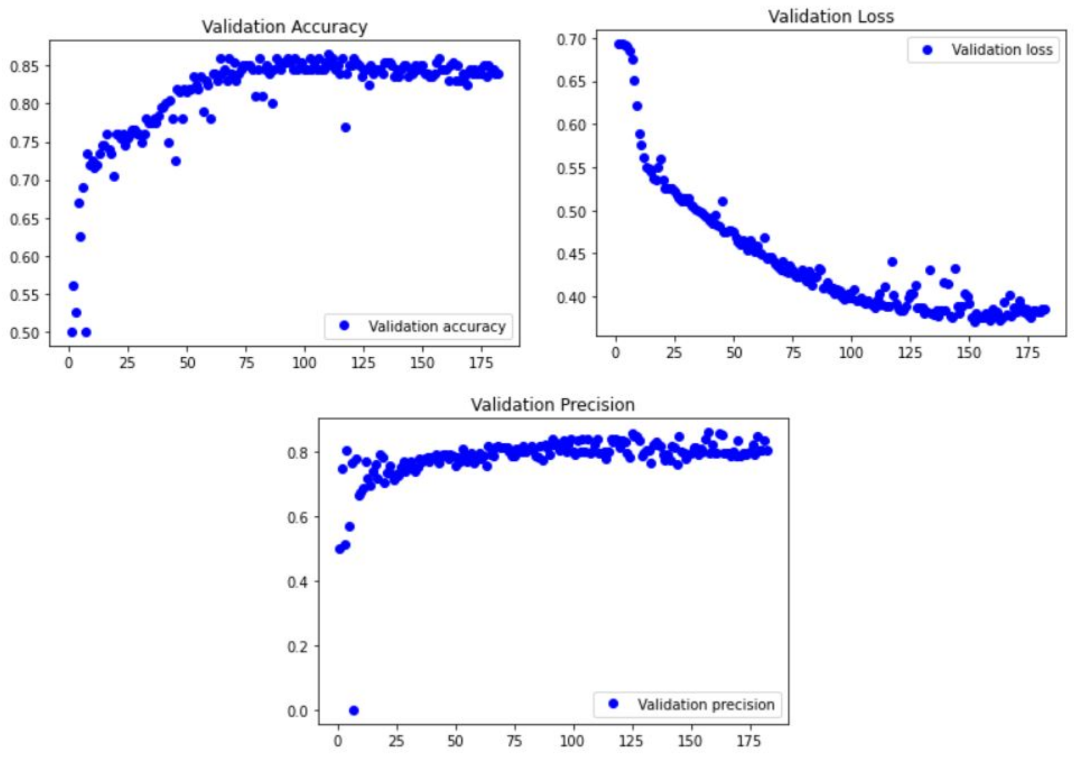
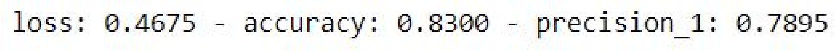


Figure 8: Validation accuracy, loss, and precision of the best performing model after splitting the data.



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

Figure 9: Results of evaluating the model using the test data.

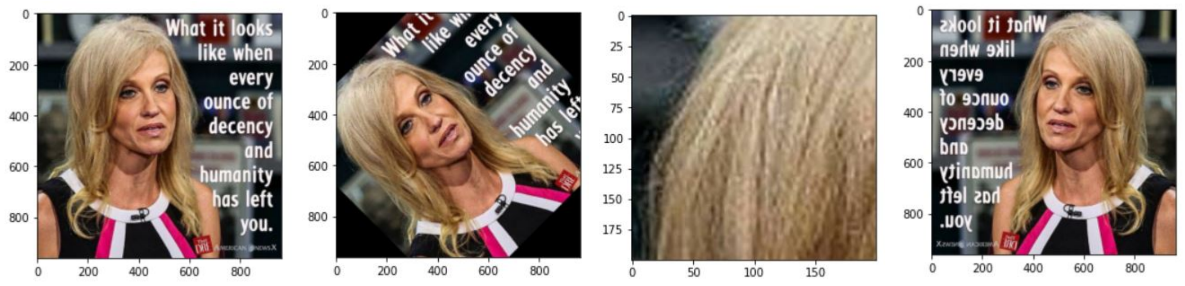

What it looks like when every ounce of decency and humanity has left you.

Figure 10: A randomly selected liberal meme and its appearance after rotation, zooming, and horizontal flipping.

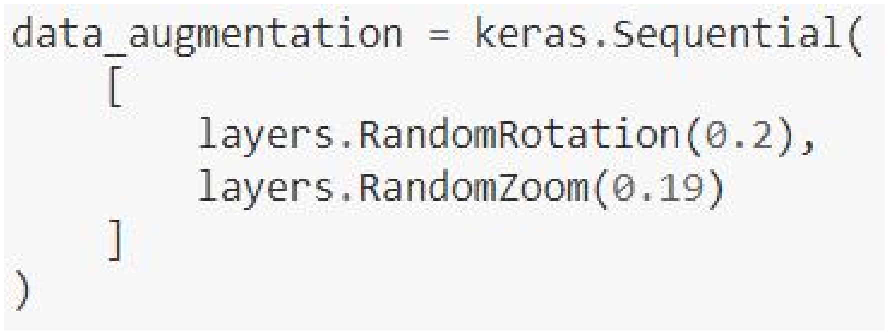
helped the model, especially since there was a relatively small set of training images available, the improvements resulting from the experimentation in this phase were modest.

On its own, the introduction of random zooming to the input data greatly hindered the model's performance. While all configurations were ineffective, the least harmful involved randomly zooming with a factor of 0.19. Similarly, the inclusion of flipping on its own, horizontally and/or vertically, hindered the model. The most effective factor by which input data was randomly rotated was 0.2.

Combining the best-performing factors for rotation (0.2) and zooming (0.19), a performance better than that of the model in the previous phase was attained (Figure 11). Compared to the model from the previous phase, the best-performing model employing data augmentation reached its greatest metrics approximately thirty epochs earlier, and thus ended training far earlier. The minimum validation loss, which was 0.3596, occurred at epoch 104, the maximum validation accuracy of 0.88 was seen at epoch 119, and the maximum validation precision hovered just under 0.85 from epoch 95 onward until maxing out at 0.8586 at epoch 118 (Figure 12). These validation metrics reflect that data augmentation made the model generally more accurate and precise throughout the validation process.

5 Normalization

Through the use of normalization techniques including dropout, batch normalization, and L1/L2 regularization, the model with data augmentation was further improved. After testing various configurations, the results of which are seen in Figures 13, 14, 15, it was clear that the best normalization implementation was using L1 and L2 regularization with amounts set to 0.0015 and one batch normalization layer. This model achieved a training accuracy of 0.86, loss of 0.356, and precision of 0.8454. The highest validation metrics achieved were an accuracy of 0.885 (notably higher than that of training), a loss of 0.3695, and a precision of 0.8348 (Figure 16).



```
data_augmentation = keras.Sequential(
    [
        layers.RandomRotation(0.2),
        layers.RandomZoom(0.19)
    ]
)
```

Figure 11: The most effective data augmentation configuration found after experimentation.

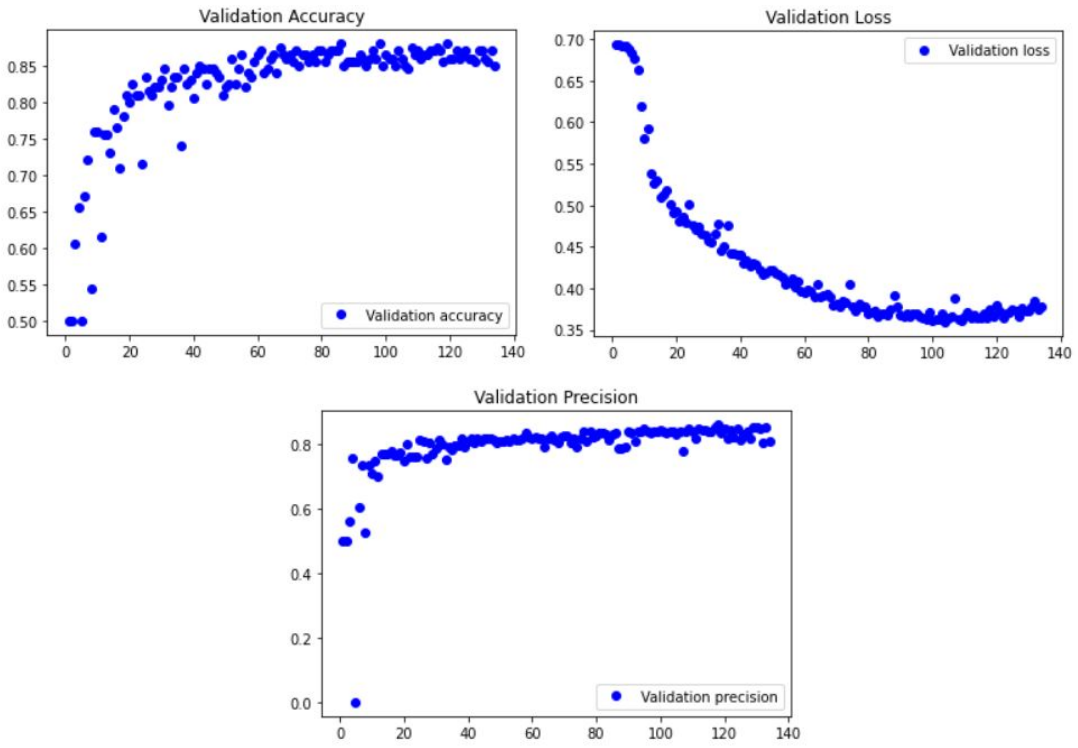
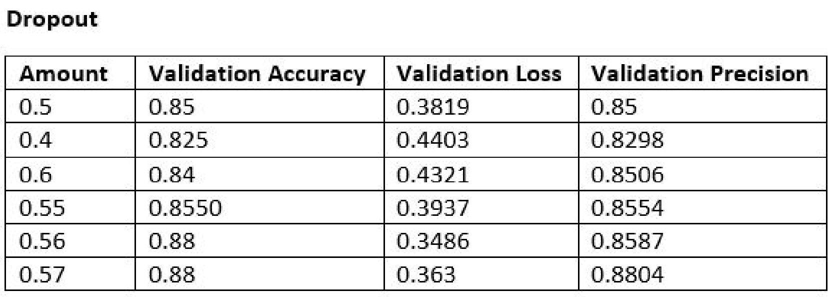


Figure 12: Validation accuracy, loss, and precision of the best-performing model using data augmentation.



Dropout

Amount	Validation Accuracy	Validation Loss	Validation Precision
0.5	0.85	0.3819	0.85
0.4	0.825	0.4403	0.8298
0.6	0.84	0.4321	0.8506
0.55	0.8550	0.3937	0.8554
0.56	0.88	0.3486	0.8587
0.57	0.88	0.363	0.8804

Figure 13: Validation results for various dropout configurations.

Batch Normalization

Placement	Validation Accuracy	Validation Loss	Validation Precision
After 3rd MaxPooling2D	0.78	0.4185	0.902
After 2nd MaxPooling2D	0.835	0.3651	0.8776
After 1st MaxPooling2D	0.855	0.3627	1

Figure 14: Validation results for various batch normalization configurations.

L1 and L2 Regularization

Amount	Validation Accuracy	Validation Loss	Validation Precision
L1=0.0005, L2=0.0005	0.85	0.431	0.8824
L1=0.0011, L2=0.0011	0.86	0.4108	0.8431
L1=0.0015, L2=0.0015	0.885	0.3695	0.8348
L1=0.0016, L2=0.0016	0.875	0.442	0.9474
L1=0.0014, L2=0.0014	0.845	0.4437	0.8367

Figure 15: Validation results for various L1 and L2 regularization configurations.

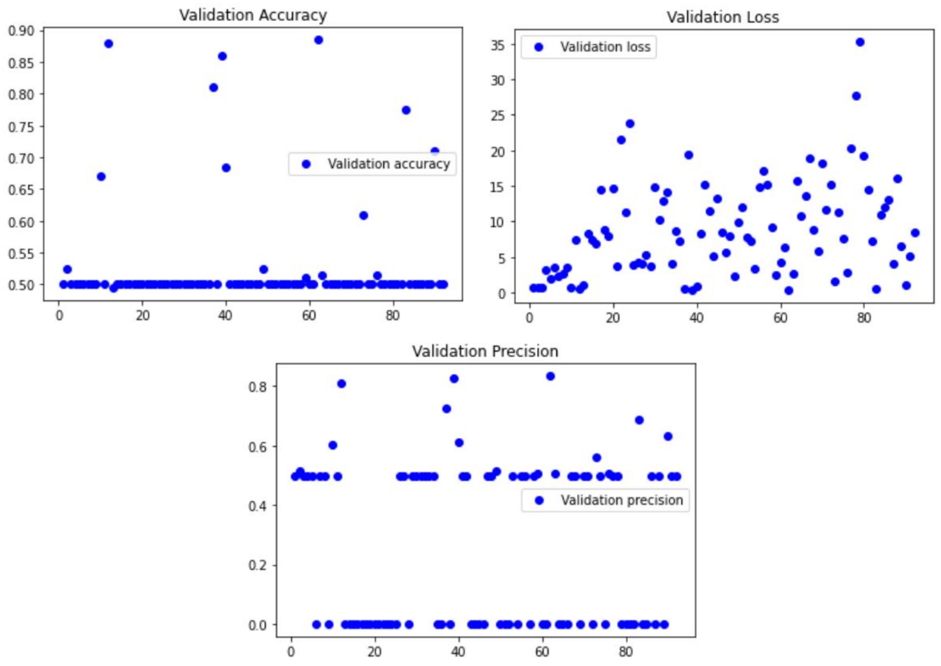
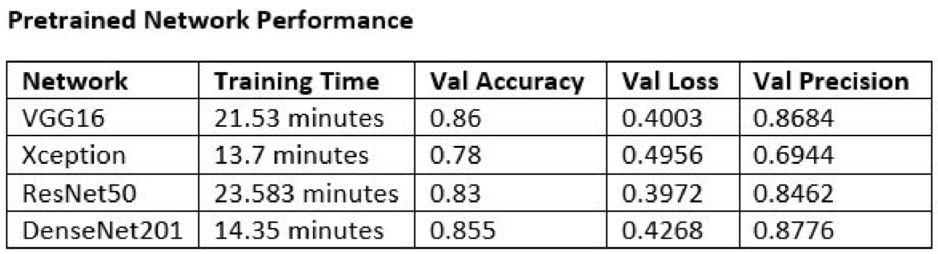


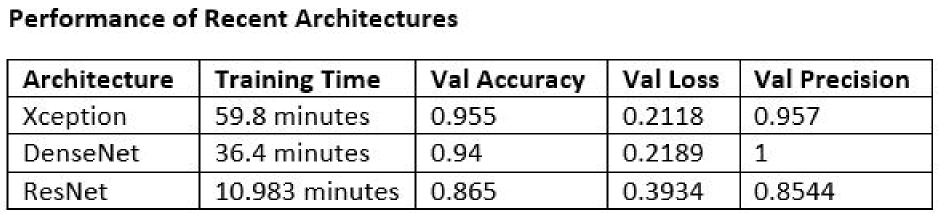
Figure 16: Validation accuracy, loss, and precision of the best-performing model using normalization.



Pretrained Network Performance

Network	Training Time	Val Accuracy	Val Loss	Val Precision
VGG16	21.53 minutes	0.86	0.4003	0.8684
Xception	13.7 minutes	0.78	0.4956	0.6944
ResNet50	23.583 minutes	0.83	0.3972	0.8462
DenseNet201	14.35 minutes	0.855	0.4268	0.8776

Figure 17: Training times and validation metrics for each pre-trained network incorporated into the model.



Performance of Recent Architectures

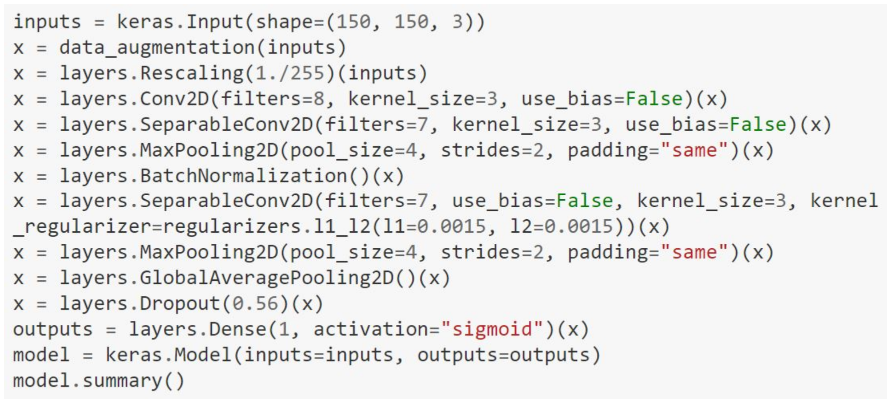
Architecture	Training Time	Val Accuracy	Val Loss	Val Precision
Xception	59.8 minutes	0.955	0.2118	0.957
DenseNet	36.4 minutes	0.94	0.2189	1
ResNet	10.983 minutes	0.865	0.3934	0.8544

Figure 18: Training time and validation performance metrics of the three recent architectures that were tested.

6 Pre-Trained Models and Recent Architectures

In this phase of the project, the validation performance of the final model from the previous phase was improved through the use of two techniques: using pre-trained networks as convolutional bases, and rebuilding the network using recent architectures. The following frozen convolutional bases were tested: VGG16, Xception, ResNet50, and DenseNet201, each of which were fed into layers from the best-performing model of the previous phase. Compared to the outcome of using normalization techniques in the previous model, the inclusion of frozen convolutional bases did not improve the performance of the model as a whole, though the results of each experiment remained relatively strong (as seen in Figure 17). Furthermore, the runtimes of the best-performing models using pre-trained networks were much faster than those of the models constructed using recent architectures.

Three recent architectures were experimented with in this phase: Xception, DenseNet[3], and ResNet[4]. The Xception and DenseNet models performed extremely well, and outperformed the evaluation metrics achieved in the previous phase (Figure 18). The strongest model overall was that which used the Xception architecture, based on a similar model from *Deep Learning with Python*[5]. The model was almost identical to the convolutional model from the previous phase, except for the inclusion of SeparableConv2D layers and a GlobalAveragePooling2D layer to replace most of the Conv2D layers and flatten layer, respectively (Figure 19). This model achieved a validation accuracy of 0.955, loss of 0.2118, and precision of 0.957.



```
inputs = keras.Input(shape=(150, 150, 3))
x = data_augmentation(inputs)
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=8, kernel_size=3, use_bias=False)(x)
x = layers.SeparableConv2D(filters=7, kernel_size=3, use_bias=False)(x)
x = layers.MaxPooling2D(pool_size=4, strides=2, padding="same")(x)
x = layers.BatchNormalization()(x)
x = layers.SeparableConv2D(filters=7, use_bias=False, kernel_size=3, kernel
_regularizer=regularizers.l1_l2(l1=0.0015, l2=0.0015))(x)
x = layers.MaxPooling2D(pool_size=4, strides=2, padding="same")(x)
x = layers.GlobalAveragePooling2D()(x)
x = layers.Dropout(0.56)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
model.summary()
```

Figure 19: Code for the Xception-like architectural model.

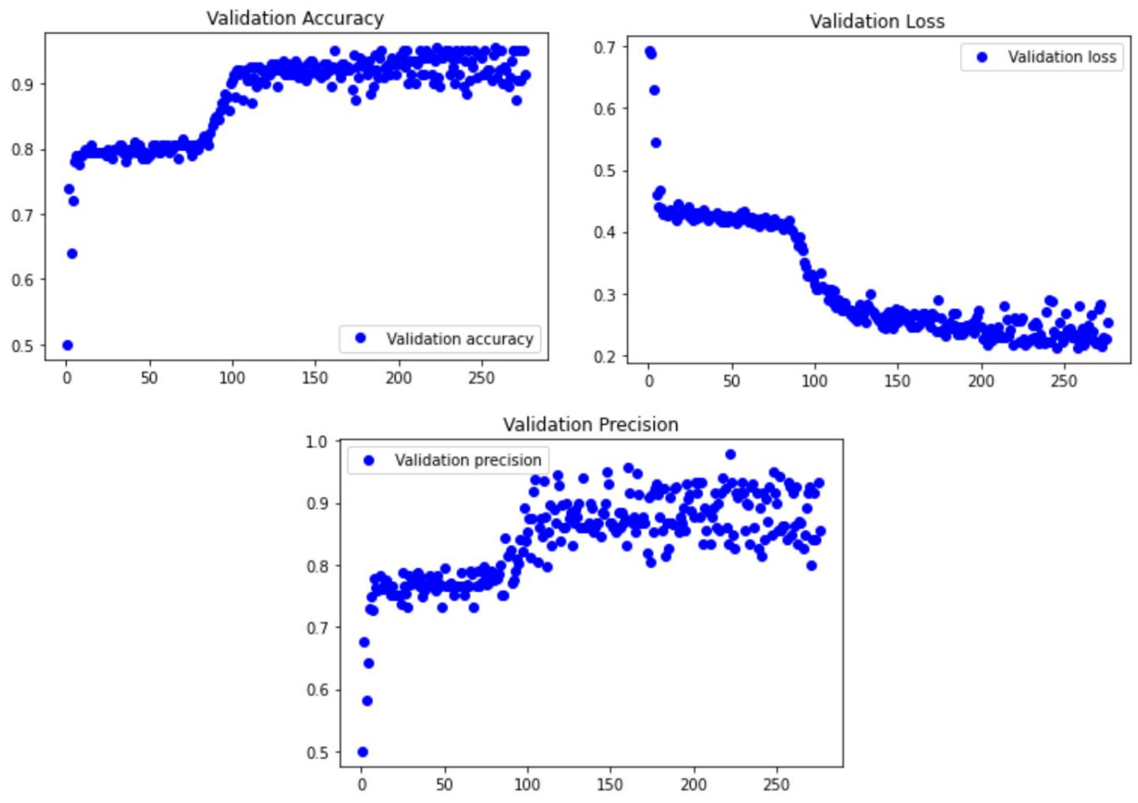


Figure 20: Validation accuracy, loss, and precision of the Xception architectural model.

7 Conclusion

Compared to the expected results of a random classifier, which in this case would be an accuracy of 0.5 since the memes were evenly distributed, the final metrics attained by the model demonstrate that it is capable of recognizing distinct patterns belonging to both classes of memes. Based on the results of experimentation, these characteristics are best perceived using various forms of data augmentation and normalization built into a model implementing the Xception architecture, which alludes to the importance of point-wise convolutions in learning the extremely local patterns of the data.

The results of this project are technically very encouraging, and show that deep learning is capable of helping answer complicated questions from the realm of social sciences. Although neural networks have been shown to adeptly distinguish physical objects and other entities, classifying the tone and motivation of an image is a more complex task. Furthermore, while an internet user can clearly describe what constitutes a meme in general, explaining the characteristics that define a conservative or liberal meme is a far more complicated endeavor, and this model was likely able to capture the visual information needed to discern such meanings and definitions.

In order to prove that the model was able to recognize the patterns that characterize each class, it is now necessary to turn to the techniques of explainability, which will not only provide insight into its decision-making process, but perhaps shed light on the two major viewpoints in question, and highlight within these philosophies motifs that a human perspective may have missed.

References

[1] Merriam-Webster. ”meme”. `https://www.merriam-webster.com/dictionary/meme`, 2022. [Online; accessed 3-February-2022].

[2] 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].

[3] Arjun Sarkar. ”Creating DenseNet 121 with TensorFlow”. `https://towardsdatascience.com/creating-densenet-121-with-tensorflow-edbc08a956d8`, 2020. [Online; accessed 14-April-2022].

[4] Yashowardhan Shinde. ”How to code your ResNet from scratch in Tensorflow?”. `https://www.analyticsvidhya.com/blog/2021/08/how-to-code-your-resnet-from-scratch-in-tensorflow/`, 2021. [Online; accessed 14-April-2022].

[5] Francois Chollet. *Deep Learning with Python.* Manning Publications, 2021.