Political Meme Classification Phase 5

Kate Arendes
Introduction to Deep Learning

April 19, 2022

Contents

Overview

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.

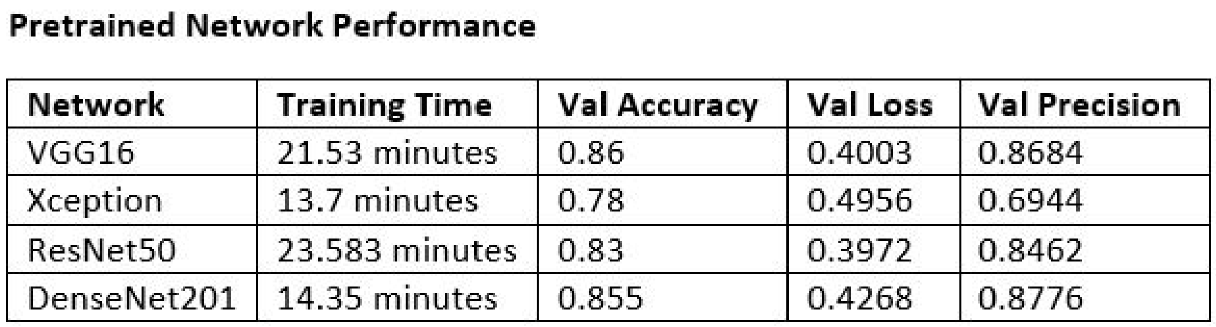
1 Pre-Trained Models

To attempt to improve the network's performance, 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.

When layers from the previous phase's model were not included, and only a dense layer was used as a classifier, the network's performance was reduced drastically. For instance, with only a dense layer for classification connected to the Xception base, the model's validation accuracy remained stuck at 0.5 throughout training.

1.1 Results of Various Pre-Trained Networks

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 1). Furthermore, the



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

runtimes of the best-performing models using pre-trained networks were much faster than those of the models constructed using recent architectures.

Out of the four models tested, the network with the VGG16 convolutional base achieved the best overall validation metrics, with an accuracy of 0.86 and precision of 0.8684. Only the ResNet50 model reached a lower validation loss, but only marginally. The validation performance of the VGG16 model over the course of its training is shown in Figure 2.

2 Recent Architectures

Due to the importance of including the layers from the previous phase in crafting an effective model, the means by which the network could be further improved was through altering its preexisting structure to emulate recent architectures. The architectures tested for this part of the phase were the Xception, DenseNet[1], and ResNet[2] models.

2.1 Performance of Recent Architectures

Out of the three architectures experimented with in this phase, the Xception and DenseNet models performed extremely well, and outperformed the evaluation metrics achieved in the previous phase (Figure 3). The ResNet model did not achieve the same metrics as the other two networks, but rendered results similar to those of the best pre-trained networks.

Despite their promising validation results, the training times of the Xception and DenseNet models were significantly slower than those of the pre-trained models, as the Xception architecture took 246 epochs to attain its best validation loss.

3 Best Performing Model from this Phase

As demonstrated by the validation results of the three different architectures, the strongest model overall was that which used the Xception architecture. Based on a similar model from *Deep Learning with Python*[3], the model was almost identical to the convolutional model from

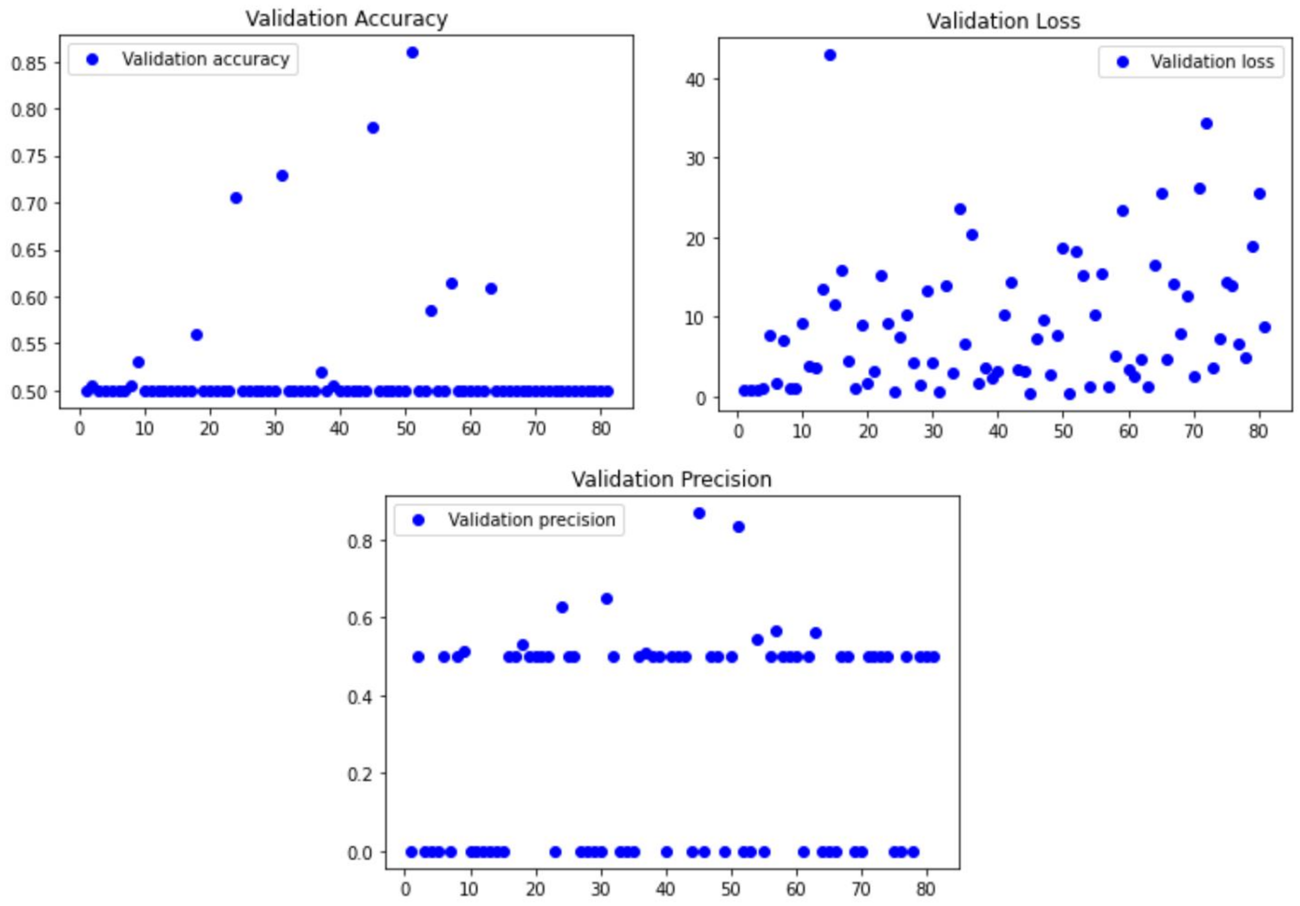
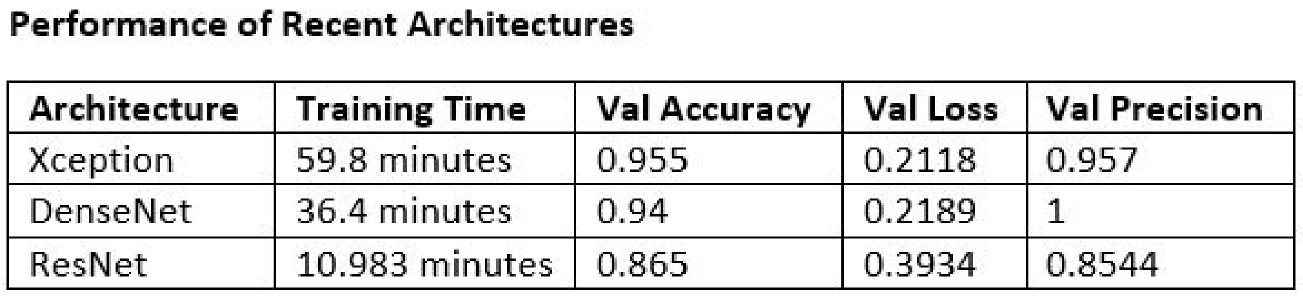


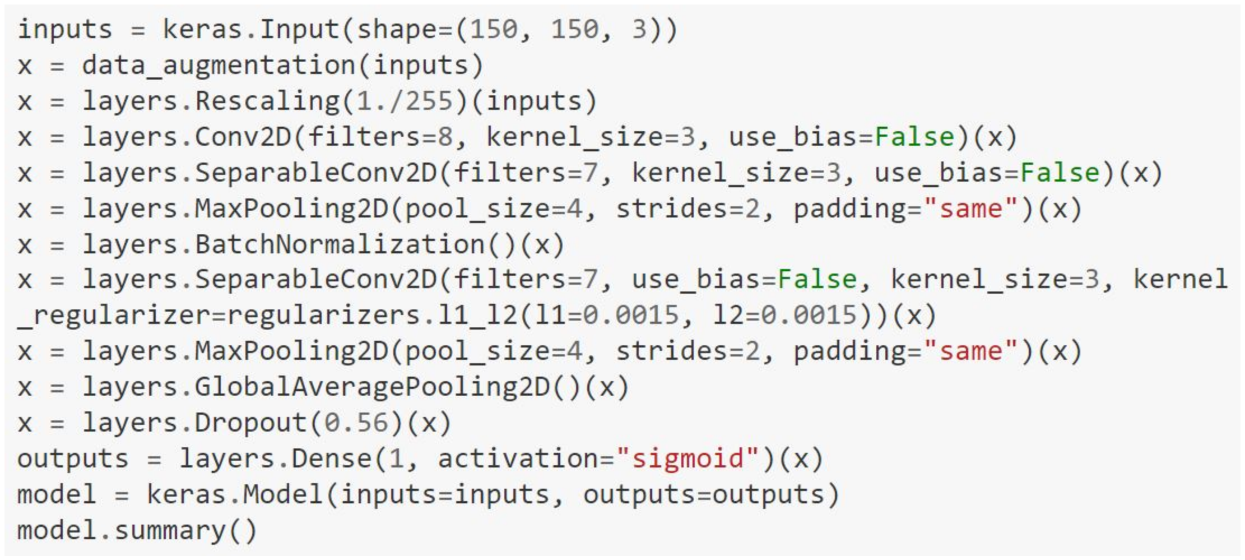
Figure 2: Validation accuracy, loss, and precision of the model using the VGG16 pre-trained network as a convolutional base.



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



```
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 4: Code for the Xception-like architectural model.

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 (Figures 4 and 5). Unlike the DenseNet and ResNet models, the Xception architecture did not make use of any residual connections.

One of the most notable differences between the Xception-like network and the architecture of the model from the previous phase, as well as other recent architectures, is its extremely small parameter count, which was about a third of that of the normalized model. The DenseNet model, which provided a similar validation performance, was much larger and had a total of 6,835,693 parameters, while the Xception model only produced 492 (Figure 6).

3.1 Training and Validation

Out of all models tested in this phase, the Xception architecture trained for the longest time. However, because the training and validation accuracies did not converge, as seen in Figures 7 and 8, the model may have been able to reach an even higher validation accuracy if the EarlyStopping callback allowed training to continue for even more epochs.

This model achieved a validation accuracy of 0.955, loss of 0.2118, and precision of 0.957, all of which were significant improvements over the pre-trained networks and other previous models.

3.2 Evaluation

The evaluation performance of the model using the Xception-like architecture was strong, reaching a test accuracy of 0.905, loss of 0.294, and precision of 0.8462, thus demonstrating the power of the using point-wise convolutions to approach this classification task.

Because the DenseNet-like architecture also performed well in both validation and evaluation, it

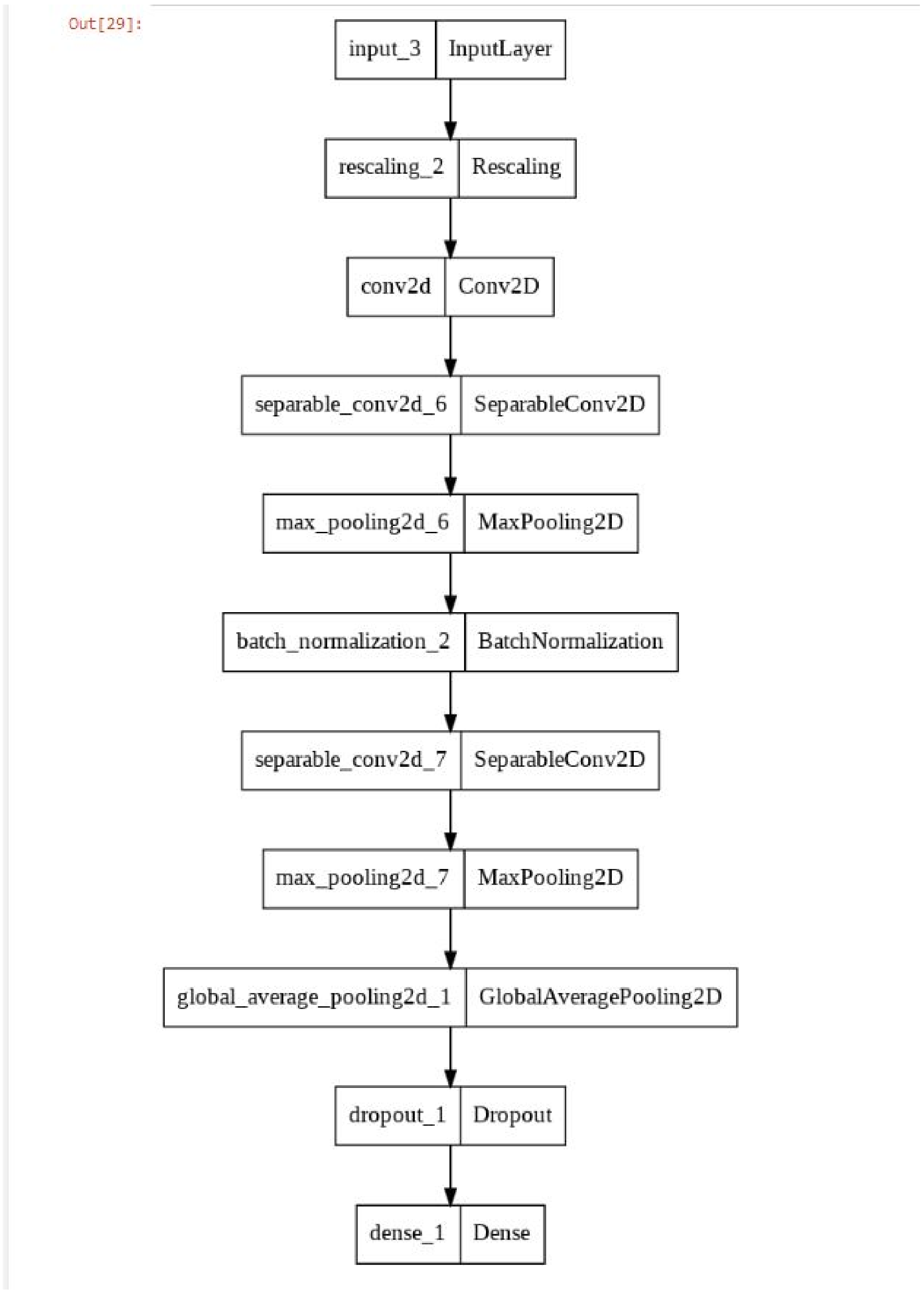
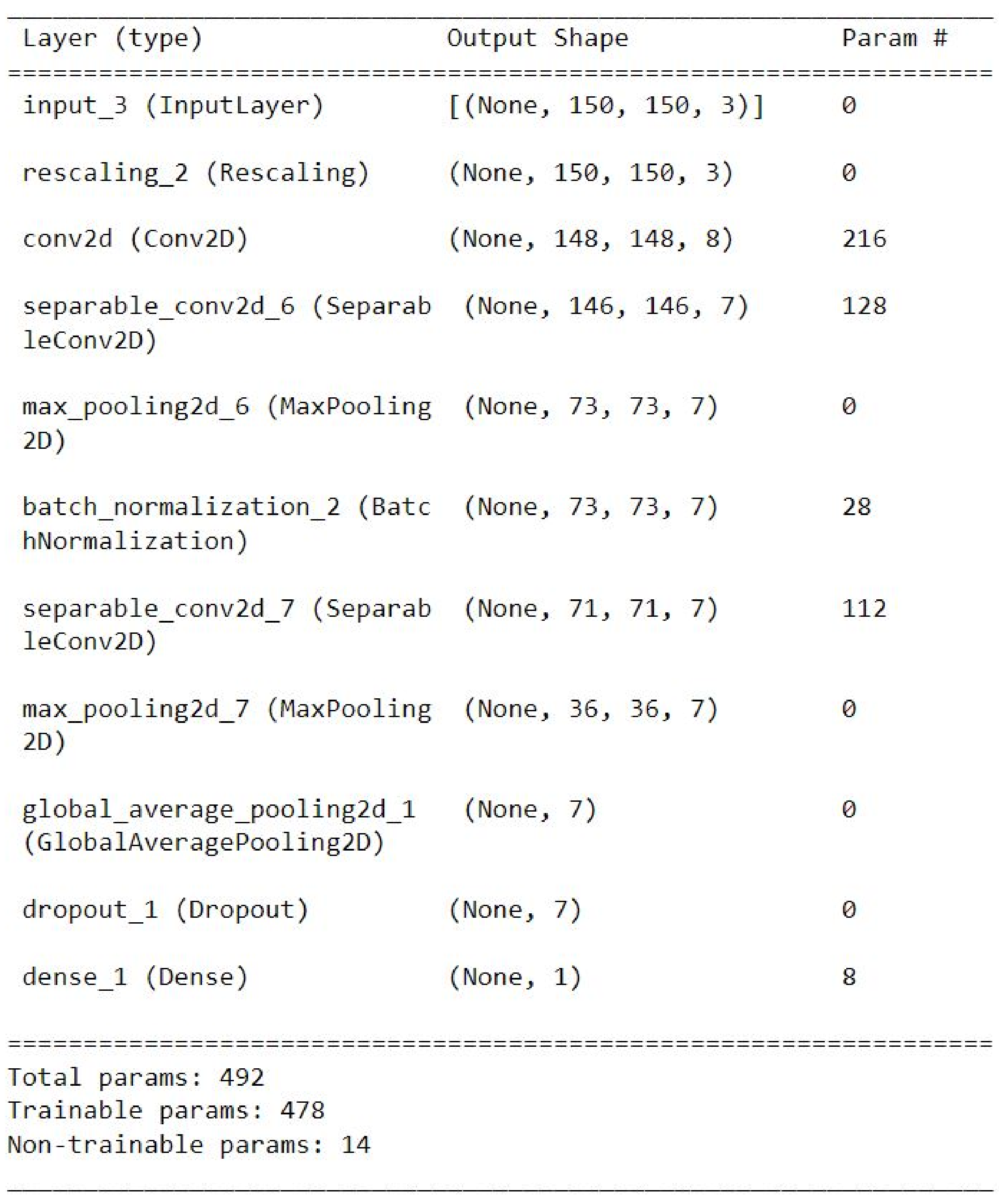

Out[29]:
input_3 | InputLayer
rescaling_2 | Rescaling
conv2d | Conv2D
separable_conv2d_6 | SeparableConv2D
max_pooling2d_6 | MaxPooling2D
batch_normalization_2 | BatchNormalization
separable_conv2d_7 | SeparableConv2D
max_pooling2d_7 | MaxPooling2D
global_average_pooling2d_1 | GlobalAveragePooling2D
dropout_1 | Dropout
dense_1 | Dense

Figure 5: Illustration of the Xception model's architecture.



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 150, 150, 3)]     0

 rescaling_2 (Rescaling)     (None, 150, 150, 3)       0

 conv2d (Conv2D)             (None, 148, 148, 8)       216

 separable_conv2d_6 (Separab  (None, 146, 146, 7)      128
 leConv2D)

 max_pooling2d_6 (MaxPooling  (None, 73, 73, 7)        0
 2D)

 batch_normalization_2 (Batc  (None, 73, 73, 7)        28
 hNormalization)

 separable_conv2d_7 (Separab  (None, 71, 71, 7)        112
 leConv2D)

 max_pooling2d_7 (MaxPooling  (None, 36, 36, 7)        0
 2D)

 global_average_pooling2d_1   (None, 7)                0
 (GlobalAveragePooling2D)

 dropout_1 (Dropout)         (None, 7)                 0

 dense_1 (Dense)             (None, 1)                 8

=================================================================
Total params: 492
Trainable params: 478
Non-trainable params: 14
_________________________________________________________________
```

Figure 6: Parameter distribution for the Xception model.

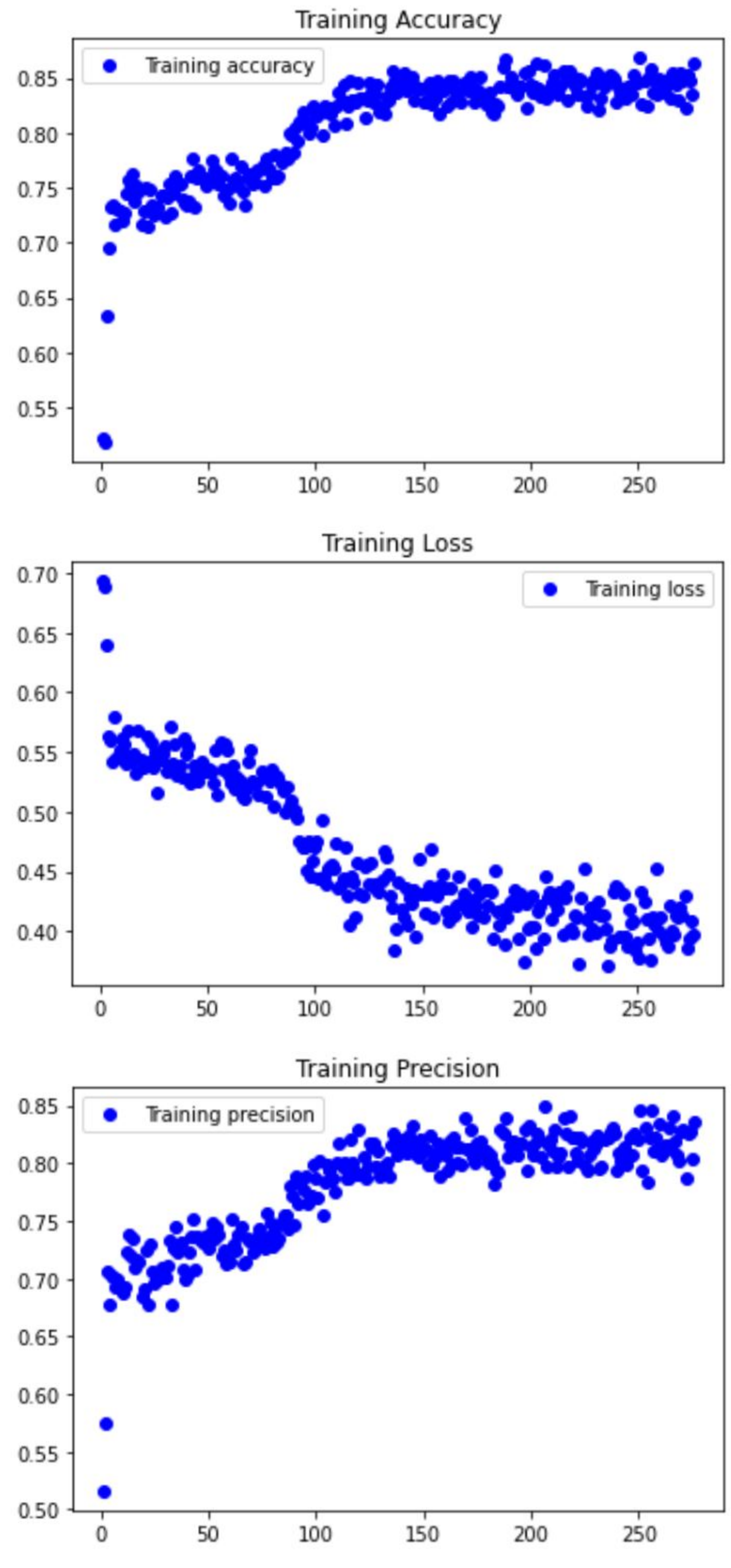


Figure 7: Training accuracy, loss, and precision of the Xception architectural model.

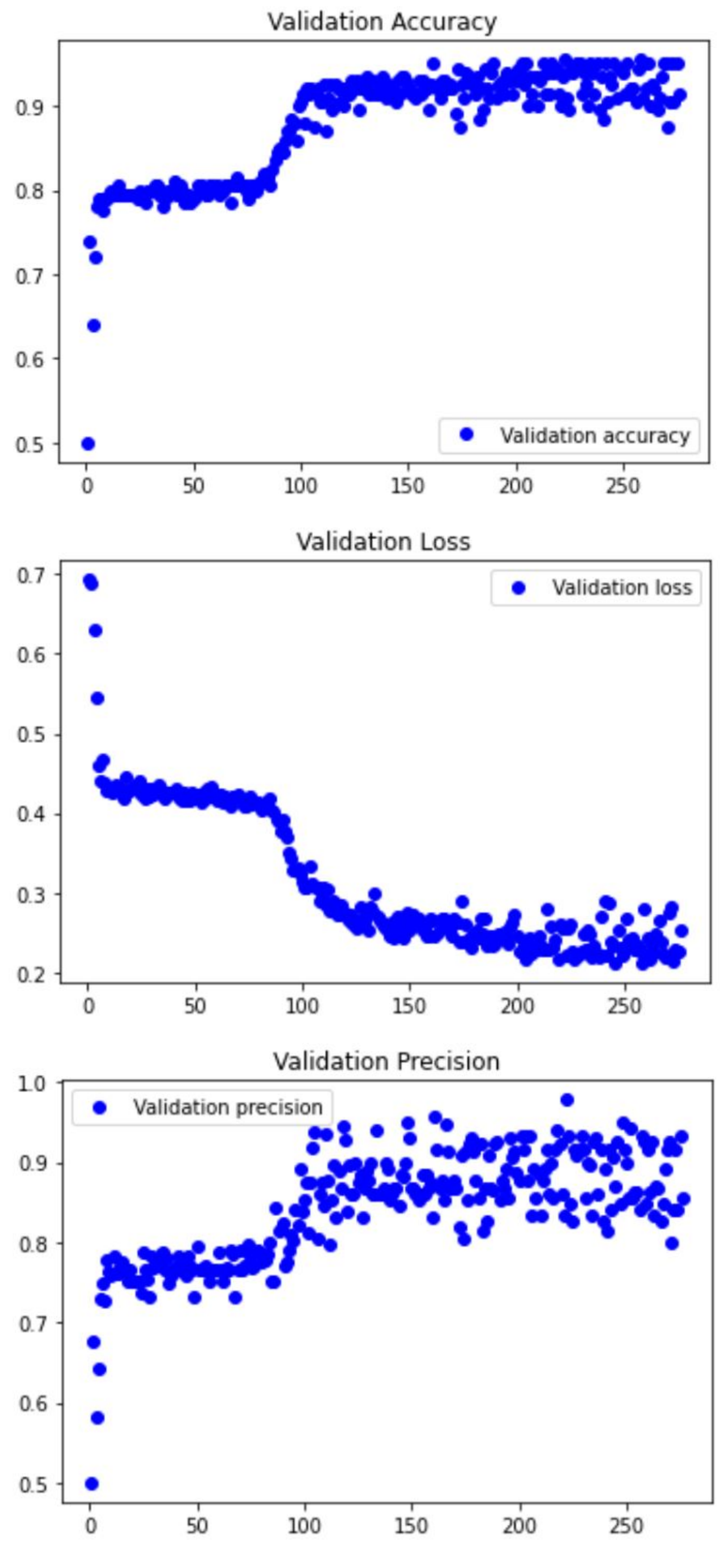
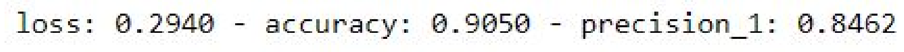


Figure 8: Validation accuracy, loss, and precision of the Xception architectural model.



```
loss: 0.2940 - accuracy: 0.9050 - precision_1: 0.8462
```

Figure 9: Evaluation accuracy, loss, and precision of the Xception architectural model.

was shown that directly altering the model's architecture, rather than solely using a pre-trained network, was the most effective means of improving its performance.

Overall, the results for the Xception architecture show that the network was capable of identifying the patterns, especially those on the extremely local level, necessary to recognize the political affiliation of memes on the internet.

References

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

[2] 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].

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