**Report**

**Model Architecture**

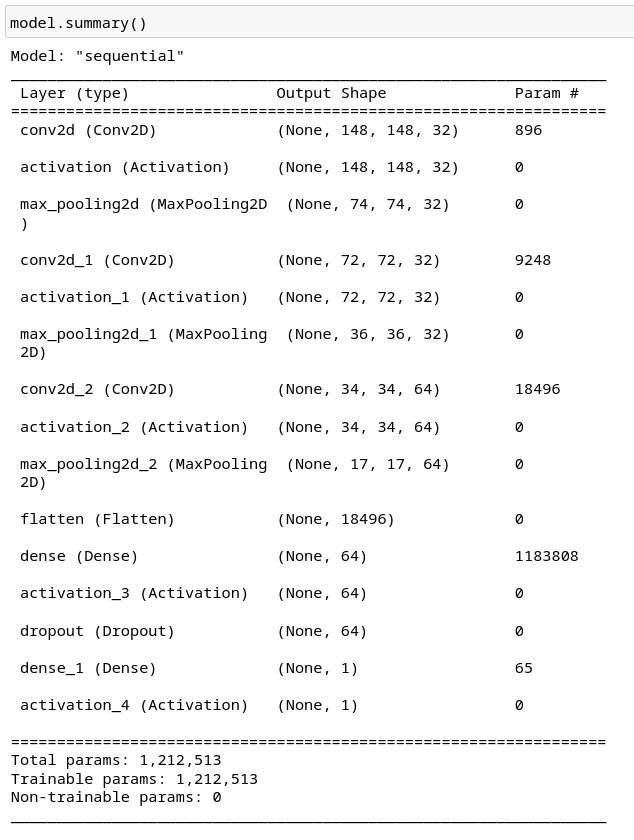
This model is a convolutional neural network (CNN) designed for image classification tasks. It is built using the Keras library and the Sequential model API. The model starts with a convolutional layer (Conv2D) of 32 filters, each with a size of 3x3, and an input shape of 150x150x3, which corresponds to the size and number of channels of the input images. This layer applies the convolution operation on the input images to extract features.

The output of the convolutional layer goes through an activation function, specifically ReLU, which introduces non-linearity to the model, allowing it to learn more complex representations. The output is then passed to a max pooling layer (MaxPooling2D) with a pool size of (2, 2), which reduces the spatial dimensions of the feature maps by a factor of 2. This helps to reduce the computational complexity of the model, and also helps to make the model more robust to small translations in the input images.

The output of the first max pooling layer is then passed through another two convolutional layers, each followed by a ReLU activation and a max pooling layer. The second and third convolutional layers have the same architecture as the first, but the number of filters is increased to 32 and 64 respectively. This increases the capacity of the model to learn more complex features.

After the third max pooling layer, the feature maps are flattened and passed through a fully connected layer (Dense) with 64 units and a ReLU activation. A dropout layer with a rate of 0.5 is added to prevent overfitting. The output of the dropout layer is passed to the final dense layer with 1 unit and a sigmoid activation, which gives the probability that the input image belongs to a certain class.

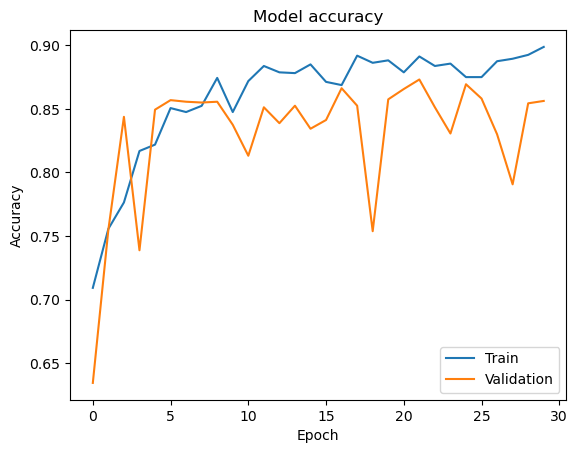
In summary, this model applies a series of convolutional and max pooling layers to extract features from the input images, then uses fully connected layers with dropout to make the final prediction. The model has the capacity to learn complex features from the images and generalize well on unseen data. However, it is important to note that this model is not trained or fine-tuned and it may not perform well on the task it was intended for.



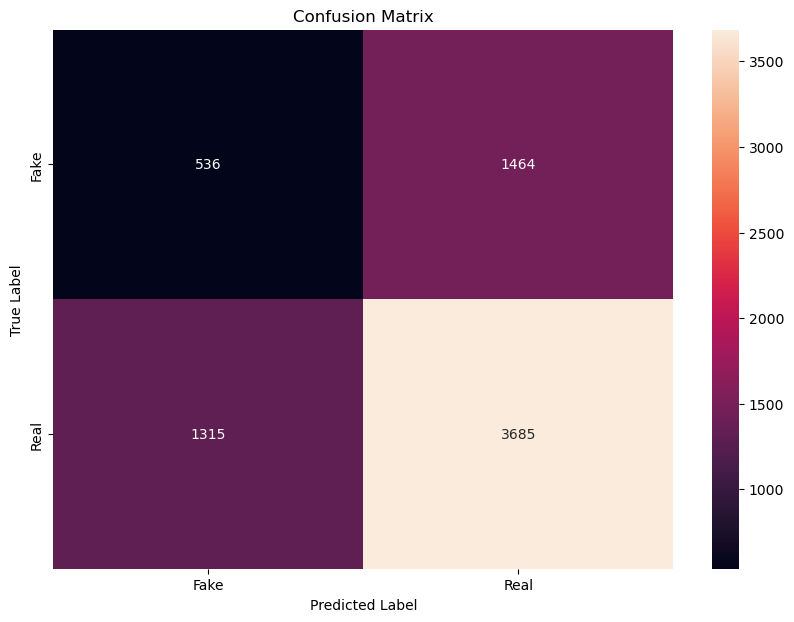
**Model Loss V/s Epoch Graph**

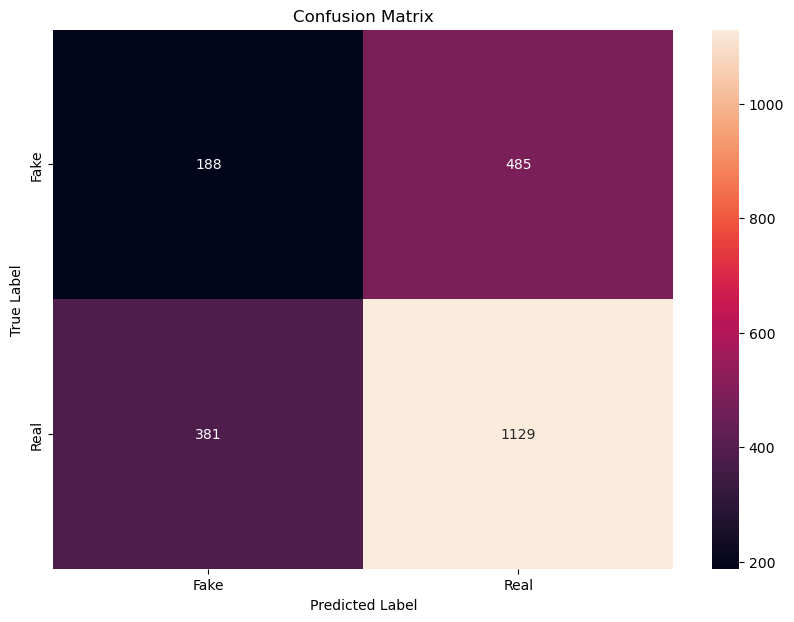
****

**Model Accuracy V/s Epoch Graph**

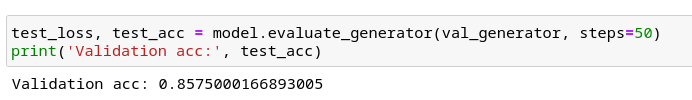
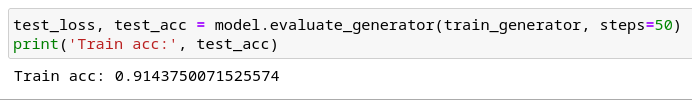
****

**Model Train Confusion Matrix**

****

**Model Validation Confusion Matrix**

**Model Evaluation**

****

The model scored an accuracy of 0.91 approx on the training dataset and 0.85 approx on the validation dataset .**train\_generator and val\_generator are** the training image data and validation image data respectively which containes both the x\_train as well as y\_train. For evaluation purpose here model.evaluate\_generator is used.

The **model.evaluate\_generator()** function in Keras is used to evaluate the performance of a model on a set of data using a generator. A generator is an object that yields batches of data for the model to evaluate on. The function takes in two main arguments:

* generator: An instance of a generator that yields batches of input/output data for the model to evaluate on.
* steps: The number of steps (i.e. batches of data) to evaluate the model on.