Project consist of perform transfer learning by using a pre-trained network for feature extraction. Pre-trained model going to be used is ResNet101.

Colab link for notebook:
https://colab.research.google.com/drive/1y9Sclg05StPZE7PcvcbV4N69SOb-0VUe

Preparing the Data

The data is already splitted into Train and Test dataset. There we total 15 categories. Each category of training part has 360 images out of which 80% (288 images) used for training and 20% (72 images) used for validation purpose. Test has 120 images.
(trainData be the trainingg imges, valData be the validation data and testData be the test data.

Image data generator from Keras has been used to load the data. Same function helped to,

- Re-size the images in 224 × 224 dimension.
- Re-scale the image pixel values from 0 to 1.
- Horizontal flipping
- Shifting
- Rotation
- ResNet preprocessing

Training, validation and test datasets is given as.

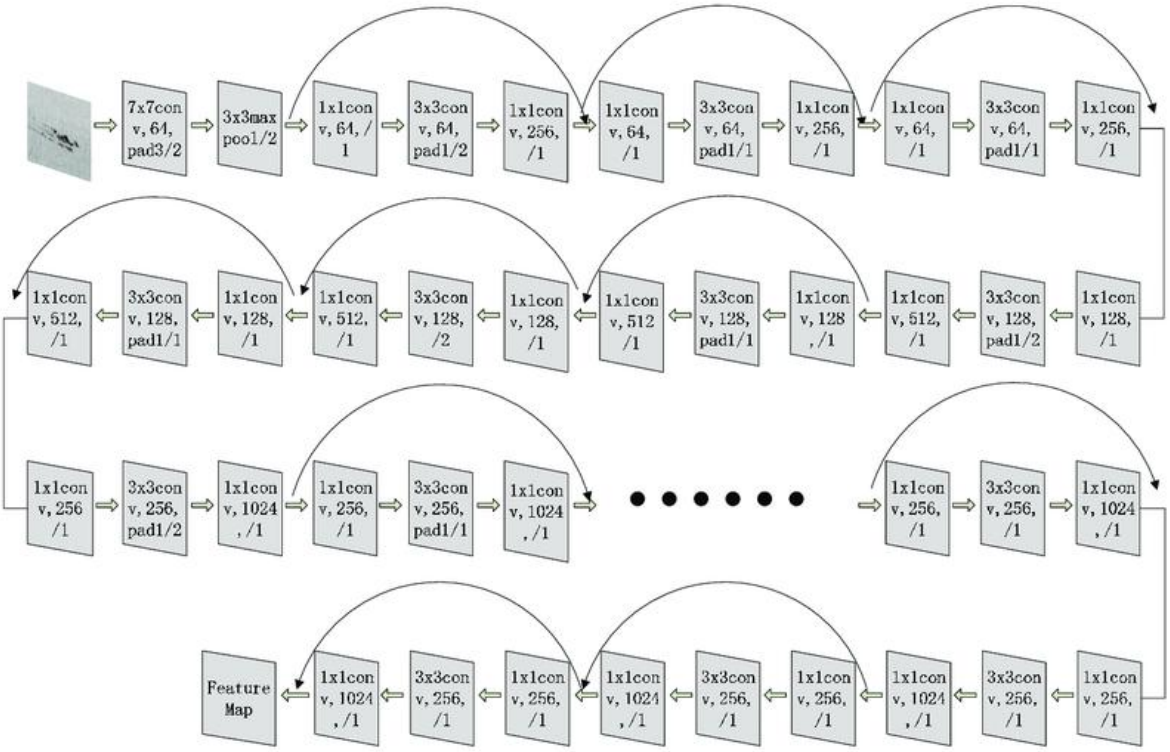
```
Found 4320 images belonging to 15 classes.
Found 1080 images belonging to 15 classes.
Found 1800 images belonging to 15 classes.
```

Model Overview and results

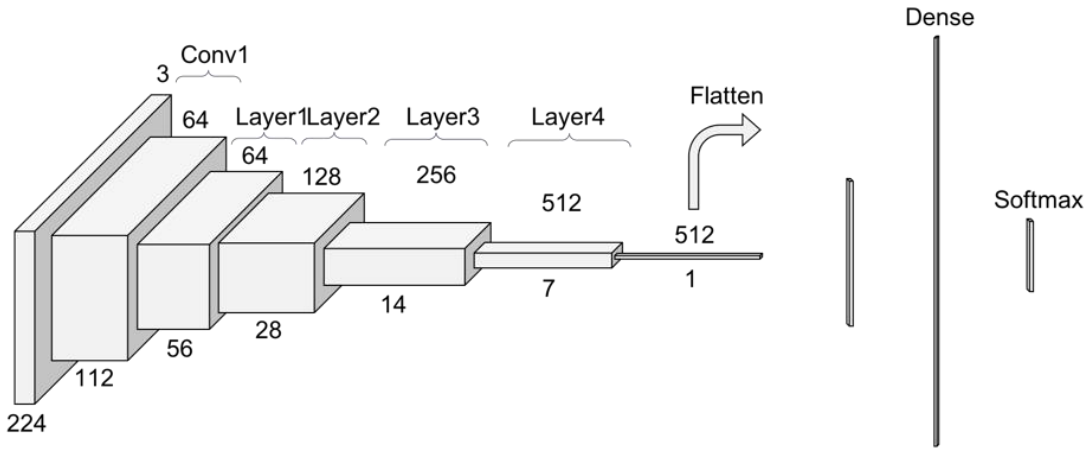
Model 1 (ResNet101)

Below is the architecture of ResNet101 model with classification head,

Feature extraction



Model with 101 layers is difficult to show, below the image of ResNet 34 is given. ResNet 101 has more layers in the same order.



Adding a classification head

 # Feature Extraction=> Average Pooling => Dense(15) => Softmax

To generate predictions from the block of features, average over the spatial 5 × 5 spatial locations layer to convert the features to a single 1280-element vector per image.

Finally a output layer with 15 nodes and activation function “Softmax”

Compiling and training

Loss function: “Categorical crossentropy”

Optimizer: ADAM

Learning rate: 0.001

Number of epoch: 50

Batch size: 32

ResNet freeze

Here we Freeze the convolutional base created by feature extractor. We only train the top classifier.

Output (Training and test result)

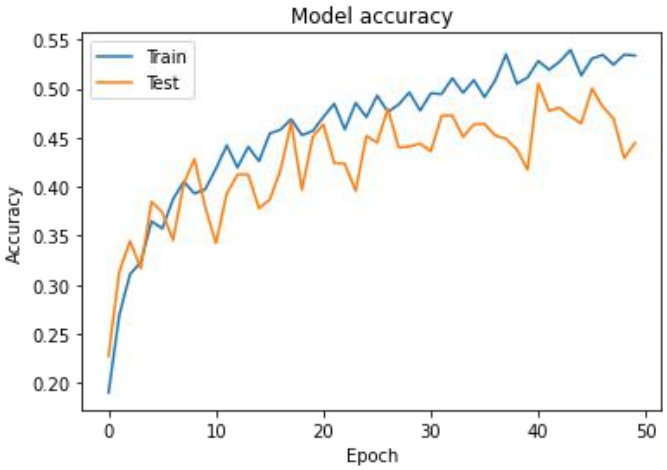
Training accuracy : 53.35 %

Validation accuracy : 44.49%

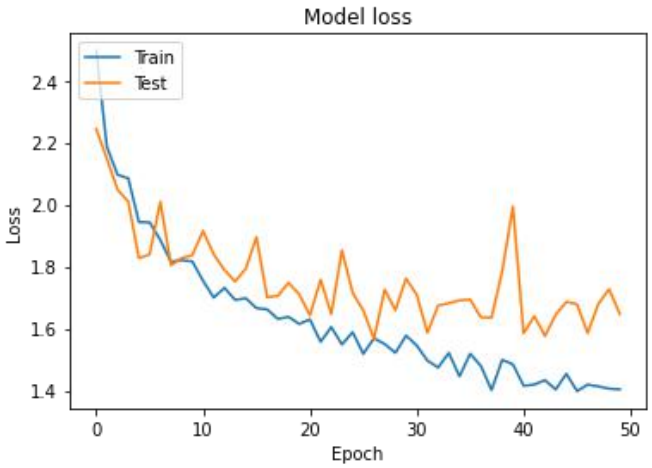
Test accuracy : 46.61%

Plots (Training and Validation data)

1. Training and validation accuracy vs epoch, (Accuracy/Y axis in range 0 to 1)



2. Training and validation loss vs epoch (Loss/Y axis in range 0 to 1)



Conclusion: There is a consistent improvement can be seen but there are some variance. With more epoch it might increase a bit but still model performance is poor.

ResNet unfreeze (Fine tuned)

Here we unfreeze the top layers of convolutional base and set the bottom layers to be un-trainable.

Output (Training and test result)

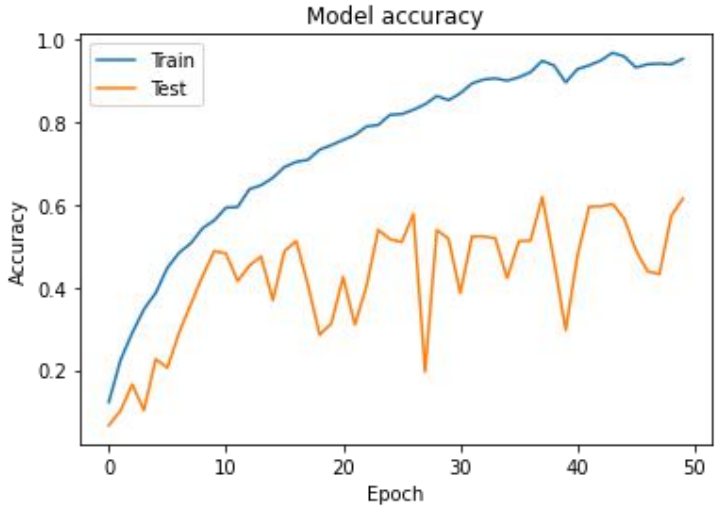
Training accuracy : 95 %

Validation accuracy : 61.52%

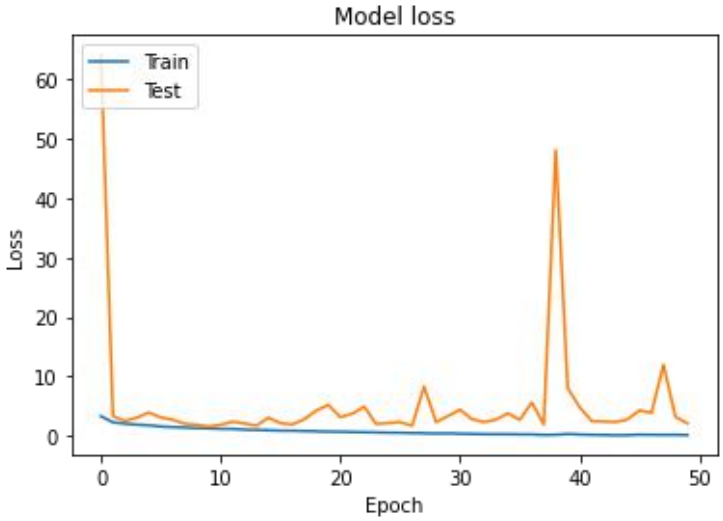
Test accuracy : 63.2%

Plots (Training and Validation data)

3. Training and validation accuracy vs epoch, (Accuracy/Y axis in range 0 to 1)



4. Training and validation loss vs epoch (Loss/Y axis in range 0 to 1)



Conclusion: Model performance is far better on training data but there is definitely overfitting and lots of variance present in the model.

Model 1 (ResNet101 V2)

Adding a classification head

 # Feature Extraction=> Average Pooling => Dense(15) => Softmax

To generate predictions from the block of features, average over the spatial 5 × 5 spatial locations layer to convert the features to a single 1280-element vector per image.

Finally a output layer with 15 nodes and activation function “Softmax”

 # Compiling and training

Loss function: “Categorical crossentropy”

Optimizer: ADAM

Learning rate: 0.001

Number of epoch: 20

Batch size: 32

Output (Training and test result)

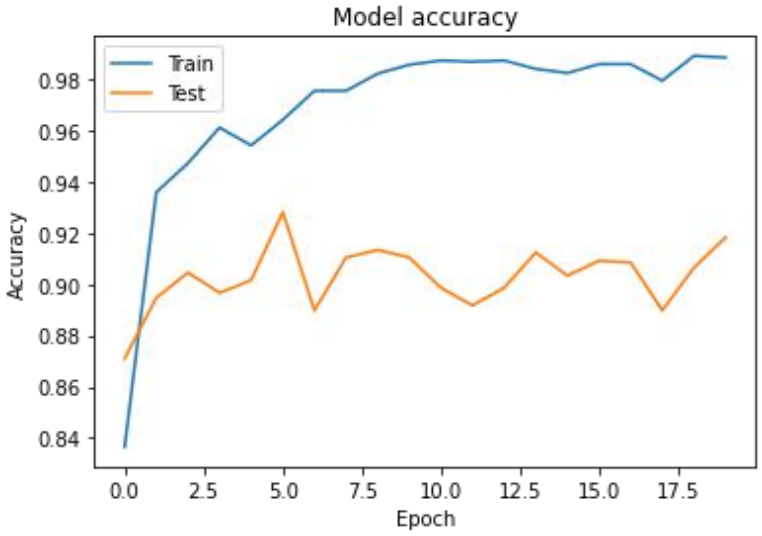
Training accuracy : 98.84 %

Validation accuracy : 91.83%

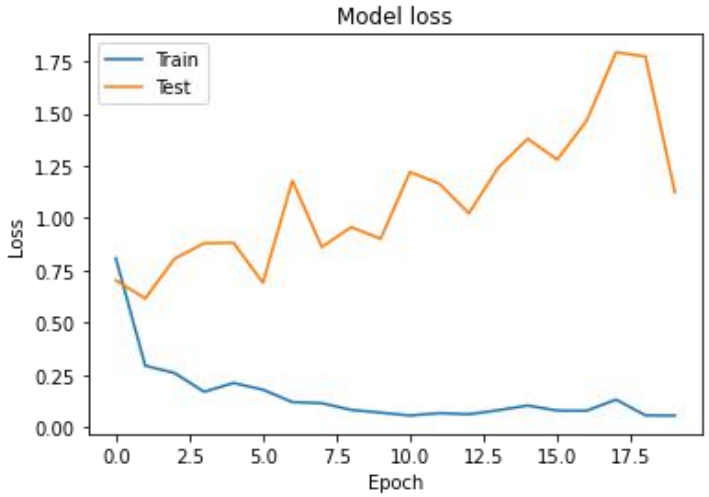
Test accuracy : 90.72%

Plots (Training and Validation data)

5. Training and validation accuracy vs epoch, (Accuracy/Y axis in range 0 to 1)



6. Training and validation loss vs epoch (Loss/Y axis in range 0 to 1)



Conclusion: Model is way better than the two above. Incomparable improvement can be seen but but there are some weird pattern also. Firstly there is some negligible overfitting which can be avoided after fine-tuning. Also we can notice that there is improvement in the validation accuracy but the loss increased.
This can be explained as two phenomenons might be happening at the same time :

- Some examples with borderline predictions get predicted better and so their output class changes (for example, with a binary classification, a cat image predicted at 0.4 to be a cat and 0.6 to be a horse becomes predicted 0.4 to be a horse and 0.6 to be a cat). Thanks to this, accuracy increases while loss decreases.

- Some examples with very bad predictions keep getting worse (eg a cat image predicted at 0.8 to be a horse becomes predicted at 0.9 to be a horse) AND/OR (more probable, in particular for multi-class ?) some examples with very good predictions get a little worse (eg a cat image predicted at 0.9 to be a cat becomes predicted at 0.8 to be a cat). With this phenomenon, loss increases while accuracy stays the same.