Deep Learning Assignment 1

Uploading data:

We mounted the drive to save our important data and to be able to use it whenever it's needed. We use the image data generator to load and split the images with their labels lazily, because if we try to load them all at once, the GPU may crash due to not having enough memory. Using a batch size of 64 speeded up training at the cost of accuracy, so we decided to settle with 32.

Visualization of data:

We use the Counter class (returns a dictionary whose keys are the given list's values and whose values are the number of occurrences of those keys) to count the number of samples for each of the 104 classes.

We took a batch from the train dataset and plotted 10 of them. We used the classes txt file to add the class name as the plot's title.

Common helpers:

We created a section that has some functions that compute important parts of the model instead of repeating them multiple times.

Those functions are:

test_model: Tests a model using the test dataset.

get_predicted_labels: Returns the labels predicted by the model.

get_confusion_matrix: Returns the confusion matrix of a model as a DataFrame.

compute_f1score: Returns the F1 score for each of the classes.

CNN Model from Scratch:

Structure:

We have 4 blocks of convolution, max pooling, and batch normalization; All blocks have 2 convolution layers, max pooling of size 2 with stride equal to pooling size (when latter is changed the stride is changes accordingly), no padding because using "same" padding either decreased the model's accuracy or kept it the same, kernel size (3x3). They are the same for all 4 blocks, but the only thing that changes is the size of filters as it doubles in each block.

Making the model any deeper didn't result in better performance. Using batch normalization after max pooling raised validation accuracy from the range of 30% to the range of 40%.

Then we transition to the fully connected neural network. First, we flatten the data from 2-d to 1-d, then we pass the flattened data into 2 fully connected layers each of size 1024, and using activation function Relu. Finally the last layer is of size 104, one neuron for each class, and using activation softmax so that each neuron represents the probability of that class being predicted. Of the 104 classes, the predicted one is the one with the highest probability.

We used the Adam optimizer with learning rate 10^{-3} and epsilon 0.1, and using Sparse Categorical Crossentropy loss because our labels are encoded as integers.

During Fitting:

We stop training early if after 5 epochs, validation accuracy doesn't increase. We trained the model 4 times. In the first 2, the model was giving good results, then the other 2 the model didn't improve much as it used the stop early many times.

Problems faced:

Used the architecture of vgg16 but accuracy wasn't exceeding 30%, so we used padding = same gave same or maybe worse accuracy so we turned it back to valid.

Then we tried to use more layers, but it decreased the performance and took more time to train, so we decreased the layers.

We tried dropout (0.2), and it improved the accuracy a little bit. So, we tried to add batch normalization and dropout (0.5) and it increased accuracy from around 30% to more than 40%.

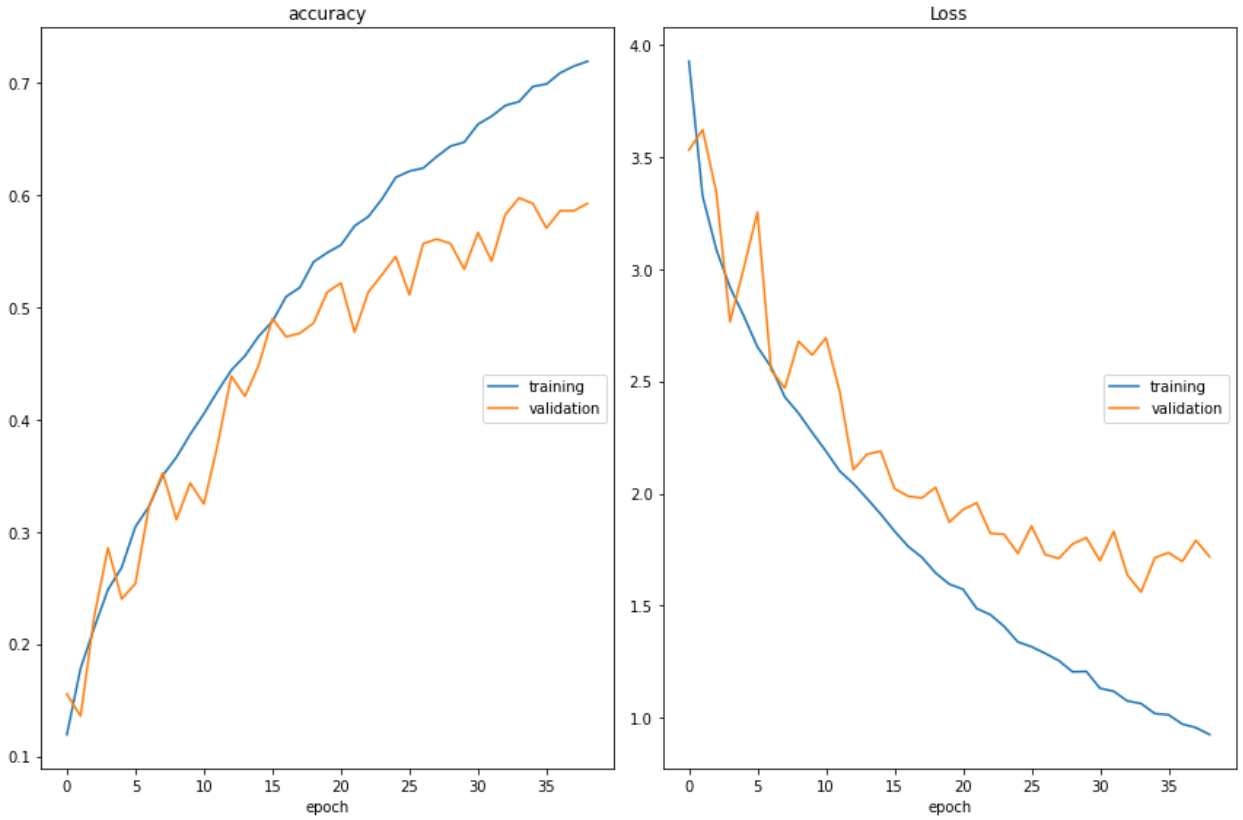
We tried learning decay, but it slowed down the model training and it didn't give good accuracy, so we removed it.

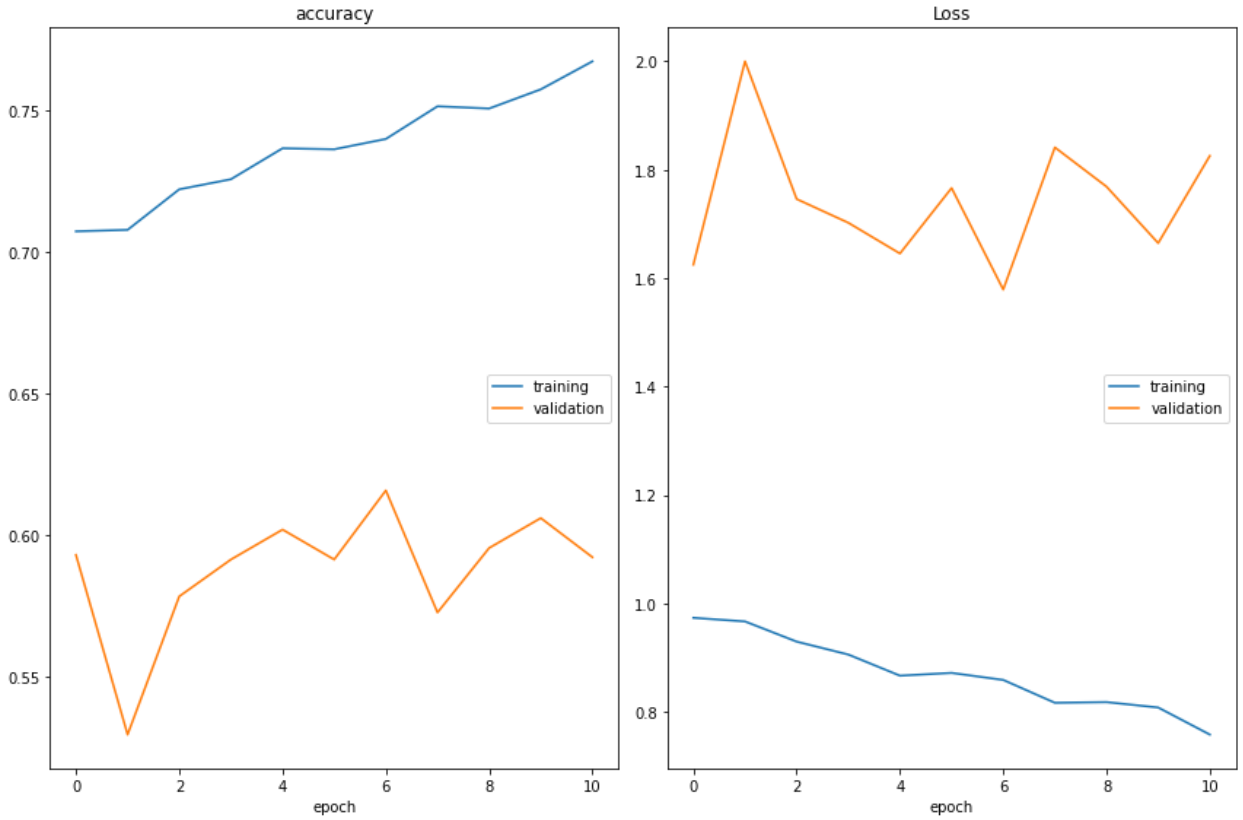
We used image augmentation and it increased accuracy from 40% to around 50% to 60%, but it was slower in training.

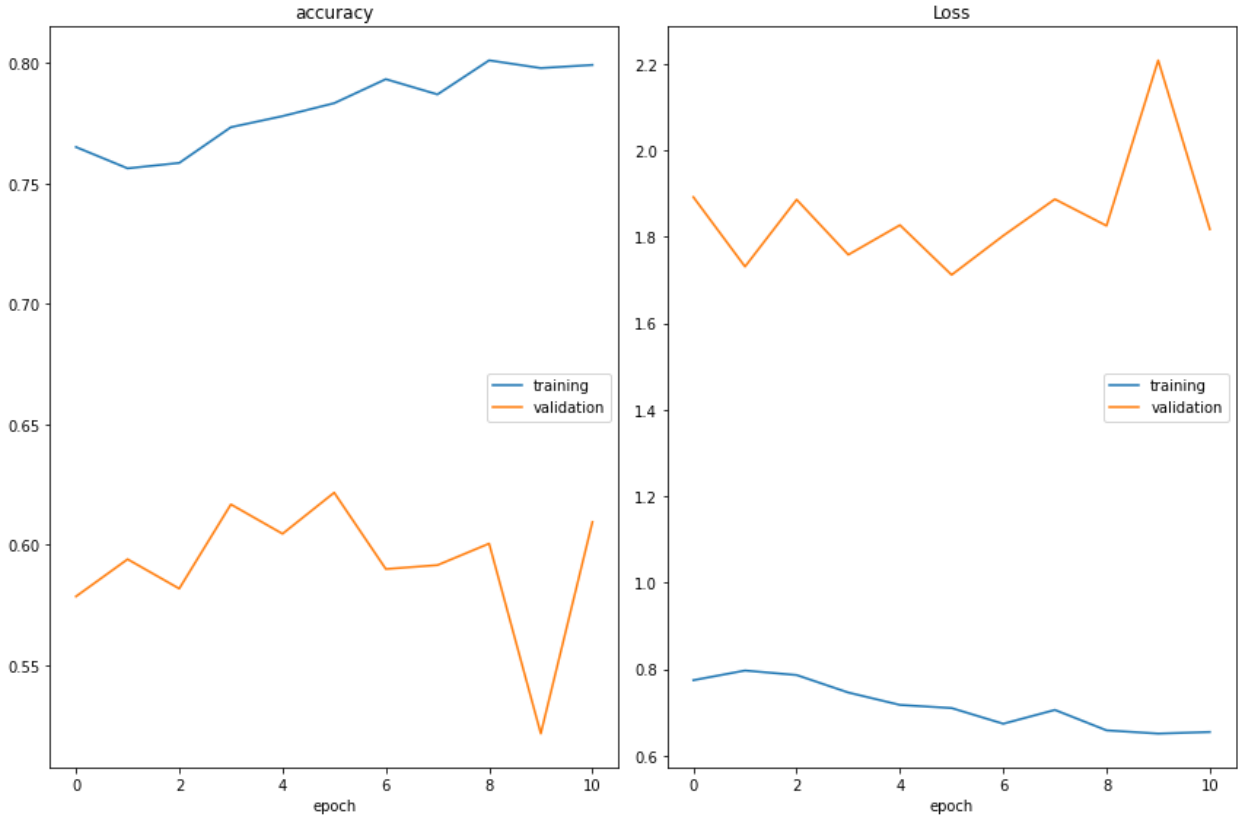
Testing:

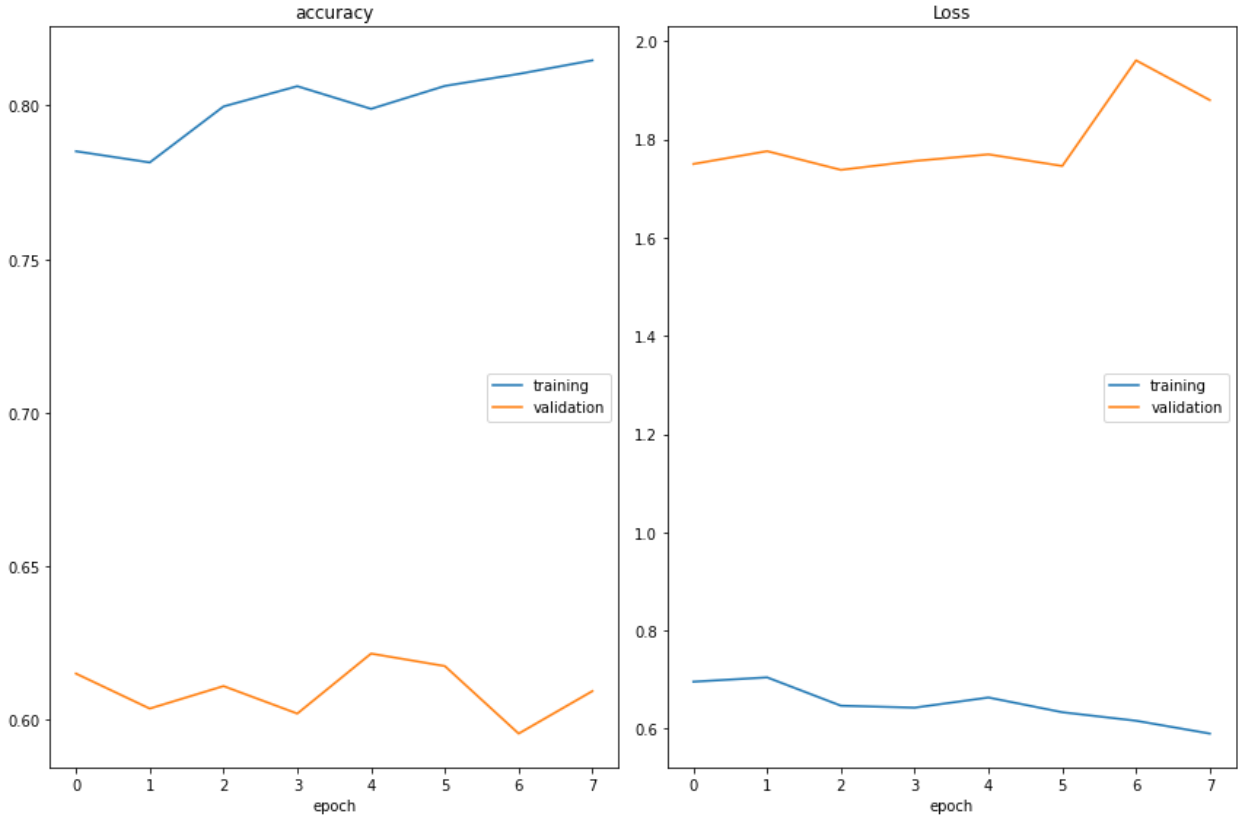
We evaluate the model to check its accuracy; if it is not giving good results, we go back and retrain our model. We send the model to the test model function. It returns the outputs of the model and the true labels.

Training Plot:






accuracy
training
validation
epoch
Loss
training
validation
epoch


accuracy
training
validation
epoch
Loss
training
validation
epoch

VGG16 Model from Scratch:

VGG is a straightforward architecture where it has less complex architecture than other models, but on the other hand it is more complex in parameters as it contains much more params than other models, as a result the taken training time is increased.

Structure:

We have 5 blocks of convolution and max pooling; the first block we have 2 conv layer, max pooling of size 2, stride 2, padding = same, kernel size (3x3) and they are not changeable at the most part of it, after max pooling the size is decreased into half the actual size, the second block is the same as the first, but the only difference is that we increased number of filters into double the first, the third block we have 3 conv layer and the size of filters is doubled, fourth block is the same as the third but doubling the size of filters, the last block just repeats the fourth.

Then we flatten the data and pass them into 2 dense layers of size 4096 and using activation relu.

We used the Adam optimizer with learning rate 10^{-3}, and using Sparse Categorical Crossentropy loss.

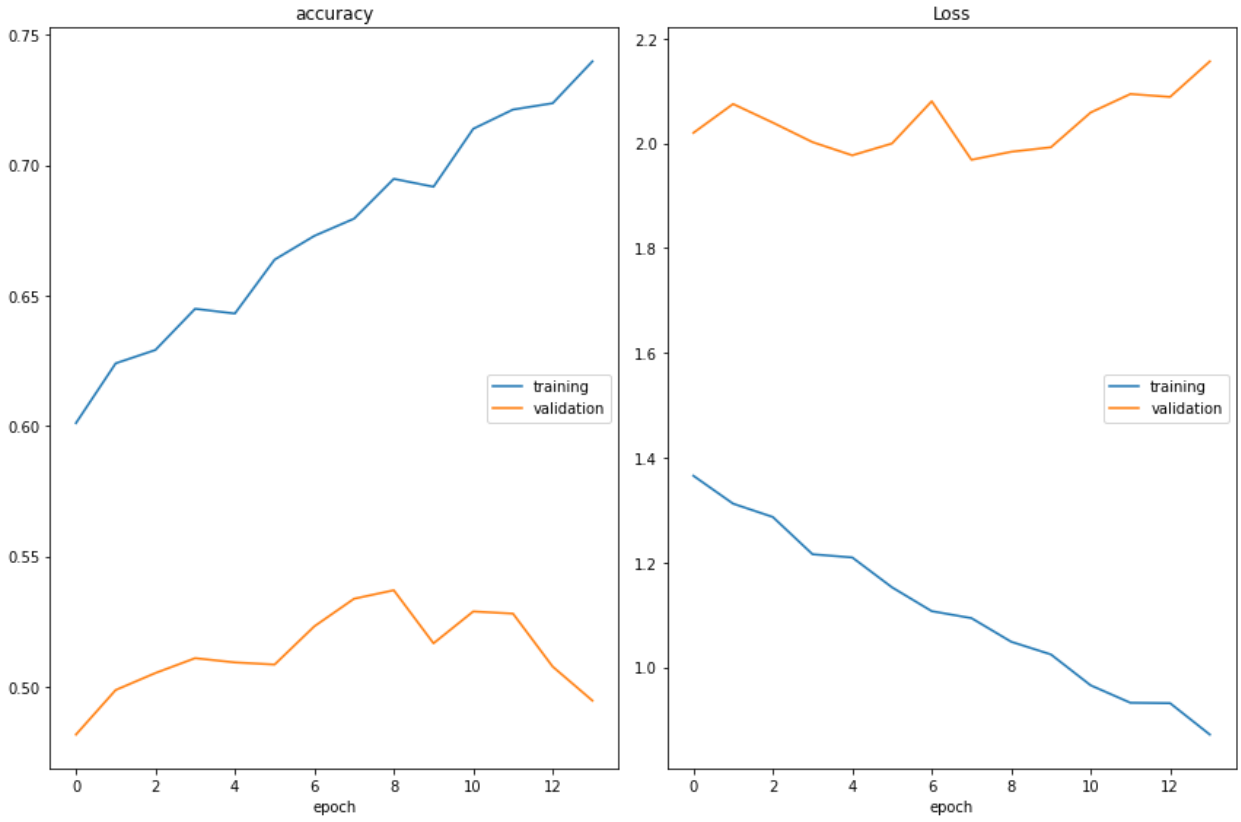
During Fitting:

Validation accuracy started decrease gradually so we used early stopping and where we were using epoch of 30 where it was split into 16 epochs with learning rate 10^{-5} and rest with learning rate 10^{-3}, then we saved the best weights acc to validation accuracy that was provided.

Problems Faced:

Accuracy wasn't the best when using higher learning rates where it gives accuracy around 20% to 30%, after decreasing the learning rate we got accuracy of 40% and we decreased the overfitting. After using image augmentation, the accuracy reached 53%.

Training Plot:



ResNet50 From Scratch:

ResNet first introduced the concept of skip connection to add the output from an earlier layer to a later layer helping it mitigate the vanishing gradient problem, they also allow the model to learn and identity function which ensures that the higher layer will perform at least as good as the lower layer, and not worse.

Structure:

Using a convolution with a kernel size of 7 * 7 and filter of size 64 all with a stride of size 2 giving us 1 layer, we use max pooling with also a stride size of 2.
Then we use 4 blocks of resnet convolution:

In the 1st block there is a 1 * 1 kernel with 64 filters, then a 3 * 3 with 64 filters and at last a 1 * 1 with 256 filters, These three layers are repeated in total 3 time so giving us 9 layers.

In the 2nd block kernel of 1 * 1 kernel with 128 filters, then 3 * 3 with 128 filters and at last 1 * 1 with 512 filters this step was repeated 4 times so giving us 12 layers.

In the 3rd block kernel of 1 * 1 with 256 filters and then two more 3 * 3 with 256 filters and 1 * 1 with 1024 filters and this is repeated 6 times giving us a total of 18 layers.

In the 4th block kernel of 1 * 1 with 512 filters and then two more 3 * 3 with 512 filters and 1 * 1 with 2048 filters and this was repeated 3 times giving us a total of 9 layers.

After that we do an average pool and end it with a fully connected layer containing 1000 nodes and at the end a softmax function, so this gives us 1 layer.

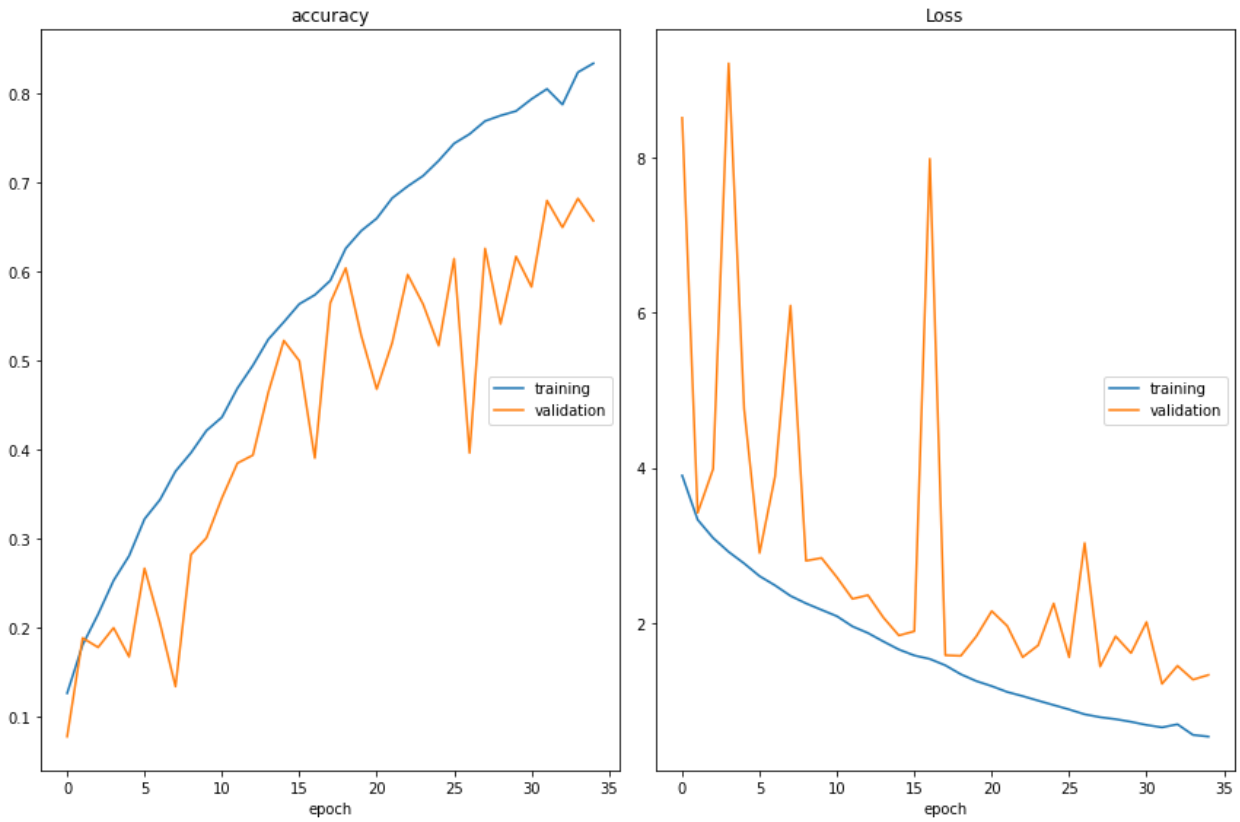
We don't count the activation functions and the max/ average pooling layers. so, total is 1 + 9 + 12 + 18 + 9 + 1 = 50 layers Deep Convolutional network.

Problems Faced:

Using the optimizer SGD made the model overfit so we changed it to Adam optimizer, so it gave optimal results.

We increased the learning rate from 10^{-4} to 10^{-3} as it gave better results.

Training Plot:



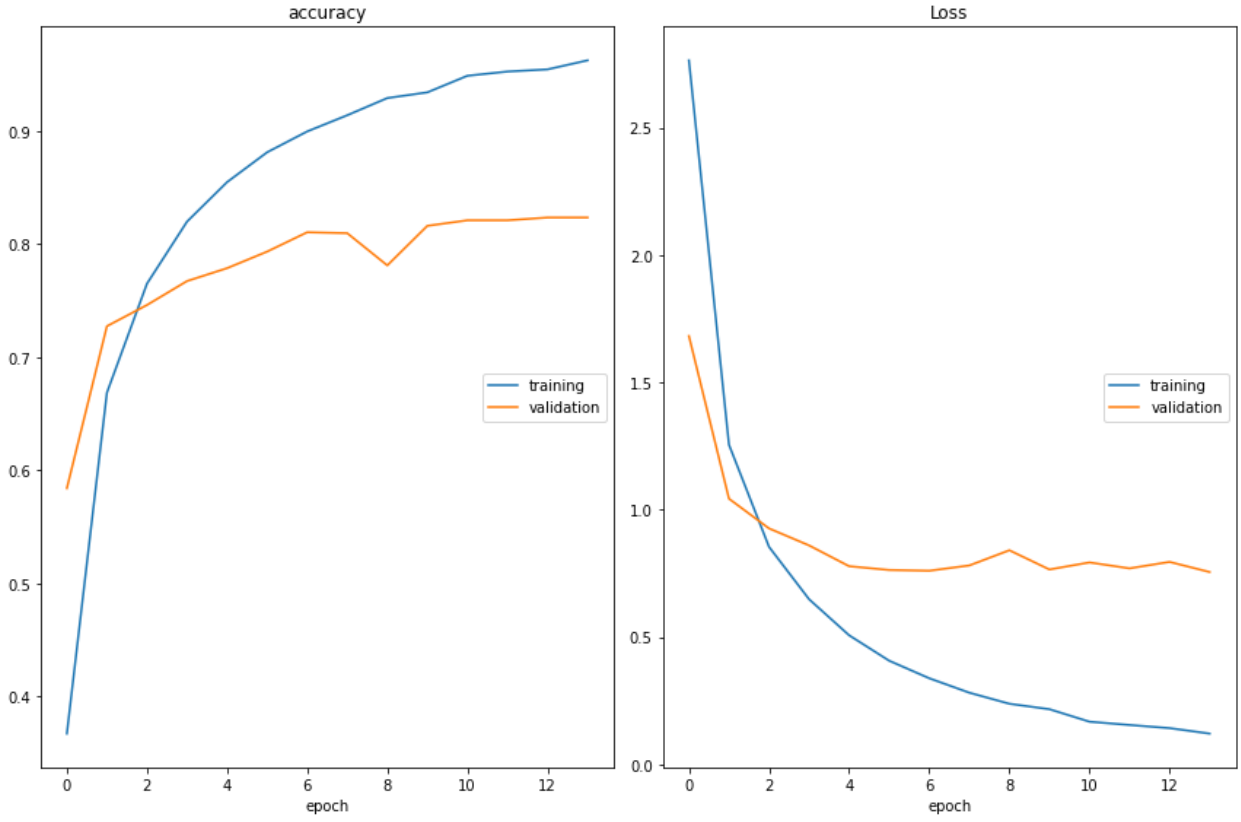
Pretrained VGG19:

We Used the vgg19 class from keras library, we used pretrained weights from imagenet and, we used include top false because the vgg have 1000 output and we have 104 classes, using optimizer Adam with learning rate $10^{-5}$and we added dense layers 4096 with activation relu and using Sparse Categorical Crossentropy loss

Problems Faced:

We faced overfitting because we used large learning rates (10^{-2}, 10^{-3}) so we tried 10^{-6}, but it resulted in underfitting, so we found that 10^{-5} gave the most optimal results.

Training Plot:



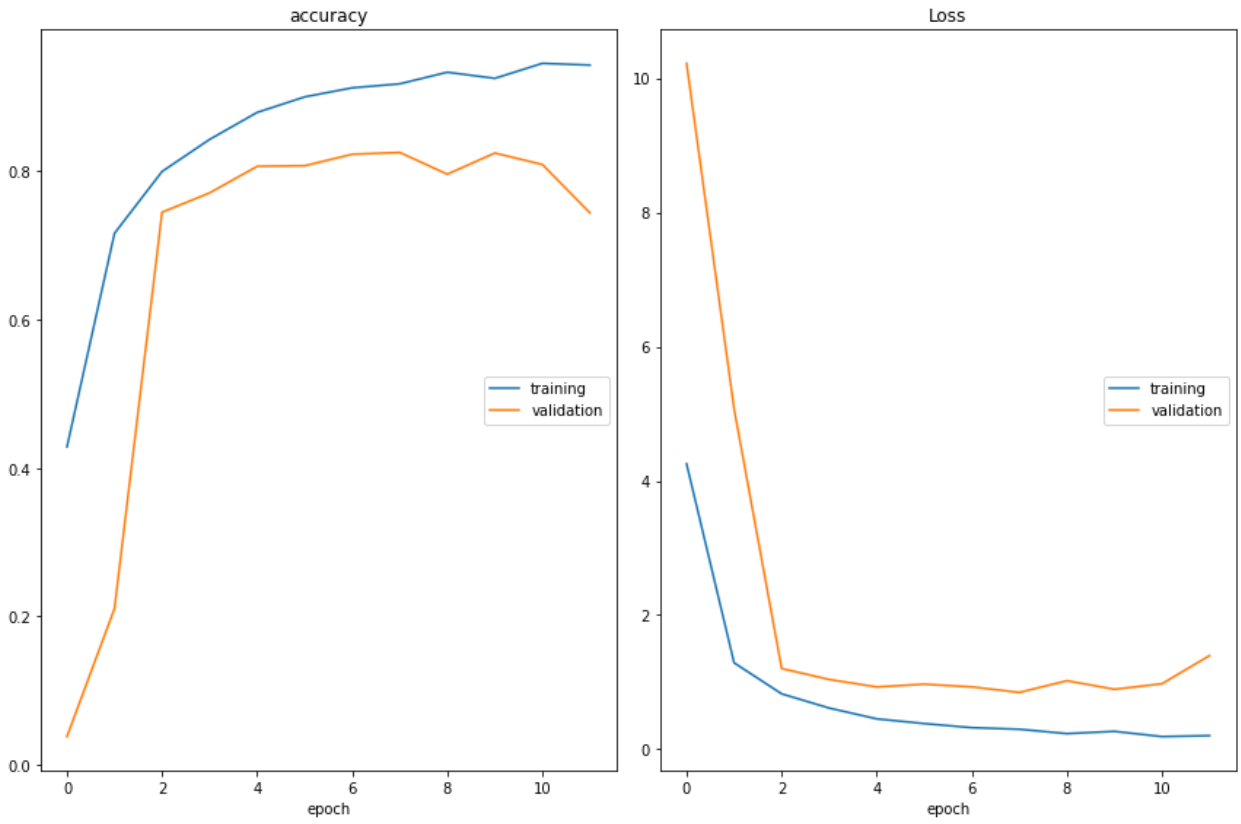
Pretrained ResNet50:

We Used the ResNet50 class from keras library, we used pretrained weights from imagenet, and we used include top false because resnet have 1000 output and we have 104 classes, using optimizer Adam with learning rate $10^{-4}$and we added dense layers 104 Sparse Categorical Crossentropy loss

Problems Faced:

We faced overfitting because we used large learning rate 10^{-3} so we used 10^{-4} and it gave better results.

Training Plot:



	Scratch	VGG 16	VGG 19	ResNet	ResNet pre	Ensemble
Accuracy	0.6501	0.5515	0.8225	0.6926	0.8106	0.7597
Loss	1.6879	2.8348	0.8649	1.2908	1.0654	0.9909
F1 Score	0.61001	0.5288	0.8015	0.6455	0.7848	0.7360